International Journal of Semantic Computing
(2025) 1–38
© World Scientific Publishing Company
DOI: 10.1142/S1793351X25300018

Current AI: A Survey and Critique

George F. Luger
Professor Emeritus, Computer Science, Linguists, and Psychology
University of New Mexico, Albuquerque, NM 87113, USA
luger@cs.unm.edu
https:/www.cs.unm.edu/~luger

Received 29 September 2024
Accepted 9 October 2024
Published

With the recent attention of the user community to the "successes" of *large language models* (*LLMs*), it is easy to overlook the extension and depth of AI research and practice over the past seventy years. Beginning with the 1956 Dartmouth summer workshop where the name *artificial intelligence* was first suggested, this new discipline branched out in multiple directions. In fact, many of these early AI research areas were to become the foundation for the later emergence of what we now call computer science. In this paper, we review the highlights of the major research and application areas of artificial intelligence including the symbol-based, genetic/emergent, probabilistic and neural network/deep learning. We conclude with several comments on the problems and promise of AI and the current generation of LLMs.

Keywords: Symbol-based AI; genetic/emergent AI; probabilistic AI; Geep learning; AI critique.

1. Introduction

The word *artificial* is derived from two Latin words: first the noun, *ars/artis*, meaning "skilled effort", i.e., artist or artisan, and second, the verb *facere*, "to make". The literal meaning, then, of *artificial intelligence* is that something, namely intelligence, is made by skilled effort. Our first definition of Artificial Intelligence is that proposed near the end of the Dartmouth summer workshop proposal:

> For the present purpose the artificial intelligence problem is taken to be that of making a machine behave in ways that would be called intelligent if a human were so behaving.

This definition can be seen as directly related to Turing's test described in the journal *Mind* [1]. Turing conjectures that if observers are not able to determine whether they are interacting with a human or with a computer, the software on the computer must be seen as intelligent. The Dartmouth workshop attendees thought that problem solving in humans could be sufficiently understood that it could be "captured" as computer algorithms. Supporting the view that the mechanisms of intelligence can be automated, the workshop proposal claims:

> The (workshop) study is to proceed on the basis of the conjecture that every aspect of learning or any other feature of intelligence can in principle be so precisely described that a machine can be made to simulate it.

This definition still suffers from the fact that human intelligence itself is not very well defined and understood. Most of us are certain that we know intelligent behavior when we see it, but we cannot define intelligence in specific enough detail to simulate it with a computer program.

A further question asks whether it is necessary to pattern an intelligent computer program after what is known about human intelligence, or is a strict "engineering" approach to developing "intelligent" results sufficient? Is it even possible to achieve general intelligence on a computer, or does an intelligent entity require the richness of sensation, emotion and experience found only in a biological existence, as critics have suggested [2, 3]?

For these reasons, any definition of artificial intelligence falls short of unambiguously defining the field. If anything, it has only led to further questions and the paradoxical notion of a field of study whose major goals include its own definition. This difficulty in arriving at a precise definition of AI is entirely appropriate. Artificial intelligence is a young discipline, and its structure, concerns and methods are less clearly defined than those of more mature sciences such as physics.

For the time being, we will simply say that AI is *the collection of problems and methodologies studied by artificial intelligence researchers.* This definition may seem silly and meaningless, but it makes the important point that artificial intelligence, like every science, is an evolving human endeavor, and perhaps is best understood from that perspective.

In the following sections we summarize AI's continuing evolution across four research themes: the *symbolic*, the *genetic/emergent*, the *stochastic* and the *neural network* or *connectionist.* Each of these approaches to AI has made important contributions; we describe these briefly and offer examples. Several AI researchers have been given the ACM Turing Award and we describe their work. The Turing award is the highest recognition offered to researchers in computer science: the *Nobel* award for computer science.

2. Symbol-Based AI

Symbol-based AI, sometimes called *GOFAI* or *good old-fashioned AI* [4], requires that explicit symbols and sets of symbols reflect the world of things and relations within a problem domain. Euler's graph theory offers a foundation for what AI researchers called *state-space search.* Several different representational languages are available for describing states of the symbol-based world including the propositional and the predicate calculi. Symbol-based AI was also enabled by association-based representations including *semantic networks* and *conceptual graphs* [5].

Examples of symbol-based artificial intelligence include game-playing programs for chess, checkers and go; expert systems, where knowledge is encoded in explicit rule relationships; and control algorithms for robots and craft exploring deep space. The explicit-symbol system approach has been highly successful, although, as its critics point out, the resulting programs can be inflexible, poorly representing an evolving world. For example, how can an explicit symbol system adjust when a problem changes over time and is no longer exactly as encoded as in the original program? How can such a system compute useful results from incomplete or imprecise information?

The symbol-based approach to modeling human intelligence began even before the 1956 Dartmouth summer workshop. The *Logic Theorist* [6], created at Carnegie Institute of Technology, was a program to solve problems in the propositional logic, solving many of the problems in Whitehead and Russell's [7] second volume. A second early program was Gelernter and Rochester's [8] program at IBM that solved secondary school geometry problems. Samuel [9], also working at IBM, designed a program that played checkers. With these and other early game-playing programs, symbol-based AI became an important component of early AI.

The psychological and philosophical foundations for symbol-based AI came primarily from Newell and Simon working at what is now Carnegie Mellon University. Their representational mechanism was the *production system* originally proposed as a model for computation by Emile Post in the 1930s. Newell and Simon's [10] Turing Award lecture was entitled *Computer Science as Empirical Inquiry: Symbols and Search*. In this lecture, they hypothesized that the production system offered a *necessary and sufficient* model of human intelligence. Edward Feigenbaum, the designer at Stanford University of the expert system technology, Sec. 2.2, was Herb Simon's PhD student.

Some historical highlights of the symbol-based approach to AI include:

In the early 1960s, Marvin Minsky and John McCarthy began the MIT AI Laboratory. For his early AI research, Minsky received the ACM Turing Award in 1969.

In 1962, John McCarthy moved to Stanford University where he began the Artificial Intelligence Project. He was awarded the ACM Turing Award in 1971.

In the 1960s, through the 1980s researchers including Ross Quillian, Yorick Wilks, Roger Schank and John Sowa were created *semantic networks*, *conceptual dependencies*, *scripts* and *conceptual graphs*, association-based data structures intended to model human language use.

The development of the LISP (McCarthy), PROLOG (Marseille and Edinburgh Universities) and Smalltalk (Xerox Palo Alto Research Center) programming environments was critical for designing the early representations for the challenges of AI. In 2003, Alan Kay received the ACM Turing Award for his work in developing Smalltalk, the prototype for object-oriented languages.

> The *Physical Symbol System* hypothesis, research of Allen Newell, Herbert Simon and colleagues at CMU, described how intelligence could be modeled in humans and represented on machines. Newell and Simon received the ACM Turing Award in 1975.
>
> In 1965, Ed Feigenbaum, whose PhD research was at CMU with Herb Simon, his dissertation director, went to Stanford University. Feigenbaum led his research group developing the first expert systems. Feigenbaum is considered the "Father of Expert Systems" and received the ACM Turing Award in 1994.
>
> In 1966, Raj Reddy received his PhD at Stanford, as John McCarthy's student. Reddy moved to the faculty at CMU where he became the founding director of the *Robotics Institute*. He developed *Hearsay I*, the first program capable of continuous speech recognition. Reddy received the ACM Turing Award in 1994.

Symbol-based artificial intelligence was the primary focus of research in AI from the 1950s through the 1990s. It didn't disappear at that time, rather its successes became an important component of, and merged into, modern computer-based problem-solving practice. As an example, the inheritance hierarchies of the 1960s semantic networks [11] built into the early Smalltalk language, became a critical component of modern object-oriented programming.

2.1. *Symbol-based example: Game playing and heuristics*

After the creation of a state space of problem situation, a search algorithm is needed to explore this space. *Best-first* or *heuristic search* takes the "best" next state from all possible "next" states in the state space graph. Consider, for example, a heuristic in the game of tic-tac-toe, Fig. 1. The costs for exhaustive search for tic-tac-toe are high but not insurmountable. Each of the nine first moves have eight possible continuations, which in turn have seven moves, and so on through all possible board placements. An analysis puts the total number of states considered in exhaustive search as $9 \times 8 \times 7 \times \cdots \times 1$ or 9!, or 362,880 paths.

Symmetry reduction decreases this search space. Using symmetry there are not nine possible first moves but only three, as seen in Fig. 2: a corner, the center of a side, or the center of the grid. Use of symmetry on the second level further reduces the number of paths through the space to $12 \times 7!$, a best-first search heuristic can, in this case, eliminate search almost entirely, as seen in Fig. 3.

If you have first move and x, plan to go to the state in which x has the most possible winning opportunities. The first three states in the tic-tac-toe game are measured in Fig. 2. The best-first algorithm selects and moves to the state with the highest number of opportunities. In the case of states with equal numbers of potential wins, take the first such state found. In our example, x takes the center of the grid. Note that not only are the other two alternatives eliminated, but so will be all their descendants. Two-thirds of the full space is pruned away with the first move!

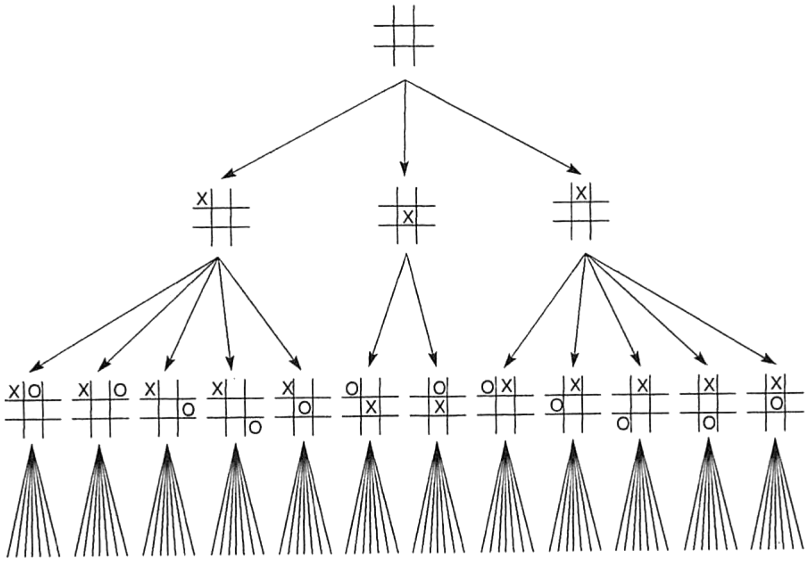


Fig. 1. The first three moves in the tic-tac-toe game where the state space is reduced by symmetry. The figure is adapted from [5].

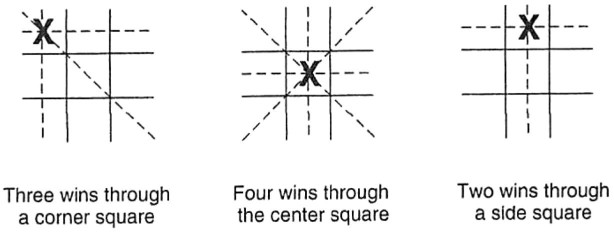

Three wins through a corner square
Four wins through the center square
Two wins through a side square

Fig. 2. The "most wins" strategy applied to the first move in tic-tac-toe.

After the first move, the opponent, o, can choose either of two moves, as seen in Fig. 3. Whichever state is chosen, the "most winning opportunities" heuristic is applied again to select among the possible next moves. As search continues, each move evaluates the children of a single node. Thus, exhaustive search is not required. Figure 3 shows the reduced search after three steps in the game, where each state is marked with its "most wins" value. For the first two moves of the X player only seven states are considered, considerably less than the 72 considered in exhaustive search. For the full game, "most possible wins" search has an even larger savings when compared to exhaustive search.

