Module 1: AI and Risk – Introduction and Overview

Learning Objectives

The application of artificial intelligence (AI) introduces a novel set of risks to all organizations that use it. This module offers a historical perspective on AI, an overview of both machine learning methodologies and generative AI, and an introduction to the risks associated with using AI/ML.

After completing this module, you should be able to:

- Explain some central principles of Classical AI, including search methods and recursion.
- Describe the limitations of classical AI.
- Articulate the potential and limitations of deep learning.
- Identify key breakthroughs leading to advances in AI and ML.
- Compare and contrast reinforcement, supervised, unsupervised, and semi-supervised learning, and identify practical applications for each technique.
- Discuss risks associated with inscrutability in AI and ML.
- Discuss risks associated with over-reliance on AI systems.
- Discuss ways in which AI exposes individuals, organizations, and society to risk.

1.0 Classical AI

Intelligence – or "reason" – has long been seen as the human quality that, more than any other, distinguishes us from other species and gives us our unique power over the world. But there has been speculation through the ages about the possibility of duplicating the power of human reason with artificial technology. This began to seem practically feasible after the industrial revolution, and in the early 19th century, Charles Babbage conceived of a mechanical, gear-driven "difference engine" that could replace error-prone human calculation in the efficient production of mathematical tables. His more-ambitious design for an "analytical engine" is widely seen as anticipating the digital computer, though bringing it to successful completion would be a huge feat of engineering, even with modern technology.

The crucial breakthrough occurred when the digital computer emerged as the brainchild of Alan Turing in 1936, although his "computing machine" (universally known as a "Turing machine") was a theoretical invention – designed to prove fundamental results about mathematics and what we now call theory of computation – rather than a practical device. Turing argued convincingly that such a machine would in principle be able to follow any given recipe for calculation that we are able to devise, and in that sense would be a universal programmable computer.[1] These ideas strongly informed his work at Bletchley Park during the Second World War, devising mechanical methods of decrypting the Nazi Enigma codes, which gave a massive impetus to the development of the electronic computer after the war (on both sides of the Atlantic). Turing himself had visionary thoughts about the possibility and potentially tremendous power of intelligent machines, which he famously presented in his paper, "Computing Machinery and Intelligence" of 1950. This centers on the idea of an "imitation game" – now known as a "Turing Test" – in which a computer proves itself to be intelligent if it can generate textual conversation of a quality indistinguishable from that of an intelligent human, ranging over any topic chosen by the human "interrogator." Though highly controversial philosophically, the Turing Test remains influential, and has become particularly salient recently, as "chatbots" based on large language models (such as ChatGPT) have provided – for the first time – examples of computer programs that are relatively plausible contenders.

[1] Turing's seminal paper, entitled "On Computable Numbers, with an Application to the *Entscheidungsproblem*", appeared in the *Proceedings of the London Mathematical Society* 42 (1936-37), pp. 230-65. For links to relevant online talks and papers, see https://www.philocomp.net/videos/turing.htm, and for a set of Oxford

University lectures on “Alan Turing on Computability and Intelligence”, see https://www.philocomp.net/videos/turing_lectures.htm.

1.1 Specific versus General AI

Media discussion of AI, and futuristic speculation, is often concerned with the question of whether an artificial system could surpass human *general* intelligence. This may in part be a legacy of the Turing Test, but it probably also reflects a natural concern about challenges to our primacy in reason and understanding – abilities that are so closely bound up with human identity. Such concerns are also commonly associated with worries about existential risk arising from an artificial general intelligence (AGI). It is far less threatening to consider that machines might outperform us in very specific domains, such as calculating mathematical tables or generating complex statistics. But it is the latter kind of machine – designed to be *intelligent at solving specific problems* – that will mostly concern us in what follows.

Some would argue that a system can only properly count as "intelligent" if it has the sort of general ability that we do. It has become commonplace, however, to talk of intelligent or "smart" systems that are relatively limited. Even Turing's legacy does not point very heavily toward a purely general interpretation of AI. In 1950, his aim was to devise a thought experiment that demonstrated the possibility of AI by imagining a computer system that could not reasonably be denied being intelligent. In that context, it made good sense to consider a conversational system of comparable versatility and skills to a human, as a sort of conceptual existence proof of that possibility.[2] But once we have been convinced that there is no objection *in principle* to artificial intelligence, it seems odd to deny that a system can be intelligent in addressing specific tasks, without having more general competence.

[2] For more on this understanding of the Turing test, see the 2021 paper linked from footnote 1.

1.2 Good Old-Fashioned AI (GOFAI): Logic and Games

The term "artificial intelligence" was coined in 1956 by John McCarthy of Dartmouth College, who along with others (Marvin Minsky of Harvard, Nathan Rochester of IBM, and Claude Shannon of Bell Laboratories) organized a seminal two-month summer conference on the topic.[3] Echoing Turing's notion of universality, this was based on "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."[4] Here the paradigm of intelligence was accordingly *explicit, precise processing*, as typified by symbolic logic or other formal rule-governed systems. And in arguing for the universality of his computing machines in 1936, Turing had emphasized their ability to mimic logical proofs, constructed step-by-step from a given set of "axioms," and systematically applying such rules.

Following in this tradition, in 1955 Allen Newell, Herbert Simon, and Cliff Shaw (all working at the Rand Corporation) started developing their *Logic Theorist* program, designed to generate proofs in *propositional logic*, the simplest and most fundamental form of modern symbolic logic. Newell and Simon presented this program at the Dartmouth conference (though not, apparently, to great acclaim at the time), and they later used it to confirm most of the theorems in Chapter 2 of the monumental *Principia Mathematica* by Alfred North Whitehead and Bertrand Russell.[5] This famous three-volume work had aimed to demonstrate the *logicist* thesis, that all of mathematics could be deduced from suitable axioms by pure logic, so it was an excellent choice for a high-profile application of machine intelligence, made all the more impressive when *Logic Theorist* was able to find a new and more elegant proof (of Theorem 2.85).

Another paradigm aspiration of AI was to create a program capable of playing "intellectual" games such as chess at a high level. This again went back to Alan Turing, although his attempt to implement such a program proved impractical on the hardware then available, and he never got beyond a "paper" implementation of chess-playing rules. Chess would eventually be conquered with the advent of much more powerful machines and sophisticated techniques, culminating with the defeat of world champion Garry Kasparov by *Deep Blue* in 1997. But by far the most prominent early success in this direction came with the program developed in the late 1950s by Arthur Samuel of IBM, which focused on the much simpler game of checkers (or draughts).

Quite apart from their intrinsic interest (especially to mathematically minded thinkers) and their promotional value in raising public awareness of the possibility of AI, such

intellectual games provided an attractive testbed for the development of AI for at least three related reasons. First, they are relatively *simple*, compared with the complexities of normal life, and hence could plausibly be tackled even when computing resources would have been severely stretched in even recording the details of a physical environment, let alone taking on the immensely complicated challenge of reasoning about it. Secondly, such games are straightforwardly *rule-governed*, with clear and explicit regulations regarding the setup of the game, how the pieces are permitted to move, the effects of these moves on the board position, and criteria for winning, losing, and drawing. Thirdly, they are *competitive*, and we are already very familiar with the idea that players can be rated according to their relative skill. Hence such games lend themselves very naturally to the *assessment* of that skill, so that it is straightforward to judge researchers' progress as they endeavor to produce a program capable of competing at ever higher levels.

---

[3] McCarthy and Minsky are widely considered to be among the "founding fathers" of AI. In 1959 they co-founded the AI Project at MIT, which later became the AI Lab (and ultimately CSAIL, the Computer Science and Artificial Intelligence Laboratory). McCarthy went on to devise the AI computer language LISP in 1960, and to found the Stanford Artificial Intelligence Lab (SAIL) in 1963.

[4] Quoted from the first paragraph of "A Proposal for the Dartmouth Summer Research Project on Artificial Intelligence", by McCarthy, Minsky, Rochester, and Shannon, and dated August 31, 1955 (a transcript of which can be found at https://www-formal.stanford.edu/jmc/history/dartmouth/dartmouth.html).

[5] Alfred North Whitehead and Bertrand Russell, *Principia Mathematica*, 3 vols (1910, 1912, 1913), Cambridge: Cambridge University Press.

1.3 Simple Reinforcement Learning

The three aforementioned characteristics also combine to make it possible to develop a program that can *learn* to play such a game, at least in principle. Indeed, Samuel's program did this, thus becoming the first substantial example of *machine learning* (a term he coined). The first technique is reinforcement learning, illustrated with the game of Nim*,* a game of strategy in which two players take turns removing (or "nimming") objects from the game board. There are several versions of Nim, but this one starts with 20 coins being placed on a table. Now taking turns, each player removes either 1, 2, or 3 coins, and the winner is the player who gets to remove the last coin(s) from the table. There is, in fact, a simple winning strategy for the second player in this game: Depending on whether player *A* has removed 1, 2, or 3 coins on his turn, player *B* should respectively remove 3, 2, or 1 on his turn. A total of 4 coins is removed by *A* and *B* together, and so after 5 rounds all 20 coins will have been removed, with *B* taking the last group.

Suppose, however, that we are new to the game, with no clue what strategy to adopt, and we want to write a machine-learning program that will learn to play it for us. One easy way of doing this is to set up a program that, in any given position, will choose its move *randomly* based on a given set of *weights*, and that learns by adjusting those weights according to how well each move turns out in practice. For simplicity, suppose that these weights can vary between 0 and 9, and we set them all up initially to be 5. Then we play against the system, and depending on which player wins or loses each game, the weights are changed to reflect that success or failure. **Figure 1.1**, for example, is how things might look after a few games, in the middle of a game in which there are currently 11 coins remaining and the computer is about to move:[6]

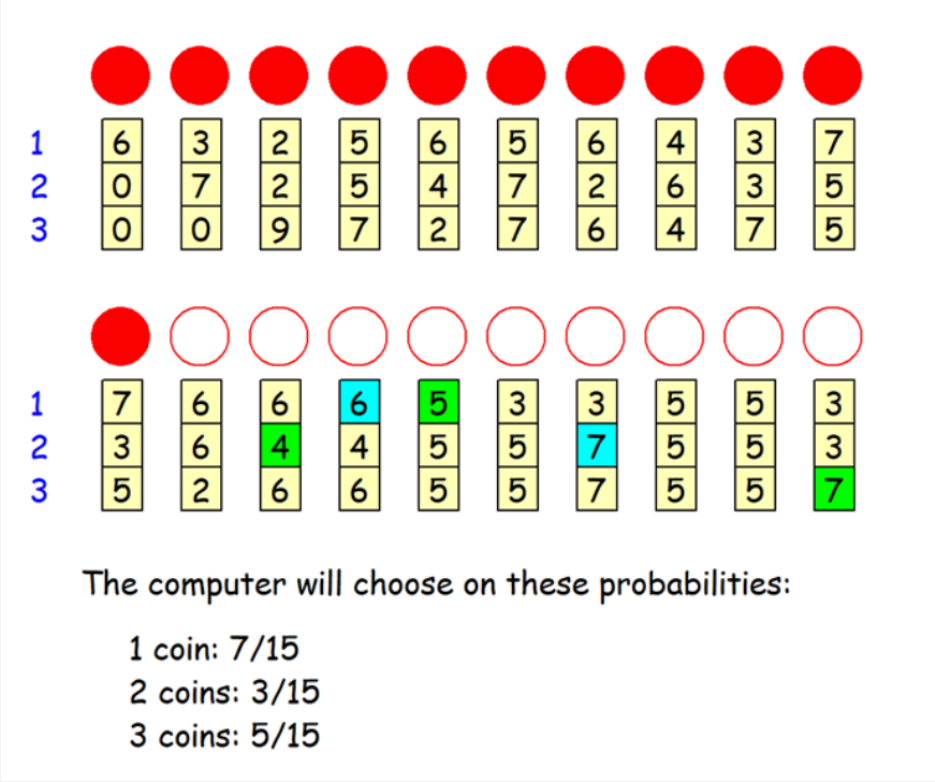

Figure 1.1

In this ongoing game, the human player started by removing 3 coins (indicated by the green square on the far right, in what we might call the 20/3 position). Then, with 17 coins remaining, the computer chose to remove 2 coins (indicated by the cyan square in the 17/2 position). The next three moves (human taking 1, computer taking 1, human taking 2) are shown by the other green and cyan squares. Here, as we see, the computer is choosing its move randomly based on the three *weights* displayed under the highest-numbered remaining coin (i.e. coin 11), so that it has probability 7/15 of choosing 1 coin, 3/15 of choosing 2, and 5/15 of choosing 3 (15 being the sum of the three weights). So it is most likely to remove 1 coin, reflecting that this has so far been the most successful move when 11 coins remain. This process of alternate moving continues until the game ends, after which reinforcement learning takes place as follows. Whichever player *wins*, all of his chosen moves that led to that victory will be *positively* reinforced by having their weight incremented by one (with a maximum of 9), whereas the moves not chosen in those positions will be *negatively* reinforced by having their weight decremented by one (with a minimum of 0). Conversely, for the player who *loses*, all of his chosen moves that led to that loss will be *negatively* reinforced by having their weight decremented by one (with a minimum of 0), whereas the moves not chosen in those positions will be *positively* reinforced by having their weight incremented by one (with a maximum of 9).[7]

