Evolution of AI Learning Systems

Gurvinder Singh
CIIT, Indore
gurvinderengg@yahoo.com

ABSTRACT
Learning can be defined as a change in behavior due to practice or experience. For learning individual may need to repeat the experience in order for the behavior to change. Sometimes learning may be so strong that may result in change of behavior. Learning can be said as an adaptive process in which the tendency for a particular behavior is changed by the experience. As the conditions and situations change people learn new things, may or may not be eliminating old ones. New and old things can be accumulated together so as to bring more fruitful results of learning. Learning basically consists of attention, in which a individual may notice or observe something in the environment, retention individual remembers what was noticed, reproduction, individual replicates what was noticed and motivation a consequence is generated for further learning. Process of learning also involves the trial and error, such is the case when individual is having no teacher or reference for instruction him. Learning in AI is generally based on empirical data and is associated with non-symbolic AI, scrubby AI and soft computing. Learning by AI is developed so as to develop more effective and efficient learning systems. Our paper discusses some of the important learning systems developed by the AI.

KEYWORDS
Learning, exploration, EPAM

1. INTRODUCTION
Artificial Intelligence (AI) is a way for computers to attempt to mimic human intelligence. This page will discuss three ways an AI learns. One way is through failure-driven learning. The second is learning by being told. And the third is learning by exploration.

Failure-Driven Learning
Failure-driven learning is based on creating a program that will learn by making mistakes and then finding a solution so that mistake doesn't happen again. This is similar to the way humans learn. If we make a mistake we usually try to learn from that mistake to improve upon ourselves so we don't make it again.

Learning by being told
Learning by being told is another area of AI learning. It's simply interaction of a teacher (human) and the AI student. The teacher is there to teach the AI how to do things in the real world. Because the teacher has a grasp on the real world situation, it virtually eliminates the need for induction by the AI.

The only problem is communication between the teacher and the AI. Preferably the teacher would want to teach in english, but the AI doesn't understand english. There isn't a sufficient english to code translator around. One solution is for the teacher to use partial english. This reduces the need to interpret unnecessary parts of the sentence such as some pronouns and articles i.e. Instead of saying "It's easier to move the little boxes first" the teacher could say "move little boxes first." This reduces the command down to a verb, adjective, noun, and word telling the program in what order to move the boxes. Another solution is for the teacher to actually put the instructions into code. This is not preferable since many reasons for AI is to get it to interpret english commands, sometimes on the fly. Short instructions are no problem to put into code but should the instruction set be lengthy, the teacher will spend a lot of time coding the instructions, instruction by instruction, until the AI understands the way the teacher is teaching it. This can be time consuming. Especially if the AI doesn't learn it, or learns it incorrectly and new instructions need to be created to nullify what it has learned.

Learning by Exploration
Learning by exploration is a little different than the other ways of learning. The purpose of learning to explore is to just gather information, and not really pursue a goal. All it tries to do is find interesting information so it can store and learn from it. But it doesn't explore until it has nothing left to explore. It will follow a series of tasks. It will perform one task, which may add more tasks, and then move onto the next task. This causes a database of concepts to continue to grow. The program will organize the tasks in order of "interstingness." And it will also not always look at each task. Sometimes it need to determine what would be a waste of time exploring. This causes a problem because it needs some way of determining what task is worth exploring, and should it choose not to explore a task make sure it's not missing out on anything by ignoring it. Sometimes the program will find that the tasks it has left are not interesting enough to explore. If this happens it will go through its tasks and explore a "suggestions" slot so it can make the tasks more interesting. This way the program will more than likely not run out of tasks to explore. The program should also be able to generate concepts from what it already contains in it's database. This way it can generate more tasks to explore or just create new concepts that may have purpose in the real world.

2. RELATED WORK
[2] have said that Discovering complex associations, anomalies and patterns in distributed data sets is gaining popularity in a range of scientific, medical and business applications. Various algorithms are employed to perform data analysis within a domain, and range from statistical to machine learning and AI based techniques. Several issues need to be addressed however to scale such approaches to large data sets, particularly when these are applied to data distributed at various sites. As new analysis techniques are identified, the core tool set must enable easy integration of such analytical components. Similarly, results from an analysis engine must be sharable, to enable storage, visualisation or further analysis of results. We describe the architecture of PaDDMAS, a component based system for developing distributed data mining applications. PaDDMAS provides a tool set for combining pre-developed or custom components using a dataflow approach, with components performing analysis, data extraction or data management and translation. Each component is wrapped as a Java/CORBA object, and has an interface defined in XML. Components can be serial or parallel objects, and may be binary or contain a more complex internal structure. We demonstrate a prototype using a neural network analysis algorithm.

