International Journal of Engineering and Manufacturing Science. ISSN 2249-3115 Vol. 7, No. 1 (2017)

DeepCoder: An Approach to Write Programs

Anshita Saxena [1*], Aniket Saxena [2], Jyotirmay Patel [3]
Meerut Institute of Technology, National Highway 58, Bye-Pass Road, Baral Partapur, Meerut, Uttar Pradesh

Abstract- This paper presents a solution for solving simple and short problems in a competitive programming by bringing an approach of deep learning techniques from the set of input-output examples. In the deep learning approach, we usually train a neural network in an order for it to forecast the behavior or the properties of that program which generates outputs from inputs within the corresponding input-output example. This approach uses the predictions in order to increase the amount of search techniques including enumerative search technique and that of satisfiability modulo theories based on solving techniques
Keywords: DeepCoder, Enumerative Search Techniques, Neural Network, Satisfiability modulo Theories (SMT).

1. Introduction

DeepCoder is the way to learn how to write programs. It is highly influenced from the field of Deep Learning, an artificial neural network that composed of many layers. It writes code of six to seven lines means short codes that are basically for solving programming competition-style problems from input-output examples. The DeepCoder is based on the following notions:

(a) To learn strategies that generalize across problems so that any problem can solve in one go.
(b) Combine several neural network architectures to increase the optimality of output rather than replacing one neural network with another like feed-forward network and RNN (recurrent neural network).

Integration of several neural networks is an important task to make the system to work in an optimized manner as the RNN has the property of preservation of the network state that is it can remember its previous state but feed-forward neural network cannot remember its previous state while the performance of feed-forward network is better than the RNN since feed-forward neural network pass the data forward from input to output, while recurrent network can be fed back into the input at some point before it is fed forward again for further processing and final output.

This approach is quite a contrast to the usual work on differentiable interpreters in which the main comes by defining a mapping from both source code and inputs to outputs which has a differentiable nature. Via differentiable interpreters, the computer programs are colossally induce from input-output examples which is called as Programming by Example (PBE). Recent work on differentiable interpreters, on the other hand, insist onto the contiguous space of programs so that gradient- based optimization can be used in order to search over several programs present in the search space in order to obtain suitable program corresponding to the inputs and outputs generated. So rather than introducing a new hybrid system consists of traditional Program Synthesis and methods like gradient descent, DeepCoder uses the approach in which augment existing methods. Hybrid system uses Program Synthesis consists of Task, Data types such as int, int[], bit vectors, string, float and Method such as gradient descent. Augment existing methods include two methods Program synthesis and Deep learning. Program synthesis consists of Task, Data types such as int, int[], bit vectors, string, float and Methods such as enumerative search and constraint solving. Deep learning consists Task, Data types such as perceptual and big data and Method such as gradient descent. DeepCoder makes use of the inductive program synthesis (IPS) problem that produces the program which has behavior consistent with input-output examples and for that we use the approach called LIPS (Learning inductive program synthesis).

In this paper, DSL (Domain Specific Language) has been introduced for the DeepCoder for reducing the complexity and search space. DeepCoder uses TerpreT as DSL which is discussed in sect.4.

2. The Synthesis Program

Synthesis Program provides the simplest program. DeepCoder uses the Synthesis Program because of the mentioned property as it provides minimal cost among many programs in order to fit the examples in the language. For the sake of simplest program, synthesis is based on three main technical notions: inductive generalization, deduction and enumerative search.

These notions are applied on the hypothesis that is the program that may have placeholders for the missing expressions.

