Paul Taylor

Deep Learning

In March this year, over 10 days in Seoul, tens of millions of people watched on live internet feeds, as AlphaGo, a computer program, defeated Lee Seedol, the best player in the world at Go, the most intellectually demanding of board games. The game is still relatively unknown in the West but hugely popular in the East. It originated in China but developed into its current form in Japan, enjoying a long golden age from the 17th to the 19th century. Classic games from the period include the Blood Vomiting game in which three moves of great subtlety were allegedly revealed to Honinbo Jowa by a ghost, allowing him to defeat Intetsu Akaboshi, his rival's young protégé, who admitting defeat after four days of continuous play, knelt down and collapsed, to die of TB shortly afterwards. Another, the Ear Reddening game, turned on a move of such strength that it caused a discernable flow of blood to the outer ears of the master Inoue Genan Inseki. That move was, until March 13th this year, probably the most talked-about move in Go. The title probably now belongs to move 78 in game four of last month's match, a moment of almost inexplicable intuition which gave Lee Seedol a single victory in the five game series. The move has been christened the Touch of God and discussed not just by fans of Go but by all kinds of people with an interest in what differentiates human from artificial intelligence.

Deepmind, the London-based company behind AlphaGo, was acquired by Google in January 2014. The £400 million price tag seemed large at the time: the company was mainly famous for DQN, a program that played Atari video games from the 1980s. Mastering Space Invaders might not seem, on the face of it, much to boast about compared to beating a champion Go player, but it is the approach Deepmind has taken to both problems that is impressive. Traditional computer programming requires that knowledge or expertise be made

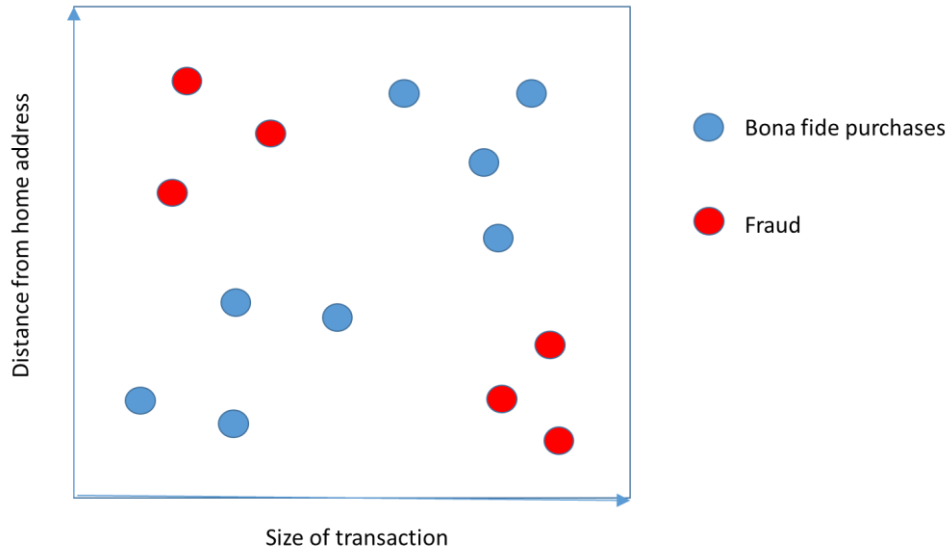
explicit; writing chess programs involves identifying and encoding the principles that underpin sound play. That isn't how Deepmind's software works. DQN doesn't know how to repel an invasion, it doesn't even know that the electronic signals it is processing depict aliens. They are just an array of pixels[i]. Deepmind searches the game data for correlations, which it interprets as features. It then learns how those features are affected by the choices it makes and uses what it learns to make choices that will, ultimately, maximise a desirable outcome. After just a few hours of training, the software is, if not unbeatable, then at least uncannily effective.

Demis Hassabis, the CEO of Deepmind, learned to play chess at the age of four. When he was 12 he used the winnings from an international tournament to buy a Sinclair ZX Spectrum computer. At 17 he wrote the software for *Theme Park*, a hugely successful simulation game. He worked in games for a further ten years before getting a formal education. He completed a PhD in cognitive neuroscience at UCL, then did research at Harvard and MIT. In 2011 he founded Deepmind with, he has said, a two-step plan to 'solve intelligence, and then use that to solve everything else'.