2.2. *Symbol-based example: The expert system*

The expert system, enabled by the production system technology, was created by Edward Feigenbaum's research group at Stanford University in the mid-1960s [5].

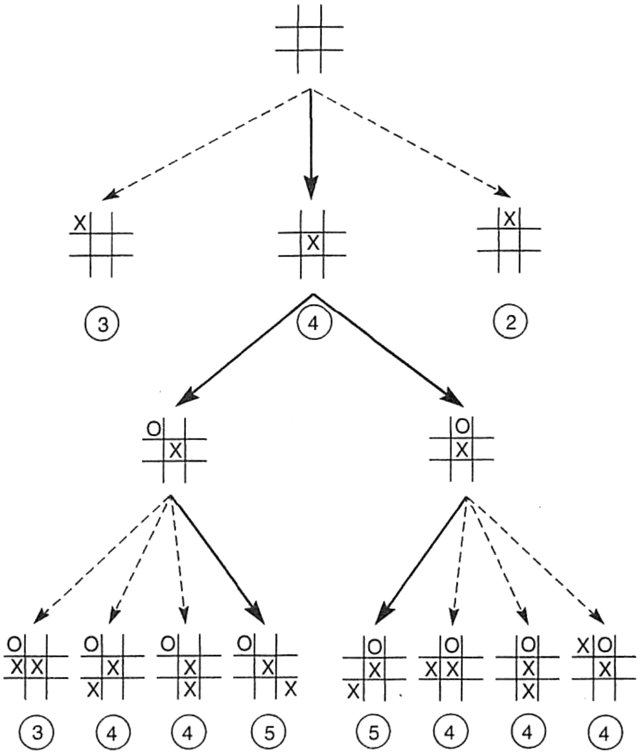


Fig. 3. The state space for tic-tac-toe reduced by using best-first search. The "most wins" strategy is used, with the bold arrows indicating the best moves. The figure is adapted from [5].

In a goal-driven rule-based expert system, where focus is on a problem to be solved or a diagnosis to be made, the problem's goal is considered: the problem is X. The program then finds an if... then... rule whose *conclusion* matches that goal and then focuses on that rule's *premises.* This action corresponds to working back from the problem's goal to supporting sub-goals. The process continues in the next iteration, with these sub-goals becoming the new goals to match against the rules' conclusions. This process continues until sufficient sub-goals are found to be true and thus indicate that the original goal is satisfied.

In an expert system, if a rule's premises cannot be determined to be true by given facts or using rules in the knowledge base, it is common to ask the human user for help. Some expert systems specify certain sub-goals that are to be solved by the user. Others simply ask the user about any sub-goals that fail to match rules in the knowledge base. Consider an example of a goal-driven expert system that has user queries when no rule conclusion is matched. This is not a full diagnostic system, as it contains only four simple rules for the analysis of automotive problems. It is intended to demonstrate the search of a goal-driven expert system, the integration of new data and the use of explanation facilities. Consider the rules:

```
Rule 1:
If
    the engine is getting gas, and
    the engine will turn over,
then
    the problem is spark plugs.

Rule 2:
if
    the engine does not turn over, and
    the lights do not come on
then
    the problem is battery or cables.

Rule 3:
if
    the engine does not turn over, and
    the lights do come on
then
    the problem is the starter motor.

Rule 4:
if
    there is gas in the fuel tank, and
    there is gas in the carburetor
then
    the engine is getting gas.
```

This is a very simple example. Not only is its automotive knowledge limited at best, but it also ignores several important aspects of actual implementations, including that the rules are phrased in English, rather than in a computer language. On finding a solution, an actual expert system will tell the user its diagnosis, although our example simply stops. If it had failed to determine that the spark plugs were bad, our expert system would have needed to back up to the top level and try Rule 2 next. Despite its simplicity, however, this example underscores the importance of rule-based search and representation by and/or graph search.

An important advantage of the expert system is its transparency in reasoning. First, all the rules are considered independent of each other, so in debugging an expert system, rules can simply be removed and replaced by "better" rules. This is an important example of the iterative refinement process: when the program produces results that are "wrong" in some sense, we replace them with better rules. The designer corrects his/her understanding of a problem by continuous improvement of rules that generate solutions.

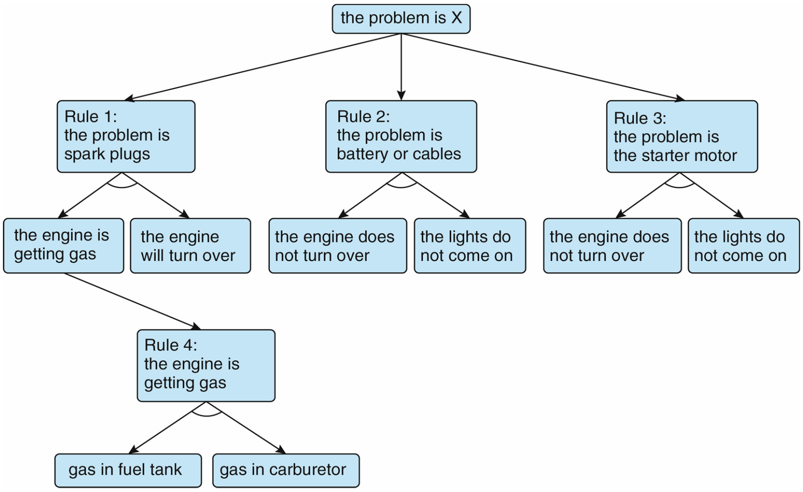

the problem is X
Rule 1: the problem is spark plugs
Rule 2: the problem is battery or cables
Rule 3: the problem is the starter motor
the engine is getting gas
the engine will turn over
the engine does not turn over
the lights do not come on
the engine does not turn over
the lights do not come on
Rule 4: the engine is getting gas
gas in fuel tank
gas in carburetor

Fig. 4. The and/or graph the expert system searches to determine the problem is spark plugs. A rule's and constraints are linked by an arc. The figure is adapted from [5].

A second advantage of the expert system is the transparency of the reasoning process. For example, suppose the automotive expert system has an explanation facility, and assume that the consultation has reached the stage represented by Fig. 4. The dialogue begins by the expert system asking the user about a goal it is unable to prove. User responses are in bold type:

gas in fuel tank?
yes
gas in carburetor?
yes
engine will turn over?
why

Here the user enters a why query, i.e., the user asks, "why do you want to know whether the engine will turn over?" The system responds with the presentation of the rule under question:

It has been established that:
the engine is getting gas,
therefore if
the engine will turn over,
then the problem is spark plugs.

Because the user answered **yes** to the first two questions, Rule 4 is satisfied, verifying the first premise of Rule 1. When the **why** query was made, the system was trying to solve the second premise of Rule 1. Note that the explanation for the system's behavior is simply a restatement of the current rule under consideration as this explains why the program wants to know whether the engine will turn over.

In the response to the why query, the system states that it has concluded that the engine is getting gas.

Suppose the user then asks:

how the engine is getting gas

The system will respond to how it concluded that the engine is getting gas. The response is a trace of the reasoning that led to this conclusion, working back from the goal along the rules that support it to the user responses:

```
This follows from Rule 4:
if
    gas in fuel tank, and
    gas in carburetor
then
    engine is getting gas
gas in fuel tank was given by the user
gas in carburetor was given by the user
```

2.3. *Symbol-based example: Controlling a deep space vehicle*

Planning has been an important component of symbol-based AI from the beginning. Planning takes a set of world-based descriptions and searches through them to find a goal or solution state. Planning has been used in the early days for finding a configuration of a set of blocks that satisfy certain constraints [12] or to get a robot to solve a particular task. It has advanced from these simple tasks to address more complex situations, such as assembly line production.

An example of a more complex planning task is designing a control system for Livingstone, NASA's deep-space reconnaissance vehicle, as seen in Fig. 5. Williams and Nayak [13, 14] created a model of the propulsion system for space

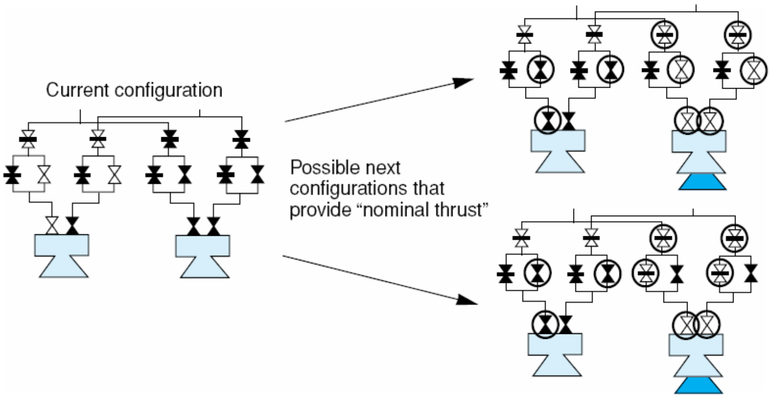


Fig. 5. The *Mode Configuration* of the propulsion system and the operations that change the state of the system. The figure is adapted from Williams and Nayak [13].

vehicles and a set of logic-based reasoning rules to address possible adverse situations, such as a blocked valve or a disabled thruster. The spacecraft's controller addresses these failings by changing the state of the system represented by a graph of possible next states that are chosen by the control system. As seen in Fig. 5, different control operations can take the system to different states. The Williams and Nayak [13, 14] approach for controlling space vehicles was successful. It remains to be seen if these control algorithms will be successful in even more complex situations, such as self-driving vehicles [15, 16].

2.4. *Symbol-based example: Information-based decision trees: ID3*

One final success story of symbol-based AI is data mining technology. Decision tree analysis programs, such as ID3 [17], are used on large sets of human data. The analysis of purchasing patterns, for example, is often used to predict a person's future credit needs and options. We next present the ID3 algorithm of a small sample of data.

Suppose a bank or department store wants to analyze the credit risk for new customers whose annual income is below $50,000. The bank or store considers earlier records of customers in this same income group. It then asks for equivalent information from the new credit applicants: that is, to build a profile of known customers' data to determine the risk for the new customers that wants credit. In a simplified example, Table 1 presents the data of 14 previous customer applicants.

In Fig. 6, ID3 uses information theory [18] to build a decision tree that analyzes the previous customers' data to determine credit RISK. The algorithm, using Shannon's formula, considers each of the four information sources, CREDIT HISTORY, DEBT, COLLATERAL and INCOME, to determine which piece of information best divides the population in the question of credit RISK. INCOME does this as is reflected in the first choice of Fig. 6.

Since the group in the left-most branch of Fig. 6 all have high credit RISK, that part of the search is finished: if you earn $15,000 or less, your credit rating is high RISK. The algorithm then considers the group on the middle branch to see which factor divides these people best for credit RISK and that factor is CREDIT HISTORY. The search continues until the decision tree of Fig. 6 is produced. Note

Table 1. The data of 14 lower income people that applied for credit. Data table is adapted from Quinlan [17].