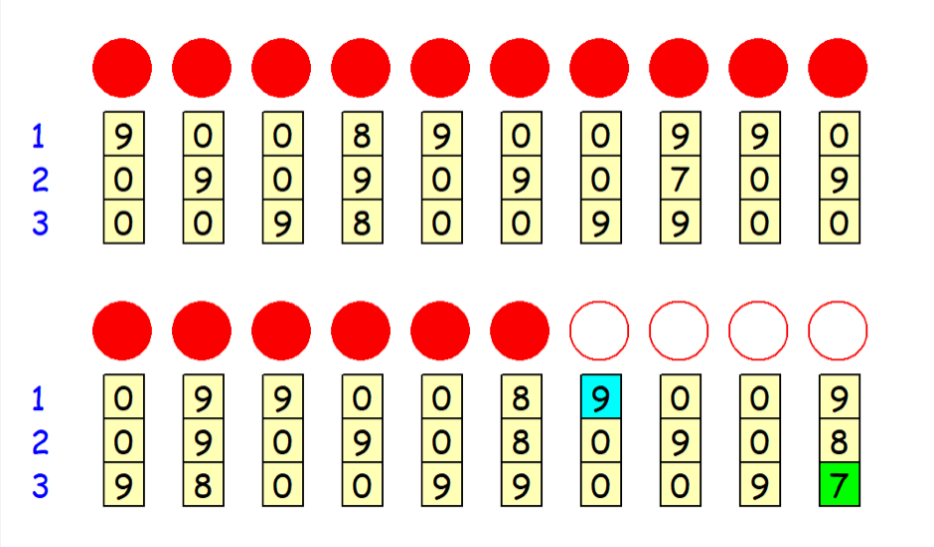

Figure 1.2

Though extremely simple, we find that this mechanism quickly discovers the winning strategy, especially if we replace the human player with an automated opponent following the same learning strategy.[8] After around 100 games, we find that the weights have been adjusted in the sort of way shown below. In this particular game (**Figure 1.2**), the human player started by removing 3 coins, and the computer responded by removing 1 coin, leaving 16 coins remaining for the human's next move. Note that the computer's last move was inevitable, given the 9-0-0 weights that it has learned when 17 coins remain, and note that a similar point applies when 18 or 19 coins remain – in all three cases, there is a 9/9 probability that the computer will reduce the remaining coins to 16. The same pattern can be seen also in the rest of the table: *whenever it can do so, the computer will reduce the coins remaining to a multiple of 4*. And thus eventually, as explained earlier, the number reduces to 0, with the computer player winning by removing the last coin(s). The program has thus learned for itself the optimal winning strategy by reinforcement learning. As the various possible moves were tried out, those that ended up leading to victory in any particular game were *positively reinforced*, in that the weights were adjusted to make them more probable in the future, whereas moves that led to loss were *negatively reinforced*, by being made less likely in the future.

This very crude kind of reinforcement learning – accumulating experience of the results that arise from the different available moves in very specific positions in a game – can only be applied to relatively simple games in which the number of possible positions is limited (although we see later in this module how reinforcement learning can be extended to more complex games such as checkers).

[6] Image created by author using his own *Turtle System* software, available free from www.turtle.ox.ac.uk. Interested readers can thus try out these programs, and examine or adapt them if desired.

[7] In the current case, for example, the weights at the initial 20-coin position are 3/3/7, with me having chosen the last of these, while the weights at the 17-coin position are 3/7/7, with the computer having chosen the second of these. So if the computer now ends up beating me in this game, the 20-coin weights will change to 4/4/6 – downweighting the choice I made, and upweighting the other two – while the 17-coin weights will change to 2/8/6 – upweighting the choice it made, and downweighting the other two.

[8] Learning is much slower if the automated "experimental" opponent plays randomly, because then the system will often end up winning games that it should have lost due to an early error, while the opponent will lose games that it should have won, resulting in misleading reinforcement. The best opponent for quick learning is one that is good at punishing errors, so that the system learns quickly to avoid those errors.

1.4 Lookahead

In our Nim-learning program, the computer has not at any point "looked ahead" to see what outcomes are likely to emerge depending on its possible choices. The program has ultimately developed a perfect winning strategy, but it has done so by a sort of evolutionary process, whereby successful moves have been made progressively more probable as future choices, and unsuccessful moves have been made progressively less probable. It now plays perfectly, but one might hesitate to call it genuinely "intelligent" when it is just blindly choosing moves according to specified probabilities, without any more-sophisticated "lookahead" or planning. Such intelligence as it displays seems to lie more in the learning process that devised its playing method than in the method itself.

Lookahead becomes essential when our aim is to form a multistep plan to achieve some goal, within a context where there is a very large number of possible situations overall, but with a relatively constrained and predictable range of options in any particular situation. A familiar example would be attempting to solve a Sudoku puzzle, where we are unsure whether to write a 4 or 7 in a particular cell, so we start looking ahead to identify the consequences that will follow: "If I put 4 here, then that means this other cell will have to be 7, in which case this third cell cannot be 7 and must be 2, and then …". Here we are following a sequence of implications, with the idea that eventually we will either find a contradiction (in which case, the original cell must be 7 rather than 4, so we rewind – or "backtrack" – the entire sequence), or generate a complete solution.

For another example, consider the Tower of Hanoi puzzle (see **Figure 1.3**).[9] In this case, the tower consists of nine differently sized and differently colored disks, which were originally piled in order – from largest to smallest – on the leftmost pillar (where the three largest disks remain). There are two other pillars, and *disks can be moved from one pillar to another, but only one at a time, and never putting a disk on top of a disk that is smaller than itself*. Subject to these rules, the original task was to move the entire tower from the leftmost pillar to the rightmost pillar, and this picture is a snapshot taken part-way through the execution of this task. Suppose, then, that we encounter this unfinished task and wish to complete it. Here we have just three possible legal moves: (a) moving disk 1 (the smallest, red one) from back to left; (b) moving disk 1 from back to right; and (c) moving disk 4 (the purple one) from right to left. But which do we choose? Here we might try tactical lookahead to narrow down the sequences of options that look plausible (e.g., that give the appearance of making progress and don't involve "wasted" moves). But more promising is a strategic planning approach, whereby we try to identify what shorter-term outcomes would usefully contribute to the completion of our task. Here we might notice that to move, say, the light blue disk 7 from the leftmost

pillar, we need first to pile up disks 1 to 5 on the orange disk 6, so as to leave the rightmost pillar clear. To achieve this, we need to move green disk 5 onto orange disk 6, and to achieve that, we need first to pile up disks 1 to 4 (the purple one) on the leftmost pillar. So we can conclude that the best move here is to move the purple disk from right to left, and similar thinking will enable us to work out the rest of the puzzle.

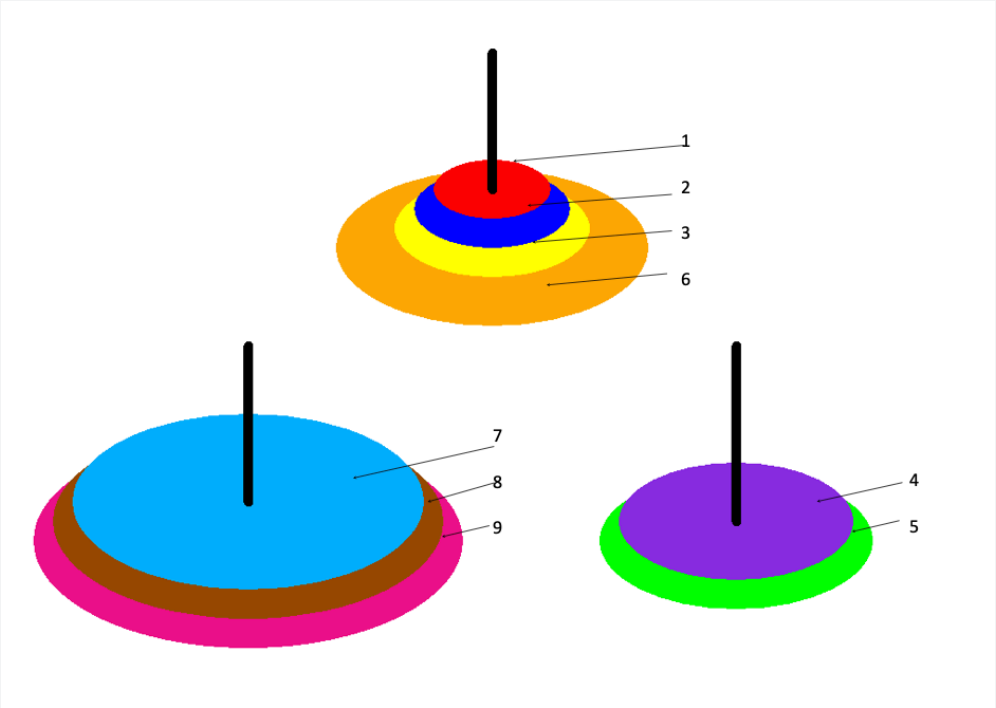

Figure 1.3

[9] Image created by author using his own *Turtle System* software, available free from www.turtle.ox.ac.uk.

1.5 Search

Notice that in working through these various options, we are conducting a search through the possible outcomes to find one that will best achieve our aims. And indeed, generating sets of possible outcomes and searching through them is one of the most fundamental techniques in AI (see a worked-out example of searching through a lookahead game tree in the Noughts-and-Crosses algorithm of section 1.7 below).

The first of these involves a number-guessing program, which provides a simple illustration of "binary search," whereby a search space is progressively divided in two until the desired target has been found. The user is invited to choose a number within a given range (say, between 1 and 1,000), and the program's job is to guess this number as quickly as possible. To accomplish this, we use two numeric variables *lowest* and *highest* to keep track of the current range, and having initialized these to 1 and 1,000, respectively, we enter a loop in which the program guesses the rounded mean of *lowest* and *highest*, and then asks the user to say whether this is *correct*, *too high,* or *too low*. If it is too high, then *highest* is revised downward to one less than the guess, whereas if it is too low, then *lowest* is revised upward to one greater than the guess. So whenever a guess is unsuccessful, the "search space" is reduced by at least half, removing the part that has been ruled out by the user's response. Assuming the responses are consistent, this will find the correct value within at most 10 guesses (since 2^{10} = 1,024, and there are 1,000 possibilities).