In 1965 the philosopher Hubert Dreyfus published a critique of artificial intelligence, later worked up into the book *What Computers Can't Do*, in which he argued that computers programmed to manipulate symbolic representations would never be able to complete tasks that require intelligence. His thesis was unpopular at the time, but by the turn of the century, decades of disappointment had led many to accept it. One difference between human intelligence and digital computation Dreyfus identified is that humans interpret information within a context that is not explicitly and exhaustively represented. Typically, someone reading such sentences as 'the girl caught the butterfly with spots,' or 'the girl caught the butterfly with a net,' doesn't register their ambiguity. It seems likely that one's intuitive interpretation in each case arises naturally from the association of connected ideas, not by

logical inference on the basis of known facts about the world. The idea that computers could be programmed to work in a similar way, learning how to interpret data without the programmer having to provide an explicit representation of the all rules and concepts that the interpretation might require**[1]**, has been around for almost as long as the kind of symbol-based AI **[2]** that Dreyfus wrote so scathingly about, but it has taken until now to really make it work. It is this kind of 'machine learning' that is behind the recent resurgence of interest in AI.

The best-known example of an early machine-learning was the Perceptron, built at Cornell in 1957 to simulate a human neuron. Neurons function as simple computational units: each receives multiple inputs and has only a single output – on or off. Given numerical data about examples of a particular phenomenon, the Perceptron could learn a rule and use it to sort further examples into sets. Imagine the Perceptron was trained using data on credit card transactions, some of which were known to be fraudulent and the rest of them above board. To begin with, each element of information (for example the size of the transaction, the time since the previous transaction, the location, any information about the vendor) fed to the Perceptron is given a random weight, and the machine classifies cases according to whether the total reaches an arbitrary threshold. Details of the training examples are entered, and whether the computer assigns an example to the right side of the threshold (fraud or not fraud) monitored, the weights given to the various inputs then gradually adjusted so as to improve the machine's success rate. **[3]**

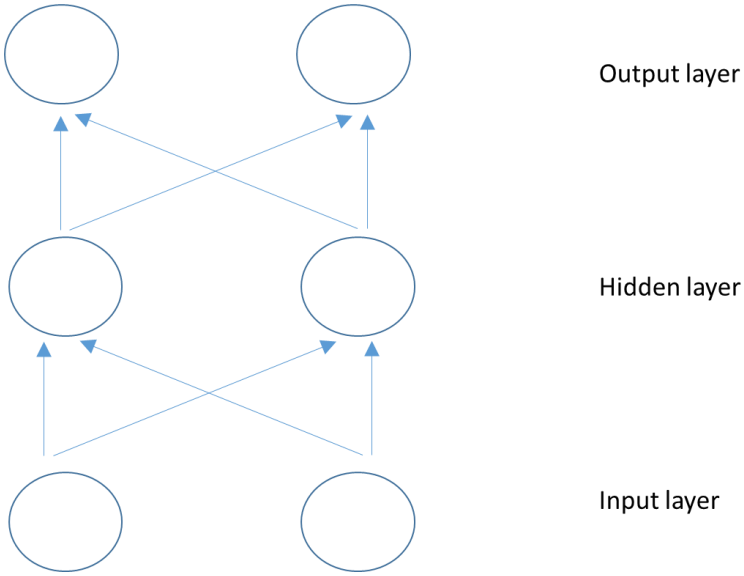


Given enough data and a well-structured problem the Perceptron could learn a rule that could be applied to new examples. Unfortunately even very simple problems turned out to have a structure that is too complex to be learned in this way. Imagine that only two things are known about credit card transactions: their amount, and where they take place (since both must be expressed as numbers, let's assume the location is expressed as the distance from the cardholder's home address). If fraud is found to occur only with large purchases or only with remote ones, the Perceptron can be trained to distinguish fraudulent from bona fide transactions. However, if fraud occurs in small remote purchases and also in large local purchases, the task of classification is too complex. This kind of system only works on problems that are 'linearly separable' and, as should be clear from Figure 1, no single straight line through the space will separate the fraud cases from the others.

