Inductive Programming by Expectation Maximization Algorithm

Roger Hsiao
InterACT, Language Technologies Institute
Carnegie Mellon University
wrhsiao@cs.cmu.edu

Abstract

This paper proposes an algorithm which can write programs automatically to solve problems. We model the sequence of instructions as a n-gram language model and the sequence is represented by some hidden variables. Expectation maximization algorithm is applied to train the n-gram model and perform program induction. Our approach is very flexible and can be applied to many problems. In this paper, we will concentrate on function approximation and traveling salesperson problem.

1 Introduction

This paper proposes a novel machine learning algorithm which can write programs automatically to solve problems. Program, in the context of this paper, is known as a sequence of user defined instructions which operate on some continuous or discrete variables. The proposed algorithm allows users to plug in any instruction set, so as a result, it provides a very flexible framework which allows this algorithm can be applied to many kinds of problems.

Formally speaking, given some data, this algorithm induces a program for a machine which comes with an user defined instruction set and a memory module. Memory in this context is defined as a set of continuous or discrete variables. Instruction set are some functions which operates on this memory. One can imagine the PCs nowadays are examples of this kind of machines. The automatic procedure of inducing a program from data is known as program induction or inductive programming.

Different from the previous approaches discussed in [2][6], this paper proposes a probabilistic framework for inductive programming. The induction process is driven by the expectation maximization algorithm (EM) [3]. In which, each program is modeled by a N-gram language model (LM) and the instruction sequences are hidden variables.

The ultimate goal of this project is to explore whether it is possible for a machine to write programs automatically. And if it is possible, we would like to know what kind of problems would be suitable to use this approach. We believe that this algorithm allows ones to get a fairly good solution for a very difficult problem immediately without serious study on the problem. As a preliminary study, we try to explore two applications: function approximation and traveling salesperson problem (TSP). The function approximation problem studied in this paper is different from a standard approximation problem, since in addition to get

good approximation, we are interested in whether the new algorithm can recover the true functional form of the target function. For TSP, as a NP-hard problem, we interpret the tour, i.e. sequence of cities, as a sequence of instructions. Our approach allows us to use EM algorithm to tackle TSP which is difficult to solve in general.

This paper is organized as follows: Section 2 will briefly describe some existing work about program induction. In section 3, we present how to apply EM algorithm to inductive programming. Section 4 is about experiments and finally, we have a conclusion and a discussion about the future work.

2 Related Work

The term "inductive programming" sometimes may refer to some methodology in software engineering or any existing learning algorithms in machine learning [5][7]. However, this paper is going to adopt a more specific definition: Design an algorithm which can write a program automatically given some information or data.

One of the existing approaches for inductive programming is using genetic algorithm [2], in which, a program is made up of some instructions and they are treated as an organism. Through an evolution process, one can obtain a program for the designed task.

Another approach of inductive programming is using functional programming languages. [6] suggests an algorithm which can induce functional program by generalization of the training examples. However, a drawback of this approach, as mentioned in the paper, all training examples have to be correct, which is unlikely to be true in real application.

3 Program Induction by EM algorithm

Unlike previous approaches, this paper describes how to apply EM algorithm for inductive programming. We propose a probabilistic framework for program induction.

A machine is defined by a tuple (Φ, M) where Φ is the instruction set which is a set of user defined functions, $\{f_1, \ldots, f_N\}$; M is the memory which is a set of continuous or discrete variables, $\{m_1, \ldots, m_K\}$. A program is considered as a sequence of functions. Suppose a program has T steps, then it is denoted by $F = (f_1, \ldots, f_T)$. Functions are operators working on the variables in the memory, so given a program and an initial configuration of the memory, one can compute the final state of the memory by using the user defined functions as illustrated in figure 1.

