International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395 -0056
Volume: 04 Issue: 06 | June-2017 www.irjet.net p-ISSN: 2395-0072

DEEPCODER TO SELF-CODE WITH MACHINE LEARNING

Sumit Thappar[1], Ameya Parkar[2]

[1] Student, Masters of Computer Application Department, VESIT, Maharashtra, India
[2] Asst. Professor, Masters of Computer Application Department, VESIT, Maharashtra, India

---

Abstract- *Machine Learning has been focusing on the various aspects of the technology to automate the day to day needs of the human interaction like Siri, Cortana, Google Assistant. The Machine Learning is a branch of Artificial Intelligence that focuses on learning from the existing data to give expected outputs to the users. This paper's focuses on the upcoming possibility of the machines to learn to code by itself to build blocks of code that a regular programmer can do but in a quiet lesser time and better optimized. Deep-coder is a technology upcoming which is being developed by Microsoft to generate such algorithms where machines can generate code provided there are specifications provided from the user.*

Key Words: *IPS, SMT, DSL, DeepCoder.*

1. INTRODUCTION

Learning is an important parameter for a machine to develop intelligence. Deep understanding is what is required for any decision that is to be taken. Different algorithms could be used for different decisions that involves learning depending on the environment. Most of the algorithms use the concept of pattern recognition to get an optimized decision. This paper focuses on the deep learning concept to code by the machines.
Learning is also considered as a parameter for intelligent machines. Deep understanding would help in taking decisions in a more optimized form and also help then to work in most efficient method. As seeing is intelligence, so learning is also becoming a key to the study of biological and artificial vision. Instead of building heavy machines with explicit programming now different algorithms are being introduced which will help the machine to understand the virtual environment and based on their understanding the machine will take particular decision. This could eventually decrease the number of programming concepts and also machine could become independent and take decisions on their own.
Different algorithms are introduced for different types of machines and the decisions taken by them. Designing the algorithm and using it in most appropriate way is the real challenge for the developers and scientists. Pattern recognizing a concept in machine learning to make optimized decisions. As a consequence of this new interest in learning we are experiencing a new era in statistical and functional approximation techniques and their applications to domain such as computer visions.

2. Related Work

Matej Balog from the Cambridge University and Alexander L. Gaunt along with his associates[2] at the Microsoft Research developed a first line of attack for solving programming competition-style problems from input-output examples using deep learning. Their approach is to train a neural network to predict properties of the program that generated the outputs from the inputs. They used the neural network's predictions to augment search techniques from the programming languages community, including enumeration search and an SMT based solver. Factually, their approach leads to an order of magnitude speedup over the strong non-augmented baselines and a Recurrent Neural Network approach, and that we are able to solve problems of difficulty comparable to the simplest problems on programming competition websites.

In this work, they proposed two main ideas:

1. learn to induce programs; that is, use a corpus of program induction problems to learn strategies that generalize across problems,[3] and
2. integrate neural network architectures with search-based techniques rather than replace them.[3]

In more detail, their approach contrasts to existing work on differentiated interpreters. In differential interpreters, the idea is to define a differentiated mapping from source code and inputs to outputs. After observing inputs and outputs, gradient descent can be used to search for a program that matches the input-output examples.
It can be argued that machine learning can provide significant value towards solving Inductive Program Synthesis (IPS) by re-casting the problem as a big data problem. It shows that training a neural network on a large number of IPS generated problems could predict cues from the problem description can help a search-based technique. In this work, they focused on predicting an order on the program space and show how to use it to guide search-based techniques that are common in the programming languages community.[3]

This approach has three desirable properties:

first, we transform a difficult search problem into a supervised learning problem;

second, we soften the effect of failures of the neural network by searching over program space rather than relying on a single prediction;
third, the neural network's predictions are used to guide existing program synthesis systems, allowing them to use and improve on the best solvers from the programming languages community.
It represents magnitude of improvements over some optimized standard search techniques and a Recurrent Neural Network-based approach to the problems.
In summary, it defines and instantiate a framework for using deep learning for program synthesis problems like ones appearing on programming competition websites.