Interest in the approach at first faded, but at the end of the 1970s, people worked out how to tackle more complex classification tasks using networks of artificial neurons arranged in layers, so that the outputs of one layer formed the inputs of the next. Consider the network in Figure 2. Imagine the two nodes in the input layer is used to store the size and location of each credit card transaction. If the left-hand node in the middle layer can be trained to detect

just the cases in the top left of figure one – a linearly separable problem - and the right-hand node can be trained to detect only the cases to the bottom right – which again is a linearly separable problem - the two inputs to the output node would measure the extent to which a case is a) small and distant, and b) large and local. Bona fide transactions will score low on both measures, fraud transactions will score highly on one or the other, so that the two classes can be divided by a straight line. The challenge is that the network has to identify the concepts to be captured in the hidden middle layer from information about how changing the weights on the final set of links affects the final classification of transactions as fraud or bona fide. The process works by computing a measure of how a change in the final set of weights changes the rate of errors in the classification and then propagating that measure backwards through the network.

For a while multi-layer networks were a hot topic, not least because people were excited by the explicit analogy with human perception, which depends on a network of cells that compute features in a hierarchy of increasing abstraction but, as before, early promise gave way to disappointment. The backwards propagation of errors seemed a hopelessly inefficient training algorithm if more than one or two layers separated the input and the output. Such shallow networks couldn't be programmed to complete challenging tasks in vision or speech recognition, and on simpler tasks they were outperformed by other approaches to machine learning.



The challenge in machine learning is not so much finding a rule that correctly classifies the training data, as finding *the* rule that is most likely to work for future examples. One

approach that would work for a linearly separable problem would be to divide the two sets using the straight line that maximises the distance between the line and the nearest point in each of the two sets. Finding that line is mathematically relatively straightforward. But as I've said, most interesting problems can't be separated by a straight line. A mathematically elegant solution is to project the data into a higher dimensional space where a simple separation can be found, by a process of iterative search. For the data in Figure 1, the search would be to find a mathematical function that takes the values of the x and y co-ordinates for each of the points and use them to derive a z co-ordinate so that the red points hovered at a greater height than the blue ones.

The representation of credit card transactions as points on a 2-D surface or in a 3-D space in this way is, of course, metaphorical. In reality each transaction is just a set of numbers, and in most problems there will be a lot more than three numbers to deal with. A 2-D space is divided by a line, a 3-D space by a plane. A space of more dimensions than that is divided by a hyperplane. A 'support vector machine', as these classifiers are known, identifies the hyperplane that optimally separates points in an n-dimensional space. Support vector machines dominated machine learning from the 1990s until very recently; they have the sought-after property, not shared by neural networks, that if the computation converges on a solution, it is guaranteed to be the best available one.

Imagine a classifier is to be trained using a hundred images: fifty of them each contain a different handwritten 'i' in shades of grey on a white background, and the other fifty contain examples of 'j's. If each image is 32 pixels high and 32 pixels wide then it can be represented as a single point in a 1024-dimensional space, where each dimension corresponds to a pixel, and the value on the dimension ranges from 0, which represents white, to 255, which represents black. The data for the set of images is completely represented as a hundred points in this 1024-D space. A support vector machine could attempt to find a hyperplane that

divides the space so that, ideally, all the points corresponding to the images of ‘i’s are on one side and all the ‘j’s on the other. However the hundred images will form a diffuse cloud taking up only a tiny fraction of the total space and will almost certainly be unhelpfully distributed within it. This is a common problem in machine learning: the feature space is only very sparsely populated by the data.