A more complicated example is attempting to find the optimal route from one city to another by taking an appropriate sequence of journeys (by train, let us suppose), with no obvious best combination to reach our destination. In the simplest case, we are attempting to optimize on just one criterion (e.g., distance, though it might also be cost or time), and we want our program to search through all the potentially optimal routes, finding the best of them. There are several different ways in which this can be undertaken – for example, it might use breadth first search, in which we keep track of all the different possible routes as we build them up stage by stage; or it might use depth first search, in which we follow each possible route to the end before moving on to compare the next one. Both have serious disadvantages: For example, breadth-first search is inefficient and expensive in memory, whereas depth-first search may waste time focusing on poor options initially while neglecting short and straightforward paths that breadth-first search would have found far more quickly. Usually better than either of these is to employ methods of heuristic search, where we take advantage of some additional information (e.g., the crow-flies distance between cities) to help us decide which options are worth exploring first. The best-known example of this kind of search

technique is the A* ("A star") algorithm, which for each location reached so far in our search, keeps track of both the distance that has been travelled so far to reach that location, and an (optimistic) estimate of the distance still to go to our ultimate destination (for example, the crow-flies distance).[10] The locations are kept ordered according to the sum of these two values, and exploration of ongoing routes always starts from the location that currently has the shortest sum.

Heuristic techniques such as these can indeed help considerably in facilitating many search problems, but unfortunately in more complex or demanding cases – for example the task of finding the *perfectly optimal* route around a list of cities (the so-called "travelling salesman problem") – the search space grows so enormously as the number of cities increases – a combinatorial explosion as the relevant numbers of possible choices get multiplied together – that certainty of solution quickly becomes unfeasible (see sections 1.8 and 1.9 below). Much the same applies to complex games such as chess, where searching for optimality also involves additional complications because it is adversarial**,** in that we are trying to maximize our own advantage while our opponent is trying to minimize it. An essential preliminary is the important concept of recursion.

---

[10] It is crucial that the estimate be *optimistic* if the A* algorithm is to be guaranteed to find the best route, because this means that potential routes that *might* prove to be better than any yet found are "kept in the game" (and the search halts when a complete route is found which is shorter than any estimated route remaining). Thus the Euclidean distance is eminently suitable as an estimate, since no train journey can be shorter. This would not be true if the algorithm were to use what is called Manhattan distance (i.e. sum of differences in x and y coordinates on some predefined grid, reminiscent of distance through the streets of Manhattan). But the latter is useful in various machine learning algorithms.

1.6 Recursion

The discussion of lookahead might have given the impression that any algorithmic solution to the Tower of Hanoi problem is bound to be highly complex, with sophisticated lookahead and comparison of possible future lines of play. But in fact the problem is extremely easy to solve using a powerful technique called *recursion*, which is of huge importance within classical AI (and, indeed, in computer science generally). The Tower of Hanoi is one of the most famous and elegant illustrations of the recursive technique, whereby a complex problem is solved by being progressively *reduced* to simpler instances of the same kind of problem.

We have already implicitly noticed how this sort of reduction is implied in the Tower of Hanoi problem, because whenever we want to move one of the larger disks – say disk 5 – from one pillar to another, we need first to pile up all of the disks that are smaller than it – here disks 1 to 4 – onto a single pillar, so that disk 5 will be free to move without violating the rule that it cannot be put on top of a smaller disk. It immediately follows that if we want to move *the entire pile of disks 1 to 5* from pillar *A* to pillar *B*, using pillar *C* as an intermediary, then we have to perform the following three subtasks:

To move the pile of disks 1-5 from A to B

- Move the pile of disks 1-4 from *A* to *C*
- Move disk 5 from *A* to *B*
- Move the pile of disks 1-4 from *C* to *B*

The second of these subtasks is trivial, just involving a single disk transfer. But what about the first and third subtasks: How are those to be achieved? Well, we can apply *exactly* the same kind of reasoning that we have just applied to the problem of moving the 5-pile from *A* to *B*, to the problem of moving the 4-pile from *A* to *C* (and later from *C* to *B*). Thus the first subtask itself divides into three:

To move the pile of disks 1-4 from A to C

- Move the pile of disks 1-3 from *A* to *B*
- Move disk 4 from *A* to *C*
- Move the pile of disks 1-3 from *B* to *C*

And, in fact, this simple line of reasoning provides a complete solution to the Tower of Hanoi, dividing every problem of moving a pile of *n* disks into two smaller problems of

moving a pile of $n-1$ disks (plus one move of disk n), so that in the end everything is reduced down to single disk movements. Moving the entire pile of 9 disks by this method takes 511 steps altogether (calculated as 2^9-1).[11]

---

[11] This can be neatly proved by the method of mathematical induction, as follows. Moving one disk obviously requires just 1 move, which is $2^1 - 1$. Now suppose – in line with the hypothesis that we are trying to prove – that moving k disks involves $2^k - 1$ moves. Well, in that case, moving $k + 1$ disks will require moving the pile of k disks to the intermediate pillar ($2^k - 1$ moves), then moving disk ($k + 1$) to the destination pillar (1 move), then moving the pile of k disks from the intermediate to the destination pillar ($2^k - 1$). Thus the total number of moves required to move $k + 1$ disks will turn out to be $2 \times (2^k - 1) + 1 = (2^{k+1} - 2) + 1 = 2^{k+1} - 1$, exactly fitting our hypothesis. The upshot is that if our hypothesis is true for $k = 1$ (as it is), then it must be true for $k = 2$, and then it must be true for $k = 3$, and so on for all positive integers.

1.7 Recursive Adversarial Tree Search – Solving Noughts and Crosses (Tic-Tac-Toe)

Recursion can also be applied to solve the relatively complex problem of playing an adversarial game, by combining with lookahead and search. Here our task is to design a program that plays a perfect game of Noughts-and-Crosses (also known as Tic-Tac-Toe), in the sense that it will never lose and will always find a winning sequence of play if one exists (though it won't necessarily find the quickest path to a win). Crucial to this will be devising a function that can assess the *value* of any position *to the player whose turn it is to move in that position* (i.e., whether it is a position that, with best play on both sides, will lead to a win for that player [value +1]; a loss for that player [value −1]; or a draw [value 0]). Once we have such a function, it will be easy to construct our program.

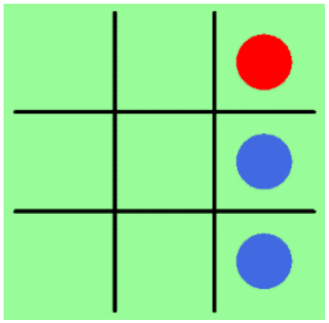

Figure 1.4

We start by considering a game between Blue (male) and Red (female) which has reached the position shown in **Figure 1.4**. It is Red's turn to move, and she has 6 choices of empty squares, so we call this a "6-space-position" or "6-position" for short. After Red makes her move, Blue will be faced with a 5-position in which to make his reply. Now suppose, hypothetically, that Red had some infallible method – some *oracle* – for assessing the value *to Blue* of any 5-position he might face: How could *Red* use this now to work out her best move? Clearly, Red should choose a move that gives Blue the *lowest possible value* in the resulting 5-position. In the current case, that means choosing one of the top two empty squares, either of which will yield a 5-position that is losing for Blue (i.e., with a value for Blue of −1).[12]

We now have a clear method for calculating the theoretical value of this 6-position to Red: We work through all the possible 5-positions that could result, pick the *lowest value* of any of these positions (here −1), and invert it (here +1) on the basis that −1 to

Blue is +1 to Red (and vice-versa). This is called adversarial search, because Red is supposed to look for whichever move is *least* advantageous for her opponent.[13]

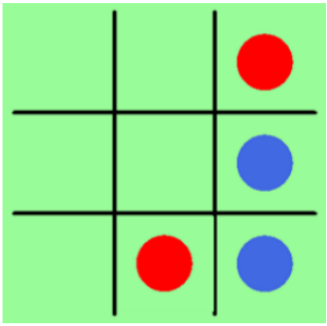

Figure 1.5

But all this depended on having an oracle that could tell us the value of any 5-position, and of course we have no such thing. But now we can repeat the same sort of reasoning at the next level, as we can see if we consider a possible (suboptimal) continuation from the position above, as shown in **Figure 1.5**. This is a 5-position with Blue's turn to move, and after he makes his move, Red will be faced with a 4-position in which to make her reply. Much as before, we imagine what Blue could do if he had some oracle for assessing the value (to Red) of any 4-position. In that case, he could work through all the possible 4-positions that could result from the current position, use the oracle to assess their value to Red, pick the *lowest* of those values (here −1, in the case where Blue moves in the middle square), and invert it to yield the value to himself of the current position (i.e., +1).

Note, importantly, that the same sort of reasoning will apply to *any* 5-position, *except where Blue has an immediate winning move*, so that one of the resulting 4-positions contains a line of three Blues – then the value to Red of that position is clearly −1, without requiring any oracle.

Spelling out the consequences of this line of reasoning:

- If we can evaluate any 5-position, then we can use these assessments to evaluate any 6-position;
- If we can evaluate any 4-position, then we can use these assessments to evaluate any 5-position;
- This pattern continues, with the upshot that if we can evaluate any 0-position (i.e., a position with no remaining moves to be played), then we can evaluate any 1-position, and hence any 2-position, and hence any 3-position, and hence any 4-position, and so on, all the way back to the initial 9-position.

- Apart from this, we need to be able to recognize when a game has ended owing to completion of a line (yielding a position of value −1 to the player whose turn it would otherwise be).

We can now specify our complete algorithm for evaluating a Noughts-and-Crosses (Tic-Tac-Toe) position in which it is player *X*'s turn to move:

1. Has *X* already lost (because the opponent has a line of three in the current position)? If so, the *X*-value of the position is −1.
2. Is the position full up (with all nine spaces filled)? If so, its *X*-value is 0 (i.e., drawn).
3. If neither of these, then construct in turn all of the positions (with *Y* to move) that can arise from the current position (i.e., try all of *X*'s possible moves in the current position in turn).

   Evaluate each of these new positions *from Y's point of view using this very same algorithm* (resulting in evaluations Y_1, Y_2, and Y_3), and then assign – as *X*-value of the *current* position – the *inverse* of the *lowest* of Y_1, Y_2, and Y_3.

When its turn comes, the computer should select a move that leads to a position with the lowest available evaluation (from its opponent's point of view).

---

[12] By contrast, if Red moves to either of the bottom two empty squares, then Blue should win (so the value of the resulting 5-position will be +1 to the player whose turn it is to move in that position); and if Red moves to either of the two middle empty squares, then the game should be drawn (value 0).

[13] In the specific context of two-player games, this is called minimax search, whereby when looking through the available moves, I presume that my opponent will choose whichever reply *maximises* their benefit, and on that assumption, I choose the move which *minimizes* that maximum benefit – i.e. I choose the move which is worst for them, on the presumption that they will be responding optimally. The same principle applies at every level, and the recursive algorithm described below ensures this.

1.8 Complexity, Heuristics, and Refinement by Reinforcement Learning

It is very surprising that an assessment algorithm that is capable of exhaustively analyzing all of the possible lines of play in a game of Noughts-and-Crosses, implicitly creating a "search tree," and keeping track of those positions' value can be implemented so simply (in fewer than 25 short lines of computer code). But there is a catch – the curse of *exponential complexity*, leading to a *combinatorial explosion*. Noughts-and-Crosses is practically solvable in this way only because the number of possible lines of play is so relatively small, with 9 possible first moves, 8 possible responses, then 7, 6, 5, and so on, making an upper limit of 9 factorial (i.e., 362,880, though the true figure will be significantly less because so many games finish sooner). If we try to solve complex games like go, chess, or even checkers in this way, then the depth and exponential growth of the search tree (with an average branching factor per move of around 250, 30, and 10, respectively) means that at any plausibly achievable speed of computation, the Earth will have been swallowed up by an aging Sun long before the program even makes its first move.