Their base contributions are:

1. defining a programming language that is expressive enough to include real-world programming problems while being high-level enough to be predictable from input-output examples;
2. models to map sets of input-output examples to program properties; and
3. experiments that show an order of magnitude speedup over standard program synthesis techniques, which makes this approach feasible for solving problems of similar difficulty as the simplest problems that appear on programming competition websites.

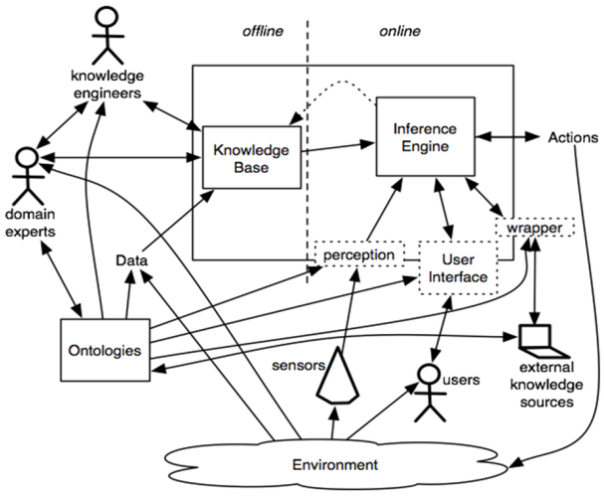


Fig1: Machine Learning Model

About Deep-Coder:

Deep coder is a machine learning system that can write its own code. It does this by using a Technique called program synthesis. It has the ability to create new programs by taking the line of code from the existing programs from other software. This program synthesis can determine which line of code in particular can be useful to get the desired output for any particular user.

This approach is to train a neural network to predict properties of a program that generates output from the inputs. Neural network predictions can be used to augment search techniques from programming language community, as tested by the team led by Alexander Gaunt from Microsoft Research and Matej Balog from Cambridge. [3]
The system, called Deep-Coder, basically searches a corpus of code to build a project that works to spec. This system gets smarter as it keeps practicing, figuring out which code snippets work best together and when to use a certain snippet in place of another. Hence it learns the system to get faster as it builds more programs.
Deep-Coder successfully plowed through the basic, input-output style challenges usually set by programming competitions. It was able to search through multi-lines of code more scrutinized and widely than a human coder could do, grouping together code in a manner humans might not think of and in a more quicker way. Since Deep-Coder is essentially a deep learning algorithm, every time it's given a new problem, it gets better at combining lines from source codes. Ultimately, this algorithmic technique can make programming accessible to non-coders, allowing anyone and everyone to easily build simple programs.
Researcher Marc Brockschmidt, one of Deep-Coder's creators from Microsoft Research in Cambridge, UK, believes that their approach would make it possible for non-coders to just describe a program and leave the system to build it. This could innovate the programming drastically, in no one way that programmers could have thought of.[2]
Deep-Coder's current version only allows it to handle programming challenges with around five lines of code. "The potential for automation that this kind of technology offers could really signify an enormous [reduction] in the amount of effort it takes to develop code,"
Defining the machine learning is a task that is vitally important to understand scope of the problem and the limits involved in any potential solution.

In the standard programming term setting, we have two pieces of information:

1. A textual description of the problem
2. One or more example input / output pairs

A simple example is provided below.

Program 0:	**Input-output example:**	*Description*:
k ← int b ← [int] c ← SORT b d ← TAKE k c e ← SUM d	*Input*: 2, [3 5 4 7 5] *Output*: [7]	A new shop near you is selling n paintings. You have $k < n$ friends and you would like to buy each of your friends a painting from the shop. Return the minimal amount of money you will need to spend.