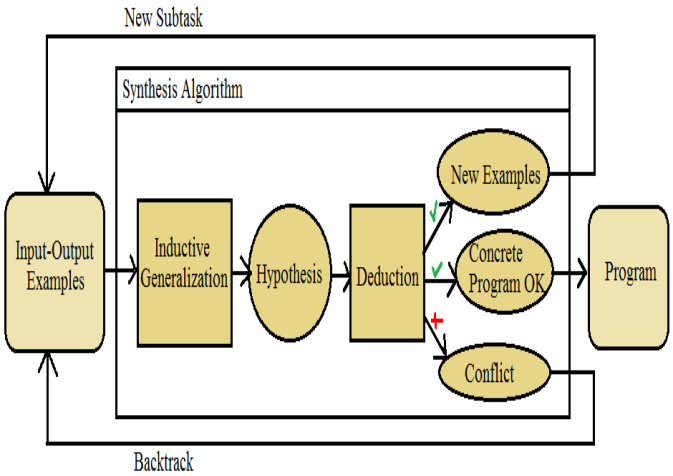

Fig. 1 Process of enumeration search
(Source: Synthesizing Data Structure Transformations from Input-Output Examples [1])

(a) Inductive Generalization: It generalizes the input-output examples into the hypothesis based on the target program structure rather than blindly search for the target program.
(b) Deduction: It is used for learning a new input-output examples that is to guide the search.
(c) Enumeration Search: Generalization and deduction are complemented by enumerative search that is used to identify the hypothesis that can be realized into programs that fit the top level goals. The input-output examples are generalized into hypothesis using Inductive Generalization, the deduction is performed on the hypothesis. If there are new examples synthesized then creation of new subtask is taken place, if the program is found correctly then the program is satisfied and if the conflict is taken place then this corresponds to a form of backtracking in the overall algorithm. The synthesis algorithm performs the Enumeration Search.

3. Concept of Inductive Program Synthesis (IP

There are two main problems which arise while forming an IPS system that is:

(a) To find a consistent program from the appropriate set of the possible program with the help of searching over the set
(b) We should rank if there are multiple programs which are identified to be consistent with input-output examples.

Domain Specific Language and search techniques are the foundation for inductive program synthesis.

The language which is used in DeepCoder is the DSL (Domain Specific Language). These are the programming languages that are suitable for the specialized domain and more restrictive than the full-featured programming language for example C++. The search space will be enlarge and involve complex synthesis if there is a full-featured language. So, for searching, it will use enumeration and pruning (eradication of irrelevant search space).

The constraints written in the programming language is solved by the SMT (Satisfiability Modulo Theories) that integrate SAT-style search with arithmetic and inequality theories. SMT is the form of constraint satisfaction problem (CSPs are mathematical problems defined as a set of objects whose state must satisfy a number of constraints or limitations.) and thus a certain formalized approach to constraint programming (It is the programming paradigm wherein relations between variables are stated in the form of constraints).So, program synthesis engines(Sketch and Brahma) convert the semantics of a DSL into a set of constraints between variables representing the program and input-output values and call an SMT solver to find the satisfying setting of the program variables. But this approach is not sufficient for the complex DSLs.

Since DSL has less features so it reduces the difficulty of ranking problem. There are two approaches for solving ranking: firstly, to choose the shortest program from the pool of consistent programs and secondly, to assign scores to program such that higher scores to the ground truth programs than the other programs that satisfy the input-output specification, where the ground truth programs are given.

4. Introduction to Learning Inductive Program Synthesis (LIPS)

DeepCoder uses learning inductive program synthesis (LIPS) which includes DSL (Domain Specific Language that is choice of the language particular to the problem), data generation strategy, machine learning model (to encode input-output program space) and searching algorithm (searching over the program space). Categorization of LIPS is shown in the figure 2.

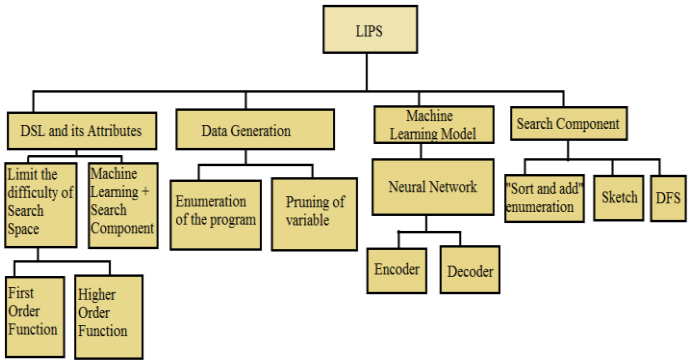

Fig. 2 Categorization of learning inductive program synthesis (LIPS)

