Rebooting Computers as Learning Machines[1]

Erik P. DeBenedictis, Sandia National Laboratories

Artificial neural networks could become the technological driver that replaces Moore's law, boosting computers' utility through a process akin to automatic programming—although physics and computer architecture would also factor in.

The slowdown and eventual end of Moore's law will limit the IT sector's growth. Moore's law relies on the shrinking of device size to drive performance, but semiconductor device scaling will soon hit its physical limit. Thus, it's urgent that we explore new computational approaches that circumvent physics' roadblocks.

Artificial neural networks (ANNs) have had some very public computational successes recently. Yet understanding how these networks could re-enable the growth of general-purpose computing requires a complex sequence of ideas. Could an ANN-based learning become the "new Moore's law" scaling driver?

Background on Neural Networks

Google's AlphaGo's recent defeat of Lee Sedol, the top-ranked human Go player, was a breakthrough for ANNs [1]. The computational method combined ANNs that evaluated game board positions with a conventionally coded tree search. Both aspects were computationally significant, with the system comprising 1,920 CPUs and 280 GPUs. However, subsequent community discussion in articles and blogs questioned whether an ANN alone could learn to play equally well. The answer is at the heart of whether and how ANNs will play a role in the next generation of computing.

We can explain the ANN used in many of the recent breakthroughs in terms of current computers—with the ANN assuming the role of both computer and programmer. The simplified diagram in Figure 1 shows a diagnosis of cancer type based on patient symptoms, and highlights the parallels between natural and artificial neural networks.

Information flows from left to right, starting with symptom information in nerve bodies, which are represented by the circles on the left. This information then moves through the axons, synapses, and dendrites, which are represented by lines whose weights modulate the information's strength. One of the three intermediate nerve body circles activates when the sum of the modulated information exceeds a threshold. Finally, the intermediate information flows to two output neurons that represent a diagnosis of either benign or malignant cancer. The system's knowledge of how to solve the problem is in the strengths of the neural network's synapses or weights.

[1] Sandia National Laboratories approved for unlimited unclassified release SAND2017-8677 J
Published as DeBenedictis, Erik P. "Rebooting Computers as Learning Machines." *Computer* 49.6 (2016): 84-87.
DOI: 10.1109/MC.2016.156

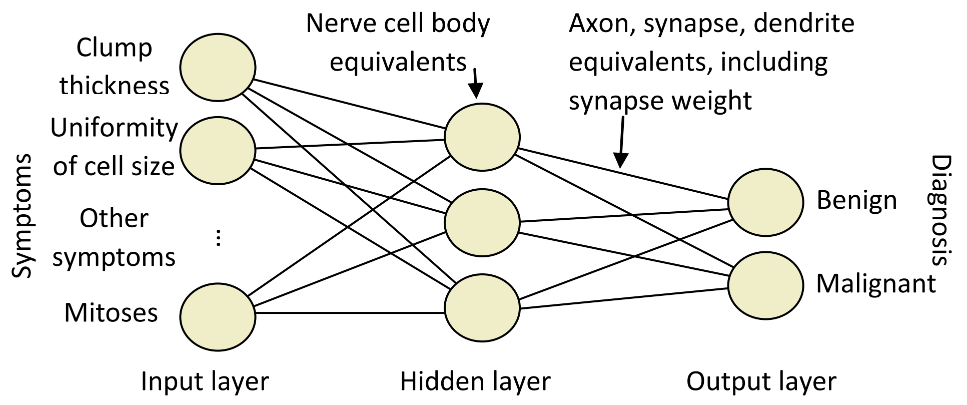

Fig 1. Medical diagnosis example using perceptron neural model

By applying the threshold gate interpretation of neural networks, we can geometrically alter the network in Figure 1 to produce the electronic circuit in Figure 2 [2]. The knowledge is now in the interconnections between Boolean gates. The information produced by Figure 1's nerve body circles becomes the output voltage of threshold gates, drawn as AND or NAND gates in Figure 2. The NAND gate inputs would need to be analog voltages (more on this later). The mat of crossing wires in Figure 1 has been redrawn in Figure 2 as a crossbar where signals applied to one dimension of the array are routed to the other dimension after having their amplitudes multiplied by the synapse or weight at the row–column intersections. Instead of the summation implicit in the wires, the logic-gate notation sends each input signal individually to the gate, which then sums the signals. The gate produces an output when the sum exceeds a threshold.