No.	Risk	Credit history	Debt	Collateral	Income
1.	High	Bad	High	None	$0–15k
2.	High	Unknown	High	None	$15–35k
3.	Moderate	Unknown	Low	None	$15–35k
4.	High	Unknown	Low	None	$0–15k
5.	Low	Unknown	Low	None	over $35k
6.	Low	Unknown	Low	Adequate	over $35k
7.	High	Bad	Low	None	$0–15k

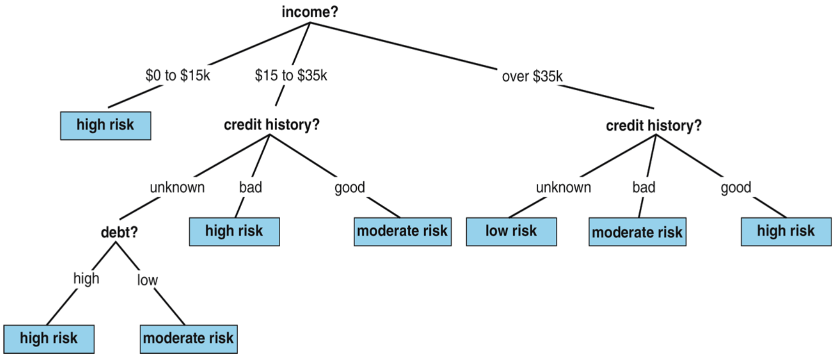


Fig. 6. The final decision tree produced from the data of Table 1 to be used for assessing the credit risk of new lower income customers. Example and figure are adapted from Quinlan [17].

that the COLLATERAL factor is not important. ID3 helps minimize the amount of information needed for analysis of new customers applying for credit: their amount of COLLATERAL is not useful for determining RISK. Full details on ID3 are available in Luger [19, Sec. 10.3.2].

3. Genetic and Emergent AI

A second theme of current artificial intelligence is the genetic and emergent approach to problem solving. Holland [20] of the University of Michigan, was a primary designer of *genetic algorithms*. Holland's algorithms are a natural extension of the "Randomness and Creativity" goal of 1956 Dartmouth workshop. Genetic algorithms use operators including *mutation*, *inversion* and *crossover*, to produce potential solutions for problems. The best of these possible solutions is then selected, using a *fitness function*, to create the next generation of possible solutions.

Genetic algorithms are another example of the space–space search we saw in Sec. 2.1. The difference is that new states are created by the "genetic" operators that transform the current states of a problem. Further, instead of focusing on single states as we saw in the examples of the previous section, these operators transform the "fittest" members of the current population of states. In this sense, they emulate the "survival of the fittest" members of that population.

Evolutionary programming goes back to the creation of digital computers. In 1949, John von Neumann asked what levels of organizational complexity were necessary for self-replication. John von Neumann's goal according to Burks [21] was

> ... not trying to simulate the self-reproduction of a natural system at the level of genetics or biochemistry. He wished to abstract from the natural self-reproduction problem its logical form.

Finite state automat offered the representational medium for artificial life research and von Neumann's research in the late 1940s. The development of genetic algorithms and programming began in the 1960s and extended into the late 1990s. Research in a-life, that now includes work in synthetic biology and chemistry, continues to the present [5, Sec. 6.3].

Some highlights of the genetic and emergent approaches to AI problem solving include:

> In the late 1940s, John von Neumann studied finite state machines for properties of self-organization and replication.
>
> From the 1960s through the 1990s, John Holland at the University of Michigan created genetic algorithms and classifier systems.
>
> The *Game of Life*, created in the 1970s by the mathematician John Horton Conway and made popular in *Scientific American* by Martin Gardiner, popularized a-life technology.
>
> In the 1990s, John Koza at Stanford University used concepts from genetic algorithms to create *genetic programming*.
>
> Current researchers [22–24] continue to explore creative issues in artificial chemistry and biology.

There are several genetic operators that produce offspring having features of their parents; the most common of these is *crossover*. Crossover takes two candidate solutions and divides them, swapping components to produce two new candidates. Figure 7 illustrates crossover on bit string patterns of length 8. The operator splits them in the middle and forms two children whose initial segment comes from one parent and whose tail comes from the other. Note that splitting the candidate solution in the middle is an arbitrary choice. This split may be at any point in the representation, and indeed, this splitting point may be randomly adjusted or changed during the solution process. Before showing problems solved by genetic algorithms, we present pseudocode for that algorithm:

```
Let P(t) be a list of n possible solutions, $x_1^t$, at time t:
P(t) = {$x_1^t$, $x_2^t$,..., $x_n^t$}
procedure genetic algorithm;
begin
    set time t to be 0;
    initialize the population P(t);
    while the termination condition of the problem is not met do
    begin
        evaluate fitness of each member of the population P(t);
        select pairs of members from population P(t) based on fitness;
        produce the offspring of these pairs using genetic operators;
        replace, based on fitness, weakest candidates of P(t);
        set new time to be t +1
    end
end.
```

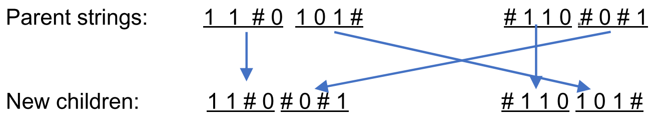


Fig. 7. Use of crossover on two bit strings of length eight. # is a “don’t care” bit.

For example, suppose the target class is the set of all possible bit strings of length 8 beginning and ending with a 1. Both the parent strings in Fig. 6 would have performed relatively well on this task. However, the first offspring would be much better than either parent: it would not have any false positives and would fail to recognize fewer strings that were in the solution class. Note also that its sibling is worse than either parent and will likely be eliminated over the next few generations.

Mutation is another important genetic operator. Mutation takes a single candidate and randomly changes some aspect of it. For example, mutation may select a bit in the pattern and change it, switching a 1 to a 0 or #. Mutation is important in that the initial population may exclude an essential component of a solution. In our example, if no member of the initial population has a 1 in the first position, then crossover, because it preserves the first four bits of the parent to be the first four bits of the child, cannot produce an offspring that does. Mutation would be needed to change the values of these bits. Other genetic operators, for example *inversion*, which reverses the order of the components of the representation, could also accomplish this task.

We next present three problems and discuss representation issues and fitness functions appropriate for their solutions. Three things should be noted: first, all problems are not easily or naturally encoded as bit level representations. Second, the genetic operators must preserve crucial relationships within the population, for example, producing usable components of code for a genetic programming problem. Finally, we discuss an important relationship between the fitness function(s) for a problem and the encoding of that problem.

3.1. *Genetic/emergent example: CNF satisfaction*

The conjunctive normal form (CNF) satisfiability problem is straightforward: an expression of the propositional calculus is in *CNF* when it is a sequence of clauses joined by an and ($\wedge$) relation and each of the sequence of clauses is in the form of a disjunction, the or ($\vee$) of the literals. For example, if the literals are, a, b, c, d, e and f, and ~ indicates that a literal is false, then the expression

$$(\sim a \vee c) \wedge (\sim a \vee c \vee \sim e) \wedge (\sim b \vee c \vee d \vee \sim e) \wedge (a \vee \sim b \vee c) \wedge (\sim e \vee f)$$

is in CNF. This expression is the conjunction of five clauses, each clause is the disjunction of two or more literals. CNF satisfiability means that we find an assignment of true or false (1 or 0) to each of the six literals, so that the CNF expression evaluates to true. The reader should confirm that one solution for the CNF expression is to assign false to each of a, b and e. Another solution has e false and c true.

A natural representation for the CNF satisfaction problem is a sequence of six bits, each bit, in order, representing true, 1, or false, 0, for each of the six literals, again in the order of a, b, c, d, e and f. Thus:

1 0 1 0 1 0

indicates that a, c and e are true and b, d and f are false, and the example CNF expression is false.

We require that the actions of each genetic operator produce offspring that are truth assignments for the CNF expression, thus each operator must produce a six-bit pattern of truth assignments. An important result of our choice of the bit pattern representation for the truth values of the literals of the CNF expression is that any of the genetic operators discussed to this point will leave the resulting bit pattern a legitimate possible solution, i.e., crossover, inversion and mutation all leave the resulting bit string a possible solution of the problem. Even other less frequently used genetic operators, such as *exchange,* interchanging two different bits in the pattern, leave the resulting pattern a legitimate possible solution of the CNF problem. In fact, from this viewpoint, it is hard to imagine a better suited representation than a bit pattern for the CNF satisfaction problem.

The choice of a fitness function for this population of bit strings is not quite as straightforward. From one viewpoint, an assignment of truth values to literals will make the expression either true or false. If a specific assignment makes the expression true, then the solution is found; otherwise, it is not. At first glance it seems difficult to determine a fitness function that can judge the "quality" of bit strings as potential solutions.

There are several alternatives, however. One would be that the full CNF expression is made up of the conjunction of five clauses. Thus, we can make up a rating system that will allow us to rank potential bit pattern solutions in a range of 0–5, depending on the number of clauses that pattern satisfies. The pattern:

1 1 0 0 1 0 has fitness 1,
1 0 0 0 1 0 has fitness 2,
0 1 0 0 1 1 has fitness 3, and
1 0 1 0 1 1 has fitness 5 and is a solution.

This genetic algorithm offers a reasonable approach to the CNF satisfaction problem. One of its most important properties is the use of the implicit parallelism that is afforded by operating on the entire population of potential solutions. The genetic operators have a natural fit to this representation. Finally, the solution search seems to fit naturally a parallel "divide and conquer" strategy, as fitness is judged by the number of problem components that are satisfied.

3.2. *Genetic/emergent example: Genetic programming*

Koza [25, 26] has suggested that a working computer program might evolve through successive applications of genetic operators. In genetic programming, the structures adapted are hierarchically organized segments of computer programs. The learning algorithm maintains a population of candidate programs. The fitness of a program is measured by the ability to solve a set of tasks, and programs are modified by applying crossover, mutation and other operators to a program's subcomponents.

Genetic programming searches a space of computer programs of varying size and complexity. The search space is of possible computer programs composed of functions and terminal symbols appropriate to the problem to be solved. These pieces consist of standard mathematical functions, logical and domain-specific procedures and other related programming operations; this search is random, largely blind and yet surprisingly effective.

The production of new programs comes with application of genetic operators such as crossover and mutation. These operators must be customized to produce new computer programs. The fitness of each new program is then determined by seeing how well it performs on the problem under consideration. Programs that do well on this fitness task survive to produce the children of the next generation.

We next present several examples, adapted from Koza [25] of genetic operators producing new programs. The Lisp computer language, created in the late 1950s by John McCarthy, one of the organizers of the 1956 AI Dartmouth summer workshop, is *functional*. Lisp program components are *symbol expressions*, or *s-expressions*. These symbol expressions have a natural representation as trees, where the function is the root of the tree and the arguments of the function, either terminating symbols or other functions, descend from the root. Figure 8 offers examples of s-expressions represented as trees. Operators map these structures of s-expressions into new trees that are Lisp program segments.

When setting up a domain for creating programs to address a set of problems, first analyze what terminal symbols are required. Next select program segments sufficient for producing these terminals. As Koza notes [25, p. 86] "... the user of genetic programming should know ... that some composition of the functions and

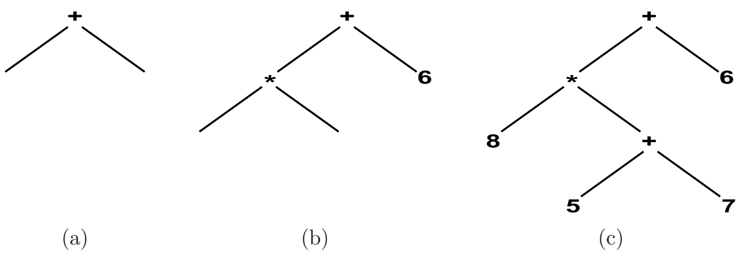


Fig. 8. The random generation of a program of Lisp s-expressions. The operator nodes, +, *, are from the set of Lisp functions. The figure is adapted from Koza [25].

terminals he supplies can yield a solution of the problem." The functions can be as simple as {+, *, −, /} or more complex functions such as sin(X), cos(X), or matrix operators. Terminals can be the integers, real numbers, matrices, or more complex expressions. The terminal symbols must include all symbols that the create function set defined can produce.