4.1 Significance of DSL and its attributes

In Lips, the DSL is important because it is restricted to limit the difficulty of the search as well as it is expressive to consider the particular problem while the attribute is important to connect machine learning model and the search component of LIPS because by machine learning, it would be predictable from input-output examples and by search component its value reduces the search space size. Therefore, DSL contains constructs that are high level so the prediction of their occurrences from input-output examples can be learned successfully.

The example given below is framed in the TerpreT Language released by Microsoft in August 15, 2016. For DeepCoder, the chosen DSL is the TerpreT Programming Language. TerpreT is a popular probabilistic language programming language for program induction in which we express program synthesis problems.

TerpreT has two main benefits:

(1) It allows rapid exploration of range of domains and interpreter models.
(2) It separates model specification from inference algorithm.

From a single TerpreT specification, it enables automatically performing inference using four different back-ends. These are based on Gradient-descent, linear program (LP) relaxations for graphical models, Discrete Satisfiability solving and Sketch programming synthesis system. DSL has two types of functions that are first order functions such as HEAD, LAST, TAKE, DROP, ACCESS, MINIMUM, MAXIMUM, REVERSE, SORT, SUM, and the higher-order functions MAP, FILTER, COUNT, ZIPWITH, SCANL1. Behavior of higher order functions would be fully specified if it has suitable functions such as for MAP our DSL provides lambdas (+1), (-1), (*2), (/2), (*(-1)), (**2), (*3), (/3), (*4),

(/4); for FILTER and COUNT there are predicates (>0), (<0), (%2==0), (%2==1) and for ZIPWITH and SCANL1 the DSL provides lambdas (+), (-), (*), MIN, MAX.

For Example:

Consider the following example solving by using mixture of higher and lower level functions of Domain Specific Language (DSL):
Rishika is very enthusiastic about her dinner and intend to have chapattis in a big tray arrange in non-increasing order according to the sizes.
Given an array 'p' consisting of sizes of all the chapattis in a row in inches, compute how many inches she require to cut the chapattis in total in order to obtain chapattis having arranged in the desired manner.
Input-Output Example:
Sample Input: [8 5 7 2 5]
Sample Output: 5
Program:

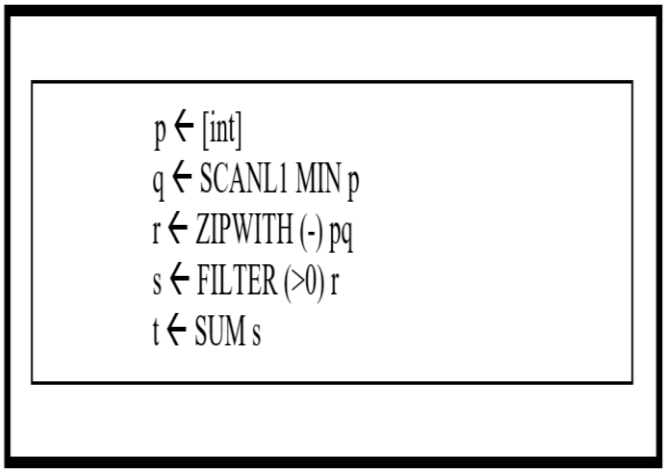

Fig. 3 The program is written of above question

Description about the program:

Program is shown in figure 3, there are first order function SUM and three higher order functions.
In first statement, assign an input integer array to a name 'p'.
In second statement, a function SCAN1 MIN consists a higher order function SCAN1 and INT → INT → INT lambda function MAX which maps integer pairs to integers. In our example, it returns an array q of length 5 as per p does whose i^{th} element is the minimum of first I elements in p, that is, q becomes an array having values like [8 5 5 2 2].
In third statement of our program, we consider higher-order statement 'ZIPWITH' with a lambda function (-) being applied to corresponding elements of both arrays 'p' and 'q' resulting in an array. The resulting array has length which is minimum of lengths of 'p' and 'q'. At the end of execution of third statement, we get an array 'r' as [0 0 2 0 3].
In fourth statement, we use a higher-order function 'filter' with the predicate (>0) mapping from integers to truth values returns number of elements in 'r' satisfying the predicate so as 's' becomes an array with the values [2 3].
In the final statement, we intend to use a first-order function 'SUM' which simply returns the sum of elements of array 's' as '5', so is our output.