3. MODEL

Important Learning Systems

Following are the some of the important learning systems:

1. Epam
   It was developed by Feigenbaum. Its full form is Elementary Perceiver and Memorizer. It memorized pairs of nonsense syllables by using discrimination nets in which it could find images of syllabus it had seen. It stored as its cue only enough information to disambiguate the syllable from others seen at the time the association was formed. Thus old cues might be inadequate to retrieve data later, so the system could "forget."

2. Samuel's Checker Program
   It was developed by Arthur Samuel. The program used minimax search and a static evaluation in the game. It rote-learned positions by their minimax values. It could thus reuse the value in other games where the situation reappeared. He used a least-recently-used approach to caching. To learn, the system used sixteen features multiplied by weights. The weights changed through learning. The program actually knew 38 features, but only used 16 for evaluation at any given time; occasionally one was deleted and replaced by another in the evaluation function.

3. Winston's Learning Program
   It was developed by Winston. It is a nearly structural learning program that used semantic nets to describe block structures. Matching and hill climbing allowed the program to learn more complex structures; one major problem was that a teacher had to guide the program through a set of helpful examples.

4. Lex
   It was developed Mitchell and Utgoff. It is a concept learning program that created and refined heuristics that suggested whether a given operator would be applied to a given problem state in a forward-search problem solver. Each operator had associated with it a set of states in which it "should" be applied. LEX's domain was integral calculus. It used candidate elimination to determine good and bad instances of operator applications. The algorithm refined the version spaces of the operators. Multiple boundary sets were used to handle noise.

5. Meta-DENDRAL
   It was developed Buchanan and Mitchell. It is a program that does automatic theory formation using the version space learning algorithm. It is designed to infer theories (rule-sets) for heuristic DENDRAL, which represents knowledge about mass spectroscopy as production rules.

6. Hacker
   It was developed by Sussman. HACKER learns by doing; it tries to generalize subroutines and bugs. It has a gallery of critics which inspects the plans it generates for known generalized bugs. It uses the meaning of operators to guide the generalization process.

7. Prodigy
   It was developed by Minton. It is a general-purpose problem-solving system that uses several learning methods. Most learning is directed at automatically constructing a set of control rules to improve search in a given domain. It can acquire control rules in three ways:
   1. Through hand coding by programmers.
   2. through a static analysis of the domain's operators.

Proceedings of the 5th National Conference; INDIACom-2011

Copy Right © INDIACom-2011 ISSN 0973-7529 ISBN 978-93-80544-00-7
Evolution of AI learning systems

[Cheeseman, 1988] Given a set of training cases, it hypothesizes a set of classes. For any given case it provides probabilities to predict the class into which it will fall. AUTOCLASS found meaningful new classes of stars from infra-red spectral data.

9. Net
A connectionist, hardware implementation of a semantic network using local representations. Each node is a simple processing unit, capable of storing a few single-bit markers and performing simple Boolean operations on them. Each link is also a simple processor, wired to two or more units. Link units can also perform Boolean operations, which usually consist of passing a marker from one node to another.

10. AM / Eurisko
It was developed by Lenat Being an Amateur Mathematician he worked from basic concepts of set theory to discover a fair amount of standard number theory. It used a variety of general-purpose AI techniques, including a frame system to represent concepts. One of AM's major tasks was to create new concepts and fill their slots. An agenda controls the discovery process. AM operates in cycles, each cycle choosing the most promising task on the agenda. AM discovered the concept of prime numbers, the Unique Factorization Theorem, Goldbach's Conjecture (even numbers greater than 2 are the sum of two primes). AM's performance was limited by its static heuristics; the main source of its power was the large set of heuristics about what things are "interesting." AM has an implicit bias toward learning number theory concepts.

11. BACON
It was developed by Langley. A model of data-driven scientific discovery. BACON creates proportionalities in order to derive relations between data values. It uses other heuristics to postulate intrinsic properties of objects and to reason by analogy. It "discovered" Kepler's third law, Ohm's law, Joule's law, and conservation of momentum. BACON was criticized for lack of robustness and noise tolerance.

CONCLUSION
Learning provides important role in everyday's life.

FUTURE SCOPE
Such systems can provide adequate help in education

REFERENCES