The goal is to produce a working program that is
(a) consistent with all provided input / output examples and
(b) consistent with the task as described.
Notice that solving (a) does not imply (b).[2]
The above is called Inductive Program Synthesis (IPS),where a given input-output example, produces a program that has behavior consistent with the examples and the defined task.

3) What does Deep-coder do?

3.1 The program space with our infinite monkeys

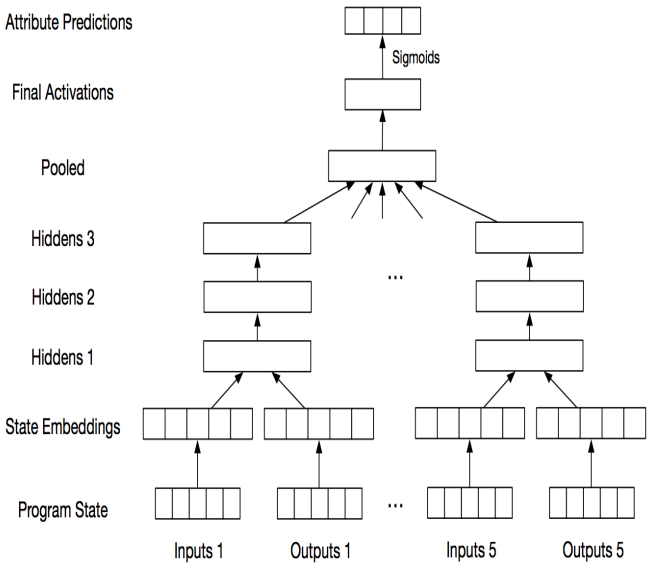
In the experiments in the paper, each problem is given five example input / output pairs. Deep-coder defines a domain specific language (DSL) that, when composed together, can solve the specified problems. This DSL contains 34 different first order and higher order functions and allows all integers from -255 to 255. Essentially, you have a set keyboard of options that you can use to piece together a solution.[1]

First-order functions: HEAD, LAST, TAKE, DROP, ACCESS, MIN, MAX, REVERSE, SORT, SUM
Higher-order functions: MAP, FILTER, COUNT, ZIPWITH, SCANL1.
Higher-order MAP allows: (+1), (-1), (*2), (/2), (*(-1)), (**2), (*3), (/3), (*4), (/4)
Higher-order FILTER and COUNT allows: (>0), (<0), (%2==0), (%2==1)
Higher-order ZIPWITH and SCANL1 allows: (+), (-), (*), MIN, MAX

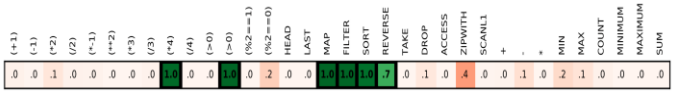
Note that this DSL does not contain any explicit control flow such as for loops, while loops, or branching.
This results in an enormous potential set of programs- one of which is guaranteed to hold the solution. This is referred to as the program space and is equivalent to throwing the infinite monkey theorem at the problem which states if there are n number of monkeys placed in front of specialized keyboards having buttons labeled with functions (SORT, TAKE, SUM), variables (k, b, c), and so on - one of the monkeys would eventually produce the correct program. Obviously monkeys are slow, temperamental, and require bananas, so we'd hope there's a better option. One best options is depth first search (DFS), where one weights the search towards programs similar in composition to previous working programs we've seen in training and keep testing the program against the input / output examples to see if it works. This is one of the baselines that Deep-coder competes against. Given infinite time, DFS would solve our proposed problems using the DSL specified above.[1]

3.2 Neural networks are better than infinite monkeys

Deep-coder gives the output after getting the input pairs and predicts the presence or absence of individual functions from the DSL.
Note that Deep-coder doesn't even read the problem description and yet to help decide which functions are most likely!