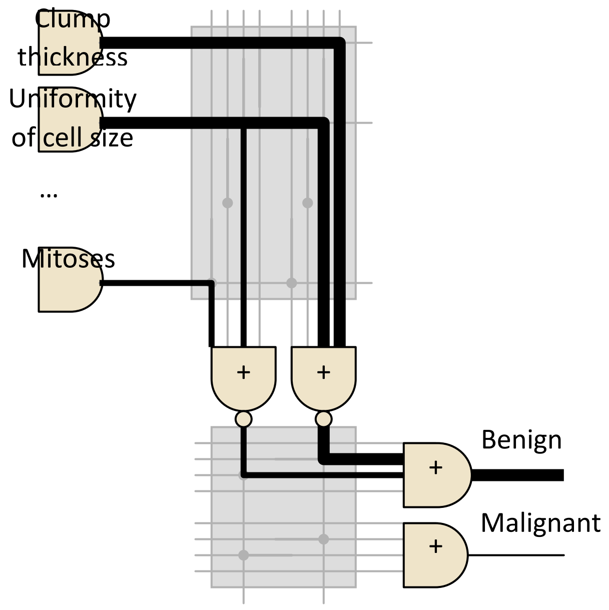

Fig. 2. Neural network drawn as a circuit

Figure 2 extends Boolean logic to a form that can learn. If the signals and weights are limited to 0s and 1s, the weights in Figure 2 become the wiring pattern of Boolean logic. A weight of 1 creates a full-strength connection that can be viewed as a wire between an output and an input to a gate. A weight of 0 indicates no wire. If a threshold gate sums k inputs whose values are either 0 or 1 and compares the sum to a threshold of k-1/2, the gate will compute the AND function. Inverting the output switches a gate to the NAND function.

A neural network figures out the weights during learning. Weights are analog values that correspond to strong versus “tentative” wires. Different ANN approaches learn differently, but for the purposes of this article, a tentative wire passes a signal with probability p, $0 \leq p \leq 1$, where p is the weight. A learning phase such as back-propagation adjusts weights during learning to make the neural network’s output better correspond to training or observational data.

In the latest ANN developments, GPUs have emulated neural networks by using floating-point numbers in lieu of the analog biological values. Thus, learning becomes a gradient descent algorithm. ANN operation—known as performance—becomes a vector-matrix multiply for each layer.

The behavior demonstrated in Figure 2 is equivalent to an expression in a computer language. Although Figure 2 is based on an already simplified tutorial, the gradient descent actually creates the following line of source code. (Note that I’ve abbreviated the medical diagnosis example from
http://neuroph.sourceforge.net/tutorials/PredictingBreastCancer/PredictingBreastCancer.html several times, so the expression no longer represents the problem.)

Benign = !(Uniformity_of_cell_size ^ Mitosis) ^ !(Clump_thickness ^ Uniformity_of_cell_size)

The connection above between ANNs and expressions enables a rough yet quantitative comparison with software projects. Figure 2 has input and output layers and one intermediate layer of three neurons. In contrast, recent ANNs might have 6–50 intermediate layers, each comprising a repetition of hundreds of sets containing hundreds of neurons. Published papers have reported that several of the larger ANNs have billions of variables or weights [1] [3]. Let’s suppose that each weight in Figure 2 creates a term in the behavioral expression and that a line of source code has 10 terms. One of these billion-variable ANNs would be equivalent to 100 million lines of expensive and time-consuming human-generated source code—within the size range of an OS distribution [4].

Training ANNs takes a long time in absolute terms, but the duration is stunningly short compared with the only other way to create the same result—human programming. For example, the ANNs that David Silver [1], Quoc Le [3], and their respective colleagues described needed a few days of supercomputer cluster runtime to learn the values of billions of weights. If we take the $1 billion over a few years that’s estimated to be required for developing an OS and divide it by $100,000—representing the energy and depreciation of a few days’ use of a supercomputer cluster—ANNs yield a cost advantage of 10,000× and a speed advantage of 300×.

Similar to Moore's law, these factors could lead to a new exponential improvement path. However, this path would be based on a qualitatively different improvement type. In lieu of more energy-efficient electronics powering current software, ANNs would improve computers through increasing diversity and sophistication in new applications.

The ANN's internal computer language

ANNs invent computational primitives that are eerily familiar to us because our brains do similar things subconsciously. But today's ANNs can only learn a limited range of behaviors.

ANNs discover template-like computational primitives as they learn to classify input data with those templates. The ANN starts out like a newborn child who's never seen anything. As the ANN experiences inputs, the first templates become edges and lines—the visual primitives of images. Learning advances to templates with higher-level structure, such as faces or other objects. Although ANNs discover patterns during learning, their task is to classify inputs, not report these patterns. However, researchers can run what are basically diagnostic algorithms on ANNs to view the templates. For example, Le and his colleagues ran an ANN in reverse to identify the most frequent input image for the largest output class [3]. Based on training an ANN with Internet images, this principal template revealed itself as the "spooky face" in Figure 3a [3]. This template makes a lot of sense to us because—as owners and operators of a human brain—we're intuitively aware of how we perceive images.

