Stochastic Grammar Based Incremental Machine Learning Using Scheme

Eray Özkural and **Cevdet Aykanat**
Bilkent University
Ankara, Turkey

Introduction

Gigamachine is our initial implementation of an Artificial General Intelligence (AGI system) in the O'Caml language with the goal of building Solomonoff's "Phase 1 machine" that he proposed as the basis of a quite powerful incremental machine learning system (Sol02). While a lot of work remains to implement the full system, the present algorithms and implementation demonstrate the issues in building a realistic system. Thus, we report on our ongoing research to share our experience in designing such a system. In this extended abstract, we give an overview of our present implementation, summarize our contributions, discuss the results obtained, the limitations of our system, our plans to overcome those limitations, potential applications, and future work.

The reader is referred to (Sch04; Sol02; Sol09) for a background on general-purpose incremental machine learning. The precise technical details of our ongoing work may be found in (Ozk09), which focuses on our algorithmic contributions. The discussion here is not as technical but assumes basic knowledge of universal problem solvers.

An overview of Gigamachine

Gigamachine is an incremental machine learning system that works on current gigaflop/sec scale serial computers. It is the testbed for our experiments with advanced general purpose incremental machine learning. Here, we describe version 1 of Gigamachine. Incremental machine learning can act as the kernel of complete AGI systems such as Solomonoff's Q/A machine or the Gödel Machine. The present implementation solves operator induction over example input/output pairs (any Scheme expression is an example). Our work may be viewed as an alternative to OOPS in that regard.

Design and implementation choices

Reference machine In Algorithmic Probability Theory, we need to fix a universal computer as the reference machine. (Sol09) argues that the choice of a reference machine introduces a necessary bias to the learning system and looking for the "ultimate machine" may be a red herring. In previous work, low-level universal computers such as FORTH and Solomonoff's AZ have been proposed. Solomonoff suggests APL, LISP, FORTH, or assembly. We have chosen the LISP-like language Scheme R5RS, because it is a high level functional language that is quite expressive and flexible, directly applicable to real-world problems. We have taken all of R5RS and its standard library with minor omissions.

Probability model of programs We use a stochastic Context-Free Grammar (CFG) as the "guiding pdf" for our system as suggested by Solomonoff (Sol09). Although Solomonoff proposes some methods to make use of the solution corpus using stochastic CFG's, we introduce complete algorithms for both search and update.

Implementation language We have chosen O'Caml as our implementation platform, as this modern programming language makes it easier to write sophisticated programs yet works efficiently. The present implementation was completed in about a month.

Implementation platform We have used an ordinary desktop computer and serial processing to test our ideas, we will use more advanced architectures as we increase the complexity of our system.

Contributions

We have made several contributions to incremental machine learning regarding both *search* and *update* algorithms. A good part of our contributions stem from our choice of Scheme as a reference computer. It would seem that choosing Scheme also solves some problems with low-level languages. A drawback in systems like OOPS is that they do not make good use of memory, better update algorithms may eventually alleviate that drawback.

Search algorithms

For the search algorithm, we use Depth-First Search in the space of *derivations* of the stochastic CFG. Thus, only syntactically correct candidates are generated. We use leftmost derivation to derive programs from the start symbol. We keep track of defined variables and definitions to avoid generating unbound references and definitions by extending CFG with procedural rules. We generate variable declarations and integers according to the Zeta distribution which has empirical support. We use a probability horizon to limit the search depth. We also propose using best-first search and a new memory-aware hybrid search method.

Update algorithms

We have designed four update algorithms that we will summarize. The former two have been implemented and they contain significant innovations on top of existing algorithms. The latter two are completely novel algorithms. All of them are well adapted to stochastic CFG's and Scheme.

Modifying production probabilities We use derivations of the programs in the solution corpus to update production probabilities. Since each derivation consists of a sequence of productions applied to nonterminals in sentential forms, we can easily compute probabilities of productions in the solution corpus. This is easy to do since the search already yields the derivations. However, this would cause zero probabilities for many productions if we used those probabilities directly, thus we use exponential smoothing to avoid zero probabilities.

