How to Win at Deep Learning

What happens when you increase the number of layers in an artificial neural network?

By Pradeep Mutalik

"Deep learning" is the new buzzword in the field of artificial intelligence. As Natalie Wolchover reported in a recent *Quanta Magazine* article, "'deep neural networks' have learned to converse, drive cars, beat video games and Go champions, dream, paint pictures and help make scientific discoveries." With such successes, one would expect deep learning to be a revolutionary new technique. But one would be quite wrong. The basis of deep learning stretches back more than half a century to the dawn of AI and the creation of both artificial neural networks having layers of connected neuronlike units and the "back propagation algorithm" — a technique of applying error corrections to the strengths of the connections between neurons on different layers. Over the decades, the popularity of these two innovations has fluctuated in tandem, in response not just to advances and failures, but also to support or disparagement from major figures in the field. Back propagation was invented in the 1960s, around the same time that Frank Rosenblatt's "perceptron" learning algorithm called attention to the promise of artificial neural networks. Back propagation was first applied to these networks in the 1970s, but the field suffered after Marvin Minsky and Seymour Papert's criticism of one-layer perceptrons. It made a comeback in the 1980s and 1990s after David Rumelhart, Geoffrey Hinton and Ronald Williams once again combined the two ideas, then lost favor in the 2000s when it fell short of expectations. Finally, deep learning began conquering the world in the 2010s with the string of successes described above.

[puzzles]

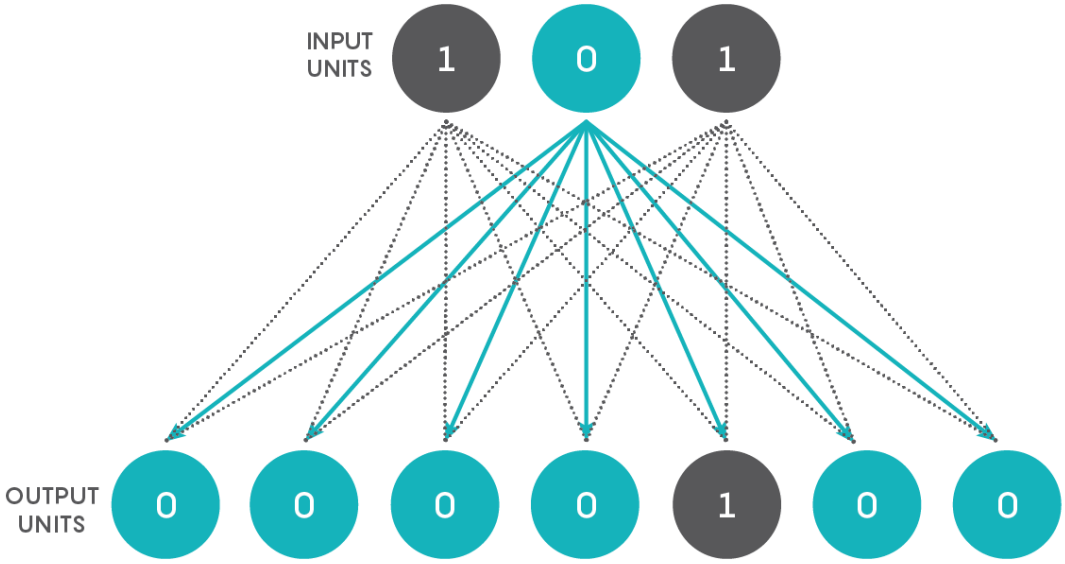
What changed? Only brute computing power, which made it possible for back-propagation-using artificial neural networks to have far more layers than before (hence the "deep" in "deep learning"). This, in turn, allowed deep learning machines to train on massive amounts of data. It also allowed networks to be trained on a layer by layer basis, using a procedure first suggested by Hinton.

Can merely increasing the number of layers in a network produce such a big qualitative difference? Let's see if we can demonstrate it as we explore a simple neural network in this month's puzzle questions.