As Samuel recognized, therefore, a practicable checkers program could not possibly work by exhaustive search through all possible combinations of moves. Instead, in any position it must limit its exhaustive search to, say, the first *n* moves (e.g., just 3 in the case of Samuel's program, perhaps 8 [i.e., 4 moves for each player] in a modern program).[14] Then to assess the positions reached through this analysis (and thus to select which path is best to follow), it would use some set of *heuristics* – in other words, calculable criteria or rules of thumb that can usefully guide the choice of position to aim for, even though they cannot be guaranteed to reach an optimal solution. In the case of checkers, perhaps the most obvious criterion for assessing a position is based on material balance of our pieces as compared with our opponent's. A second plausible criterion might be the relative mobility of those pieces. A third could be how far advanced those pieces are (on the way to becoming "kings"), and a fourth the relative number of "kings" already achieved. Suppose, then, that we have come up with six such plausible criteria,[15] and have worked out a method of calculation for each of them such that, for any given position on the board, we are able to calculate a single number (say, c_1 to c_6) representing how far each criterion is achieved or not (e.g., in the case of material balance, this might be simply a positive or negative integer, representing the number of my pieces minus the number of yours). To compare one position with another overall, we also need to assign some relative *weight* to these six criteria (say, w_1 to w_6), and we can then calculate a *value* for each position by calculating the sum:

$$v = w_1 c_1 + w_2 c_2 + w_3 c_3 + w_4 c_4 + w_5 c_5 + w_6 c_6$$

Choosing a move in a particular position now becomes a matter of examining the positions that would arise from all the analyzed *possible* sequences of *n* moves in that position (i.e., all of the *n*-move sequences that are available, given the rules of the game), calculating the *value* of each resulting position, and selecting whichever move yields the highest value position, using the same kind of adversarial search algorithm that we saw in section 1.7 above. Where several moves seem equally good (or approximately so), we choose randomly among them. This has the twin advantages of making the program less predictable (and hence less easy to beat consistently), and providing a more-varied basis for learning.

Human judgement – informed by practical experience with the game – is involved here in identifying plausible criteria for assessing positions, which indeed might seem fairly straightforward for an expert player. But how much relative weight should we give to each of these criteria, when they tell in different directions? That is far more difficult to assess, but fortunately there is no need to rely on human judgement when selecting the crucial *weights* that feed into our move-selection algorithm, for this is where reinforcement learning can play a role. To start with a very crude method, we might initially assign w_1 to w_6 an equal value (say, 1,000). This defines our initial *current* strategy. We then make some random variation to these weights (e.g., by assigning a random change between +100 and −100 to each of them). This generates a *competing* strategy. Then we play a sequence of games between these two strategies (e.g., four games with *current* moving first, and four with *competing* moving first), either played out to completion or at least to a stage where both strategies agree that one player has a significant advantage. If *competing* does better overall than *current* in these games, we then replace *current* with *competing* (so that becomes our new *current* strategy).[16] Either way, we then go on to generate a new *competing* strategy randomly as before. Continuing iteratively in this way, the weights in our *current* strategy are progressively refined (by simulated evolution, so to speak), and we hope ultimately to achieve something close to an optimal set of weights, and thus the best possible strategy of this kind. Depending on the game and suitable choice of criteria, this sort of machine-learning technique can enable a program to learn to play better than its designer.[17]

---

[14] However, as Samuel observed ("Some Studies in Machine Learning Using the Game of Checkers", 1959, p. 538), it is vital to extend the analysis beyond *n* moves when a capturing combination or piece exchange is in progress, since otherwise the position will give a misleading impression of the material balance. Hence analysis would normally continue until the resulting position is *quiescent*. Note in passing that because the counting of "moves" is potentially ambiguous (e.g. does the chess sequence "1. e4 e5 2. f4 exf4" count as

two moves or four?), it is standard to use instead the term "ply", so that one move for one player is "1 ply", while two moves for each of two players (as in this case) is "4 ply".

[15] Samuel devised 38 criteria (in addition to piece advantage), of which 16 would be used at any time within the position scoring function (1959, p. 544)

[16] Of course there is scope for far more subtlety here – for example, the weights might be adjusted more or less, depending on how emphatically *competing* defeats *current*.

[17] Note that in the context of a complex game, which cannot possibly be analysed to its end owing to the *combinatorial explosion* of possible lines of play, it is not possible to find the optimal weights by mathematical optimisation methods – practical experiment and feedback from the eventual game results is crucial.

1.9 Limits of Classical AI

We have now reviewed some of the main ideas and techniques of classical AI, illustrated mainly within the simple context of familiar rule-governed games. And indeed, it was within such contexts that classical AI would ultimately prove particularly effective, its most conspicuous success coming in 1997 with the sensational defeat of World Chess Champion Garry Kasparov – then considered by many to be the strongest human chess player of all time – by *Deep Blue*. In less well-disciplined contexts, however, classical AI techniques struggled to fulfil their apparent early promise. One serious difficulty was how to represent the characteristics and relationships of things in the world, and especially the "rules" that govern their behavior. Even human experts find it very hard to elucidate the implicit background assumptions that guide their judgements, and spelling out our ordinary common-sense understanding of things proves to be extraordinarily difficult.[18]

A related issue was the so-called "frame problem," keeping track of which aspects of a situation change when some action is performed, but also which aspects stay the same. Codifying these things in detail added yet more fuel to the sort of combinatorial explosion that we have already mentioned, which quickly resulted when attempts were made to apply the general-purpose search mechanisms of classical AI to problems of any serious complexity. This became all the worse if the data in question were at all ambiguous or uncertain, requiring yet more complex calculation if *probabilities* were to be taken into account (quite unlike the situation in games of "perfect information" such as chess and checkers, which had been such popular testbeds within classical AI). Acquiring real-time data about physical things was inevitably very error-prone, with computer vision systems relying on tailor-made algorithms first to identify such things as edges, shapes, textures, and colors, and then to synthesize these features into representations of familiar objects. Such synthesis also proved to be immensely difficult except within specific domains (for example where a limited range of objects could be expected, conforming to familiar templates and with known dimensions of variation).

In response to such limitations, the focus of much classical AI research moved – especially from the 1980s – toward so-called *expert systems*, which were designed to capture knowledge within specific domains, often using *logic programming* as a general-purpose reasoning mechanism.[19] These systems were sometimes very successful, but their functionality could not easily be generalized beyond those specific domains without hitting the same old problems. Indeed, AI systems in general were notoriously "brittle," failing in unexpected ways when applied beyond their familiar narrow boundaries, or to situations that had not been explicitly foreseen by their designers. So, while progress continued in many areas where precise codification of facts and rules was achievable,

and computational techniques were refined to achieve increasingly impressive results in those areas, a pessimistic consensus emerged that Alan Turing's dream of a *general artificial intelligence* was likely to remain unfulfilled, at least for the foreseeable future.

---

[18] For example, Douglas Lenat's decades-long *Cyc* project, running from 1984, attempted to codify human common-sense knowledge, and by 2017 was reckoned to have absorbed over 1,000 person-years of effort, with around 25 million rules and assertions.

[19] The boom in expert systems (also known as *intelligent knowledge-based systems*) was provoked in particular by the Japanese government's massive investment in the *Fifth Generation Computer Systems Project*, which aimed to harness the logic programming language PROLOG as the inferential medium. Part of the impetus behind this came from the idea that a distinction could be drawn between a topic-specific "knowledge base" and a more general "inference engine", making it easier for "knowledge engineers" to extend the technology to new domains by gathering and codifying expert knowledge about those domains.

2.0 Neural Nets

Alan Turing had been contemplating, discussing, and writing about the possibility of AI as early as 1941,[20] while he was at Bletchley Park engaged in the even more important project of helping to defeat the Nazis. But what is often considered the first published contribution to modern AI development appeared soon after, in the *Bulletin of Mathematical Biology* of 1943, in the form of "A logical calculus of the ideas imminent in nervous activity" by Warren McCulloch and Walter Pitts. This paper, intended as a contribution to theoretical neurophysiology, described the activity of neurons in the brain, and proposed that the behavior of networks of these neurons could be analyzed in terms of propositional logic.[21] Ironically, in view of the later contrast that would be drawn between logical and neural approaches, the paper talks of neural nets as equivalent to Turing machines, and hence as limited by Turing computability (p. 129), saying that even "in psychology, … the fundamental relations are those of two-valued logic" (p. 131).

Donald Hebb's 1949 book *The Organisation of Behaviour*, however, put aside such approaches from "the application of mathematics" and instead found inspiration in biology (pp. xi-xii). He suggested a way in which associative learning could take place within networks of neurons, based on a "neurophysiological postulate" that "*When an axon of cell* A *is near enough to excite a cell* B *and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that* A*'s efficiency, as one of the cells firing* B*, is increased*" (p. 62). Frank Rosenblatt later took this further, proposing a specific computational model for how the brain learns and stores information in which, rather than treating memories as encoded representations of experiences, he instead favored the Hebbian approach. The latter, he suggested, "stems from the tradition of British empiricism" and treats "the central nervous system … as an intricate switching network, where retention takes the form of new connections" (1958, p. 386). Rejecting any symbolic or algorithmic approach, he accordingly formulated his new model – which he called the Perceptron – "in terms of probability theory rather than symbolic logic" (pp. 387-8).

---

[20] See for example Jack Copeland, "Artificial Intelligence", p. 353 in his excellent edited collection of Turing's writings, *The Essential Turing* (Oxford University Press, 2004).

[21] McCulloch and Pitts use the notation of Russell and Whitehead (whose influence was evident in §1.2 above), together with symbolism from Rudolf Carnap (a student of Gottlob Frege who had devised the foundations of modern logic). Their only other explicit reference is to Hilbert and Ackermann's famous work in theoretical logic, to which Turing's 1936 paper had been in part responding. So overall, although their paper concerns neuropsychology, it is absolutely immersed in the same logical background as Turing's.

2.1 Artificial Neurons

Rosenblatt's Perceptron led to a standard model of an artificial "neuron," which takes a number of numerical inputs (i_1, i_2, i_3 etc.) and outputs a single value v. Each input is multiplied by some weight (w_1, w_2, w_3 etc.) and these are added together (potentially with some constant bias term w_0) to make the net input. This is then fed into an activation function f to determine the output value v:

$$v = f\,(w_0 + w_1\, i_1 + w_2\, i_2 + w_3\, i_3 + \ldots)$$

The activation function is chosen to be non-linear, and often approximates to a step function that gets triggered when the net input is greater than some threshold (so the neuron's output is "all or nothing").[22] This non-linearity is crucial if a network is to be able to learn non-linear behavior.