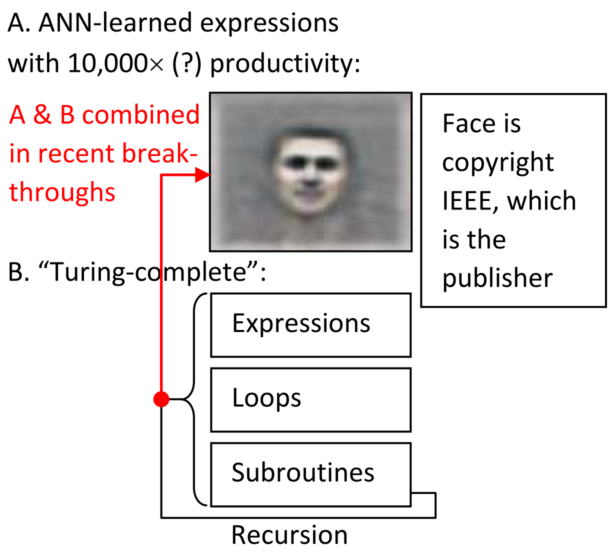
A. ANN-learned expressions with 10,000× (?) productivity:
A & B combined in recent break-throughs
Face is copyright IEEE, which is the publisher
B. "Turing-complete":
Expressions
Loops
Subroutines
Recursion

Fig. 3. ANN, conventional, and hybrid computing

Turing's mathematical theory of computability is the basis of today's programming languages, which can express everything that's computable as a collection of recursive functions. A computer language for these functions needs primitives such as expressions, loops, and subroutines that can call each other recursively. Figure 3b shows a schematic of the traditional elements of a general-purpose, or Turing-complete, language.

The ANN structure shown in Figures 1 and 2 has a fundamental leftto-right information flow that's insufficient to produce the loops and subroutines in Figure 3b. It's easy to imagine extending Figure 2 into a deep, wide Boolean logic network, but a loop would require the ANN to wrap around itself. A recursive subroutine would also need a way to store arguments or local variables during recursion. Wrapping the output of the networks in Figures 1 and 2 back onto their inputs would indeed let the network perform like a general-purpose computer, yet these nonlinear structures would need different learning algorithms. Relevant learning algorithms such as recursive and recurrent networks, liquid state machines, and genetic algorithms have been successful, although not at large scale.

A New Human-Computer Partnership

Recent ANN results combine Figures 3a and 3b in a way that might demonstrate a new human–machine partnership. An expert human transforms a problem statement into a solution plan at the level of the traditional expressions, loops, and subroutines combined with new ANN modules—in essence, a structure using the four parts of Figures 3a and 3b as building blocks. The human also devises a training regimen so the ANN portion can learn vast numbers of expression templates with a huge increase in productivity compared with human programming. The previous steps use human intelligence to create the diverse variety of structures needed to produce any computable function. The ANN's training regimen then automatically programs many more lines of code than the human, yet it's only filling in the details of the program structure that was created by the human and that the ANN lacks the intelligence to change—atleast so far.

Diverse applications

A few ANNs have been used in applications originally developed for computers, although the majority just duplicatehuman behaviors. Some work has even augmented supercomputers with neural networks, which is significant given that government funding typically stresses supercomputing.

For example, ANNs have been retrofitted into supercomputer weather and climate simulation codes. Humans programmed the original atmospheric simulations using expressions, loops, and subroutines. ANNs have replaced the programmed code in specific areas, such as radiation transport between atmospheric layers [5]. One project that used neural networks to emulate or duplicate human programming achieved an 80× speedup. Speed improved because the loops and subroutines in the human-programmed code were replaced with the structure in Figure 2, which was more efficient in this case. In another project, a neural network learned radiation flow properties from measured data and implicitly created a new model for radiative transport, achieving more accuracy than the human-programmed model.

ANNs could power computer advancement based on an entirely different principle from Moore's law. Computers have long been considered suitable for numerical and data-intensive tasks, whereas learning and reasoning have been the exclusive domain of humans. ANNs could alter this decades-old human–computer split by allowing automation of new tasks. The resulting boost in utility, enabled by learning, would reduce the reliance on advances in device physics for growth of the IT sector, which are becoming increasingly challenging.

Computational energy efficiency will still be important if ANNs continue to demonstrate potential. Google's AlphaGo showed how 1,920 CPUs and 280 GPUs could beat Lee Sedol four times out of five at Go. Although giving large numbers of people 100-kW supercomputers might raise human productivity, this isn’t economically or environmentally feasible. New energy-efficient technologies optimized for ANNs’ new learning features such as memristors2 must be developed.

References

[1] Silver, David, et al. "Mastering the game of Go with deep neural networks and tree search." *Nature* 529.7587 (2016): 484-489.

[2] Mountain, David J., et al. "Ohmic Weave: Memristor-Based Threshold Gate Networks." *Computer* 12 (2015): 65-71.

[3] Le, Quoc V. "Building high-level features using large scale unsupervised learning." *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*. IEEE, 2013.

[4] See link (not sure how to cite) https://en.wikipedia.org/wiki/Source_lines_of_code

[5] Krasnopolsky, Vladimir M., Michael S. Fox-Rabinovitz, and Dmitry V. Chalikov. "New approach to calculation of atmospheric model physics: Accurate and fast neural network emulation of longwave radiation in a climate model." *Monthly Weather Review* 133.5 (2005): 1370-1383.

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.