Re-using previous solutions Modifying probabilities is not enough as there is only so much information that it can add to the stochastic CFG. We extend the stochastic CFG with previously discovered solutions (Scheme definitions) and generate candidates that use them. This is very natural in Scheme as the syntactic extension of a function is not a function. In principle, this is synergistic with modifying production probabilities as the probabilities of new productions can be updated, although in practice this depends on implementation details.

Learning programming idioms We can learn more than the solution itself by remembering syntactic abstractions that lead to the solution. Syntactic abstraction removes some levels of derivation to obtain abstract programs and then remembers them using the algorithm of re-using previous solutions.

Frequent sub-program mining Similar to code re-use, we can find frequently occurring sub-programs in the solution corpus as Scheme expressions and add them to the grammar.

Training Sequence and Experiments

We have tested a simple training sequence that consists of the identity, square, addition, test if zero, fourth power, NAND, NOR, XOR, and factorial functions. Details can be found in (Ozk09). Our experiments show beyond doubt the validity of our update algorithms. The search times decrease dramatically for similar subsequent problems showing the effectiveness of modifying probabilities. The fourth power function demonstrates code re-use in its solution of `(define (pow4 x ) (define (sqr x ) (* x x)) (sqr (sqr x ) ))` which takes shorter than solving the square problem itself. The factorial function took more than a day, so we interrupted it. It would be eventually found like other simple problems in the literature, however, we think that it showed us that we should improve the efficiency of our algorithms.

Discussion, Applications and Future Work

The slowness of searching the factorial function made us realize that we need improvements in both the search and the update algorithms. Some doubts have been raised whether our system can scale up to AGI since the search space is vast. In AGI, the search space is always vast, whether is the solution space, program space, proof space, or another space. Since no system has been shown to be able to write any substantially long program, we think that these doubts are premature. The path to bootstrapping most likely lies in more sophisticated search and update/memory mechanisms for a general purpose induction machine. Therefore, we think that we should proceed by improving upon the existing system. Regarding search, we can try to avoid semantically incorrect programs and try to consider time and space complexity of candidate solutions. The approach of HSEARCH may be applied. The probability model can be further advanced. For update, ever more sophisticated algorithms are possible. Another major direction that Solomonoff has suggested is context-aware updates, which may require significant changes.

The most important promise of *initial* AGI implementations will be to decrease the human contribution in *current* AI systems. The heuristic programmers of old school AI research can be replaced by programs like the Gigamachine. General induction might act as the "glue code" that will make common sense knowledge bases and natural language processing truly work. Many problems in AI are solvable only because the researchers were clever enough to find a good representation. AGI programs may automate this task. The current programs in machine learning and data mining may be supplemented by AGI methods to yield much more powerful systems. In particular, hybrid systems may lead to more intelligent ensemble learners and general data mining.

We will develop a more realistic training sequence featuring recursive problems, optimizing search and implementing the remaining two update algorithms. After that, we will extend our implementation to work on parallel multicore and/or GPU hardware. Those new architectures are extremely suitable for our system which will not require much synchronization between cores and requires little memory per core. We will then complete the implementation of Phase 1, implement the Phase 2 of Solomonoff's system, and attempt implementing other AGI proposals such as the Gödel Machine on top of our AGI kernel.

References

[Ozk09] Eray Ozkural. Gigamachine: incremental machine learning on desktop computers. `http://examachine.net/papers/gigamachine-draft.pdf`, December 2009. Draft.

[Sch04] Juergen Schmidhuber. Optimal ordered problem solver. *Machine Learning*, 54:211–256, 2004.

[Sol02] Ray Solomonoff. Progress in incremental machine learning. In *NIPS Workshop on Universal Learning Algorithms and Optimal Search*, Whistler, B.C., Canada, December 2002.

[Sol09] Ray Solomonoff. Algorithmic probability: Theory and applications. In M. Dehmer and F. Emmert-Streib, editors, *Information Theory and Statistical Learning, Springer Science+Business Media*, pages 1–23. N.Y., 2009.