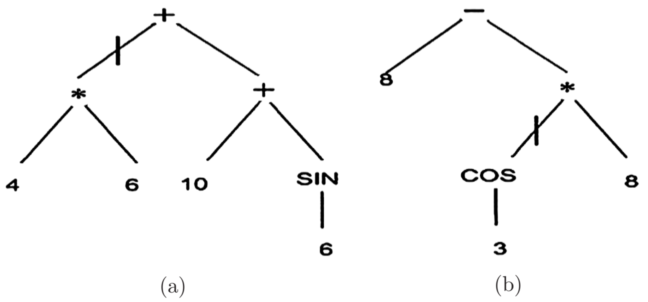
A population of initial "programs" is generated by randomly selecting elements from the union of the functions and terminals. Start with an element from the functions, say +, and get a root node of a tree with two potential children. Suppose the initialization then selects *, with two potential children, as the first child, and then terminal 6 from as the second child. Another random selection might yield the terminal 8 and then the function +. Assume it concludes by selecting 5 and 7 from the terminals.

The program just produced is represented in Fig. 8. Figure 8(a) gives the tree after the first selection of +, Fig. 8(b) after selecting the terminal 6 and Fig. 8(c) the final program. A population of similar programs is created to begin the genetic programming process. Sets of constraints, such as the maximum depth for programs to evolve, can help control population growth. A more complete description of these constraints, as well as different methods for generating initial populations, may be found in Koza [25].

The discussion to this point addresses the issues of representation, s-expressions, and the set of tree structures necessary to initialize a situation for program evolution. Next, we require a fitness measure for populations of possible programs. The fitness measure is problem-dependent and usually consists of a set of tasks the evolved programs are intended to solve. The fitness measure itself is a function of how well each program does on these tasks. One example fitness measure is called *raw fitness.* This score adds the differences between what a program produces and the results that the actual task of the problem required. Thus, raw fitness is the sum of errors across a set of tasks. *Normalized fitness* divides raw fitness by the total sum of possible errors which puts all fitness measures within the range of 0 to 1. Fitness measures can also include an adjustment for the size of the program, for example, to reward smaller, more compact programs. An example of a fitness test is presented in Sec. 3.3.

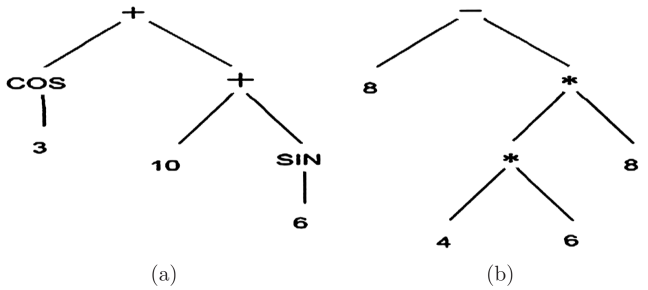
Genetic operators, besides transforming program trees, also include the exchange of structures between trees. Koza [25] describes the primary transformations as *reproduction* and *crossover*. Reproduction simply selects programs from the present generation and copies them unchanged into the next generation. Crossover exchanges subtrees between the trees representing two programs.

For example, suppose we are working with the two parent programs of Fig. 9, and that the random points indicated by | in parents a and b are selected for crossover. The resulting children are shown in Fig. 10. Crossover can also be used to transform a single parent by interchanging two subtrees within that parent. Two identical parents can create different children with randomly selected crossover points. The root of a program can also be selected as a crossover point.



(a) (b)

Fig. 9. Two programs, selected for fitness, are randomly chosen for crossover. The "|" represents the point selected for crossover. The figure is adapted from Koza [25].



(a) (b)

Fig. 10. The child programs produced by the crossover operator applied in Fig. 8. The figure is adapted from Koza [25].

The state of the solution is reflected by the current generation of programs. There is no record keeping for backtracking or any other method for skipping around the fitness landscape. From this viewpoint, genetic programming is like *hill-climbing* [27] where the "best" children are selected at any time, regardless of what the ultimate best program might be. The genetic programming parallels nature in that the evolution of new programs is a continuing process. Nonetheless, lacking infinite time and computation, termination conditions are often necessary.

The fact that genetic programming is a technique for the computational generation of computer programs also places it within the *automated programming* research tradition. From the earliest days, AI practitioners have worked to create programs that automatically produce programs and solutions from fragmentary information. Genetic programming is another tool in this important research domain.

3.3. *Genetic/emergent example: Kepler's third law of planetary motion*

Koza [25, 26] describes many applications of genetic programming that solve interesting problems, but most of his examples are rather large and very complex. Mitchell [28] has created an example that illustrates many of the concepts of

Table 2. A set of observed planetary data, adapted from Urey [29], used to determine the fitness of each evolved program. A is Earth's semi-major axis of orbit and P, the length of time for an orbit, is in units of earth-years.

Planet	A (input)	P (output)
Venus	0.72	0.61
Earth	1.0	1.0
Mars	1.52	1.87
Jupiter	5.2	11.9
Saturn	9.53	29.4
Uranus	19.1	83.5

genetic programming. Kepler's *Third Law of Planetary Motion* describes the functional relationship between the orbit period, P, of a planet and its average distance, A, from the sun, as is shown in Table 2.

Kepler's Third Law, with c a constant, is

$$P^2 = cA^3.$$

If we assume that P is expressed in units of earth years, and A in units of earth's average distance from the sun, then $c = 1$. An s-expression representing this relationship is

$$P = (\text{sqrt } (* \ A \ (* \ A \ A))).$$

Thus, the program we want to evolve for Kepler's third Law is represented by the tree structure of Fig. 11. The selection of the set of terminal symbols in this example is the single real value given by A. The set of functions are {+, −, *, /, sq, sqrt}.

We begin with a random population of programs. This population might include:

$$(* \ A \ (- \ (* \ A \ A) \ (\text{sqrt } A))), \text{ with fitness} = 1,$$

$$(/ \ A \ (/ \ (/ \ AA) \ (/ \ AA))), \text{ with fitness} = 3,$$

$$(+ \ A \ (* \ (\text{sqrt } A) \ A)), \text{ with fitness} = 0.$$

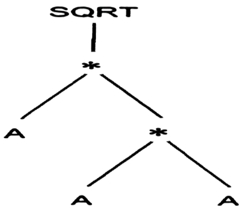


Fig. 11. The target program for relating the orbit P to the period for Kepler's Third Law. A is the average distance of the planet from the sun. The figure is adapted from Mitchell [28].

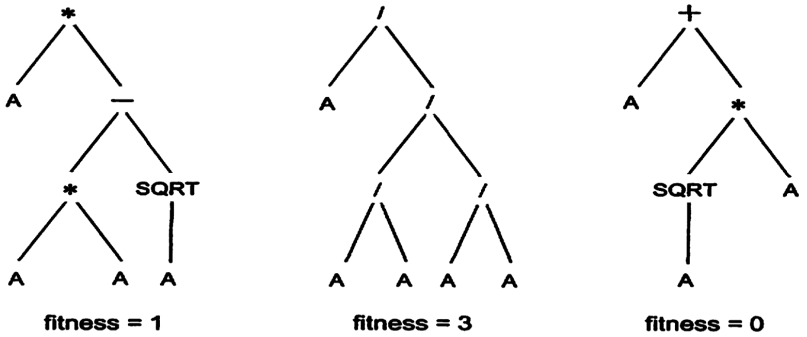


Fig. 12. Members of the initial set of random programs generated to solve the orbital/period problem. The figure is adapted from Mitchell [28].

As noted earlier, the initial population often has a priori limits, both of size and search depth, given knowledge of the problem. These three examples are represented with the program trees of Fig. 12. We next determine a fitness test for the population of programs using the planetary data we want our evolved program to explain, i.e., the data of Table 2.

Since our task is to create a function reflecting the data points of Table 2, a fitness measure is the number of results, within a 20% tolerance, from running each program. Using this criteria, Fig. 12 shows the fitness of each program. It remains for the reader to create more members of this initial population, to build crossover and mutation operators that can produce further generations of programs, and to determine termination conditions.

4. Probabilistic Models

The third theme for contemporary AI research and practice is *probabilistic* model building for diagnosis. In the mid-18th century, a Church of England clergyman, Bayes [30], proposed a formula for relating information already learned, the *prior*, to newly observed data, the *posterior*.

In many complex situations, the task of computing all Bayesian relationships, i.e., all the probabilistic information required to support full Bayesian reasoning, can be prohibitive. Where there are multiple hypotheses and large amounts of supporting data, the information required for Bayesian analysis is large. For example, a medical application where there are 200 possible diagnoses with 2000 possible symptoms requires the collection of more than 800,000,000 distributions [19, p. 185]. Nonetheless, Bayes' theorem has been used to support the reasoning process in several early expert systems, for example, in searching for mineral deposits and diagnostic systems for internal medicine [19, Sec. 5.3].

Pearl's 1988 book [31], *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*, introduced the *Bayesian belief network*, or *BBN*, technology. The BBN is a reasoning graph with the assumptions that the network is directed, reflecting causal relationships, and is acyclic, i.e., has no loops from a node back to itself. With the creation of the BBN, several computationally efficient algorithms became available for reasoning. Pearl [32] also wrote *Causality: Models, Reasoning and Inference*, in which his *do-calculus* offered a mathematical framework for building models in which, over time, supported reasoning about plausible causal relationships or "what if" scenarios.

The stochastic approach is used extensively in machine learning and robotics. It is especially important for human language understanding, leading to important results in computer-based speech and written language analysis. *Dynamic Bayesian networks* or *DBNs*, offer a representation able to characterize how the activity of complex systems can be modeled and understood across time. In the examples that follow, we demonstrate how stochastic representation and reasoning schemes are sufficient to capture components of human perception and reasoning.

Some historical highlights in the probabilistic or stochastic approach to AI include:

> Bayes [30] proposed a formula for relating information already learned, the *prior*, to data newly observed, the *posterior*. Other 18th century mathematicians, Pascal and later Laplace, considering the domain of gambling, began the development of a probabilistic calculus.
>
> In the 1950s and 1960s, probabilistic models for digit and other character recognition tasks were developed at Bell Labs [33] and elsewhere [34].
>
> Authorship attribution studies used probabilistic language models. The analysis of text samples supported assigning known authors to anonymous literature [35].
>
> The 1990s demonstrated many new probabilistic algorithms for natural language understanding and generation [36].
>
> Judea Pearl introduced the Bayesian Belief Net technology [31]. Pearl's book, *Causality* [32] described the potential of dynamic probabilistic systems. Judea Pearl received the ACM Turing Award in 2011 for his development of a calculus for probabilistic and causal reasoning.

The primary current interest in probabilistic techniques in AI has been from the 1990s to the present, supporting the goals of language understanding, visual scene analysis and other classification problems.

4.1. *Example of probabilistic reasoning: Bayesian belief networks*

Data collection is a limiting factor for using full Bayesian inference for diagnoses in complex environments. As just noted, to calculate probabilities in medicine, where there can be hundreds of possible diagnoses, and thousands of possible symptoms, the data collection problem becomes intractable.

The *Bayesian belief network* or *BBN* [31, 32] is a graph whose nodes are represented by probabilities and whose links are conditional probabilities. The graph is *acyclic*, in that there are no link sequences from a node back to itself. It is also *directed*, in that links are conditioned probabilities that represent causal relationships between the nodes. With these assumptions, it can be shown that a BBN's nodes are independent of all their non-descendants, nodes that they are not directly or indirectly linked to, given knowledge of their parents, i.e., nodes linking to them.