4.2 Data generation

It is used to generate a dataset of programs in
DSL and ensure that it is feasible to generate a large dataset. The approaches towards generation are enumeration of the programs in DSL and pruning of the irrelevant or redundant variable. With the help of constraints, the range of valid values are obtained for each input. If the range for one of the inputs are empty, the program is discarded, else pre-computed ranges are used to generate the input-output pairs by picking inputs and then we get output values after the execution of the program. The explanation of this process is shown in the figure 4.

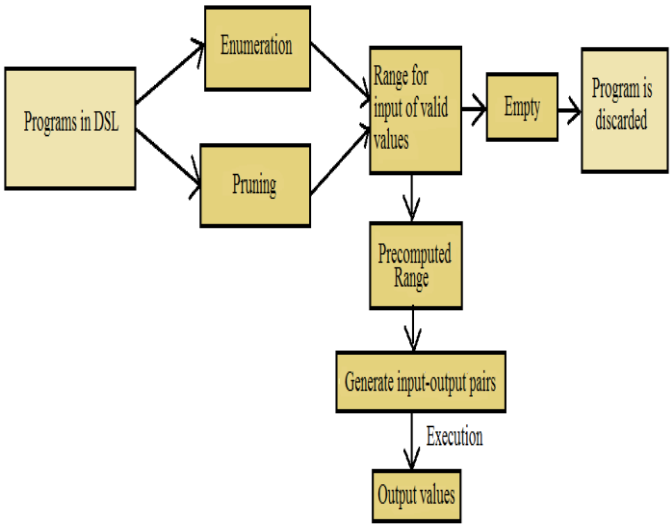

Fig. 4 Explanation of the Data Generation Procedure

4.3 Machine Learning Model

It is used for learning a distribution of attributes given input-output examples. We will recognize the patterns in the input-output example for utilizing them to predict the presence and absence of individual functions. At training time, training uses a maximum likelihood objective to observe the attributes. So, for the mapping of the input-output examples to attributes, we make a use of neural network that consists of two parts: Encoder and Decoder.

4.4 Encoder

It is used for differentiable mapping from a set of input-output examples to a latent real-valued vector. Feed-forward network is used in Encoder. Choosing Feed-Forward network because it is easy to train as compared to RNN as discussed earlier in Sect. 1 but there is also a disadvantage because it needs an upper bound on the input and output length of arrays.

4.5 Decoder

It is used to map the latent vector to predictions of the direct program's attributes. Encoder and Decoder both corresponds to a standard sequence-to-sequence model by using RNN. But the more modernized RNN decoder or training procedure is not so successful according to the experimental data.

4.6 Search Component

It is used for guiding the search. Neural
Network is used to guide the search for a program consistent set of input-output examples and not directly predict the entire source code. So the various search techniques are used such as DFS, "Sort and add" enumeration and Sketch.
"Sort and add" enumerative search maintains and performs DFS only on the active functions (which are not pruned), if search is not successful then take the next function that would be added in the active set and then the search is again applied to this active set. But re-exploration of some parts of the search space is several times is the limitation of this enumeration search procedure.
"Sketch" is the SMT-based program synthesis tool as discussed in Sect. 1 which is used to fill "holes" (Synthesize Program) in the incomplete source code to match specified requirements. So, sketch utilizes the neural network predictions from the "Sort and add" search procedure as the possibility of function and its arguments that is holes that is restricted to the current active set.
"DFS" (Depth First Search Technique) is used when sometimes several neural network models are used to guide the search in program space encountered with search procedures that tend to extend a partially executable program then it examines several external functions which arranges in an order accordance to their predicted probabilities from trained neural network.

