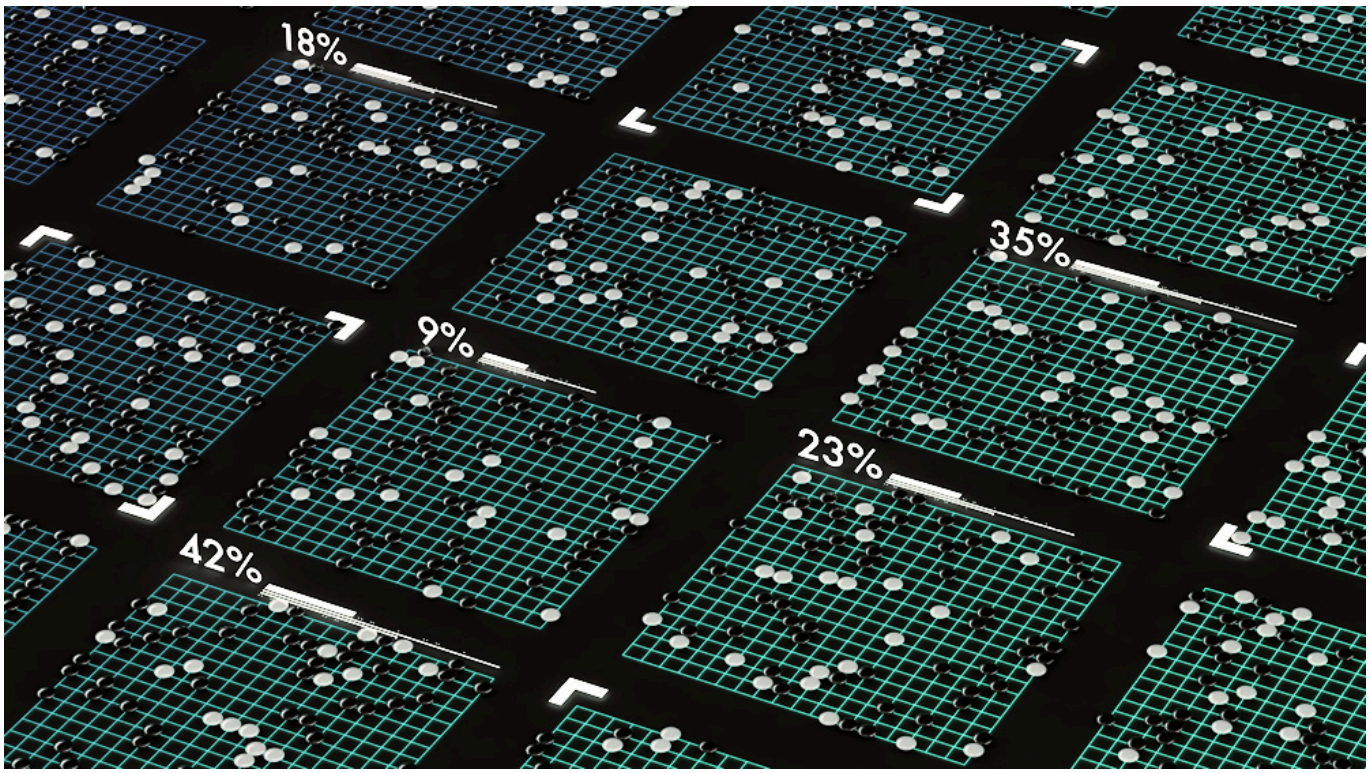
Quantamagazine

Is AlphaGo Really Such a Big Deal?

The Go-playing program captures elements of human intuition, an advance that promises far-reaching consequences.

By Michael Nielsen



In 1997, IBM's Deep Blue system defeated the world chess champion, Garry Kasparov. At the time, the victory was widely described as a milestone in artificial intelligence. But Deep Blue's technology turned out to be useful for chess and not much else. Computer science did not undergo a revolution.

Will AlphaGo, the Go-playing system that recently defeated one of the strongest Go players in history, be any different?

Quantized

A monthly column in which top researchers explore the process of discovery. This month's columnist, Michael Nielsen, is a computer scientist and author of three books.