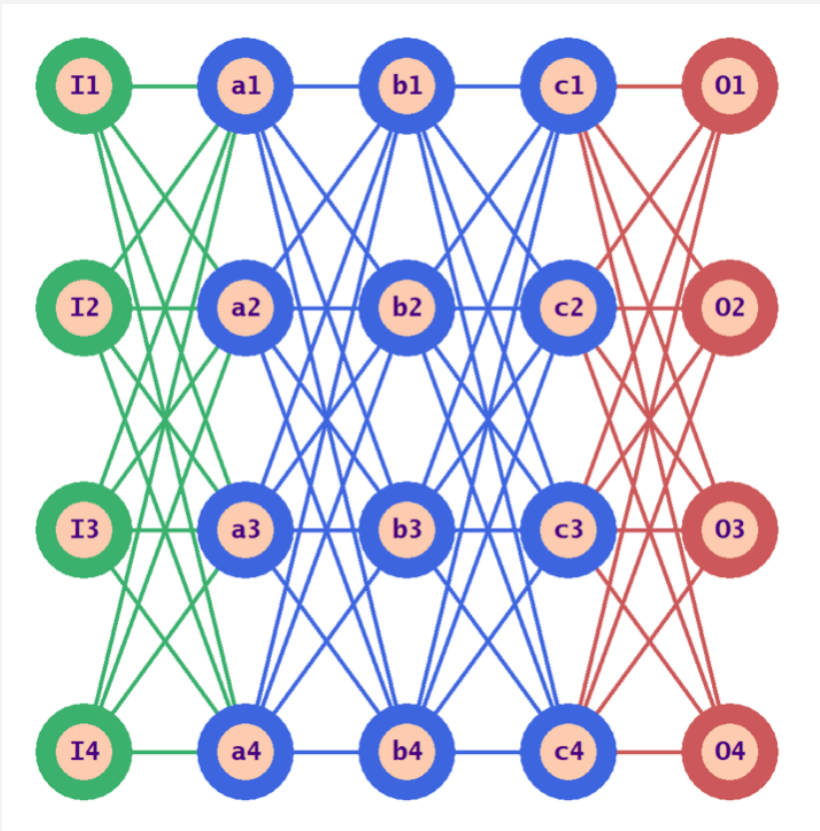

Figure 1.6

In recent years, artificial neurons of this general kind have been powerfully linked together in "deep" networks that have multiple additional layers between the input and output layers. **Figure 1.6** is a simplified diagram of a neural net with five layers: the input layer (labelled "I"), three hidden layers (labelled "a", "b," and "c"), and the output layer (labelled "O"). Each neuron in the input layer is given a specific numeric level of activation, in such a way that the overall pattern of activations represents the input data. For instance, each neuron's activation might store the color value of one of the pixels in an image (in which case, the actual number of input neurons will be far larger than the mere four shown here). In this example, every neuron in the input layer is connected to every neuron in the first hidden layer (i.e., the a-layer), so that the level of activation of the a-neurons will depend – in the way described above – on their activation function f_a as applied to the net input that they receive from the input layer. In the same way, every neuron in the second hidden layer (i.e., the b-layer) will depend on their activation function f_b as applied to the net input that they receive from the a-layer, and so on. Typically, the activation functions will be consistent across the neurons, but *the weights of the individual connections will vary*, thus influencing the propagation of activity through the network from the input layer, through the hidden layers, to the output layer O. It is these changing weights that represent the learning of the network. Hence in a successfully trained network, the weights will have evolved in such a way that the activation of the output neurons does indeed provide the desired response to the relevant input. In a network trained for recognition of digits, for instance, there are likely to be ten output neurons – one for each of the digits from "0" to "9" – whose activation should tell us which digit is represented by the image whose pixel values are reflected in the activation of the input neurons.

---

[22] For example, the sigmoid and hyperbolic tangent functions have commonly been used. In deep networks, however, the Rectified Linear Unit (ReLU) function – which does not closely approximate a step function – has become more popular, partly because it is more computationally efficient. This makes $f(x)$ equal to x if x is positive, and zero otherwise.

2.2 Connectionism and its Early Challenges

Rosenblatt's work inspired widespread interest in artificial neural networks, and an appreciation of some of their particular strengths as learning mechanisms. But in 1969, Marvin Minsky and Seymour Papert published a fierce critique in their book *Perceptrons*, which severely undermined this enthusiasm. In particular, they showed that Perceptrons were incapable of learning some simple logical functions (notably anything depending on an "exclusive or"), thus apparently wrecking any prospect that they might provide a route toward "intelligent" information processing of any complexity. But it later turned out that this critique applied only to neural networks consisting of just a *single layer* (as in Rosenblatt's original design), and that more-complex functions could be learned by arranging artificial neurons (such as Perceptrons) into multiple layers. A network with a large number of such layers is called a deep neural network (see section 2.1).

Adding additional layers, however, significantly complicated the process of learning, and so this route was largely ignored until the back-propagation learning algorithm rose to prominence in the 1980s. To illustrate how this works, suppose that we have a network of the type described in section 2.1, with input data representing the pixels of an image, and ten output neurons intended to identify the pictured digit. In this setup, we would hope that if an image of a "6" is input, then the output neuron corresponding to "6" would have maximal activation (close to 1) and the other nine would have minimal activation (close to 0). If the actual pattern of activation is very different, however, then learning is required, so we calculate how much a small change in the weights of the final (output) layer would contribute (positively or negatively) to improving the match.[23] Back-propagation is then the process of working back through the layers, doing a similar calculation for all the other weights in the network.[24] Once this has been done, the weights are adjusted by a small amount, in the direction that would bring improvement. This process is then repeated with other images – potentially thousands or millions of times – and so the network learns by iteratively adjusting its weights.

The discovery and success of back-propagation led to a resurgence of interest in what became known as "connectionism," encouraged in 1986 by the two-volume *Parallel Distributed Processing: Explorations in the Microstructure of Cognition* by David Rumelhart, James McClelland, and the PDP Research Group. This displaced the pessimism of Minsky and Papert, demonstrating how layers of simple interacting "neurons" could achieve learning of complex, cognitively relevant functions. The approach was attractive for at least four reasons. First, it was biologically inspired, and consequently had the potential to shed light on how we think. Second, and relatedly, it seemed to learn in much the way that we do, through association and feedback in

response to success or failure. Third, this method of learning was quite general, and could apparently be applied to a huge variety of tasks and domains, without requiring any special programming. Finally, the storage of this learned information, rather than being explicitly represented, was obscurely "distributed" through the network of weights, which made the learning process more robust in response to "noisy" or ambiguous data, better able to generalize, and less liable to break down entirely as the quality of data declines (i.e., it exhibited "graceful degradation").

These advantages became well known, but enthusiasm for connectionism steeply declined in the 1990s as research failed to deliver the sort of practical breakthroughs that had been expected. But widespread turning away from neural networks would again prove to be premature, for their day would come once computer technology was able to deliver the computational power required by multilevel learning.

---

[23] Technically, this involves calculating the partial derivative of the loss function (which measures the magnitude of the mismatch) with respect to the given weight.

[24] Suppose we have already calculated the partial derivative of the loss function for every weight in layer n. Then back-propagation involves using the *chain rule* of differentiation to calculate – based on these layer-n results – the relevant partial derivatives for layer $n-1$. Thus we work backwards through the network.

2.3 Deep Learning Proves its Potential

Work on neural networks continued, with a particular focus on image and speech recognition, problems that very much played to their strengths. Consider again the familiar task of identifying digits (and letters) in print or handwriting – something of great potential practical value in applications such as sorting mail or processing bank checks. Here an important breakthrough came in 1998, when Yann LeCun and colleagues published their work on LeNet-5, a *convolutional* neural network designed for recognition of digits.[25] This is a deep network with a number of "convolutional" layers, each of which has the effect of applying a local filter (or "kernel") repeatedly to points across the grid, which in this case is a grid of pixel values in a (grayscale) digitized image. The filter consists of a small matrix of numerical "weights," and each of these weights is multiplied by the pixel activation value at its position, with the sum of these products providing the corresponding activation value in the convolutional layer (at the filter's central point). The weights within the filter matrix are initially set randomly, and subsequently "learned" in the same sort of way as other weights in the network. The virtue of such a convolutional layer is to enable efficient identification of local features in an image, which can then feed into the remainder of the network. LeNet-5 consisted of 8 layers, starting with the input layer that stores the pixel values of the digital image, then two sequences of a convolutional layer, followed by a "pooling" layer (which reduces the dimensions of the "feature map" by dividing it into 2 × 2 squares and averaging), then two further layers, and finally the output layer that classifies the digit.[26]

Startling vindication of the amazing power of deep networks for image recognition came in 2012, when AlexNet – developed by Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton – soundly thrashed the other competitors in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC), a high-profile international competition to identify images taken from the large ImageNet collection built up by Fei Fei Li.[27] The winning score in 2011 had been an error rate of 25.8%; AlexNet in 2012 had an error rate of 15.3%! Compared with LeNet-5, AlexNet was gigantic, nearly 100 times bigger with almost 900,000 neurons. The computational power required had become possible thanks to the rapid development of graphics processing units in response to the demand for high-quality animated computer games. To enhance speed of animation, these are designed to process image pixels *in parallel* (with recent versions having as many as 10,000 cores), and this has turned out to be perfect for the sorts of huge matrix operations that are required by neural networks.

[25] Yann Lecun, Léon Bottou, Yoshua Bengio and Patrick Haffner (1998), "Gradient-based learning applied to document recognition", *Proceedings of the IEEE* 86 (11), pp. 2278–2324.

[26] In a bit more detail, the input image was 32 × 32 pixels, which after convolution with a 5 × 5 filter reduces to 28 × 28 (because the center point of the filter never reaches the pixels within 2 of the boundary). In fact 6 different filters were used here, so now there are 6 "channels" (i.e. 6 neurons for each point of the 28 × 28 "feature map"). In the next "pooling" layer the values are averaged pairwise, reducing the size to 14 × 14 × 6, after which a further convolution with 5 × 5 filters reduces the size to 10 × 10 (though now with 16 channels), a pooling layer takes this down to 5 × 5 × 16, and the final three layers have respectively 120, 84 and 10 neurons. Altogether this adds up to 9,118 neurons in the 8 layers.

[27] Alex Krizhevsky, Ilya Sutskever and Geoffrey Hinton (2012), "ImageNet Classification with Deep Convolutional Neural Networks", *Communications of the ACM*, 60 (6), pp. 84-90. Fei Fei Li started building ImageNet in 2006, crowdsourcing images and their "labels" (i.e. categorization) using Amazon's Mechanical Turk. By 2009 it consisted of 3.2 million images labelled into 5,247 categories; at the time of writing it has around 14.2 million images, in nearly 22,000 categories.

2.4 Deep Learning Beats Symbolic AI at its Own Game

Another major deep-learning landmark came in 2013, when the London company DeepMind (founded in 2010) announced its success in programming a deep convolutional network to learn to play vintage Atari video games from the 1970s, including *Breakout* and *Pong*, on which it learned to perform vastly better than an expert human. The remarkable thing about this achievement is that the system was not given any information about the *goal* of the games, the *significance* of the screen images, or the *effect* of the user's actions (e.g., pressing buttons). It had to learn what to do entirely on the basis of knowing that a certain number of distinct actions were available, and trying these out in response to pixel information about the changing images and the game score. Everything else was done by deep reinforcement learning based on the score feedback, so in a sense, the system was teaching itself how to play from scratch.

In 2014 DeepMind was acquired by Google, and soon after another of its deep-learning programs – named *AlphaGo* – made even more headlines when it defeated European champion Fan Hui 5-0 in a match of Go, a game that until then had been considered too subtle and complicated for computer algorithms to master in the foreseeable future. The following year, *AlphaGo* sensationally went on to defeat World Go Champion Lee Sedol, 4-1. And in 2017 DeepMind revealed *AlphaZero*, a more generalized program capable of teaching itself new games, including both Go and chess. It did not quite do this "from scratch" in the way of the 2013 Atari program, for it was given the starting position, the moves of all the pieces, the various rules (including how to identify a win, draw, and loss), and had efficient recursive adversarial tree searching (as in section 1.7 above, but hugely optimized) built into it. But in contrast to sophisticated traditional AI programs like *Deep Blue* (which had scored a spectacular triumph by defeating Garry Kasparov 20 years before), *AlphaZero* learned how to play by competing against itself (in much the same way as the Nim program and Samuel's checkers program) without any input from human experts or game databases. Self-training in this way for only a few hours, it was even able to defeat the 2017 world champion chess program *Stockfish*. And now *Deep Blue*'s 1997 achievement looked relatively modest, for it (like *Stockfish*) had been specifically programmed to play chess, with input from expert players and using algorithms that were finely tuned to reflect established theory. *AlphaZero* was an altogether more impressive system, teaching itself new skills to an even higher level than human ingenuity had been able to achieve, and thus representing massive and potentially frightening progress toward Artificial General Intelligence.