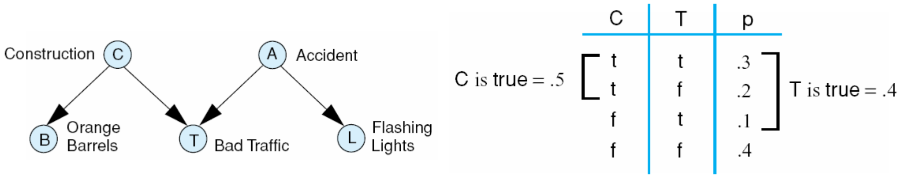
Judea Pearl's proposed Bayesian belief networks assume that their links reflect causal relationships. With the demonstrated independence of states from their non-descendants, given knowledge of their parents, Bayesian technology comes to an entirely new importance. Most importantly, the independence assumption that splits, or *factors*, the reasoning space into independent components, makes the BBN a transparent representational model that captures causal relationships in a computationally useful format. We demonstrate, in our next examples, how the BBN supports both transparent and efficient reasoning.

The BBN, before new data are presented, represents the a priori state of an expert's knowledge of an application domain. These networks of causal relationships are usually carefully crafted through many hours working with human experts. When new data are given to the BBN, e.g., road traffic slows as we see next, the network "infers" the most likely explanation, given its a priori model of the situation.

Figure 13 shows a BBN model for a typical traveling situation. Suppose you are driving your car in a familiar area where you are aware of the likelihood of traffic slowdowns, road construction and accidents. You are also aware that flashing lights often indicate emergency vehicles at an accident site and that orange traffic control barrels indicate construction work on the roadway. We name these situations T, C, A, L and B, as seen in Fig. 13. The likelihood of each parameter is reflected in the partial probability table of Fig. 13, where the top row indicates that the probability of both construction, C and bad traffic, T, being true, t, is 0.3.

For full Bayesian inference, this problem would require a 32-row probability table of 5 variables, each either true or false. In the separation, or *factoring*, that BBN reasoning supports, this becomes a 20-row table where Flashing Lights is independent of Construction, Orange Barrels is independent of Accident and Construction and Accident are also independent. Figure 13 presents a portion of this table.

Suppose that as you drive along and without any observable reasons, the traffic begins to slow down; now Bad Traffic, T, becomes true, t. This new fact means



C	T	p
t	t	.3
t	f	.2
f	t	.1
f	f	.4

Fig. 13. Bayesian belief network (BBN) for the driving example and a partial table of probability values for Construction, C and Bad Traffic, T. The figure is adapted from Luger [19].

that the probabilities of the table in Fig. 13, Bad Traffic is no longer false. The sum of the probabilities for the first and third lines of the table goes from $t = 0.4$ to $t = 1.0$. This new higher probability is then distributed proportionately to the probabilities for Construction and Accident and, as a result, both situations become more likely.

4.2. *Example of probabilistic reasoning: The dynamic Bayesian network*

A *dynamic Bayesian network*, or *DBN*, is a sequence of identical Bayesian networks whose nodes are linked in the directed dimension of time. This representation extends BBNs into multi-dimensional environments and preserves the same tractability in reasoning toward best explanations. With the factoring of the search space and the ability to address complexity issues, the dynamic Bayesian network becomes a potential model for exploring diagnostic situations across both changes of data and time. We next demonstrate the DBN.

Continuing the driving example, suppose as you travel farther you notice Orange Barrels, B, along the road that partially redirect traffic. This indicates that, on another probability table not shown here, B is true, with its probabilities summing to 1.0. The probability of Construction, C, gets higher, approaching 0.95. As the probability of Construction gets higher, with the absence of Flashing Lights, L, the probability of an Accident decreases. The most likely explanation for what you now experience is road Construction. The likelihood of an Accident goes down and is said to be *explained away*. The calculation of these higher probabilities as new data are encountered is called *marginalization* and, while not shown here, may be found in Luger and Chakrabarti [37].

Figure 14 represents the changing dynamic Bayesian network for the driving example just described. The perceived information changes over the three time periods: driving normally, cars slowing down and seeing orange traffic control barrels. At each new time with new information, the values reflecting the probabilities for that time and situation change. These probability changes reflect the best explanations for each new piece of information the diver perceives.

Finally, in Fig. 14, consider the state of the diagnostic expert at the point where Time = 2. Once traffic has slowed, the driver may begin an active search for Orange Barrels or Flashing Lights, to try to determine, before Time = 3, what might be the most likely explanation for the traffic slowdown. In this situation, the driver's

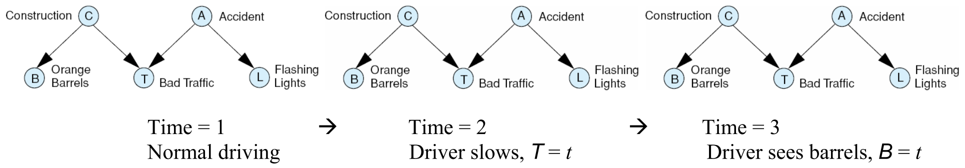


Fig. 14. Dynamic Bayesian network. Each time the driver perceives new information, the DBN's probabilities change to reflect these observations. The figure is adapted from Luger [19].

expectations motivate his/her active search for supporting information. These changing situations and their explanations are shown again in the next example.

4.3. *Probabilistic reasoning: Monitoring the nuclear production of energy*

Our final example comes from building computational models to monitor potential problems in producing electric power using a sodium-cooled nuclear reactor. Nuclear accidents are rare, but their effects are extremely harmful for people, the environment and the economy. Jones *et al.* [38] and Darling *et al.* [39] in research supported by the US Department of Energy, designed a computational monitoring system based on dynamic Bayesian networks to support the observations and knowledge of the human nuclear power experts. There are several reasons for employing the DBN technology in this challenging environment.

First, the Bayesian network is composed of nodes and links that reflect expert human knowledge and judgment in the field of reactor physics. This fact is important as the day-to-day monitors are usually not as skilled as the experts that designed the system. The probabilities of the DBN also reflect the results of multiple tests on individual components of the power system, such as sensors, as well as on simulations of the full working environment. Thus, the resulting model contains both explicit human physics and engineering knowledge as well as a probabilistic account of the reactor's running health.

Second, in the very complex environment on nuclear power generation, the DBN can produce faster than real-time analytic and diagnostic results. Murphy [40] has described the transparent and tractable reasoning powers of DBN-based technology. The human monitor can be made aware of events as soon as, and often before, they happen. The monitor also receives from the model suggestions and recommendations for remediating potential problems.

Figure 15 presents a schematic for a sodium-cooled nuclear reactor to produce electric power. The reactor system's model has multiple sensors monitoring the states of the pumps, the temperatures of the various vessels, the positions of the control rods and the turbine speeds. The 10 monitors of the state of the power generation system are represented by the rectangular boxes of Fig. 16. The circles of Fig. 16 represent the nodes of the DBN and the cylinders extending off the circles represent the values of each circle changing over time.

Training the dynamic Bayesian network takes place as the power generation system runs across multiple scenarios and time cycles. First, we train the basic components of the model on near normal data to establish a state of equilibrium for the system. The model, in a near normal running situation, can also provide approximate values for missing sensor data from the system. The values proposed for missing information or for damaged sensors are what the model determines to be most likely, given the current state of the running system [41, 42].

Once the equilibrium model was trained, the research group generated multiple accident sequences using their simulation system. In each scenario, for example, having

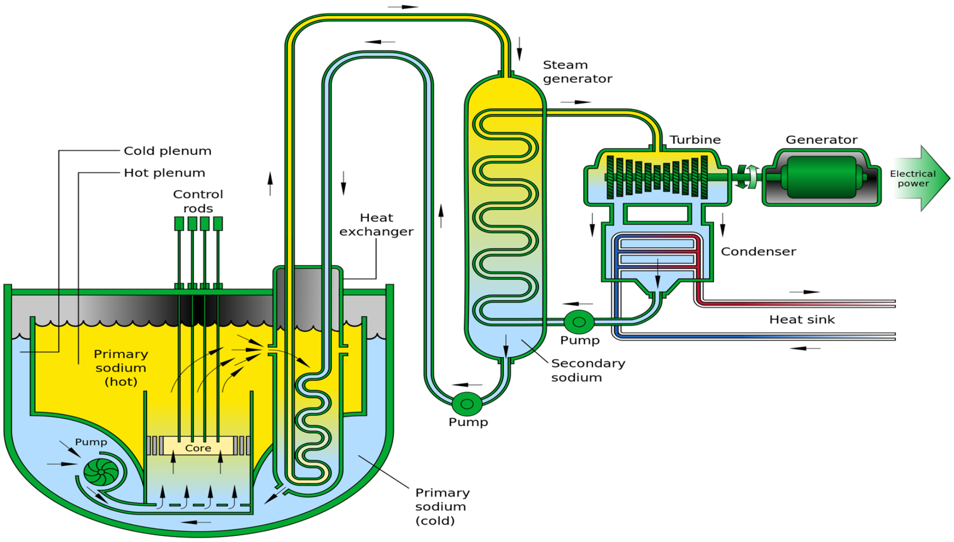


Fig. 15. The schematic of a sodium-cooled nuclear power generation system. The various reservoirs, pumps, control rods, turbine, etc. have sensors reporting their states to the DBN, as seen in Fig. 16. The figure is adapted from [39].

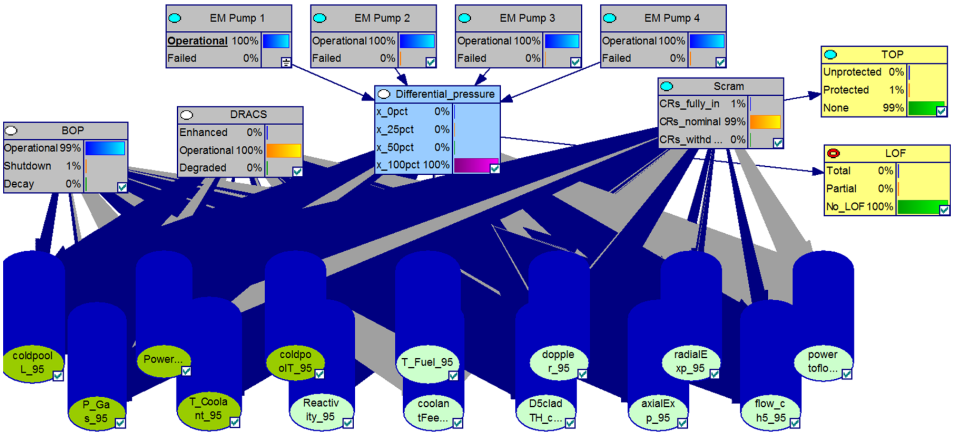


Fig. 16. The dynamic Bayesian network of the sodium cooled reactor of Fig. 15. The ten rectangular boxes represent monitors collecting reactor sensor data. The circles represent the nodes of the DBN, lines from the rectangles to the circles represent the probabilistic links of the network, and the cylinders emanating from these nodes represent the nodes changing over time. The figure is adapted from [39].

differential pressures within the plant's cooling system or performing control rod insertion, the model captured the state of the system as the "accident" evolved. This allowed visualization of all parameters related to each accident as well as presented options for remediation. The fact that these options could be realized in faster than real time supports the human operators' ability to make decisions for remediation.

Once the DBN model was trained, the research group generated multiple accident sequences using their simulation system. In each scenario, for example, having differential pressures within the plants cooling system or performing control rod insertion, the model captured the state of the system as situations evolved. This allowed visualization of all parameters related to each accident as well as presented options for remediation. The fact that these options could be realized almost incautiously supports the human operators' steps toward remediation.

Hypothetical reasoning is supported by the fact that the computational model offers an accurate reflection of the power-producing reactor. The knowledge-based probabilistic model allows monitors to try out different control strategies and get almost immediate feedback on what would happen, as well as the time sequence for it to occur. Examining these possible responses can direct the reactor monitors to make the most informed decisions at appropriate times. This trained computational model captures the human-like reasoning that an informed diagnostic expert would offer in similar situations.