Problem 1

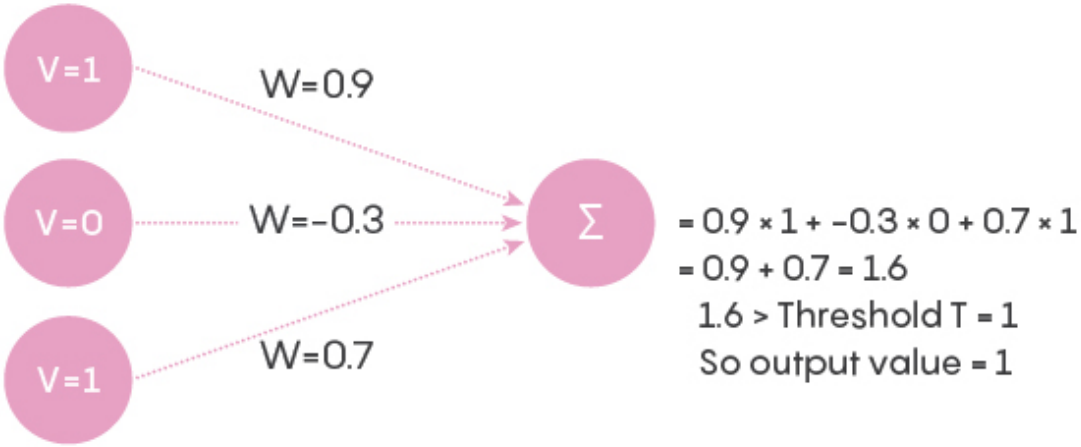
> We're going to create a simple network that converts binary numbers to decimal numbers. Imagine a network with just two layers — an input layer consisting of three units and an output layer with seven units. Each unit in the first layer connects to each

unit in the second, as shown in the figure below.



Olena Shmahalo/Quanta Magazine; source: Pradeep Mutalik

As you can see, there are 21 connections. Every unit in the input layer, at a given point in time, is either off or on, having a value of 0 or 1. Every connection from the input layer to the output layer has an associated weight that, in artificial neural networks, is a real number between 0 and 1. Just to be contrary, and to make the network a touch more similar to an actual neural network, let us allow the weight to be a real number between –1 and 1 (the negative sign signifying an inhibitory neuron). The product of the input's value and the connection's weight is conveyed to an output unit as shown below. The output unit adds up all the numbers it gets from its connections to obtain a single number, as shown in the figure below using arbitrary input values and connection weights for a single output unit. Based on this number, the output number decides what its state is going to be. If it is more than a certain threshold, the unit's value becomes 1, and if not, then its value becomes 0. We can call a unit that has a value of 1 a "firing" or "activated" unit and a unit with a value of 0 a "quiescent" unit.



Our three input units from top to bottom can have the values 001, 010, 011, 100, 101, 110 or 111, which readers may recognize as the binary numbers from 1 through 7.

Now the question: Is it possible to adjust the connection weights and the thresholds of the seven output units such that every binary number input results in the firing of only one appropriate output unit, with all the others being quiescent? The position of the activated output unit should reflect the binary input value. Thus, if the original input is 001, the leftmost output unit (or bottommost when viewed on a phone, as shown in the top figure) alone should fire, whereas if the original input is 101, the output unit that is fifth from the left (or from the bottom on a phone) alone should fire, and so on.

If you think the above result is not possible, try to adjust the weights to get as close as you can. Can you think of a simple adjustment, using additional connections but not additional units, that can improve your answer?

Problem 2

Now let’s bring in the learning aspect. Assume that the network is in an initial state where every connection weight is 0.5 and every output threshold is 1. The network is then presented with the entire set of all seven input values in serial order (10 times over, if required) and allowed to adjust its weights and thresholds based on the error difference between the observed value and the desired value. As in Problem 1, the connection weights must remain between –1 and 1. Can you create a global, general purpose learning rule that adjusts the connections and thresholds such that the optimum condition of Problem 1 can be reached or approached? For example, a learning rule could say something like “increase the connection weight by 0.2 within the permitted limits, and decrease its threshold by 0.1, if the output unit remains quiescent in a situation where it should have fired.” You can be as creative as you like with the rule, provided the specified limits are followed. Note that the rule must be the same for all the units in a given layer and must be general purpose: It should allow the network to perform better for different sets of inputs and outputs as well.

Problem 3

> Finally, let's add an extra layer of five units between the input and output layers. Analyzing the resulting three-layer network will probably require computer help, such as using a spreadsheet or a simple simulation program. Following the same rules as above, with extra rules for the hidden layer, how does the three-layer network perform on Problems 1 and 2 compared to the two-layer network?

That's it for now. Happy puzzling. Hopefully, this exercise will slightly alter the connection weights and thresholds of some of your neurons in constructive ways!

Editor's note: The reader who submits the most interesting, creative or insightful solution (as judged by the columnist) in the comments section will receive a Quanta Magazine *T-shirt. (**Update**: The solution has been published here.) And if you'd like to suggest a favorite puzzle for a future Insights column, submit it as a comment below, clearly marked "NEW PUZZLE SUGGESTION." (It will not appear online, so solutions to the puzzle above should be submitted separately.)*

Note that we may hold comments for the first day or two to allow for independent contributions by readers.