I believe the answer is yes, but not for the reasons you may have heard. Many articles proffer expert testimony that Go is harder than chess, making this victory more impressive. Or they say that we

didn’t expect computers to win at Go for another 10 years, so this is a bigger breakthrough. Some articles offer the (correct!) observation that there are more potential positions in Go than in chess, but they don’t explain why this should cause more difficulty for computers than for humans.

In other words, these arguments don’t address the core question: Will the technical advances that led to AlphaGo’s success have broader implications? To answer this question, we must first understand the ways in which the advances that led to AlphaGo are qualitatively different and more important than those that led to Deep Blue.

In chess, beginning players are taught a notion of a chess piece’s value. In one system, a knight or bishop is worth three pawns. A rook, which has greater range of movement, is worth five pawns. And the queen, which has the greatest range of all, is worth nine pawns. A king has infinite value, since losing it means losing the game.

You can use these values to assess potential moves. Give up a bishop to take your opponent’s rook? That’s usually a good idea. Give up a knight and a bishop in exchange for a rook? Not such a good idea.

The notion of value is crucial in computer chess. Most computer chess programs search through millions or billions of combinations of moves and countermoves. The goal is for the program to find a sequence of moves that maximizes the final value of the program’s board position, no matter what sequence of moves is played by the opponent.


Michael Nielsen

Early chess programs evaluated board positions using simple notions like “one bishop equals three pawns.” But later programs used more detailed chess knowledge. Deep Blue, for example, combined more than 8,000 different factors in the function it used to evaluate board positions. Deep Blue didn’t just say that one rook equals five pawns. If a pawn of the same color is ahead of the rook, the pawn will restrict the rook’s range of movement, thus making the rook a little less valuable. If, however, the pawn is “levered,” meaning that it can move out of the rook’s way by capturing an enemy pawn, Deep Blue considers the pawn semitransparent and doesn’t reduce the rook’s value as much.

Ideas like this depend on detailed knowledge of chess and were crucial to Deep Blue’s success. According to the technical paper written by the Deep Blue team, this notion of a semitransparent levered pawn was crucial to Deep Blue’s play in the second game against Kasparov.

Ultimately, the Deep Blue developers used two main ideas. The first was to build a function that incorporated lots of detailed chess knowledge to evaluate any given board position. The second was to use immense computing power to evaluate lots of possible positions, picking out the move that would force the best possible final board position.

What happens if you apply this strategy to Go?

It turns out that you will run into a difficult problem when you try. The problem lies in figuring out how to evaluate board positions. Top Go players use a lot of intuition in judging how good a particular board position is. They will, for instance, make vague-sounding statements about a board position having "good shape." And it's not immediately clear how to express this intuition in simple, well-defined systems like the valuation of chess pieces.

Now you might think it's just a question of working hard and coming up with a good way of evaluating board positions. Unfortunately, even after decades of attempts to do this using conventional approaches, there was still no obvious way to apply the search strategy that was so successful for chess, and Go programs remained disappointing. This began to change in 2006, with the introduction of so-called Monte Carlo tree search algorithms, which tried a new approach to evaluation based on a clever way of randomly simulating games. But Go programs still fell far short of human players in ability. It seemed as though a strong intuitive sense of board position was essential to success.

What's new and important about AlphaGo is that its developers have figured out a way of bottling something very like that intuitive sense.

To explain how it works, let me describe the AlphaGo system, as outlined in the paper the AlphaGo team published in January. (The details of the system were somewhat improved for AlphaGo's match against Lee Sedol, but the broad governing principles remain the same.)

To begin, AlphaGo took 150,000 games played by good human players and used an artificial neural network to find patterns in those games. In particular, it learned to predict with high probability what move a human player would take in any given position. AlphaGo's designers then improved the neural network by repeatedly playing it against earlier versions of itself, adjusting the network so it gradually improved its chance of winning.

How does this neural network — known as the policy network — learn to predict good moves?