Below, Figure from the paper, is an example that predicts the probability of each DSL function appearing in the source code. The neural network in this case is particularly interested in trying to use MAP,(*4), SORT, FILTER, and REVERSE to solve the problem. If the neural network's prediction is accurate, it may be possibly to find the relevant program quickly and easily without exhaustively searching the program space. [1]



Important to note, the maximal length of the

Program 4:
x ← [int]
y ← [int]
c ← SORT x
d ← SORT y
e ← REVERSE d
f ← ZIPWITH (*) d e
g ← SUM f

Input-output example:
Input:
[7 3 8 2 5],
[2 8 9 1 3]
Output:
79

Description:
Xavier and Yasmine are laying sticks to form non-overlapping rectangles on the ground. They both have fixed sets of pairs of sticks of certain lengths (represented as arrays x and y of numbers). Xavier only lays sticks parallel to the x axis, and Yasmine lays sticks only parallel to y axis. Compute the area their rectangles will cover at least.

programs is length 5, as in the example below.

This is still far away from being useful in real world tasks though it does represent a strong speed improvement over previous methods that could exhibit these capabilities.

3.3 Algorithm Synthesis with *DeepCoder*

The authors of "*DeepCoder*: Learning to Write Programs" employ deep learning and artificial neural networks(ANNs). First, a domain-specific programming language is chosen. Then, an ANN is trained on a variety of example programs. This ANN is then used to guide the search in the space of possible programs when trying to solve a new programming task. Basically, this search can exhaustively explore a certain subspace of the space of all possible programs. The subspace could be "all programs of at most 5 instructions." The search could then be a depth-first search (DFS) up to depth 5. Whenever the DFS needs to choose the next instruction to add to its current program, it could pick the one the ANN deems to most likely to be the right choice. Since it would still explore the subspace exhaustively and the ANN just provides the order of the exploration, it would still find the right program (if one exists), even if the ANN would be faulty. By using this concept, the authors achieve a considerable speedup on existing technologies.

An example taken from the paper *DeepCoder*: Learning to Write Programs.[2]

Program

s <- [int]

b <- REVERSE s

c <- ZIPWITH (-) b s

d <- FILTER (>0) c

e <- SUM d

Input Example

Input: [1 2 4 5 7]

Output: 9

Description

Vivian loves rearranging things. Most of all, when she sees a row of heaps, she wants to make sure that each heap has more items than the one to its left. She is also obsessed with efficiency, so always moves the least possible number of items. Her dad really dislikes if she changes the size of heaps, so she only moves single items between them, making sure that the set of sizes of the heaps is the same as at the start; they are only in a different order. When you come in, you see heaps of sizes (of course, sizes strictly monotonically increasing) s[0], s[1], … s[n]. What is the maximal number of items that Vivian could have moved?

4. Misconception about Deep-coder Steal code:

What did people claim Deep-coder could do?
Like a game of telephone / Chinese whispers, errors seem to accumulate in each retelling of scientific research. The various allegations like "Deep-coder copy pastes from Stack Overflow" were a major widespread rumors that followed. [1]
Simple statements meant to improve reader comprehension take on a life of their own. a journalist started describing it as "piecing together lines of code taken from existing software" Instead they could describe that the program was able to use 34 different first order and higher order functions from a domain specific language. A journalist also described it as "using machine learning to scour databases of source code" Instead of stating the process of training the algorithm from a specific set of problem descriptions and input / output pairs.
Neither of these are bad- especially within the original article context. A full explanation of the paper is out of scope for such an article. These minor simplifications are used to allow a broader audience of readers to follow the story. This is good- more people should have the opportunity to understand these advances. Many articles are even reasonable in their claims, stating that Deep-coder for now only works with programs of length five or less and specifically only over an extreme subset of programming competition problems.
The issue comes as the story is relayed, poorly, over and over again. The incorrect but helpful narration *"piecing together lines of code"* suddenly becomes copy and paste. This helpful *"scouring a database"* becomes *"stealing from other software"* which then jumps to *"stealing from Stack-Overflow".* Deep-coder even becomes an active competitor in online programming competitions and capable of already assisting programmers. Reality falls away awfully quickly...[1]