5. Neural Networks: Deep Learning

The final major approach to current AI technology is the *neural,* or *connectionist, network*. Neural network research began with conjectures in the 1940s by the neuroscientist McCulloch and the logician Pitts [43] and by Hebb [44] a psychologist.

The basis for neural network computing is the artificial neuron, an example of which may be seen in Fig. 17. The artificial neuron consists of *input signals* x_i, and often a *bias* that comes from the environment or from other neurons. There is also a *set of weights,* w_i. that enhance or weaken the strengths of the input values. Each neuron also has an *activation level,* $\Sigma w_i x_i$, the value of net: the summed strengths of the input times their weight measures. Finally, there is a *threshold function, f,*

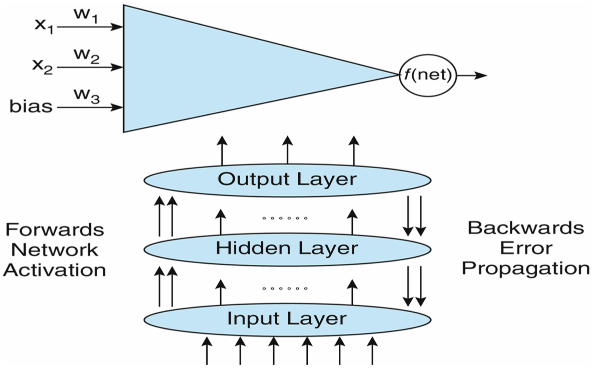


Fig. 17. (Top) A single artificial neuron whose input values, multiplied by trained weights, produce a value, net. Using some function, f(net) produces an output value that may, in turn, be an input for other neurons. (Below) A *supervised learning* network where input values move forward through the nodes of the network. During training, the network's weights are differentially adjusted, error propagation, for incorrect responses to input values.

that computes the neuron's output by determining whether the neuron's activation level is above a predetermined threshold.

$$\text{net} = x_1 w_1 + x_2 w_2 + (\text{bias}) w_3.$$

In addition to the properties of individual neurons, the network also has global properties including the numbers and pattern of connections between the individual neurons and the different layers of neurons. Further properties include learning rates, training batch sizes and activation functions. Finally, there is the *encoding scheme* that interprets problem data input as well as the output result. Encoding/decoding determines how an application is presented to the nodes of the network as well as how the results from the network are interpreted after network processing.

There are two primary approaches to neural network learning: *supervised* and *unsupervised*. Figure 16 (below) shows *supervised learning*. In *unsupervised learning*, there is no feedback to the input weights w_i, given output values. In fact, some algorithms don't require weights at all. Output values are calculated by the structure of the data itself interacting with the network or combining with other outputs as they self-organize into useful clusters.

With the advent of very high-performance computing and parallel algorithms for computing, it has now become common to have networks with multiple internal layers and complete networks passing data off to other networks. This approach, sometimes referred to as *deep learning,* has brought an entirely new dimension to the power and possibilities of connectionist computing.

Some historical highlights of the neural network approach to AI include:

The research of D.O. Hebb and W.S. McCulloch and W. Pitts at MIT in the late 1940s.

An early network, the *Perceptron*, was created by Rosenblatt [45].

Minsky and Papert's book *Perceptrons* [46] demonstrated limitations of the perceptron technology. AI interest and funding slowed at that time.

The Boltzmann machine [47] and the backpropagation algorithm [48] addressed the problem of error propagation through multiple layered networks.

In 1989, the publication of *Parallel Distributed Processing* by Rumelhart and colleagues [48] in the PDP Group at UCSD, renewed interest in neural networks.

In the 2000s, Hinton and his colleagues [49] demonstrated improvements to image processing using convolutional neural networks.

In 2018, Yoshua Bengio, Geoffrey Hinton and Yann LaCun were given the ACM Turing award for their many engineering improvements supporting deep learning technology.

In 2024, Geoffrey Hinton and John Hopfield received the Nobel Award in physics for work on neural networks.

> From the late 2010s, deep learning computing was greatly enhanced by the wide availability of tensor processors and server farms. This generation of networks, the *large language models*, or *LLMs*, is called *transformers*.

The major emphasis in neural network computing was from the mid-1940s until the late 1960s and from the late 1980s until the present day. We next present four examples.

5.1. *Neural network example: AlphaGo Zero and Alpha Zero*

AlphaGo Zero is an extension of the original *AlphaGo* program created by Google's DeepMind research group. Its major strength is that it taught itself to play go without any experience against human players. It was simply given the rules of the game and then started playing against a version of itself. After 40 days and 29 million games the program proved better than all earlier versions of *AlphaGo* [50, 51]. This result is interesting in that it demonstrates that knowing only the rules of a game and playing against itself, the program learns sufficient skills to defeat the best human players.

AlphaZero, also created by Google's DeepMind, takes deep learning coupled an important step beyond *AlphaGo Zero*. AlphaZero, joins deep learning with reinforcement learning [5, Sec. 11.4.2] to create an architecture general enough to play several different games. Besides go, *AlphaZero* also learned to play chess and shogi. With only three days of training, it was able to outperform all chess and shogi programs and a version of *AlphaGo Zero*. With reinforcement learning, *AlphaZero* searched 1000 times fewer states than did its computer-based opponents.

5.2. *Neural network example: Robot navigation: PRM-RL*

We noted in Sec. 2.3 that AI robotics programs used the state space and search to accomplish tasks. Modern robotics has taken these earlier search-based approaches to entirely new levels, using deep learning coupled with reinforcement learning to support exploring environments. At Google Brain, Faust and her colleagues [52] created a robot navigation system, PRM-RL, that uses *probabilistic roadmaps* and *reinforcement learning* to find paths in complex environments.

A probabilistic roadmap planner [53] has two phases: first, a graph-based map is built that approximates the movements the robot can make within its environment. To build this roadmap, the planning algorithm first constructs a set of possible partial paths by considering links between accessible locations it discovers in the environment. In the second phase, the actual goal for the robot is linked to the graph and the algorithm determines the shortest path to that goal.

The reinforcement learning component of PRM-RL is trained to execute point-to-point tasks, to learn constraints, system dynamics and sensor noise independent of the ultimate task environment of the robot. In the testing environment, the PRM-RL program builds a roadmap using reinforcement learning to determine connectivity. The reinforcement learning algorithm joins two configuration

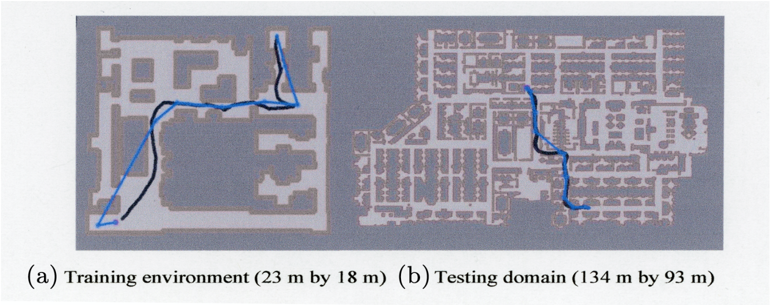

(a) Training environment (23 m by 18 m)
(b) Testing domain (134 m by 93 m)

Fig. 18. (a) represents the training environment for the robot and (b) the testing environment. The heavier line indicates the actual path taken by the robot. The figure is adapted from [52].

points only when search finds point-to-point connectivity between them, avoiding all obstacles. Figure 18(a) shows the training map within a 23 m by 18 m building. Figure 18(b) shows the testing environment, the 134 m by 93 m floor of a building.

All these approaches are, as mentioned earlier, computer time- and cost-intensive. Because of this complexity, a major challenge to deep reinforcement learning is analyzing frequently long successful search paths and identifying appropriate states within that search to “reinforce”. It is also very impressive that Google’s PRM-RL robot can be trained in one environment and then transfer that learning to a new related but different situation, as seen in Fig. 18.

5.3. *Neural network example: Deep learning and video games*

Deep learning algorithms that play video games use similar approaches to those of Google’s *AlphaZero* [50, 51], introduced in Sec. 5.1. The input values for the network are the game’s pixelated video screen and the current game score. There is no model for the game situation, as would be needed in a symbol-based approach. For example, the symbolic approach would represent the agent that is playing and learning the game, the target goals and the tools or weapons for achieving these goals, as well as rules for attack, defense, escape and so on.

Given the current screen and game score, reinforcement learning represents the state of the player and the game choices available. In video games these choices can be very large, estimated about 10^{50}, while the maximum number of choices a go player has is about 10^{2}. The video game choices are in multiple categories, including movement, weapon use and defensive strategies. The reinforcement algorithm has probabilistic estimates of the quality of each of these choices, given the current state of the game. These probabilistic measures are determined by the previous successes of making that choice, given that state.

When the reinforcement learning begins, there is very little reward information and the agent’s moves will seem both exploratory and erratic. As multiple games are played, the reward algorithm gets more “success” information, and the agent’s choices improve and eventually so does winning. The learning process for a video game playing computer requires multiple millions of games of a program playing against a version of itself and can cost several millions of 2018-dollars.

Deep learning video game programs including Google's DeepMind's AlphaStar program playing StarCraft II [54] have successfully outperformed humans in single player games. *AlphaStar* achieved Grandmaster status in August 2019. A single agent game program, with full explanations and code that uses Q-leaning, a model-independent reinforcement learning algorithm, to play the video game Snake can be found at https://medium.com/@hugo.sjoberg88/using-reinforcement-learning-and-q-learning-to-play-snake-28423dd49e9b.

Many interesting video games require teamwork. Jaderberg *et al.* [55] designed a program that plays *Quake III Arena* in *Capture the Flag* mode. Its agents learn and act independently to cooperate and compete with other agents. Again, the reinforcement learner uses only screen images and the game score as input. The population of reinforcement learning agents is trained independently and in parallel. Each agent learns its own reward pattern that supports its active interactions within the game environment.

5.4. *Deep learning example: Large language models*

From the early 1990s until about 2015, the traditional computer methods used to understand human language, summarize documents, answer questions and to translate speech and text from one language to another were probabilistic. More recently, alternative technologies, including deep learning, address these same human language tasks.

A *language model* characterizes how the components of a language work together in communication. These models capture relationships including noun verb agreement, proper use of adjective and adverbs, how clusters of words are often used together and much more. Many deep learning-based language models are currently available, with perhaps two, Google's BERT [56] and OpenAI's GPT-3 [57] most widely used; see Table 2.

Deep learning neural language models are trained networks that capture the relationships between words in a language. There are several regimens for training these models. One more traditional approach asks, given the n words that precede an unknown word, what is the most likely word that would be. Google's BERT [56] also considers the n words that follow that unknown token to determine what word is most likely to precede these words.

Training takes place by giving these learning networks an extremely large number of sentences, for example all the Wikipedia corpora. As of December 2020, this had more than 6 million articles with almost 4 billion words. Other large corpora include the *Google Books Corpus*, the *International Corpus of English* and the *Oxford English Corpus*. The original BERT training took about 4 days on 4 cloud TPUs. A TPU is a *tensor processing unit*, a special purpose processor built by Google for processing the very large arrays used in training deep neural networks. Google search engines are currently supported by the BERT technology.