An alternative is to build a new feature space, a system of co-ordinates that is adapted to the data we are interested in. For example, the origin of the new system of co-ordinates could be placed at the centre of the cloud of points and a line drawn that passes through the origin and goes as close as possible to as many points as possible. A second line through the origin could be set at 90° to the first and again positioned as close to as many points as possible; and then a third, and so on until, say, ten dimensions have been defined. Each image can now be given a set of co-ordinates in the new ten-dimensional space. Each image is no longer represented by 1024 pixels but by a set of ten numbers that is both a much better characterisation of the data and a more parsimonious input to a support vector machine. Each of the ten numbers corresponds to a value for an abstract feature which has been derived by the computer from an analysis of the data as a whole. This abstract feature will correspond to some way in which the ‘i’s and ‘j’s vary, but it may or may not correspond to an intuitive human interpretation of the data.

From around 1990 to around 2010 most research in machine learning was focused on statistical techniques such as support vector machines and the attempt to derive feature spaces that made classification easier. As computers became more and more powerful and datasets became larger and larger, it became more practical to leave it to the computers to figure out the right feature space to use. This is what seems magical about software like Deepmind’s: computers are abstracting from experience something which can then be applied in reasoning about a problem. It seems natural to say that the computer has learned a concept.

In 2006 Netflix offered a prize of $1 million to anyone who could improve on the algorithm it used to generate recommendations for its customers. To give contestants something to work with, it released a database of 100,480,507 ratings that 480,189 users had given to 17,770 movies. The prize was awarded in 2009 to a team who used a blend of different algorithms, although most of their success seemed to be down to just two of the dozens of approaches used. **[4]**

In the first approach a large matrix is created in which each movie is a row and each user a column. In roughly 1 per cent of the cells there is a number between one and five which indicates the rating a particular user gave to a particular movie. The challenge is to use the data to predict if a given user would like a movie that they haven't yet seen. This corresponds to using the values in the filled cells to predict the rating that should go into each of the empty cells. The solution makes an assumption, that there are a smallish number, say 30, of features that determine whether or not a user likes a movie. It doesn't make any assumption about what the features are (a happy ending, a big budget, a strong female lead?) just about how many there are. The problem then reduces to identifying two much smaller matrices. One has a row for each movie and a column for each feature and records the extent to which a feature is present in a movie. The other has a row for each feature and a column for each user and records the extent to which a user has a preference for a feature **[5c]**. The product of these two matrices **(footnote:** The conventional approach to multiplying matrices allows a matrix with m rows and n columns to be combined with another having n rows and p columns to create a matrix with m rows and p columns.**)** will then generate a rating corresponding to each cell in the large matrix. The problem is that none of the values in either of the smaller matrices are known. The solution, as with other approaches to machine learning, is start with an initial guess **[5e]**, see how the generated predictions for filled cells

compare with the known ratings and then make repeated adjustments to minimise the average error.

The other algorithm that seemed particularly successful in this challenge also assumed that the required predictions could be generated from a small set of latent features, but used a variant of a neural net, known as a Restricted Boltzman Machine or an autoencoder, to derive the features from the data. A traditional neural net is trained on samples with a known classification until it learns a rule. An autoencoder is trained on samples of unclassified data until it learns to generate similar patterns of data. The Netflix autoencoder looks just like the neural network in Figure Two, but with many more nodes, it has an input node for each movie, a hidden node for each feature and an output node for each movie. Every movie is linked to every hidden node. As with all neural nets, each link is associated with a weight. The state of each hidden node is determined, for a given user's set of ratings, by multiplying the ratings by the weights and applying a threshold. The process is then run in reverse, applying the same weights to the states of the hidden nodes, and adding up the products and setting a threshold to determine the rating that is 'reconstructed' at the output node. The algorithm then adjusts the weights to minimise the difference between the original and reconstructed ratings. The weights of the links and the values at the hidden nodes can then be used to generate new ratings for each user.

Although Netflix awarded their prize with a blaze of publicity, the winning approach was never implemented. In part this was because Netflix had already begun to distribute movies via a streaming service. Customers were able to pick what they wanted to watch there and then rather than having to choose DVDs to watch a week or two later, and somehow this meant that they were less likely to pick the kinds of films that earned high ratings – no one wants to watch Schindler's List after putting the kids to bed on a Tuesday night – with the consequence that predicting ratings was no longer the best way to make

recommendations. There was the added difficulty that although Netflix believed it had anonymised the data it released, it had included information about films people had watched but not rated and also the dates on which they were watched. If you knew the dates on which you and your partner had watched two or three films together, that would typically be enough to pick out his or her unique column in the data, which might mean you could find out what he or she had watched without you. This became known as the 'Brokeback Mountain problem' and in 2009 Netflix was successfully sued for breach of privacy.