5. Overview of λ^2 Synthesis Tool

λ^2 is a program synthesis tool named as lambda learner that integrates the enumerative search with deduction to prune the search space. λ^2 is used to choose the library of functions according to the neural network predictions.

While learning, the DeepCoder loses by ignoring correlation between functions and this arise the new concept the Training Loss Function. The predictions about a function is said to be marginal probabilities.

6. Evaluation of DeepCoder by using "Sort and add"

For the evaluation of DeepCoder, the recorded time of search procedures are compared that are needed to find a program consistent with M input-output examples.
If "Sort and add" without the neural network then it would exceeded the 10^4 seconds timeout, so the neural network is optimizing the runtime. If ignoring the correlations between functions, then it is not going to change for "Sort and add" enumeration owing to a Rank Loss (It is Bayes optimal to rank the labels that is the functions appearing in the direct source code, according to their marginal probabilities) that upper bounds the "Sort and add" runtime can be minimized.

7. Conclusion and Future Work

In this paper, we have presented an approach to enhance the quality of Inductive Program Synthesis (IPS) Systems by training the neural network to guide the search in program space. There have been some problems in the real competitive programming competition that can be solved with a program in Domain Specific Language (DSL) which validates the relevance of this approach.
There are the following two limitations of this approach:

(a) The programs synthesized are the simplest ones and many complicated problems may require more knowledge about making complex algorithmic solutions like dynamic programming which is beyond the approach's reach.

(b) While this approach still expecting about LIPS's applicability to be successful, DSL becomes more complex having large number of functions in it and when this moves to solve big programming problems, the input-output examples becomes significantly less informative.

This approach can lead to the best future prospects of DeepCoder by using extensive power of machine learning to program synthesis.

References

[1] John K. Feser, Swarat Chaudhuri, and Isil Dillig. Synthesizing data structure transformations from input-output examples. In Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), 2015.
[2] Wojciech Zaremba, Tomas Mikolov, Armand Joulin, and Rob Fergus. Learning simple algorithms from examples. In Proceedings of the 33nd International Conference on Machine Learning (ICML), 2016.
[3] Sainbayar Sukhbaatar, Arthur Szlam, Jason Weston, and Rob Fergus. End-to-end memory networks. In Proceedings of the 28th Conference on Advances in Neural Information Processing Systems (NIPS), 2015.
[4] Alex A. Alemi, Francois Chollet, Geoffrey Irving, Christian Szegedy, and Josef Urban. DeepMath - deep sequence models for premise selection. In Proceedings of the 29th Conference on Advances in Neural Information Processing Systems (NIPS), 2016.
[5] Alexander L. Gaunt, Marc Brockschmidt, Rishabh Singh, Nate Kushman. TerpreT: A Probabilistic Programming Language for Program Induction, 2016.
[6] Giovani Dematos, Milton S. Boyd, Bahman Kermanshahi, Nowrouz Kohzadi, lebeling Kaohzadi. Feedforward versus recurrent neural networks for forecasting monthly Japanese yen exchange rates, 1996.

[7] Sebastian Riedel, Matko Bosnjak, and Tim Rocktaschel. Programming with a differentiable forth interpreter. CoRR, abs/1605.06640, 2016. URL http://arxiv.org/abs/1605.06640, retrieved on 21st June 2017.
[8] Constraint Programming https://en.m.wikipedia.org /wiki/Constraint_programming, retrieved on 21st June 2017.
[9] Satisfiability modulo theories (SMT). URL https://en.m.wikipedia.org/wiki/Satisfiability_modulo_theories, retrieved on 22nd June 2017.
[10] Differentiable Functional Program Interpreters (DFPI). URLhttps://www.microsoft.com/en-us/research/ publication/differentiable-functional-program-interpreters/, retrieved on 23rd June 2017.