Broadly speaking, a neural network is a very complicated mathematical model, with millions of parameters that can be adjusted to change the model's behavior. When I say the network "learned," what I mean is that the computer kept making tiny adjustments to the parameters in the model, trying to find a way to make corresponding tiny improvements in its play. In the first stage of learning, the network tried to increase the probability of making the same move as the human players. In the second stage, it tried to increase the probability of winning a game in self-play. This sounds like a crazy strategy — repeatedly making tiny tweaks to some enormously complicated function — but if you do this for long enough, with enough computing power, the network gets pretty good. And here's the strange thing: It gets good for reasons no one really understands, since the improvements are a consequence of billions of tiny adjustments made automatically.

After these two training stages, the policy network could play a decent game of Go, at the same level as a human amateur. But it was still a long way from professional quality. In a sense, it was a way of playing Go without searching through future lines of play and estimating the value of the resulting board positions. To improve beyond the amateur level, AlphaGo needed a way of estimating the

value of those positions.

To get over this hurdle, the developers' core idea was for AlphaGo to play the policy network against itself, to get an estimate of how likely a given board position was to be a winning one. That probability of a win provided a rough valuation of the position. (In practice, AlphaGo used a slightly more complex variation of this idea.) Then, AlphaGo combined this approach to valuation with a search through many possible lines of play, biasing its search toward lines of play the policy network thought were likely. It then picked the move that forced the highest effective board valuation.

We can see from this that AlphaGo didn't start out with a valuation system based on lots of detailed knowledge of Go, the way Deep Blue did for chess. Instead, by analyzing thousands of prior games and engaging in a lot of self-play, AlphaGo created a policy network through billions of tiny adjustments, each intended to make just a tiny incremental improvement. That, in turn, helped AlphaGo build a valuation system that captures something very similar to a good Go player's intuition about the value of different board positions.

In this way, AlphaGo is much more radical than Deep Blue. Since the earliest days of computing, computers have been used to search out ways of optimizing known functions. Deep Blue's approach was just that: a search aimed at optimizing a function whose form, while complex, mostly expressed existing chess knowledge. It was clever about how it did this search, but it wasn't that different from many programs written in the 1960s.

AlphaGo also uses the search-and-optimization idea, although it is somewhat cleverer about how it does the search. But what is new and unusual is the prior stage, in which it uses a neural network to learn a function that helps capture some sense of good board position. It was by combining those two stages that AlphaGo became able to play at such a high level.

This ability to replicate intuitive pattern recognition is a big deal. It's also part of a broader trend. In an earlier paper, the same organization that built AlphaGo — Google DeepMind — built a neural network that learned to play 49 classic Atari 2600 video games, in many cases reaching a level that human experts couldn't match. The conservative approach to solving this problem with a computer would be in the style of Deep Blue: A human programmer would analyze each game and figure out detailed control strategies for playing it.

By contrast, DeepMind's neural network simply explored lots of ways of playing. Initially, it was terrible, flailing around wildly, rather like a human newcomer. But occasionally the network would accidentally do clever things. It learned to recognize good patterns of play — in other words, patterns leading to higher scores — in a manner not unlike the way AlphaGo learned good board position. And when that happened, the network would reinforce the behavior, gradually improving its ability to play.

This ability of neural networks to bottle intuition and pattern recognition is being used in other contexts. In 2015, Leon Gatys, Alexander Ecker and Matthias Bethge posted a paper to the scientific preprint site arxiv.org describing a way for a neural network to learn artistic styles and then to apply those styles to other images. The idea was very simple: The network was exposed to a very large number of images and acquired an ability to recognize images with similar styles. It could then apply that style information to new images. For example, the right-hand image below illustrates what happens when you transfer the style of Vincent van Gogh (center) to a photograph of the Eiffel tower (left).

It's not great art, but it's still a remarkable example of using a neural network to capture an intuition and apply it elsewhere.