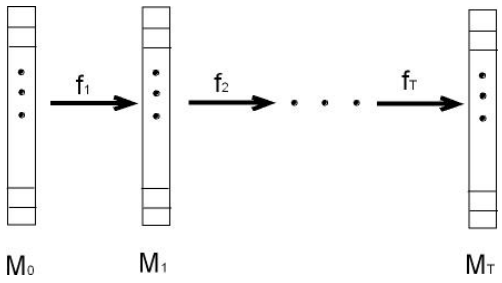

Figure 1: Program as a sequence of instructions.

Program is modeled as a n-gram language model. In order words, the probability of choosing a function at step t depends on the functions chosen in the last n steps, i.e.

$P(f_t|f_{t-1},\ldots,f_{t-n+1})$. Linear interpolation of lower order n-gram probabilities is applied for smoothing due to data sparseness for higher order n-grams. In this paper, we will call this n-gram language model as program model.

Given a problem with observable data O, we are interested in computing $P(O)$. $P(O|F)$ measures how strong the program F supports the data O. The program F is a latent variable, so we do not observe F directly. Therefore, in order to maximize the log likelihood of the data. We apply the EM algorithm which maximizes the expected complete log likelihood,

$$\begin{aligned} Q(\theta,\theta_{old}) &= E_F[\log P(O,F|\theta)|x,y,\theta_{old}] \\ &= \sum_F P(F|O,\theta_{old})\log P(O|F,\theta) \\ &+ \sum_F P(F|O,\theta_{old})\log P(F|\theta)\ , \end{aligned} \tag{1}$$

where θ and θ_{old} are the parameters of the program model. θ_{old} corresponds to the parameters of the last EM iteration and θ is the parameters to be estimated. Due to the fact that the number of possible programs is huge, equation 1 is not tractable unless we apply some approximation.

Assuming all programs must have T steps, from the initial memory configuration, we can build a search tree from step 1 to T as shown in figure 2. In order to keep the tree small enough, we prune the tree by considering the probabilities from the program model, and we will keep a fixed amount of active tokens during the search. This fixed amount is named prune width in this paper. Once we get the search tree, we assume that for all program F which is not in the tree, $P(F|O,\theta) \simeq 0$. With this approximation, equation 1 remains tractable.

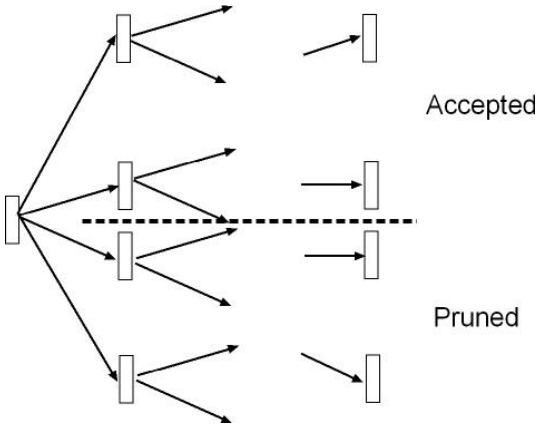

Figure 2: Building a search tree

3.1 E step

During the E step, one has to compute the posterior probability $P(F|O,\theta_{old})$. By Bayes' rule, it becomes,

$$P(F|O,\theta_{old}) = \frac{P(O|F,\theta_{old})P(F|\theta_{old})}{\sum_F P(O|F,\theta_{old})P(F|\theta_{old})}\ . \tag{2}$$

Therefore, it breaks down to program model score and likelihood. The likelihood term measures how strong the program F supports the data O. The measurement depends on the output of the program, which is encoded in the memory. Given a program $F = (f_1, \ldots, f_T)$, we can reproduce the memory sequence $M_0^T = (M_0, M_1, \ldots, M_T)$. Hence, we assume,

$$P(O|F, \theta) \quad \simeq \quad P(O|h(M_0^T), \theta) \tag{3}$$

where h is a function to extract the output/hypothesis made by the program F. $P(O|h(M_0^T), \theta)$ measures how good the hypothesis h supports the data O. The distribution of this probability may be different for different problems. The easiest way would be assume it is a Gaussian distribution just like what we did on making probabilistic interpretation of linear regression problem.