In 2006, about the time Netflix launched the competition, Geoffrey Hinton, one of a dwindling number of researchers then working on neural networks, realised that if the lower levels of a neural network could be programmed using autoencoders, the bottom of a deep neural network would learn a feature space that the top of the network could use to learn a classification. In 2009 two of his students using this approach published results for a speech recognition system that had, within a few years of development, outperformed competitors which had been refined for over 30 years. One student went on to work for Microsoft, the other for Google and by 2012 this work was the heart of the algorithm that allowed Android phones to respond reasonably reliably to spoken queries and commands. **[7]**

By then Google had begun to make a number of large investments in what was becoming known as 'deep learning'. It devoted some space in its massive computing infrastructure to build what was, until last year, the world's biggest artificial neural network. 'Inception', as it was called, was trained on a thousand machines running in parallel for three days. It analysed still images selected at random from ten million YouTube videos. Whereas earlier neural networks had been used to perform low-level image processing or speech recognition, the much taller stack of layers in this monster network made it possible to recognise human faces or (this tells us more about YouTube than it does about AI) cats' faces. If this network had been fed thousands of images labelled as 'contains cats' or 'doesn't

contain cats' and trained to work out the difference for itself by iteratively tweeking its 1.7 billion parameters **[8]** until it had found a classification rule, that would have been impressive enough, given the scale of the task involved in mapping from pixels to low-level image features and then to something as varied and complex as a cat's face. What Google actually achieved is much more extraordinary, and slightly chilling. The input images weren't labelled in any way: the network distilled the concept of 'cat face' out of the data unguided.

Last year Hinton, now a Google employee himself, gave a talk to the Royal Society at which he reviewed some of the history and spoke of new developments. Recurrent neural networks add the innovation that weighted links exist not just between nodes but between instances of the same node at successive steps in the computation. It is as if, instead of the network shown in Figure 2, there is a 3D stack of identical networks with links rising up from the page from each node to the node above. Each layer in the stack, however, doesn't represent a part of the network, but the state of the network at a point in time, so the bottom layer is the network at the start of training, the next layer is the network after the first cycle of training and so on. The point is that when a conventional neural network learns how to classify examples, be they images or sets of customer ratings, it doesn't matter in what order the training examples are processed. Recurrent networks, in contrast, are ideally suited to analysing data which is inherently sequential, data such as speech or language. Researchers at Google have for some years been programming recurrent neural networks to predict the next word in a sentence. Almost as a by-product this work creates a point in a high-dimensional feature space for each word. The features have no human interpretation, they are just the values of the hidden nodes in a network that was trained for a prediction task. But the researchers noticed that words with similar meanings had similar representations in the feature space. Even more astonishing, you could do a kind of arithmetic with the representations. Subtracting the features for 'uncle' from those for 'aunt' gives almost the

same answer as subtracting ‘king’ from ‘queen’, suggesting that this abstract, computer-derived space has a dimension that correlates with gender. One practical result of this work is an approach to machine translation that involves mapping between the feature representations of words in different languages. The process is still in its infancy and outperformed by more conventional methods, but it is getting better faster.

The extraordinary progress made in deep learning has led some to talk as if artificial intelligence is being solved. The solving of problems that until recently seemed insuperable gives the impression that the machines are acquiring capacities usually thought distinctively human. But although what happens in a large recurrent neural network does resemble what takes place in a brain more than more conventional software does, it remains the case that the similarity is limited. There is no close analogy between how neural networks are trained and what we know about how human learning takes place. It is too early to say whether scaling up networks like Inception will enable computers identify not only a cat’s face but also the general concept ‘cat’ or even more abstract ideas such as ‘two’ or ‘authenticity. And powerful though Google’s networks are, the features they derive from sequences of words are not built from the experience of human interaction in the way that our use of language is: it is unclear whether or not they will eventually be able to use language as humans do.

