

The History of Artificial Intelligence

Definitions of Artificial Intelligence

- “Systems that think like humans” – concerned with thought processes/reasoning and makes comparison to humans. Strays into cognitive science. Proof through psychological experiments.
- “Systems that think rationally” – concerned with thought processes/reasoning and makes comparison to an ideal concept. Uses formal logic – some problems cannot be transcribed into this language.
- “Systems that act like humans” – concerned with behaviour and makes comparison to humans. Proof through the Turing Test.
- “Systems that act rationally” – concerned with behaviour and makes comparison to an ideal concept. Sometimes purely logical reasoning does not yield the best solution to a problem.

Milestone Developments

Eliza – Wiezenbaum – 1966

Conversationalist. Based on a set of rules which produce certain types of question from certain types of input. If no matches are found, general questions can be asked, e.g. ‘Can you elaborate on that?’ Different scripts (rule sets) can be given. E.g., a psychiatrist script.

Physical Symbol System Hypothesis – Newell and Simon – 1975

A physical symbol system "consists of a set of entities, called symbols, which are physical patterns that can occur as components of another type of entity called an expression (or symbol structure). Thus, a symbol structure is composed of a number of instances (or tokens) of symbols related in some physical way (such as one token being next to another). At any instant of time the system will contain a collection of these symbol structures. Besides these structures, the system also contains a collection of processes that operate on expressions to produce other expressions: processes of creation, modification, reproduction and destruction. A physical symbol system is a machine that produces through time an evolving collection of symbol structures. Such a system exists in a world of objects wider than just these symbolic expressions themselves."

Mycin – Shortliffe – 1976

Diagnoses infectious diseases. Backwards reasoning (works back from patients illness to possible cause). Assigns probabilities to its diagnoses. Typical Mycin rule – “if A and B and C then suggestive evidence (confidence P) that D”. Tested against medical experts in 1982. Consistently outperformed the humans. Solves real-life problems, used expert empty shell (different knowledge bases can be used)

XCON/R1

Developed by the Digital Equipment Corporation to insure all components and software needed for the ordered system were included in the delivered product.

Some important concepts

- Forward chaining – move from initial state through various intermediate states to the goal. Used if a) many potential goals, b) goal is not known until it is reached, c) low branching factor.
- Backwards chaining – assume the goal state and work backwards towards the initial state. Used if a) the goal is given, b) there is a large branching factor or c) the data is not given.
- Strong and Weak problem solving – weak problem solving has no domain knowledge. Strong problem solving has such knowledge.
- Means-End-Analysis (as demonstrated by the General Problem Solver – Newell & Simon – 1963) uses forwards and backwards chaining together to solve different parts, or ‘sub-problems’ of the same problem. ?...?

The Turing Test – 1950 (Alan Turing)

Interrogator asks questions to A and B (one of them is a human, one a machine). Machine’s aim is to fool the interrogator into thinking it is human (could, e.g., wait for a while then give a wrong answer). The Turing Test has yet to be passed and some believe it never will. It is argued that passing the Turing Test does not show a machine is intelligent (see Chinese Room). To pass the test, the machine must be able

Revision Notes

to process natural language, represent knowledge, reason, etc. The Total Turing Test involves testing visual perception as well.

The Chinese Room – 1980 (John Searle)

Proposes that *behaving* intelligently does not prove the existence of intelligence. As demonstrated by the following:

The system includes: A human who speaks only English (the CPU), a Rule Book (in English) (the program) and two stacks of paper (one stack is blank, one contains indecipherable symbols) (memory). The human takes in the symbols, looks them up in the rule book, and performs the actions specified in the rule book (e.g. writing a different set of symbols, etc.). Ultimately, the human will pass out a piece of paper containing some symbols.

Say the input and output symbols were valid Chinese – Searle says that although this system *appears* to understand Chinese (and thus be intelligent), it is not really understanding, because neither the human, the book or the paper actually understands Chinese.

Essentially, Searle is saying that 1. Certain objects are incapable of conscious understanding, 2. The human, book and paper are objects of this kind, 3. If each object is incapable of understanding, then the system is also incapable, 4. The Chinese Room, therefore, involves no conscious understanding.

However, some say that 3. is invalid and that if you believe the human brain is simply composed of nerves which are only molecules, then either humans are not capable of understanding, or molecules are.

Oppositions to this include:

The Systems Reply

The system *as a whole* understands Chinese, even if the CPU and the program do not. Searle argues that the system could be internalised into a brain and yet the person would still claim not to understand.

The Robot Reply

Could internalise everything into a robot so it appears human. Searle does not see that this adds anything of value.

The Brain Simulator Reply

Could write a program to emulate the brain. Searle says we could also emulate the brain using a series of water pipes and valves. Are these, therefore, intelligent?