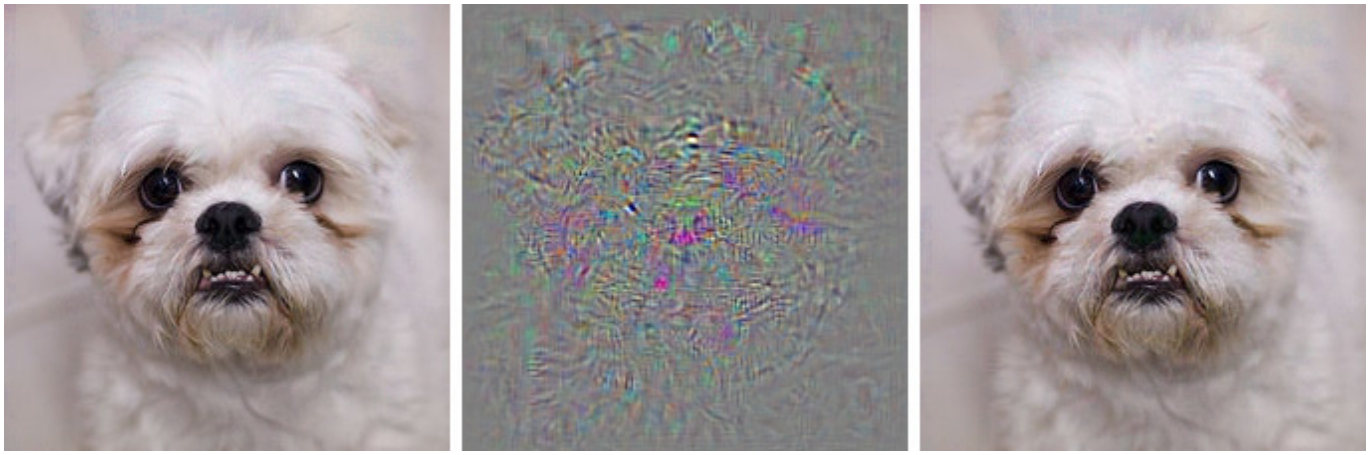
Over the past few years, neural networks have been used to capture intuition and recognize patterns across many domains. Many of the projects employing these networks have been visual in nature, involving tasks such as recognizing artistic style or developing good video-game strategy. But there are also striking examples of networks simulating intuition in very different domains, including audio and natural language.

Because of this versatility, I see AlphaGo not as a revolutionary breakthrough in itself, but rather as the leading edge of an extremely important development: the ability to build systems that can capture intuition and learn to recognize patterns. Computer scientists have attempted to do this for decades, without making much progress. But now, the success of neural networks has the potential to greatly expand the range of problems we can use computers to attack.

It's tempting at this point to cheer wildly, and to declare that general artificial intelligence must be just a few years away. After all, suppose you divide up ways of thinking into logical thought of the type we already know computers are good at, and "intuition." If we view AlphaGo and similar systems as proving that computers can now simulate intuition, it seems as though all bases are covered: Computers can now perform both logic and intuition. Surely general artificial intelligence must be just around the corner!

But there's a rhetorical fallacy here: We've lumped together many different mental activities as "intuition." Just because neural networks can do a good job of capturing some specific types of intuition, that doesn't mean they can do as good a job with other types. Maybe neural networks will be no good at all at some tasks we currently think of as requiring intuition.

In actual fact, our existing understanding of neural networks is very poor in important ways. For example, a 2014 paper described certain "adversarial examples" which can be used to fool neural networks. The authors began their work with a neural network that was extremely good at recognizing images. It seemed like a classic triumph of using neural networks to capture pattern-recognition ability. But what they showed is that it's possible to fool the network by changing images in tiny ways. For instance, with the images below, the network classified the image on the left correctly, but when researchers added to it the tiny perturbations seen in the center image, the network misclassified the apparently indistinguishable resulting image on the right.



Another limitation of existing systems is that they often require many human examples to learn from. For instance, AlphaGo learned from 150,000 human games. That's a lot of games! By contrast, human beings can learn a great deal from far fewer games. Similarly, networks that recognize and manipulate images are typically trained on millions of example images, each annotated with information about the image type. And so an important challenge is to make the systems better at learning from smaller human-supplied data sets, and with less ancillary information.

With that said, systems like AlphaGo are genuinely exciting. We have learned to use computer systems to reproduce at least some forms of human intuition. Now we've got so many wonderful challenges ahead: to expand the range of intuition types we can represent, to make the systems stable, to understand why and how they work, and to learn better ways to combine them with the existing strengths of computer systems. Might we soon learn to capture some of the intuitive judgment that goes into writing mathematical proofs, or into writing stories or good explanations? It's a tremendously promising time for artificial intelligence.

This article was reprinted on TheAtlantic.com.