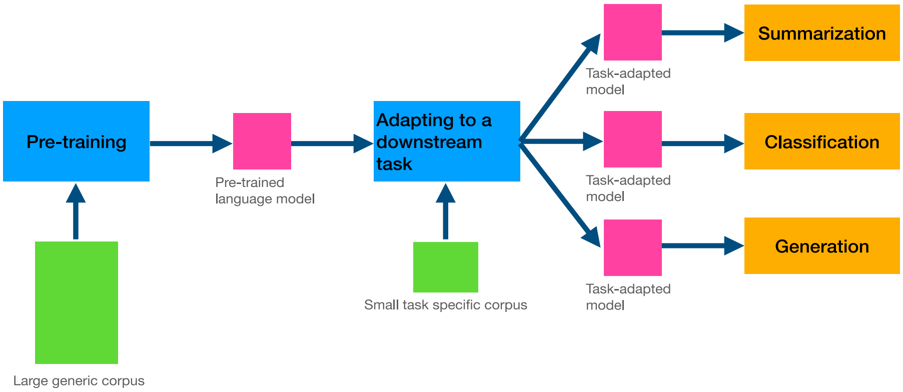


Fig. 19. The pretrained transformer transfers its learning to related domains. The new domain is conditioned by using a smaller task-specific dataset. The figure is adapted from Luger [5].

OpenAI has taken the BERT approach to language models one important step further. After creating a language model like BERT, GPT, the *Generative Pre-Trained Transformer* [57] adds a second task-specific training set. The primary training is task-agnostic and is just a general-purpose language model like BERT. With the addition of task-specific training, GPT can focus on targeted applications. For example, if the task-specific training is the writing of a specific author, the user can request what that author thinks about a particular topic. GPT's response, in the words and style of that author, is then often cogent and believable.

Since 2018 the transformer architecture has become the predominant methodology used in building LLMs. *Transfer learning* in AI is a technique that leverages the knowledge gained from one task to improve performance on another task. The "transfer" practice for transformers is, as we have just noted, to first pretrain these large-scale models on enormous corpora optimizing self-supervised learning. After pretraining, the models are then *fine-tuned* by users with data appropriate for a particular application task, as we see in Fig. 19. When using this pretraining approach, the original transformers are referred to as *foundation models* [58].

Fine-tuning consists of four steps:

(1) Determine the model to be tuned and its parameters, e.g., the learning rate.
(2) Aggregate new training data, where format and other parameters depend on the model.
(3) Compute losses, the error measure and gradients, to change the model to minimize error.
(4) Update the model through backpropagation.

Prompt engineering is the practice of querying the trained LLM with specific pieces of information to elicit the most appropriate responses from the model. There are several approaches to prompt engineering. *Zero-shot* queries request information that is not part of the model's training; the model will, however, generate a result. This technique makes LLMs useful for many different tasks. *Few-shot* prompting is a strategy where the model is given several task-specific examples before presenting the actual query. Few-shot queries enable the model to generalize over the queries. To summarize:

Chain-of-thought prompting is a style of few-shot prompting, where prompts contain a series of intermediate reasoning steps. Chain-of-thought prompting encourages the model to reason the way that the prompts are proposed, i.e., in a series of steps. Surprisingly, the answers from chain-of-thought prompting are often more accurate and interpretable than the answers from other prompts. Chain-of-thought prompting also discourages the model from generating quick easy answers.

There are now several suggestions for organizing the processes just described that move from the foundation model through fine tuning to prompt engineering. One process is called *LLM alignment*. Another is *RLHF* or *Reinforcement Learning from Human Feedback* [59]. There are still major questions about the utility of these approaches [60]; see Shen *et al.* [61] for a survey of alternative approaches. Table 3 presents a list of the transformer models currently available for exploration.

We have only touched the surface of how deep learning and LLMs support human language analysis. There are now many successes in language tasks, including finding documents that are similar such as patents [62], producing document

Table 3. Currently, June 2024, available software for generative AI. Note that information on several models is company confidential. A "token" is the piece of information used to train the model. Having more tokens or parameters does not guarantee better results, as the quality of code and processing is also critical. Interested readers should search these software tools for more current information as AI companies are known to change names, merge or simply dissolve.

Model	Capabilities	Parameters	Training data	URL
BERT	Question answering, finds semantic similarity	345 million	3.3 billion words	URL 1
PaLM 2	Generates text, essays and reports, answers questions, uses desired style and tone. Tuned to follow instructions	340 billion	3.6 trillion tokens	URL 2
ChatGPT, powered by GPT	Generates text, essays and reports, answers questions, uses desired style and tone. Tuned to follow instructions.	175 billion	300 billion tokens	URL 3
Gemini created by Google DeepMind	Generates multimodal output from multimodal input. Generates documents with both text and images.	Information not public.	Information not public.	URL 4
Llama 2	Generates text, essays and reports, answers questions using desired style and tone.	70 billion	2 trillion tokens	URL 5

Notes:
URL 1: BERT https://github.com/google-research/bert
URL 2: https://blog.google/technology/ai/google-palm-2-ai-large-language-model/
URL 3: ChatGPT https://chat.openai.com
URL 4: https://blog.google/technology/ai/google-gemini-ai/
URL 5: https://ai.meta.com/llama/

translations [63] and answering questions about speech and text [64]. Further analysis of the LLM technology may be found in [5]. For further details on LLM technology, the interested reader should consult the papers published by Google who created BERT [56] and OpenAI who created the GPT software [57].

6. Some Limitations of Current AI Practice

Symbol-based AI, Sec. 2, is created through the abstraction process. Specific symbols are created to describe the states of a problem situation. These symbols might be the specific words of a sentence, configurations of a complex mechanism, or situations in a game. Abstractions, of necessity, leave important information outside of the problem-solving process. A word only has meaning in a particular practical context. Complex systems can change in unpredictable ways. A game state, by itself, can be misunderstood, for example, if an opponent is in the process of sacrificing a game piece or position to achieve a greater goal.

There are other issues with symbol-based AI. Has the program designer identified all necessary states for useful solutions? Can the search process be accomplished in a practical time? What if part of the problem situation changes during the solution search? The symbol-based approach to AI has been very successful where it is used properly; the point of these criticisms is to be always aware of its intrinsic limitations.

Genetic and emergent AI takes a different approach to solving problems. The insight at the foundation of this approach is Darwin's notion of evolving systems and the survival of the fittest. With computation, this evolution can happen in multiple ways. We demonstrated genetic operators and example applications in Sec. 3.

In artificial life and emergent computing, the representation problem is critical. How are the input values created? What are the constraints for evolving the next generation of possible solutions? What criteria are used for measuring the "fitness" of the current generation's progeny? A deeper critique can ask how artificial life algorithms can create new species of life or how new life forms might originate. All current artificial life and genetic algorithm practice can only produce results contained within the computational closure of their originally defined domains.

Section 4 described why the computational cost of full Bayesian reasoning is not computationally usable in many practical situations such as most medical applications [19, Sec. 5.3]. The insights of Judea Pearl offered important contributions to making Bayesian-based AI models useful in practical situations. The Bayesian belief network [31, 32] allowed probabilistic representations to be viewed as causal relationships that, when factored, become computationally tractable. As a result, Bayesian belief network solutions are computable with reasonable time and memory usage.

There remains the representation problem for probabilistic reasoning, i.e., how best to abstract an application into sets of probabilistic symbols that combine to create a model for reasoning. Do the data for training a probabilistic classifier come from a normal distribution? When is a normal distribution assumption important? Section 4 offers examples of Bayesian belief net solutions, including the use of dynamic Bayesian networks.

Connectionist, or neural network problem solving, addresses many of the limiting factors of symbol-based AI. Many limitations of symbol abstraction can disappear by focusing on relationships between input symbols. There remain other issues including the choice of “appropriate” input symbols for describing a problem. When trying to recognize the meaning of spoken language expressions is it better to analyze individual words or syllables? Are picture points, pixels, optimal for recognizing different human faces? What are the best “tokens” for a network representation for a chess playing program?

Further questions relate to network architectures. What is the best number of input nodes? How many hidden layers are required? What is the connectivity scheme for the nodes on the hidden layers? These engineering questions are sometimes answered with massive computing power where the full connectivity of multiple layers, each with a very large number of nodes, is possible. For example, OpenAI’s GPT-3 has more than 175 billion parameters.

There are further problems inherent with connectionist problem solving, and we briefly name three. First the problem of *overlearning.* How much training does a network need to recognize categories of data? Too much training and only the training data itself will be recognized. Second, even with billions of parameters, many interesting problems are underspecified. Retraining with different initialized weights can produce different network solutions.

Although BERT and GPT produce impressive language models successful at multiple tasks, they only reflect the patterns found in text-based human language. There is no semantics in the human sense, only the presentation of the patters of words found in human communication. As Haven [65] claims in an MIT Technology Review, GPT-3 is completely mindless, capable of producing total nonsense, and even at times racist and sexist utterances. Finally, there is the matter of the transparency of the solution process and explanations of results, as we discuss further in Sec. 7.

Many consequential challenges remain for creating intelligent systems. We propose two general questions that need to be addressed to continue to build more intelligence into mechanical systems: the role of embodiment in intelligence and the importance of meaning or *grounding* in machine-based LLMs.

7. A Summary

One of the main assumptions of the hypothesis supporting computational “intelligence” is that the implementation of a symbolic or network model is irrelevant: all that matters is appropriate representations and algorithms. This viewpoint has been challenged by several thinkers [66, 67]. These philosophers argue that intelligent action must be supported by a physical and social embodiment that allows the agent to be integrated into the natural world of practical purposes, both personal and social.

The interfaces and architectures of modern computing do not support this degree of "situatedness". They require that an artificial intelligence interact with its world through the extremely limited window of contemporary input/output devices. If this "situated" challenge is correct, although some forms of machine intelligence may be possible, more general intelligence will, at the very least, require a very different machine than that afforded by contemporary computers.

Further, intelligence must also be regarded as a social as well as an individual construct. In a *meme-based* theory of intelligence [68], society itself carries essential components of intelligence and accumulated knowledge. It is possible that an understanding of the social context of knowledge and human behavior is as essential a component for a theory of intelligence as is an understanding of the dynamics of the individual mind/brain. Making machines that are analogues of human brains in their operations may not be sufficient: we may need machines appropriately immersed in social contexts.

What is the nature of meaning and interpretation, or how does AI address the *grounding* problem? Most computational models in traditional AI operate within an already interpreted domain. With this approach, there is an implicit and *a priori* commitment by the system's designers to a set of "meanings" for the program. Once this commitment is made, there is very little flexibility for shifting contexts, goals, or representations as the problem-solving situation evolves.

There are, of course, and will continue to be *AI winters*, a term used by the AI community to indicate deep changes in financial and research support for various projects. AI winters are the result of a schism between society's expectations of AI and the reality of AI's practice. AI winters reflect differences between AI's promises and the actual delivered product, between what we claim to produce and what we actually develop. A further topic too often overlooked in AI practice is ethics issue. There is currently much more interest both from industry and government in addressing these issues, e.g., see Luger [69, Sec. VIII].

The most exciting aspect of work in artificial intelligence is that to be coherent and contribute to the endeavor we must address a wide range of tasks and goals. To understand problem-solving, learning and language, we must expand our philosophical understanding of what it means to know and to represent our world. We are asked to resolve Aristotle's tension between *theoria* and *praxis*, to fashion a union of understanding and practice, to live between science and art.

Artificial intelligence researchers and practitioners are toolmakers. We create representations, algorithms and languages. These artifacts enable the design and building of mechanisms that exhibit intelligent behavior. Through experimentation, we test our tools' functional adequacy for addressing problems, and in that process examine our own understanding of the world we inhabit. There is a tradition for this: Descartes, Leibniz, Bacon, Pascal, Hobbes, Boole, Babbage, Turing and the AI researchers of the past seventy years. The AI vision thrives through the insights of engineering, computing, psychology, linguistics and philosophy as our exploration continues to address the nature of understanding, knowledge, meaning and intelligence.