1. Microsoft's AI writes code by looting other software
2. Deep-coder takes lines of code from existing software
3. Microsoft's new AI can code by stealing bits of code from other software
4. Microsoft Deep-coder AI Produces Its Own Code By Ripping Off Existing Software
5. Microsoft's AI 'Deep-coder' learns coding by stealing from others
6. Now, here is Deep-coder, an AI trained to use pieces of code from existing software and write a code of its own.
7. Deep-coder AI Writes Programs Using Existing Code Snippets
8. Deep-coder builds programs using code it finds lying around the system works by taking lines of code from existing programs and combining them.

Thus, the hybrid code was born-Called Deep-coder, the software can take requirements by the developer, search through a massive database of code snippets and deliver working code in seconds. They only have to describe their program idea and wait for the system to create it.
It's been used to complete programming competitions and could be pointed at a larger set of data to build more complex products. The system can search more quickly and more completely than any human coder to create a new application once it knows what the requirements are. Deep-coder successfully plowed through the basic, input-output style challenges usually set by programming competitions. In the paper, the researchers explain that Deep-coder relies on big data analysis and machine learning techniques.
To remind you why the above is stunningly incorrect:

1. Deep-coder did not (and cannot) at any point take code from another piece of software.
2. Deep-coder can't read or use any of the textual descriptions that might exist for a given problem - so anywhere "reading a problem description" is basically incorrect.
3. Solar Lezama said. No need for programmers to start updating their resumes, though, as this tech wouldn't replace humans. Instead, the system could handle more tedious parts of programming, while human coders could focus on more sophisticated work.

5. FUTURE VISION

Over the last decade, program synthesis performed a great leap forward, exempli1ed in learning complex programs from loose speci1cations in mass-market applications. I believe that its ability to perform logical reasoning and leverage domain-speci1c insight will provide a new level of capabilities to modern AI technologies. Machine learning has long realized the importance of proper representations to effective learning. Nowadays, adopting *programs* as the underlying representation of AI promises to resolve the omnipresent demand to make AI artifacts debuggable and interpretable.

While neural models by themselves are difficult to interpret, program synthesis and DSL''s can help in this regard. There are multiple ways to apply them to the deep learning artifacts:
(a) use features/subroutines that were learned by techniques from the previous paragraph as a high-level interpretation;
(b)synthesize an "interpretation" program from a supplementary DSL that most closely approximates the model as a black-box function;
(c) combine both approaches by inducing a supplementary DSL from the learned subroutines.

Any of these approaches makes ML-based AI more transparent, which helps to apply it to new domains on an industrial scale.

6. CONCLUSIONS

Machine learning has a lot of "folk wisdom" and scope that can be hard to come by, but is crucial for success. This article summarized some of the most salient items. This paper emphasizes on the possibility of the wide domain of machine learning to grow its roots for the programmers to help and code through artificial intelligence.

REFERENCES

[1] URL:https://smerity.com/articles/2017/deepcoder_and_ai_hype.html
[2] Matej Balog, Alexander L. Gaunt, Marc Brockschmidt, Sebastian Nowozin, Daniel Tarlow ,Submitted to ICLR 2017 URL: https://arxiv.org/abs/1611.01989
[3] URL:https://techxplore.com/news/2017-02-microsoft-university-cambridge-deepcoder-code.html
[4] Alexander L. Gaunt, Marc Brockschmidt, Rishabh Singh, Nate Kushman, Pushmeet Kohli, Jonathan Taylor, and Daniel Tarlow. Terpret: A probabilistic programming language for program induction.CoRR, abs/1608.04428, 2016. URL http://arxiv.org/abs/1608.04428.
[5] Alex Graves, GregWayne, and Ivo Danihelka. Neural Turing machines. CoRR, abs/1410.5401, 2014.URL http://arxiv.org/abs/1410.5401.