Ray Kurzweil has written about the Singularity, the idea that once computers are able to generate improvements to their own intelligence, the rate at which that intelligence improves will accelerate asymptotically. Nick Bostrom, the Oxford philosopher, wrote a 2014 bestseller, Superintelligence, examining the risks associated with uncontrolled artificial intelligence. Stephen Hawking has suggested that building machines more intelligent than we are could lead to the end of the human race. Elon Musk has aired similar anxieties. In truth, if there is something to be scared about here, it is the social consequences of the economic transformation that they might enable, that and the growing dominance of a small number of

corporations with access to the mammoth quantities of computing power and data that the technology requires.

There are some pretty terrifying estimates of the number of jobs that artificial intelligence could destroy. They might be overstated, optimists will argue that as with previous technological revolutions jobs will be created too. Richard Susskind, writing in the Future of the Professions, suggests that what will be destroyed is not so much jobs as tasks, the roles fulfilled by lawyers, doctors, teachers, architects and so on will evolve in the way that that of the airline pilot already has, with automation taking over the bits that human intellects struggle to complete safely or efficiently.

And the computers still haven't entirely outsmarted us. To pick the best move in a game, you need to consider all possible moves and all possible moves that might follow from that move, until the end of the game. In Go there might be 250 moves to consider and after each move branching sequences might unfold for 150 further moves, so the number of possible ways a game can unfurl is around 250^{150} and an exhaustive search through them is infeasible, even for Google. There are two ways to limit the search. One is to have a *policy*, which restricts the search to the moves that are most plausible, the other is to understand the *value* of a position, so there is need to search through the possible further moves from that position. AlphaGo contains two 13 layer neural networks, a policy network which computes a probability for each possible move, at any given state of play, and a value network which computes the probability of winning from that position. Whereas DQN learned to play Atari from scratch, the AlphaGo policy network starts with a database of 30,000,000 actual games stored on a server that allows people to play games with each other over the Internet. Training the policy network on this database allowed it to predict with 57% accuracy the move that the humans – all amateurs - made. That doesn't sound a great basis from which to improve on human play, but it is enough, and in fact the engineers say that improvements at

this stage in the process have huge consequences later on. To get from a passable prediction of human play to a policy that would beat an expert, the system was then programmed to play out games in which the current version of the policy network competed with an adversary selected at random from earlier versions and used the learning algorithms developed in DQN to adjust the weights in the networks to obtain more successful policies. The final stage in the process used another dataset of 30 million positions from games and again the software played out games from these positions, playing against itself, and adjusting weights to improve its assessment of the value of positions.

The whole thing is an incredible engineering achievement. The competition version ran over 48 CPUs with extra processing power from specialist GPU chips that are optimised for parallel computations. AlphaGo won the first three games, and therefore the match. One hour thirteen minutes, with barely 30 minutes time left on his clock Lee Sedol, playing white, thinks for 16 minutes before making an aggressive move into black's territory, on the left edge of the centre. Two or three moves follow fairly quickly, and it is clear that white will need two or three forcing moves if he is going make the attack count and it isn't obvious, at least to the commentary team, where they are coming from. Lee Seedol then takes a terrifying six minutes before placing a white stone in between two black stones at the right side of the centre. Afterwards the AlphaGo team checked and found the policy network had rated the chances of an opponent making this move at 1 in 10,000. Michael Redmond, a 9 dan professional player commentating on the game, admitted he hadn't seen it either, but he recognised its significance immediately: if black didn't find a response the needed forcing moves would be there. AlphaGo, however, didn't seem to realise what was happening. This wasn't something it had encountered in the amateur play on which the policy network was trained, nor had it emerged in the millions and millions of games it had played with itself. In the post-match press conference Lee Seedol was complimented on the move and asked what

he had been thinking when he played it. His response was that it was the only move he had been able to see.

[i] I don't like 'just zeroes and ones', everything in a computer is always just zeroes and ones.