2.5 The Inscrutability of Deep Learning

Part of the promise and threat of deep learning lies precisely in its ability to represent all kinds of information in ways that it works out for itself in response to training data and feedback. This information is stored implicitly within the weights of a neural network that consists of layers of artificial neurons. As we saw in section 2.1, the input layer of the network is set up to reflect the specific problem case, so for example the activation level of each input neuron might represent the state of one square in a game position (in which a move is to be chosen), or the color of one pixel in a digital image (which is to be classified). The output layer is set up to signal the corresponding solution, respectively the chosen move or the image label (e.g., "cat" or "dog" if the task is to classify images of pets). But in a deep network there may be many intermediate layers, with patterns of activation passing through the layers from the input layer to the output layer.[28] Beyond the input layer, the activation of each neuron depends on the inputs it receives from the neurons to which it is connected in the previous layer (possibly involving convolutions or other kinds of local processing, as briefly described in the case of LeNet-5 above). These inputs depend both on the level of activation of those previous neurons, but also – crucially – on the weights given to the relevant connections (as described in section 2.1). These weights are adjusted during the reinforcement learning process (by back-propagation or refinements thereof), which typically involves going iteratively through the training data, assessing the resulting outputs, and gradually refining the relevant weights until a sufficient match between inputs and outputs has been achieved. Neither the weights nor the roles of the neurons in the intermediate layers are predetermined when the learning process starts. By the end, the immensely complex pattern of weights implicitly represents what the network has learned, but in a way that an unaided human will find impossible to interpret, and whose behavior they will be able to predict only by experience.

All this can look almost magical, enabling a deep network to solve problems that we have no idea how to address, and using calculations that are beyond our comprehension even when they have been discovered. But we should not be misled to suppose that the machines themselves "understand" what they are doing in any reflective way, and not only because they are completely non-sentient (and hence have no awareness or conscious understanding of anything). In conspicuous contrast to Classical AI techniques, their way of working is far more closely analogous to our own unconscious pattern-recognition than it is to how we think when explicitly calculating or reasoning about something. And we are unlikely to consider that we really understand *how* we ourselves operate when recognizing an animal as a dog, or a symbol as the letter "f" – we too learn

these things by examples, building up neural structures of which we are completely unaware, and whose results we find hard to articulate. Likewise in expert chess-playing – often considered a paradigm of explicit calculation – many moves are decided by a sort of trained instinct, involving implicit recognition of a position's characteristics and appropriate strategies, to the extent that a grandmaster will often choose a move with negligible calculation, and if asked, may initially have little more to say than "in this sort of position, that is the right move to play."[29] There is an obvious analogy here to a neural network's method of position evaluation by pattern recognition. Of course, the grandmaster also needs to be able to tell when explicit calculation is needed, for example when tactical combinations are imminent. But as noted above, *AlphaZero,* too, can perform explicit calculation, exploring options down the "game tree" of branching possibilities. Thus, within its carefully specified and rule-governed context, it was able to generate its own training data and feedback by playing lots of different games against itself, learning by experience which patterns were most conducive to success. But unlike Samuel's checkers program, discussed above, *AlphaZero* was not primed in advance with the relevant patterns – it learned them all for itself.

---

[28] In *recurrent* networks activation can also pass up the hierarchy of levels, but we ignore those here.

[29] This is not to deny that the grandmaster could probably expand on the strategic significance of the move given longer to think about it, but the point is that such deeper reflection would often not feature when initially choosing the move within the game.

2.6 The Dawn of Artificial General Intelligence?

Since 2022, we have witnessed a new "AI shock" that seems even more significant and widespread than any that has come before, with the release of OpenAI's chatbot *ChatGPT*, quickly followed by similar types of systems from rival companies. These again have been developed using techniques of deep learning, but this time with colossal human textual input in the form of around 300 billion words from various sources on the internet. And now we suddenly – and unexpectedly – are faced with a technology that seemingly has the potential to pass the "Turing Test" in its full generality, to converse plausibly, flexibly, coherently, and informatively about a vast range of topics, and without relying on pre-prepared outputs.

How should we assess *ChatGPT* and its cousins from a theoretical point of view? The system was developed by applying statistical analysis on those 300 billion words of textual data to create a large language model (LLM) that – to simplify somewhat – records the probability that any given sequence of words (within its broader context) will be continued in different ways. And accordingly, *ChatGPT*'s primary method of working is to predict which individual words are most likely to follow in any particular textual context, and then to choose one of these words (but not always the most likely word, since an element of randomness enables it to respond differently if prompted again in the same way). That chosen word is then added to the existing text, and the next word chosen in the same way, and so on. Alongside the purely automated learning that generated the (reportedly) trillion or so parameters (i.e., stored probabilities) in the LLM (reflected in the deep network's weights, as explained earlier), human users improved the system's responses through supervised fine tuning, in which it was given a wide range of typical prompts (i.e., potential user inputs), together with suitable human-crafted responses. Then a stage of reinforcement learning from human feedback was applied, whereby the system generated a range of responses for each given prompt, and humans assessed the relative suitability of those responses. This feedback was then statistically analyzed to generate a "reward model", which could in turn be used to rate responses in general, and thus assist in selecting the most appropriate.

This is, of course, a hugely simplified description of how *ChatGPT* was created and how it works. But two points here are particularly worthy of emphasis. First, its method of working is based entirely on imitating – with variation – the sorts of responses that it found in the massive textual resources that were used to train it. Second, the information it stores implicitly within its trillion or so network weights has been tuned to reflect *the characteristics of the textual data*, rather than *the characteristics of whatever domain the text might concern*. Hence, for example, if you play chess against *ChatGPT* using a

common opening (e.g., a main line Sicilian Defence), it will initially respond with some of the same sensible moves that dominate its training data, and thus give the impression that it understands what is happening on the chessboard. But it stores no internal model of the board position, has no mechanisms of analysis or "lookahead" (unlike *AlphaZero*), and is therefore subject to absurd errors, such as overlooking obvious captures, or allowing your bishop to capture one of its own pieces even when the diagonal between them is blocked. Play a less common opening, and the problems will appear much more quickly (e.g., losing track of a pawn on the fourth move of a King's Gambit). Ask it to solve a simple chess puzzle and it will likely have no clue. It seems likely that before long these particular foibles will be dealt with by linking *ChatGPT* to a dedicated chess engine that really has a model of the board together with relevant analytical algorithms. But the key point here is that its apparent *general intelligence* – its ability to converse "intelligently" about a vast range of topics where it has not been augmented with specific assistance – is based on the illusion that it has some internal understanding of what is being discussed. But unless the language model in some domain is able to generate indirectly a reliable model of the reality (which looks most plausible in domains that are primarily conceptual or expressive), it will have nothing like such internal understanding, and can quite appropriately be described as a "stochastic parrot."[30]

---

[30] The term "stochastic parrot" was coined by Emily M. Bender in 2021. Recent work by my Oxford colleagues Philipp Koralus and Vincent Wang indicates that the most recent generations of ChatGPT have become better at mimicking *good* human thinking, but also *fallacious* human thinking (arXiv:2303.17276v1). This again emphasises how the system is working by modelling *human language* rather than *the relevant domain*.

3.0 Machine Learning and Its Risks

We have now seen in general terms how AI rose to prominence, initially as a relatively conventional branch of computer science (with well-understood data structures and algorithmic methods), but then took a radically different trajectory with the dramatic, accelerating, and previously unexpected rise of contemporary machine learning over the last quarter-century. Understanding this contrast is crucial to appreciating why AI is now seen as posing quite distinctive and novel risks in a way that previous computer systems did not. But before moving on to review these risks, it will be helpful to impose a bit more structure by distinguishing among four main types of machine learning, whose methods, benefits, and potential risks differ quite significantly.

3.1 Four Types of Machine Learning

So far, we have focused mostly on **reinforcement learning**, which is typically involved in situations where a sequence of actions or decisions is to be made within a changing environment to achieve some goal. Such learning involves "trial and error," with positive feedback where things turn out well, and negative feedback where they turn out badly. This is most easily illustrated in the case of competitive games, such as the Nim example in section 1.3 and the Atari, *AlphaGo,* and *AlphaZero* game-learning programs discussed in section 2.4. Reinforcement learning is also valuable in many other domains, for example:

- Robotics – teaching robots or autonomous vehicles to move and manipulate objects effectively, to interact with humans and learn from human feedback
- Healthcare – monitoring effects of treatment both for groups and individuals
- Finance – trading algorithms
- Optimization – broadly, any area in which optimization problems can be addressed by automated learning from experience

Second, there is **supervised learning**, where the aim is to learn from a set of *labelled examples* (the *training data*) either how to categorize further examples of the same kinds of things, or to predict some characteristic. As with reinforcement learning, this was long part of the repertoire of classical computation, in the form of methods such as linear regression, decision trees, and support vector classification. But the kind of supervised learning that is today seen as generating special risks is derived from *deep learning*, particularly because of its inscrutability and possible hidden biases (see section 3.3). The most prominent example of such deep learning applied to problems of *categorization* is perhaps the classification of images, as we saw in the case of LeNet-5 and AlexNet, and which has since become ubiquitous in applications from recognition of handwriting, faces, and road signs, to analysis of medical scans and visual scenes. Other applications include *speech recognition*, *machine translation*, and *recommendation systems*. Learned *prediction* (often called *regression* when the relationships involved are relatively straightforward) operates differently, in that now the "labels" are typically numerical values rather than categories (though the numbers can represent *probabilities* of category membership), and the aim of the learning process is to be able to predict the corresponding value for new instances. Here prominent examples include:

- Financial analytics of likely stock prices or future demand

- Asset valuation of house prices
- Credit scoring probability of default
- Weather prediction of wind speed, rainfall, or temperature

This can also be applied to domains already mentioned (e.g., medical diagnosis and prediction), and broadly, to any area in which a wealth of past data can be harnessed to predict future outcomes.

The third main type of learning, of which we have yet to discuss any examples, is **unsupervised learning**, where the aim is to identify patterns within the given data, but without the help of human input in the form of specified "labels." The two most prominent forms of such learning focus on cluster analysis and dimensionality reduction. Cluster analysis is a learning technique in which groups of similar objects are automatically identified without needing a training set. Dimensionality reduction, on the other hand, is a technique in which the main "dimensions" of variation among the objects are identified, enabling their range and relationships (e.g., comparative closeness) to be more easily grasped. This yet again is a form of learning that can be done using classical methods as well as using deep networks, and here – for ease of comprehension – we look at one of those classical methods that nicely illustrates both cluster analysis and dimensionality reduction. This is the important technique of *Principal Component Analysis* (PCA), and we briefly review two such applications, first to the geographical layout of a village, and then to the analysis of written texts.

The fourth type of machine learning is **semi-supervised learning**, which is a combination of supervised and unsupervised learning. It is used in cases where only part of the available data is labeled. The unlabeled data is used to determine patterns within the explanatory variables, or it is fitted with "pseudo-labels" determined by estimating a model on the labeled part of the data. We discuss this technique in Module 2.

3.2 Examples of Unsupervised Learning (Principal Component Analysis)

To aid our imagination, we start with an example from *physical* space rather than a virtual space of numerical values. Suppose, then, that we plot the position of houses in a village, starting from coordinates in terms of latitude, longitude, and height above sea-level. PCA will then transform these coordinates into a different framework, so that the various dimensions – instead of lining up with latitude, longitude, and height – will instead line up with *whatever directions best discriminate between the data items*. Thus, if the village has grown up along a fairly straight road on mainly flat land, then the dimension that gives most discriminating information (i.e., along which the houses are most spaced out) will be horizontal and in the direction parallel to the road – so that is our first "vector." To provide a proper coordinate frame, the second vector must be orthogonal (i.e., at right-angles) to the first,[31] and to preserve most information, we choose another *horizontal* vector (because the houses will vary more in distance from the road than they do in height above sea-level or in any diagonal direction).[32] Our third and last vector must be orthogonal to both of the others, and hence must be vertical. Note that unless the road runs north-south, the resulting coordinate frame will be different from the original latitude/longitude/height frame. The coordinates of each house within the new frame can be calculated from the original coordinates, and we then imagine looking into the space from different directions. To see most information, we want to view distances according to the first two coordinates, so we look from above – as indeed maps standardly do.[33] Thus the PCA analysis yields exactly what we would wish: the most informative presentation of the layout of the village.