For regression problem $y = g(x)$ ($O = (x, y)$), one may further assume that we only care about the output at the last step of the program, and the output must be in certain variable in the memory. Then, we can further simplify equation 3,

$$P(x \to y|F, \theta) \quad \simeq \quad P(x \to y|m_k^T, \theta) \tag{4}$$

where m_k^T is the k-th variable in the memory at step T. Similar to the probabilistic interpretation of linear regression, we assume $P(x \to y|m_k^T, \theta)$ follows a normal distribution:

$$P(x \to y|m_k^T, \theta) \quad = \quad \frac{1}{\sqrt{2\pi}} \exp(-\frac{(y - m_k^T)^2}{2\sigma^2}) \; . \tag{5}$$

Therefore, if a program can generate a hypothesis similar to y, it will have a higher likelihood.

For TSP, O stores the locations of all cities and $h(M_0^T)$ represents a tour. $P(O|h(M_0^T), \theta)$ is defined as a Gaussian distribution,

$$P(O|h(M_0^T), \theta) \quad = \quad \frac{2}{\sqrt{2\pi}} \exp(-\frac{(d_h - d_{lower})^2}{2\sigma^2}) \; . \tag{6}$$

where d_h is the total distance of the tour and d_{lower} is a lower bound of the optimal tour. Hence, shorter tours will receive higher likelihood.

With the likelihood, program model score and the posterior probability, one can compute the expected likelihood and finishes the E step.

3.2 M step

Because of the assumption in equation 5 and 6, the likelihood function does not depend on θ. To maximize the auxiliary function in equation 1, we have to maximize,

$$Q'(\theta, \theta_{old}) \quad = \quad \sum_F P(F|O, \theta_{old}) \log P(F|\theta) \; . \tag{7}$$

Then we can maximize Q' by using Lagrange multiplier optimization method. The solution is the maximum likelihood solution with weight $\frac{P(F|O,\theta_{old}))}{\sum_F P(F|O,\theta_{old}))}$ for the program F.

Given an initialization of the program model, one can apply EM algorithm for program induction. Although the initialization of the program model can be as simple as uniform distribution, it can be tricky because we have to apply pruning to keep the search tree in a reasonable size. Hence, if there is no reasonably good program in the first pass, the whole optimization may fail. In such a case, one may need to use a bigger prune width or define prior probabilities for the program model.

Recall that we have an assumption about all programs must have T steps. In practice, one can assign a larger T for induction and then gradually decrease it, and check the program performance on the training set. Or one can design an instruction which can terminate a program. However, this would create programs with different length. As a preliminary study, this paper will not address this issue and just pick a larger T for induction.

3.3 Program Extraction

Once the program model is trained, there are many ways to extract the programs from the program model. The simplest way would be generating the first best program in the program model. Or one can keep track of the total likelihood of the data points for each program and choose the best one. However, this is expensive since one has to keep the statistics of thousands of programs for each data point. In this project, we keep the best program for each data point and choose the program which has the largest count. The ratio which the chosen program is the best in the training data is named dominant ratio.

4 Experiments

4.1 Function Approximation

In the regression experiments, we are working on a machine with a memory module with four continuous variables, $M = \{m_0, m_1, m_2, m_3\}$ and an instruction set as shown in table 1.

Table 1: The instruction set for the machine in the experiments.

instruction set			
$\sin(m_0) \to m_1$	$\sin(m_0) \to m_2$	$\sin(m_1) \to m_1$	$\sin(m_2) \to m_2$
$\cos(m_0) \to m_1$	$\cos(m_0) \to m_2$	$\cos(m_1) \to m_1$	$\cos(m_2) \to m_2$
$add(m_1, m_1) \to m_1$	$add(m_2, m_2) \to m_2$	$add(m_1, m_2) \to m_3$	
$mul(m_1, m_1) \to m_1$	$mul(m_2, m_2) \to m_2$	$mul(m_1, m_2) \to m_3$	
$assign(m_1) \to m_3$	$assign(m_3) \to m_1$	$assign(m_2) \to m_3$	$assign(m_3) \to m_2$
$neg(m_1) \to m_1$	$neg(m_2) \to m_2$	$neg(m_3) \to m_3$	