ORCID

George F. Luger https://orcid.org/0009-0001-8164-5964

References

[1] A. A. Turing, Computing machinery and intelligence, *Mind* **59** (1950) 433–460.

[2] H. Dreyfus, *What Computers Can't Do: The Limits of Artificial Intelligence* (Harper and Row, New York, 1972).

[3] H. Dreyfus, *What Computers Still Can't Do: A Critique of Artificial Reason* (MIT Press, Cambridge MA, 1992).

[4] J. Haugeland, *Artificial Intelligence: The Very Idea* (MIT Press, Cambridge/Bradford, MA, 1985).

[5] G. F. Luger, *Artificial Intelligence: Principles and Practice* (Springer Nature, New York, 2025).

[6] A. Newell and H. A. Simon, The logic theory machine, *IRE Trans. Inf. Theory* **2** (1956) 61–79.

[7] A. N. Whitehead and B. Russell, *Principia Mathematica*, 2nd edn. (Cambridge University Press, London, 1950).

[8] H. Gelernter and N. Rochester, Intelligent behavior in problem-solving machines, *IBM J. Res. Dev.* **2**(4) (1958) 336–345.

[9] A. L. Samuel, Some studies in machine learning using the game of checkers, *IBM J. Res. Dev.* **3** (1959) 211–229.

[10] A. Newell and H. A. Simon, Computer science as empirical inquiry: Symbols and search, *Commun. ACM* **19**(3) (1976) 113–126.

[11] A. Collins and M. R. Quillian, Retrieval time from semantic memory, *J. Verbal Learn. Verbal Behav.* **8** (1969) 240–247.

[12] T. Winograd, *Understanding Natural Language* (Academic Press, New York, 1972).

[13] B. C. Williams and P. P. Nayak, Immobile robots: AI in the new millennium, *AI Mag.* **17**(3) (1996) 17–35.

[14] B. C. Williams and P. P. Nayak, A reactive planner for a model-based executive, in *Proc. Int. Joint Conf. Artificial Intelligence* (MIT Press, Cambridge, MA, 1997), pp. 1178–1185.

[15] S. Thrun, R. Brooks and H. Durrant-Whyte (eds.), *Robotics Research: Results of the 12th International Symposium*, Advanced Series in Mathematical Physics, Vol. 28 (Springer, Heidelberg, 2007).

[16] S. J. Russell, *Human Compatible* (Viking Press, New York, 2019).

[17] J. R. Quinlan, Induction of decision trees, *Mach. Learn.* **1**(1) (1986) 81–106.

[18] C. Shannon, A mathematical theory of communications, *Bell Syst. Tech. J.* **27** (1948) 623–656.

[19] G. F. Luger, *Artificial Intelligence: Structures and Strategies for Complex Problem Solving* (Addison Wesley-Pearson, New York, 2009).

[20] J. H. Holland, *Adaptation in Natural and Artificial Systems* (University of Michigan Press, Michigan, 1975).

[21] A. W. Burks, *Essays on Cellular Automata* (University of Illinois Press, Illinois, 1971).

[22] C. G. Langton, *Artificial Life: An Overview* (MIT Press, Cambridge, MA, 1995).

[23] R. Wellhausen and K. Oye, Intellectual property and the commons in synthetic biology: Strategies to facilitate an emerging technology, in *Atlanta Conf. on Science, Technology and Innovation Policy* (IEEE, New York, 2007), pp. 1–12.

[24] O. Purcell and T. K. Lu, Synthetic analog and digital circuits for cellular computation and memory, *Curr. Opin. Biotechnol.* **29** (2014) 146–155.

[25] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection* (MIT Press, Cambridge, MA, 1992).

[26] J. R. Koza, *Genetic Programming II: Automatic Discovery of Reusable Programs* (MIT Press, Cambridge, MA, 1994).

[27] J. Pearl, *Heuristics: Intelligent Strategies for Computer Problem Solving* (Addison-Wesley, Reading, MA, 1984).

[28] M. Mitchell, *An Introduction to Genetic Algorithms* (MIT Press, Cambridge, MA, 1996).

[29] H. C. Urey, *The Planets: Their Origin and Development* (Yale University Press, New Haven CT, 1952).

[30] T. Bayes, Essay towards solving a problem in the doctrine of chances, in *Philosophical Transactions of the Royal Society of London* (The Royal Society, London, 1763), pp. 370–418.

[31] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference* (Morgan Kaufmann, Los Altos, CA, 1988).

[32] J. Pearl, *Causality* (Cambridge University Press, New York, 2000).

[33] K. H. Davis, R. Biddulph and S. Balashek, Automatic recognition of spoken digits, *J. Acoust. Soc. Am.* **24**(6) (1952) 637–642.

[34] W. W. Bledsoe and I. Browning, Pattern recognition and reading by machine, in *Proc. Eastern Joint Computer Conf.* (IEEE Computer Society, New York, 1959), pp. 225–232.

[35] F. Mosteller and D. L. Wallace, Inference in an authorship problem, *J. Am. Stat. Assoc.* **58**(302) (1963) 275–309.

[36] D. Jurasky and J. H. Martin, *Speech and Language Processing*, 3rd edn. (Prentice Hall-Pearson, Upper Saddle River, NJ, 2020).

[37] G. F. Luger and C. Chakrabarti, Chapter 23: Expert systems, in *Handbook of Probability: Theory and Applications*, ed. T. Rudas (Sage Publications, Las Angeles CA, 2008), pp. 383–401.

[38] T. J. Jones, M. C. Darling, K. M. Groth, M. R. Denman and G. F. Luger, A dynamic Bayesian network for diagnosing nuclear power plant accidents, in *Proc. FLAIRS Conf.-16* (AAAI Press, 2016), pp. 179–184.

[39] M. C. Darling, G. F. Luger, T. B. Jones, M. R. Denman and K. M. Groth, Intelligent monitoring for nuclear power plant accident management, *Int. J. AI Tools,* World Scientific **27**(2) (2018) 1–25.

[40] K. P. Murphy, Dynamic Bayesian networks: Representation, inference and learning, PhD dissertation, Computer Science Department, University of California, Berkeley (2002).

[41] D. Pless and G. F. Luger, Towards general analysis of recursive probability models, in *Proc. Uncertainty in Artificial Conf. — 2001* (Morgan Kaufmann, San Francisco, 2001), pp. 429–436.

[42] D. Pless and G. F. Luger, EM learning of product distributions in a first-order stochastic logic language, in *Artificial Intelligence Soft Computing: Proc. IASTED Int. Conf.* (IASTED/ACTA Press, Anaheim, 2003), p. 6.

[43] W. S. McCulloch and W. Pitts, A logical calculus of the ideas immanent in nervous activity, *Bull. Math. Biophys.* **5** (1943) 115–133.

[44] D. O. Hebb, *The Organization of Behavior* (Wiley, New York, 1949).

[45] F. Rosenblatt, The perceptron: A probabilistic model for information storage and organization in the brain, *Psychol. Rev.* **65** (1958) 386–408.

[46] M. Minsky and S. Papert, *Perceptrons: An Introduction to Computational Geometry* (MIT Press, Cambridge, MA, 1969).

[47] G. E. Hinton and T. J. Sejnowski, Analyzing cooperative computation, in *Proc. 5th Annual Congr. Cognitive Science Society* (Rochester, New York, 1983), p. 5.

[48] D. E. Rumelhart, J. L. McClelland and The PDP Research Group, *Parallel Distributed Processing* (MIT Press, Cambridge, MA, 1986).

[49] G. E. Hinton, S. Osindero and Y. W. Teh, A fast learning algorithm for deep belief nets, *Neural Comput.* **18**(7) (2006) 1527–1554.

[50] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel and D. Hassabis, A general reinforcement learning algorithm that masters chess, shogi, and go through self-play, *Science* **362** (2017) 1140–1144.

[51] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel and D. Hassabis, Mastering the game of go without human knowledge, *Nature* **550** (2017) 354–359.

[52] A. Faust, O. Ramirez, M. Fiser, K. Oslund, A. Francis, J. Davidson and L. Tapia, PRM-RL: Long-range robotic navigation by combining reinforcement learning with sampling-based planning, in *Proc. ICRA-18* (IEEE Press, 2018), pp. 5113–5120.

[53] L. E. Kavraki, P. Svestka, J.-C. Latombe and M. H. Overmars, Probabilistic roadmaps for path planning in high-dimensional configuration spaces, *IEEE Trans. Robot. Autom.* **12**(4) (1996) 566–580.

[54] K. Arulkumaran, C. Antoine and J. Togelius, AlphaStar: An evolutionary computation perspective, in *Proc. Genetic and Evolutionary Computation Conf. Companion* (ACM Digital Library, 2020), arXiv:1902.01724.

[55] M. Jaderberg, W. M. Czarnecki, I. Dunning, L. Marris, G. Lever, A. G. Castañeda, C. Beattie, N. C. Rabinowitz, A. S. Morcos, A. Ruderman, N. Sonnerat, T. Green, L. Deason, J. Z. Leibo, D. Silver, D. Hassabis, K. Kavukcuoglu and T. Graepel, Human-level performance in 3D multiplayer games with population-based reinforcement learning, *Science* **364** (2019) 859–865.

[56] J. Devlin, M. Chen, K. Lee and K. Toutanova, BERT: Pre-training of deep bidirectional transformers for language understanding, arXiv:1810.04805 (2019).

[57] T. B. Brown *et al.*, Language models are few-shot learners, arXiv:2005.14165 (2020).

[58] R. Bommasani *et al.*, On the opportunities and risks of foundation models, arXiv:2108.07258 (2021).

[59] R. Rafailov, S. Archit, E. Mitchell, E. Stephano, C. D. Manning and C. Finn, Direct preference optimization: Your language model is secretly a reward model, in *37th Conf. Neural Information Processing (NeurIPS-23)* (Curran Associates, 2023), pp. 1–27, arXiv:2305.18290.

[60] S. Casper, X. Davies *et al.*, Open problems and fundamental limitations on reinforcement learning from human feedback, arXiv:2307.1517 (2023).

[61] T. Shen, R. Jin, Y. Huang, C. Liu, W. Dong, Z. Guo, X. Wu, Y. Liu and D. Xiong, Large language model alignment: A survey, arXiv:2309.15025v1 (2023).

[62] L. Helmers, F. Horn, F. Biegler, T. Oppermann and K-R. Muller, Automating the search for a patentfoundation modelss for language understandingme, *PLoS One* **14**(3) (2019) e0212103.

[63] Y. Wu, M. Schuster, Z. Chen, Q. V. Le and M. Norouzi, Google,Norouzi, earch for a patentfoundation modelss for language understandingment learningd plan, arXiv:1609.08144 (2016).

[64] M. M. A. Zaman and S. Z. Mishu, Convolutional recurrent neural networks for question answering, in *3rd Int. Conf. Electrical Information and Communication Technology* (IEEE Press, New York, 2017).

[65] W. D. Heaven, OpenAI's new language generator GPT-3 is shockingly good — And completely mindless, *MIT Technol. Rev.* (MIT Technology Review, Cambridge MA, 2020).

[66] P. Agre and D. Chapman, Pengi: An implementation of a theory of activity, in *Proc. 6th National Conf. Artificial Intelligence* (Morgan Kaufmann, CA, 1987), pp. 268–272.

[67] F. J. Varela, E. Thompson and E. Rosch, *The Embodied Mind: Cognitive Science and Human Experience* (MIT Press, Cambridge, MA, 1993).

[68] G. M. Edelman, *Bright Air, Brilliant Fire: On the Matter of the Mind* (Basic Books, New York, 1992).

[69] G. F. Luger, *Knowing Our World: An Artificial Intelligence Perspective* (Springer Nature, New York, 2021).