For a very different example, we can look to classic work in stylometry by John Burrows, published in 1982, and replicated in **Figure 1.7** using the *Signature* software system.[34] Burrows set out to explore whether "The Memoirs of a Lady of Quality" (denoted as "Quality" in **Figure 1.7**), published in 1751 as part of Tobias Smollett's *Adventures of Peregrine Pickle*, was in fact written by Smollett. With this in mind, he applied PCA to the relevant texts of fifteen early eighteenth-century novelists, including Smollett himself. This method starts by counting the occurrences of all words in the entire textual corpus of these authors, ordering them by overall frequency, and selecting the most common 50 ("the," "of," "and," "I," "my," "to," "a," etc.) for further processing. Then the fifteen authorial texts – together with the "Memoirs of a Lady of Quality" – are individually re-read, in each case counting the occurrences of these common words and dividing by the text's total word-count. This yields 50 fractional values for each of the 16 texts, representing the relative frequency of each word within it

(e.g., if a word occurs 61 times in a text of 10,000 words, then that text's value for that word is 0.0061).

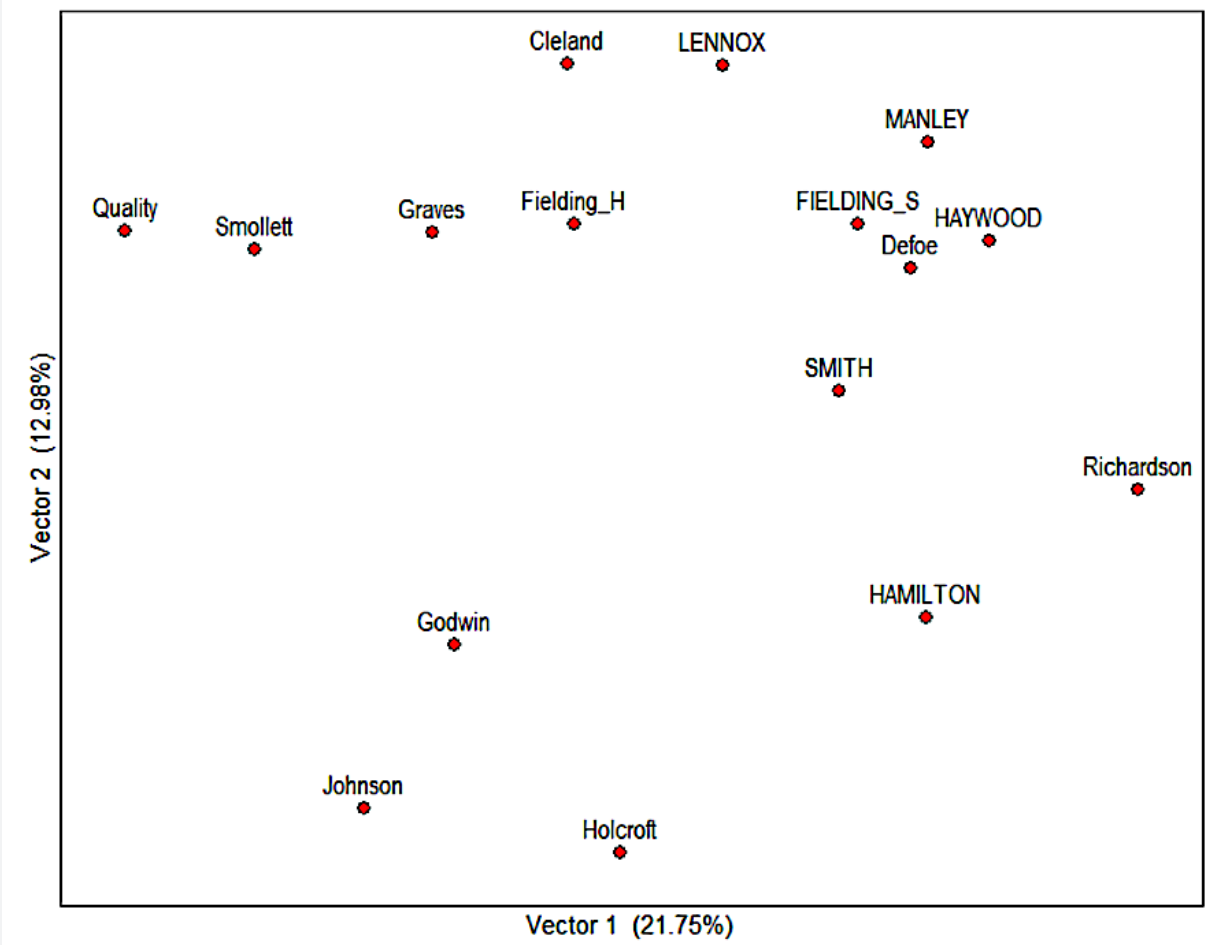

Figure 1.7

We then imagine these 16 texts placed within a 50-dimensional space (one equally extended dimension for each word), in which the 50 coordinates for each text are proportional to these relative frequencies. Thus, each of the 16 texts is represented by a point within this space, in such a way that two texts whose points are close together have broadly similar patterns of word occurrence (i.e., the same words tend to be relatively common in both texts, and the same words tend to be relatively uncommon, as compared with the other 14 authorial texts). Just as with the village map example above, PCA now gives us a way of looking into that space to see a two-dimensional projection of it *from whichever direction gives most information* – that is, whichever direction shows the texts maximally spaced out in a two-dimensional projection (so that relative similarities and differences can be most clearly seen).[35] Extensive calculation is needed to discover which direction gives the best projection, something that would be hard and extremely tedious to do by hand.[36] Hence, this technique is commonly considered as

falling under the umbrella of "machine learning," although as Burrows' example makes clear, it significantly pre-dates the recent machine-learning revolution.

What is striking about this method – and further illustrated also by other examples in Burrows' interesting paper of 1982 – is that it somehow manages to cluster the authors in such a way as to identify patterns of similarity between them, even though it has been done entirely automatically. In the current case, for example, we see clustering of the female authors (whose names are capitalized), with "Memoirs of a Lady of Quality" both distant from this cluster, and very close indeed to Smollett himself, thus yielding a fairly decisive positive verdict on his authorship. But although this is clear, it is hard to understand exactly what these similarities and differences are. The graph represents them as distances within a vector space, but there is no simple way of explaining what "Vector 1" and "Vector 2" signify, because they do not represent simple word frequencies, but rather, complicated linear functions of 50 different word frequencies.

To sum up, then, PCA provides a powerful method of *dimensionality reduction*, whereby a large array of points in multidimensional space can be "reduced" to a lower dimensional space, while effectively representing the most significant measures of closeness and distance within that multidimensional space. For the same reason, it can highlight clusters of similar items to serve as a basis for future investigation. And even though it might often be hard to pin down exactly the basis of this similarity, PCA does not reach the extreme level of inscrutability that we find with deep network learning, because here at least we can understand in general what the dimensions of variability represent, and explore their basis relatively straightforwardly.

---

[31] Technically, the mathematical processing involved in PCA identifies a sequence of *mutually orthogonal vectors* that form a *basis* of the multi-dimensional space, ordered so that those vectors that preserve most information about overall distances come before those that are less informative. For a rigorous but relatively accessible presentation of the mathematics involved and some applications, see Ian T. Jolliffe and Jorge Cadima, 'Principal component analysis: a review and recent developments', *Philosophical Transactions of the Royal Society A* 374 (2016).

[32] Note that if the ground slopes systematically in any direction, then this second vector will also slope accordingly, because the sloping distance between the houses will be greater than their horizontal distance.

[33] Other possibilities would be to focus on the first and third coordinates, thus looking from the side, or on the second and third, thus looking from one end of the road; but each of these gives far less discriminating information about the layout of the houses than looking from above.

[34] John Burrows, "Computers and the Study of Literature", in Christopher S. Butler (ed.), *Computers and Written Texts* (Oxford: Blackwell, 1992), pp. 167-204. An early version of the *Signature* software is available from https://www.philocomp.net/texts/signature.htm. Also documented there are the present author's stylometric investigations of a malicious rumour regarding Barack Obama's *Dreams from My Father* shortly before his 2008 election, and the authorship of J. K. Rowling's pseudonymous *The Cuckoo's Calling* of 2013. Both of these adventures, rather bizarrely, ended up on the front page of the *Sunday Times* newspaper!

[35] If this seems hard to imagine, start with just a three-dimensional space (e.g. based on relative frequencies of 'her', 'him', and 'me' within the texts), and imagine looking into it from different directions. Here it might help also to compare with the more familiar three-dimensional example of village houses, as given above.

[36] John Burrows did the calculations with a 1972 statistical package called *Minitab*. That is clearly preferable to hand calculation, but still far more tedious and error-prone than using software designed for the purpose.

3.3 Risks of Inscrutability

The inscrutability of deep network machine learning is problematic for a number of reasons, which can be understood in general terms given what we have discussed. First, when classification or prediction is based on supervised learning with deep networks, it is likely that any bias that exists in the labelling of the training data (e.g., owing to historical prejudice against particular categories of people) will be implicitly learned by the model and perpetuated, *but in such a way that its presence is hidden and hard to eradicate*. Suppose, for example, that a company has for many years prejudicially favored men over women, and it decides to create a computational tool to filter job applicants, based on data about its past recruitment (e.g., in the form of applicants' CVs and their acceptance or rejection). Clearly the resulting model is likely to exhibit a significant bias in favor of men, because such a bias assists it in correctly predicting the "labels" in the training data that it is attempting to match. Nor can this bias be removed by the simple expedient of removing information about the applicants' gender from the CVs before processing them, because a significant proportion of the other information in those CVs is likely to act as an effective "proxy" for gender. Lots of features of our lives, from the subjects we choose at school, to the sports and hobbies we pursue, to the careers we embark on, are strongly correlated with gender, and in societies that exhibit gender bias, these correlations are even stronger. So even if *explicit* gender information is removed from the training data, the trained models are likely to learn other correlations that have a similar effect. Another example of this type would be if financial decisions (e.g., approving a loan) have traditionally been racially biased, within a city whose areas are – not coincidentally – radically segregated to a significant extent. If information about applicants' race is completely expunged from the training data, the learning algorithm may instead come to base its judgements on the strong correlation between postal code and historical decisions, thus again unwittingly perpetuating that long-standing prejudice.

Another related risk is to *privacy*, because this deep implicit entanglement of complex information within the learned network can easily hold clues to personal characteristics that we would prefer to keep secret. In 2013, for example, researchers at Cambridge University found that Facebook "Likes" could be used "to automatically and accurately predict a range of highly sensitive personal attributes including: sexual orientation, ethnicity, religious and political views, personality traits, intelligence, happiness, use of addictive substances, parental separation, age, and gender."[37] This also carries significant risks of *manipulation*, in both the commercial and political sphere, because those who are able to identify our personal characteristics and foibles may also be able to play on them through the now familiar phenomenon of targeted advertising.

All of these problems arising from the inscrutability of deep network models are rendered more intractable because their complex incomprehensibility also makes it very hard to identify – with a view to eradication – the source of any such hidden clues or biases. Unlike a traditional expert system, for example, the deep network model can provide no *explanation* of how it reaches its decisions and predictions. The true explanation is hidden in that vast web of weights,[38] possibly billions in number, which cannot possibly be rendered humanly comprehensible. Finding ways of dealing with this and achieving some sort of explainable AI while retaining the power of machine learning is a major challenge.[39] Companies that employ deep learning systems without addressing these issues carry huge reputational risk when things go wrong.

---

[37] See Michal Kosinski, David Stillwell, and Thore Graepel, "Private traits and attributes are predictable from digital records of human behavior", *PNAS* 110 (2013), pp. 5802-5.

[38] Plus convolutions, poolings, biases, and activation functions etc., as sketched in notes 18 and 21 above.

[39] This challenge is sometimes expressed as that of achieving interpretable machine learning by finding ways of transforming *black box models* into *glass box models*.