Table 2 shows the induced programs of different tested target functions. Note that m_0 is the input x and m_3 is the output. One hundred data points were generated in the range $-\pi \leq x \leq \pi$ for each of the four cases. In the experiments, bigram is used for the program model. The length of the program is chosen to be larger than or equal to the shortest possible program. The size of the search tree is restricted to be very small in order to prevent the algorithm can find the solution in the first iteration through exhaustive search. Three EM iterations are applied to obtain the best programs. The dominant ratios for all four cases are 100%. From the result, we can observe that the proposed algorithm can discover

Table 2: The best programs for different target functions after 3 EM iterations (m_0 is x and m_3 is output).

step	$\sin(x)\cos(x)$	$2\sin^2(x) + 2\cos(x)$	$\frac{\sin(x+h)-\sin(x)}{h}$	$\sin(2\cos(x))$
1	$\cos(m_0) \to m_1$	$\sin(m_0) \to m_1$	$\sin(m_0) \to m_1$	$\cos(m_0) \to m_1$
2	$\cos(m_0) \to m_2$	$\cos(m_0) \to m_2$	$\cos(m_0) \to m_2$	$add(m_1, m_1) \to m_1$
3	$\sin(m_0) \to m_1$	$add(m_2, m_2) \to m_2$	$assign(m_2) \to m_3$	$\sin(m_1) \to m_1$
4	$add(m_1, m_2) \to m_3$	$mul(m_1, m_1) \to m_1$		$assign(m_1) \to m_3$
5		$add(m_1, m_1) \to m_1$		
6		$add(m_1, m_2) \to m_3$		

the target function though there are redundant instructions in the program. Although the current implementation requests a fixed length for the program, this limitation can be easily overcome by the methods discussed in section 3. The third experiment gives an interesting insight about the application of inductive programming since it can discover the functional form of the target function's derivative. Though in general, this algorithm provides a best fit function only.

The experiments of function approximation show the possibility of automatic program induction. However, it is difficult to show how the order of the n-gram language model may interact with the prune width or other parameters, since it is difficult to judge the complexity of the function with respect to our instruction set. It is possible that a simple function may need more instructions than a complex function (consider $g(x) = 10\sin(x)$). Therefore, we move to the next problem which allows us to have a more thorough study about this algorithm.

4.2 Traveling Salesperson Problem

The next problem that we will work on is TSP. The advantage of applying our algorithm on TSP is we know that the complexity of the problem is directly related to the number of cities. Given an instance of TSP, we can study the effect of prune width, order of n-gram model and other parameters.

Another important property of TSP is that it is a NP-hard problem, which is difficult to solve in general. Although there exists high performance heuristics to tackle TSP [1], it is interesting to see the performance of our approach, which has almost no background knowledge about the nature of TSP at all. If our approach can give reasonably good solutions, it means in case we are facing some difficult problems which we do not know how to solve, or existing algorithms are too complicated to apply, one may simply apply our algorithm to get some fairly good solutions immediately.

In our experiments, a country named Western Sahara with 29 cities is chosen (data available from [4]). Our machine consists of 29 instructions and 29 discrete variables. Each instruction allows the salesperson to visit a specific city, and the discrete variables keep track on whether the city has been visited before. The use of an instruction is considered invalid if it brings the salesperson to a visited city, and it will not expand the search tree. Due to the nature of TSP, we know that the length of the program or the tour must be equal to the number of cities.