The Combination Reply

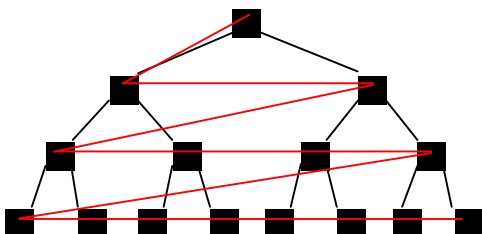
Combine all the previous three replies. Searle says this does not add anything. We know how the android operates, thus it is not carrying out strong AI.

~

Blind Searches

- Have no knowledge of the search domain. It can only distinguish between a non-goal state and a goal state.
- Use blind searches when, either you have no (or there is no) domain knowledge, or you won't know the answer until you see it.

Breadth First Search

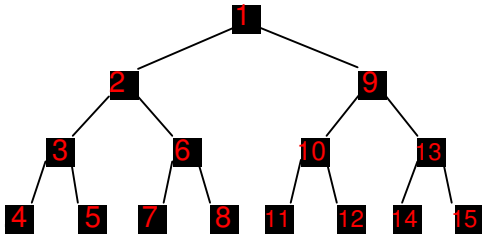


- Searches through the search tree as shown in the diagram. In terms of a queue, newly expanded nodes are placed at the END of the queue.
- Complete and Optimal
- Total maximum search space = $1 + b + b^2 + b^3 + \dots + b^d$ where b is the average branching factor and d is the depth of the solution. This type of growth is written using the O notation as $O(b^d)$

Uniform Cost Search

The cost of the path to a particular node may not always be related to its depth in the search space. In this case, Uniform Cost Search is better. In BFS, the queue is naturally ordered by depth; UCS orders the queue by path cost (often represented by $g(n)$).

Depth First Search



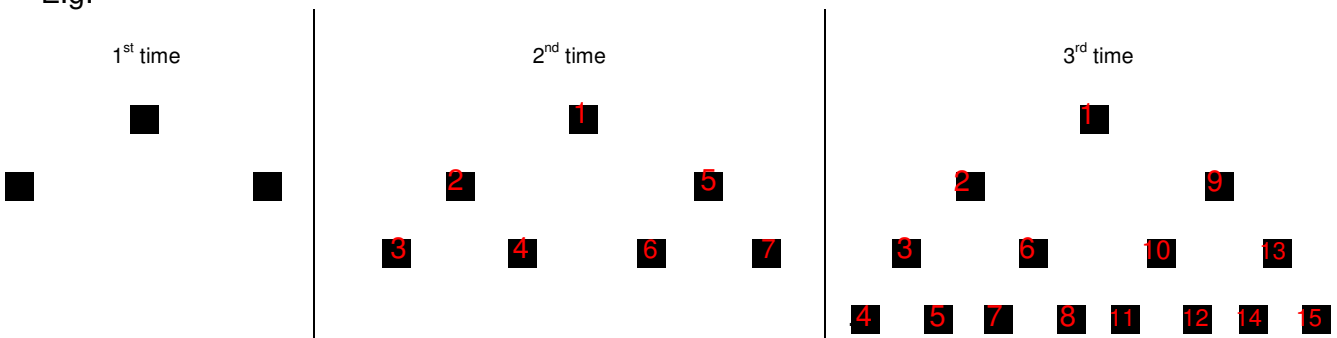
- Searches through the search tree as shown in the diagram. In terms of a queue, newly expanded nodes are placed at the START of the queue.
- Need only store the path from the root to the leaf node as well as unexpanded nodes, so memory requirements are hugely less – only $b \cdot m$ where m is the maximum depth.
- Time complexity is $O(b^m)$
- If there is an infinite branch, DFS will never end. It also may find a solution that is not the best one. Therefore, it is neither complete nor optimal.

Depth Limited Search

- Simply imposes a limit on the depth that DFS goes to, so the infinite branch is not a problem.
- Complete, as long as the limit \geq depth of the solution.
- Time: $O(b^L)$; Space: $O(bl)$

Iterative Deepening Search

- Combines the completeness and optimality of BFS with the small memory requirements of DLS.
- Runs Depth Limited Search at $L=1$, then $L=2$, then $L=3$, etc. until a solution is found.
- E.g:



- May seem wasteful, as it expands the same nodes multiple times, but the overhead is relatively small compared with the growth of an exponential search tree.
- Time complexity is $O(b^d)$ and space complexity is $O(bm)$

Repeated States

Many problems will involve generating the same state multiple times. The search time can be dramatically decreased by taking this into consideration. There are three ways to do this:

- Do not generate a node that is the same as the parent node
- Do not create paths with cycles in them
- Do not generate any state that is the same as any generated previously. This requires that every state be kept in memory (meaning a potential space complexity of $O(b^d)$).