3.4 Risks of Over-Reliance

Inscrutability can also engender over-reliance, as we come to depend on a system whose mode of operation seems completely mysterious to us, and yet appears to be impressively expert at what it does – far faster (and typically more confident) than we could possibly aspire to be, yet also far more sophisticated in its processing and comprehensive in the data on which that processing is founded. This can undermine our autonomy as responsible individuals and lead to serious dangers as we neglect to develop or apply our own judgment and fail to realize when the system is getting things wrong. As noted, neural network models have long been considered more robust than classical AI alternatives in the context of noisy, ambiguous, or incomplete data. But to the contrary, it has recently become clear that deep learning models are commonly vulnerable to deliberately designed "adversarial" examples, whereby – for example – two images that differ imperceptibly (as far as a human is concerned) may be categorized by the system quite differently, and even very confidently. One notorious, and very worrying, illustration of this was revealed by researchers at New York University who discovered that adding a sticker to road signs could cause them to be misidentified (so that a stop sign, for example, was taken to be a mere speed limit sign).[40]

The dangers of over-reliance can be seriously exacerbated by the hype surrounding conspicuous developments in AI, as for example the recent excitement about *ChatGPT* and its supposed general intelligence. One such area of concern is software development, where the apparent ability of *ChatGPT* to perform impressively in programming exercises might tempt companies to rely on it for code production, and thus hugely economize on their software engineering teams. But as in routine chess openings (see section 2.6), the reason why *ChatGPT* does so well in those exercises is not that it rigorously analyzes some internal model of what is going on in the code that it generates, but rather, that its own implicit textual model is based on analysis of the many millions of programming examples that it has been given, and it accordingly generates code by mimicking, with what appear to be appropriate textual variations of the dominant patterns. Therefore, if you have a programming task that is very standard, or which deviates from a standard task in relatively common ways (e.g., when teachers have devised non-standard programming tasks for their students, such as sorting an array of numbers but with odds preceding evens), then *ChatGPT* might well come out with a correct solution. But if the task involves any nuances (or combinations of factors) that are less common or entirely novel, then there is a serious risk that you will end up with code that has the double disadvantage of looking very plausible – and quite possibly works well in most cases – while actually being incorrect, thus making the errors

especially hard to identify.[41] None of this is to deny that in expert hands, *ChatGPT* can be a useful programming assistant, providing quick and targeted search through the mountains of online programming resources to generate "first-draft" answers that are often very helpful. But the use of *ChatGPT* in programming by those who are unaware of its foibles, or insufficiently expert to check carefully what it produces, carries the risk of generating false confidence, both in the code that it generates and in the programmers who rely on it (whose lack of understanding could be masked by their success in turning out plausible code). Programmers may find themselves spending a considerable proportion of their time fixing "bugs," so the use of generative AI to generate code doesn't necessarily guarantee quality code or time savings and could expose a firm to financial risk if managed poorly.

---

[40] See Tianyu Gu, Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg, "Badnets: Evaluating Backdooring Attacks on Deep Neural Networks", *IEEE Access*, 2019.

[41] It is well known that current large language models are prone to so-called "hallucination", in which they generate textual information which is completely false (yet often plausible). Recent work by Ziwei Xu, Sajay Jain and Mohan Kankanhalli (https://arxiv.org/abs/2401.11817) demonstrates, as their title states, that "Hallucination is Inevitable: An Innate Limitation of Large Language Models" (though this does not preclude avoidance of such errors by use of other systems in combination with the LLM).

3.5 Risks to Individuals, Organizations, and Society

The development and deployment of artificial intelligence exposes individuals, organizations, and society to a wide spectrum of risks.

Individual risk: Risks posed to individuals vary widely and may originate with organizations or institutions with whom they interact or through their own behavior. Individuals routinely interact with institutions and organizations (e.g., banks, insurers, medical care providers) where decisions are made that have an impact on the individual. If AI systems contributing to that decision making are poorly designed or trained using inappropriate data, then one could suffer because of faulty, biased, or unfair decision making. In terms of how individuals interact with technology, automation bias, or the tendency of humans to over-rely on automated systems, can lead to complacency and reduced vigilance, which can potentially impact an individual in the form of reduced decision-making autonomy or a risk to safety and well-being. AI algorithms that draw on large amounts of information about individual behaviors, activities, and preferences create the potential for manipulation (e.g., by advertisers or interest groups) by allowing them to make predictions about future actions and influence our behavior. Inferences made from that information can also threaten individual privacy.

Organizational risk: Risks posed to organizations could come in a variety of forms. As cited previously, over-reliance on AI, and the false confidence that it engenders, can lead to bad decision making, customer dissatisfaction, and/or commercial loss. Depending on the impact of the over-reliance, an organization could also be exposed to reputational, regulatory, or legal risk. Reputational risk could arise as a result of AI practices that hurt individuals or groups (e.g., a poorly designed AI algorithm that lacks explainability, breaches privacy, or yields biased results). Although AI-specific regulations are still emerging, business practices that rely on AI are still subject to existing privacy laws and model governance regulations, so in cases where such practices hurt individuals or groups, organizations are potentially subject to regulatory risk and legal risk.

Societal risk: As with prior industrial and technological "revolutions," advancements in AI create the potential for risks at the societal level. These include job losses in fields where AI-driven applications take the place of human workers, a widening wealth gap between those who have the skills to work alongside AI and those who do not, and the exacerbation of global inequality between countries well positioned to take advantage of AI (i.e., wealthier, more industrialized countries) and those less well positioned. As with many technologies, AI can be leveraged by bad actors for ill. One such example is the creation and spread of misinformation (e.g., deep fakes) via generative AI to mislead or

influence public opinion. Finally, discussions of AI in recent years have often raised a fear of “existential” risk to humanity, especially in relation to concerns about superintelligence and the possibility that an AI system, such as an autonomous weapons system, will make decisions autonomously and against the interests, values, or priorities of its creators or humanity. It should be noted, however, that there is currently no consensus regarding the nature or extent of existential risk posed by AI.

Navigating the Remainder of this Course

Our exploration here has laid the groundwork for understanding the opportunities and challenges of AI. Yet, true power lies not just in abstract concepts, but in the ability to apply and govern these emerging technologies effectively. Therefore, the remainder of this course focuses on two key areas: mastering the AI toolbox itself and navigating the ethical, regulatory, and risk-based complexities where those tools are deployed.

In our **Tools and Techniques** module, we take a deeper dive into the core building blocks of AI. We categorize machine learning algorithms for supervised, unsupervised, and reinforcement learning tasks. You'll become fluent in data preparation techniques, transforming raw information into insights fit for modeling. We cover feature selection, visualization, and the nuanced art of balancing model accuracy with explainability – a vital skill for ensuring AI aligns with critical business decision-making.

Although mastering individual tools is vital, our **Risks and Risk Factors** module takes a broader perspective. We dissect the potential unintended consequences of AI, from the amplification of existing biases to new threats to autonomy and safety. You'll gain strategies for mitigating risks related to fairness, transparency, and the complex ways errors can translate into real-world harm. As AI applications evolve, we confront the reputational risks stemming from public mistrust and the importance of proactive AI governance for building a trustworthy organization in the face of potential missteps.

In the domain of **Ethical and Responsible AI**, we move beyond risk mitigation toward principled implementation. Here, we analyze frameworks from consequentialism, deontology, and virtue ethics, applying them to real-world scenarios with AI at their core. You'll gain a thorough understanding of the evolving legal landscape, including the landmark EU AI Act and emerging US regulatory initiatives. This equips you to be proactive as regulations evolve, not merely reactive – a key leadership quality in a field where the rules are still being shaped.

Governance will be a recurring theme – not as a bureaucratic burden, but a critical lever for successful implementation. In the **Governance** module, you'll gain a holistic picture of both data and model governance. You'll learn strategies like designing a robust data strategy, ensuring data quality, and establishing clear processes for model development, testing, and ongoing monitoring. These principles ensure your AI efforts yield reliable insights, not unexpected liabilities or operational weaknesses.

Finally, we explore a variety of **Case Studies, Practitioner Perspectives,** and other content to help bring these concepts to life. The case studies examine specific

applications in finance and risk management. The other content illustrates how machine learning transforms traditional risk modeling, while discussing practical strategies to ensure explainability and robustness, particularly in high-stakes, regulated environments.

Throughout this course, the focus is on application and action. Your goal is not to become a data scientist – but rather a business leader who can speak their language and bridge the gap between technical possibilities and real-world business needs. Mastering the interplay among AI, ethics, and risk management is critical to unlocking the transformative potential of this powerful technology in a sustainable, responsible, and ultimately profitable way.

AI and Risk – Introduction and Overview: Questions

The following questions are intended to help candidates understand the material. They are not actual RAI Exam questions.

1. What are the four most basic forms of machine learning?

2. What are some reasons why inscrutability of deep network machine learning is problematic?

3. What is the risk associated with overreliance on AI systems?

4. True/False: The fact that AI-specific regulations are still emerging results in organizations having little or no AI-related regulatory risk.

5. In the context of reinforcement learning, why might Lookahead be applied?

6. Differentiate between cluster analysis and dimension reduction.

7. What are the advantages of principle component analysis (PCA)?

8. True/False: There is currently broad consensus regarding existential risk posed to humanity by AI.

AI and Risk - Introduction and Overview: Answers

The following questions are intended to help candidates understand the material. They are not actual RAI Exam questions.

1. What are the four most basic forms of machine learning?

Answer

Reinforcement learning, supervised learning, unsupervised learning, and semi-supervised learning.

2. What are some reasons why inscrutability of deep network machine learning is problematic?

Answer

The presence of bias can be hidden and hard to eradicate; the risk to privacy, because the deep implicit entanglement of complex information within the learned network can easily hold clues to personal characteristics that we would prefer to keep secret; risks of manipulation, in both the commercial and political sphere, because those who are able to identify our personal characteristics and foibles may also be able to play on them through the now familiar phenomenon of targeted advertising.

3. What is the risk associated with overreliance on AI systems?

Answer

Overreliance on AI systems, also called automation bias, can lead to complacency and reduced vigilance, which in turn can undermine our autonomy as responsible individuals. This can create the potential for a dynamic in which one neglects to develop or apply one’s own judgment and fails to realize when the system is getting things wrong.

4. True/False: The fact that AI-specific regulations are still emerging results in organizations having little or no AI-related regulatory risk.

Answer

False. Although AI-specific regulations are still emerging, business practices that rely on AI are still subject to existing privacy laws and model governance regulations, so in cases where such practices hurt individuals or groups, organizations are potentially subject to regulatory risk and legal risk.

5. In the context of reinforcement learning, why might Lookahead be applied?

Answer

Lookahead becomes essential when our aim is to form a multistep plan to achieve some goal, within a context where there is a very large number of possible situations overall, but with a relatively constrained and predictable range of options in any particular situation.

6. Differentiate between cluster analysis and dimension reduction.

Answer

Both are unsupervised learning techniques. Cluster analysis is a learning technique in which groups of similar objects are automatically identified without needing a training set. Dimensionality reduction, on the other hand, is a technique in which the main "dimensions" of variation among the objects are identified, enabling their range and relationships (e.g., comparative closeness) to be more easily grasped.

7. What are the advantages of principle component analysis (PCA)?

Answer

PCA provides a powerful method of dimensionality reduction, whereby a large array of points in multidimensional space can be "reduced" to a lower dimensional space, while effectively representing the most significant measures of closeness and distance within that multidimensional space. For the same reason, it can highlight clusters of similar items to serve as a basis for future investigation. And even though it might often be hard to pin down exactly the basis of this similarity,

PCA does not reach the extreme level of inscrutability that we find with deep network learning, because here at least we can understand in general what the dimensions of variability represent, and explore their basis relatively straightforwardly.

8. True/False: There is currently broad consensus regarding existential risk posed to humanity by AI.

Answer

False. There is currently no consensus regarding the nature or extent of existential risk posed by AI.