The initialization of the program model is an uniform unigram language model. Even in the experiments which requires a higher order, the first iteration assumes all higher order n-grams are unseen. After the first iteration, we collect the programs from the search tree to build the higher order language model. Due to the fact that, the first iteration is an uniform unigram model, it means all program scores would be the same, our current implementation chooses randomly for which tokens to be active. As a result, all the experiments have been run 3 times and averaged out before reporting the results.

Figure 3 shows the total distance of the tours generated by our algorithm across different number of EM iterations. The “greedy” line is the performance of a very simple heuristic which always ask the salesperson to the nearest unvisited city. The “optimal” line is the best tour found by using the state-of-the-art heuristic approach. From the results, we can see that for all different orders of n-gram models, the performance improves when we have more EM iterations and the improvement converges at some point. It is interesting to see that higher order of n-gram models does not have any advantages over the lower order models. One possible reason is that at the early stages of optimization, there are quite a lot errors which means bad decisions. The higher order models will cache the errors and affect the latter parts of optimization. Higher order models are less forgiving to errors compared to lower order models in this application because it has more parameters and the amount of good data is limited at the early stages. The prune width of this experiment is set to 16K.

Figure 4 shows the performance of having different sizes of prune width. In general, larger prune width gives better results. It is reasonable because the search space is larger and it is more likely to find a better tour. The order of n-gram is set to 3 for this experiment.

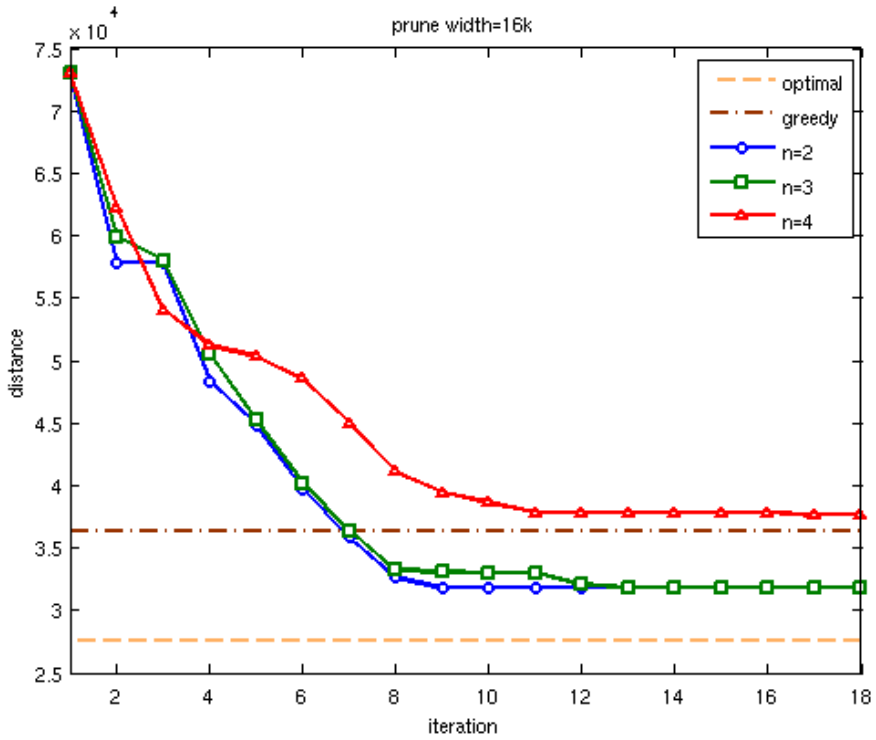

Figure 3: Effect of the order of n-gram model.

The overall results show that our algorithm can outperform the simple heuristic, but it cannot discover the optimal tour. It is possible that if we allow a larger prune width, we may further improve the performance. This result is encouraging because our algorithm does not make use of any knowledge about the nature of TSP problem but it can still output fairly good solutions. It implies that this algorithm would be helpful if we have a difficult problem which we do not know how to solve, or existing algorithms are too complicated to apply immediately.

5 Conclusion and Future Work

This papers presents a novel algorithm which allows a machine to program automatically. Our experiments show that one can obtain fairly good results without thorough background knowledge to the problem, which is encouraging.