Summary of Blind Search Methods

(b = branching factor; d = depth of solution; m = maximum depth of the tree; L = depth limit)

Evaluation	BFS	UCS	DFS	DLS	IDS
Time	$O(b^d)$	$O(b^d)$	b^m	b^L	b^d
Space	$O(b^d)$	$O(b^d)$	$O(bm)$	$O(bL)$	$O(bd)$
Optimal?	Yes	No	No	No	Yes
Complete?	Yes	Yes	No	If $L \geq d$	Yes

~

Heuristic Searches

- Blind searches are normally very inefficient.
- Heuristic searches have some knowledge of the domain and are therefore able to expand the node which is nearest to the goal state.
- The heuristic function is $h(n)$ = estimated cost of the cheapest path from the current state to the goal state.
- A heuristic is not necessarily 100% efficient, if it was, it would not be a search, but just a march from initial state to goal.

Greedy Search

- Sorts the queue by $h(n)$
- Considers the next node only, no further. Does not consider path so far. Therefore not optimal.
- Time and space complexity of $O(b^m)$ as all nodes are kept in memory.

A* algorithm

- Combines Best First Search and Uniform Cost Search.
- Heuristic function considers path so far *and* estimated path to goal ($h(n) + g(n)$)
- A heuristic must never *overestimate* the cost to the goal. Such a heuristic is termed *admissible*.
- Complete and optimal
- Number of nodes expanded is still exponential to the depth of the search space.
- The more efficient a heuristic, the lower it's effective branching factor.

Neural Networks

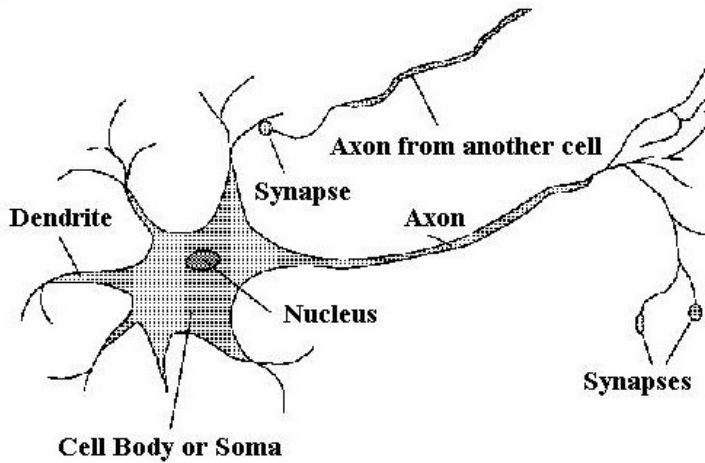
Brief History

- First neural network design: McCulloch & Pitts, 1943
- First learning rule: Hebb 1949
- 1950-60: work on the *perceptron*
- 1969: Minsky & Papert show perceptrons could only learn linearly separable functions.
- mid 1980s: Parker and LeCun (independently) discover a learning algorithm (backpropogation) that can solve non-linearly separable problems.

The Human Brain

Made up of 100s of billions of neurons.

Revision Notes



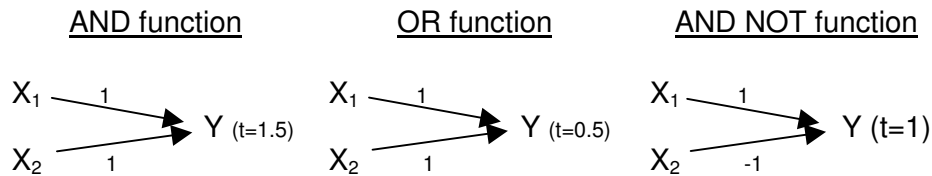
- Each neuron consists of a cell body, or soma, that contains a nucleus
- Dendrites receive connections from other neurons. Inputs
- Axon which splits into strands to make more connections. Output
- Point where neurons join is called a synapse
- A neuron may connect to 100,000 others.

Signals move from neuron to neuron via electrochemical reactions. Synapses releases a chemical to raise or lower the electrical potential in the soma. If the sum of these currents is greater than the *threshold*, the axon will fire.

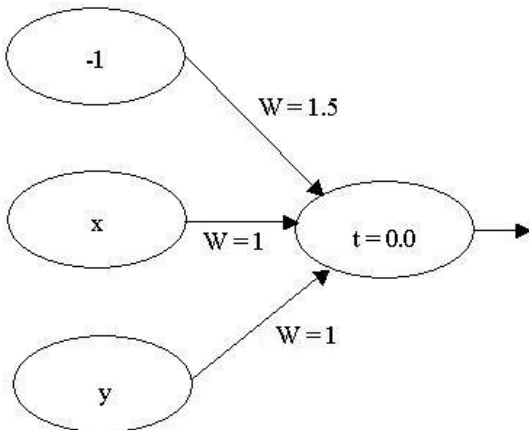
The First Artificial Neuron – McCulloch & Pitts 1943

- Binary activation (either fires or does not fire)
- Neurons joined by weighted paths
- Positive weight – excitatory; negative weight – inhibitory.
- Each neuron has a fixed threshold set that any negative input will prevent the neuron from firing.
- It takes one time step for a signal to pass over one connection.

E.g.,



From a computability point of view, it would be easiest if thresholds were standardized to one value. This can be done using a specialized input with a changeable weight, e.g.,



- with the weight of this extra input set to 1.5, this system functions as an AND function.
- change the weight to 0.5, and it's an OR

Note: the extra weight can be determined by what the threshold would have been, if the extra input had not been present (assuming other inputs to be 1 or 0). E.g., for an AND function, the threshold would be 1.5 (as above)

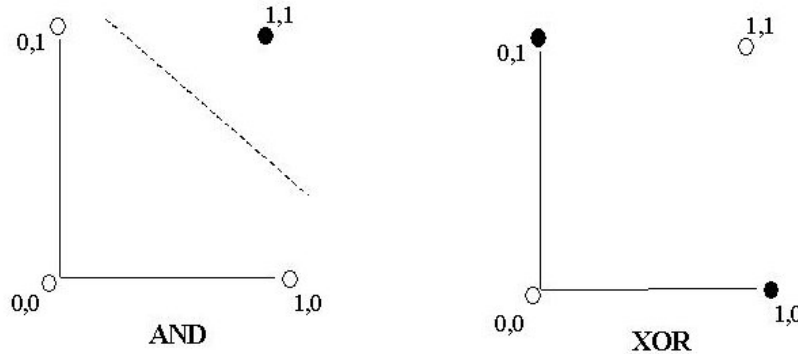
This method is particularly useful is that when 'teaching' the neuron a new function, only this extra weight need be altered, instead of thresholds and weights.

The network shown, left, is a one-layer, feed-forward network.

A single-layer, feed-forward network is also called a *perceptron*. Perceptrons cannot learn any function. They must be *linearly separable*:

Revision Notes

Boolean functions can be represented graphically:



The 'on' outputs can be separated by the 'off' outputs by a straight line in the AND function, but not in the XOR function. Hence, a perceptron cannot learn XOR. This problem was first identified by Minsky & Papert in 1969. Three or more input problems can be visualised in this way using a 3D diagram and separating with a plain, not a line.

Learning

Definitions:

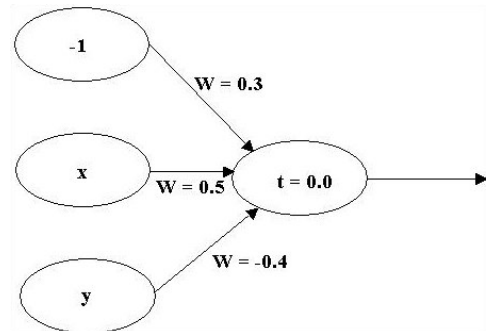
- Epoch: The entire training set (in the AND function, [0,0], [0,1], [1,0] and [1,1])
- Training value, T: the expected output
- Error: the amount by which the output is wrong. I.e., if a 0 is expected and a 1 is returned, the error is $0-1 = -1$.
- I_j – inputs presented to the neuron
- W_j – weights from input neuron to the output neuron.
- LR – the learning rate – how quickly the network converges. Typically 0.1.

The Process:

Consider this perceptron below. We will 'teach' it the AND function.

Epoch 1

- First training pair is [0,0]: Sum is -0.3 . No fire - output of 0. No error.
- Second training pair is [0,1]: Sum is -0.2 . No fire - output of 0. No error.
- Third training pair is [1,0]: Sum is 0.2 . Fire - output of 1. Error of $0-1 = -1$. Therefore, adjust the weights:
Adjusted weight = current weight + (LR * Input * Error)
 $W_0 = 0.3 + (0.1 * -1 * -1) = 0.4$
 $W_1 = 0.5 + (0.1 * 1 * -1) = 0.4$
 $W_2 = -0.4 + (0.1 * 0 * -1) = -0.4$
- Fourth training pair [1,1] (using adjusted weights): Sum is -0.4 . No fire - Output of 0. Error of $1-0 = 1$.
Adjusted weight = current weight + (LR * Input * Error)
 $W_0 = 0.4 + (0.1 * -1 * 1) = 0.3$
 $W_1 = 0.4 + (0.1 * 1 * 1) = 0.5$
 $W_2 = -0.4 + (0.1 * 1 * 1) = -0.3$



This Epoch resulted in 2 errors. Continue the process until an error-free epoch occurs. This problem takes 7 epochs to get it right.

The weights are represented by the line (of linear separation) on the graphical representation.