From the problems we studied, it can be observed that this algorithm has many potential applications. It allows us to discover the functional form when we are working on function approximation. It computes the closest functional form for numerical derivatives of functions. The TSP experiments told us this approach can give reasonably good results even though we do not have/make use of any background knowledge to the problem.

This research also opens a new area for language technologies, since we model a program as a language model. In our current implementation, we only adopt a very simple model, which we believe it can be improved a lot if we apply more sophisticated approaches.

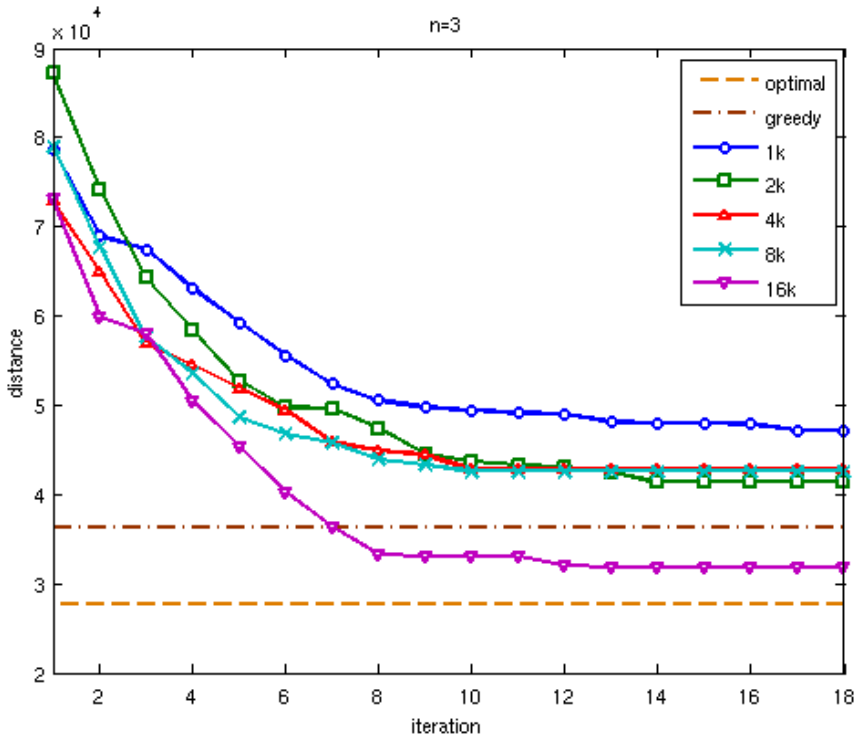

Figure 4: Effect of different prune width.

Another interesting implication of this algorithm is: what is the probability of this algorithm may generate the optimal solution? We believe it may be possible to derive a bound by considering the existence of n-grams from the optimal solution. If such analysis can be developed, it will allow us to look at the computational problems in a different way.

References

[1] David Appletgate, Robert Bixby, Vašek Chvátal, and William Cook. On the solution of traveling salesman problems. In *Proceedings of the International Congress of Mathematicians*, 1998.

[2] Dan Ashlock and James Lathrop. Program induction: Building a wall. In *Proceedings of the 2004 Congress on Evolutionary Computation*, 2004.

[3] J. Bilmes. A Gentle Tutorial on the EM Algorithm and its Application to Parameter Estimation for Gaussian Mixture and Hidden Markov Models, 1997.

[4] William Cook. http://www.tsp.gatech.edu/index.html.

[5] Pierre Flener and Derek Partridge. Inductive programming. *Automated Software Engineering*, 8(2):131–137, 2001.

[6] Emanuel Kitzelmann and Ute Schmid. Inductive synthesis of functional programs: An explanation based generalization approach. *Journal of Machine Learning Research*, 7:429–454, 2006.

[7] Derek Partridge. The case for inductive programming. *Computer*, 30(1):36–41, 1997.