Nobel Lecture: Boltzmann machines*

Geoffrey Hinton

Department of Computer Science, University of Toronto, Ontario, Canada

(published 25 August 2025)

To solve difficult computational tasks, artificial neural networks need to construct appropriate internal representations by adapting the weights on their connections in the direction that improves performance. In the 1980s, there were two promising techniques for computing the gradients required to adapt the weights. One technique was backpropagation, which is now used in almost all artificial intelligence systems. The other was the Boltzmann machine learning algorithm, which is no longer used. After describing two of the major successes of backpropagation, this Nobel Lecture explains the Boltzmann machine learning algorithm, which uses a property of the Boltzmann distribution to compute gradients in an elegant and unexpected way.

DOI: 10.1103/RevModPhys.97.030502

CONTENTS

I. Introduction 1
II. Learning to Recognize Objects in a Feedforward Neural Network 2
III. Learning the Meanings of Words 2
IV. Hopfield Nets as a Model of Memory 3
V. Searching for Good Interpretations of Images Using Noisy Binary Neurons 4
VI. The Boltzmann Machine Learning Procedure 6
VII. Restricted Boltzmann Machines 8
VIII. Stacking Restricted Boltzmann Machines 9
IX. Deep Boltzmann Machines 10
X. Conclusion 10
Acknowledgments 10
References 10

I. INTRODUCTION

This Nobel Lecture aims to explain some of the historically important advances in artificial neural networks in a way that makes the ideas accessible to people who lack a strong background in physics or machine learning. To achieve this goal, I have suppressed many details that are not essential for understanding the essence of the ideas.[1] I have minimized the number of equations and omitted mathematical derivations, focusing instead on intuitive explanations, including explanations of some concepts that are very familiar to physicists. I present the learning procedure for Boltzmann machines as a narrative that describes how Terrence Sejnowski and I arrived at the idea.

Artificial neural networks were inspired by the brain, but they use highly simplified neurons. The connections between neurons have weights that change with learning and these weights determine how the output of a neuron depends on the activities on its input lines, which are typically the outputs of other neurons. The central problem in neural network research is to figure out how to adapt the weights so that the whole network becomes better at some complex task like classifying objects in images or predicting the next word in a document. When the neural network has a large number of weights, it is hopelessly inefficient to simply change one of the weights and see if this helps because many different examples need to be run through the network to evaluate whether the change helped. Learning is much more efficient if there is some way of computing a gradient for every weight: the rate at which the performance of the whole network improves as the weight increases. All the weights can then be changed in parallel in the direction that improves performance.

In the 1980s, when most researchers in artificial intelligence were confident that artificial neural networks were a dead end, I collaborated with David Rumelhart on investigating the backpropagation learning procedure and with Terry Sejnowski on investigating the Boltzmann machine learning procedure. These are two very different ways of getting the gradient information required for adapting the weights. At the time, both procedures had their pros and cons and it was not clear which method would give rise to the best technology or which method had the best chance of explaining how real neural networks learn.

Most of this Nobel Lecture is about the Boltzmann machine learning procedure, which makes use of a property of thermal equilibrium to derive an unexpectedly simple way of estimating the required gradients. At the time, this procedure seemed like a far better bet than backpropagation for how the brain learned to do perception.[2] Sadly, Boltzmann machines did not work nearly as well as backpropagation, which gets the exact gradients in a fairly obvious way by using the chain rule of differentiation (Rumelhart, Hinton, and Williams, 1986).

I start by briefly describing two of the most influential successes of backpropagation that eventually led to its

*The 2024 Nobel Prize for Physics was shared by John J. Hopfield and Geoffrey E. Hinton. This paper is the text of the address given in conjunction with the award.

[1]I have, for example, ignored the fact that artificial neurons generally have bias terms.

[2]Francis Crick thought that backpropagation was wildly implausible as a model of how the brain learns and urged Terry Sejnowski and me to focus our efforts on Boltzmann machines.

widespread use as the workhorse of current artificial intelligence (AI). After acknowledging the success of backpropagation, I then return to the Boltzmann machine learning procedure, which is impractical, but far more intriguing.

II. LEARNING TO RECOGNIZE OBJECTS IN A FEEDFORWARD NEURAL NETWORK

The backpropagation algorithm was invented many times (Linnainmaa, 1970; Werbos, 1974; LeCun, 1985; Parker, 1985; Rumelhart, Hinton, and Williams, 1986; Amari, 1993). By the early 1980s, computers were fast enough to explore its ability to construct interesting feature-based representations in multilayer neural nets, and it became popular with cognitive scientists interested in finding alternatives to the prevailing view that cognition consisted of using symbolic rules to manipulate symbolic expressions. In the early 2000s, with the advent of graphics processing units (GPUs) and larger datasets, backpropagation gained popularity among computer scientists, engineers, and statisticians who discovered that it often worked better than their existing methods for making predictions from data.

In 2012, Alex Krizhevsky, Ilya Sutskever, and I developed a deep neural network, which we later called AlexNet, that finally convinced researchers in computer vision that deep neural networks trained with backpropagation were much better than their existing technology at recognizing objects in images (Krizhevsky, Sutskever, and Hinton, 2012). We got almost half the error rate of the best non-neural vision systems. This advance was made possible by a number of different factors.

First, there was the dataset. For its time, AlexNet had a lot of connections and therefore required a lot of training data—many more labeled images than had traditionally been used for training neural nets to recognize objects. Fortunately, Deng *et al.* (2009) had created a very large hand-labeled dataset and had started an annual competition with about a million training images containing a thousand different types of object. Some of the object classes were very similar in appearance, like different breeds of dogs or species of mushrooms.

Second, there was the computer power. Training a large neural net on a lot of images required a lot of compute. Fortunately, Nvidia GPUs and their CUDA software made it relatively easy to parallelize computations across a very large number of simple processors. This allowed neural networks to learn at least an order of magnitude faster. AlexNet was a multilayer convolutional neural network of the type pioneered in Toronto by Yann LeCun (LeCun, 1989). Each layer learned a set of linear filters that were applied, at every point in the image, to the nonlinearly transformed outputs of the filters in the previous layer. Making convolutional neural nets run really efficiently on a single GPU was a very tricky programming problem and getting multiple GPUs to collaborate was also difficult. Alex's code for solving these problems was excellent and is now on display in the computer museum in Mountain View, California.

Third, we used some technical innovations to make AlexNet learn faster and generalize better to images it had not been trained on. To speed training we used the recently introduced rectified linear nonlinearity (Nair and Hinton, 2010) and to improve generalization we used a method called "dropout" which randomly eliminates half of the neurons before processing each image, but a different random half each time (Srivastava *et al.*, 2014). Dropout prevents the neural net from learning delicately balanced solutions in which one neuron cancels out the bad side effects of another neuron.

The remarkable performance of AlexNet opened the floodgates for neural networks and it also convinced Nvidia that the GPUs they had designed for computer graphics might be more valuable as cheap supercomputers for training neural networks.

III. LEARNING THE MEANINGS OF WORDS

By 2013, neural networks had been remarkably successful for computer vision and speech recognition (Hinton *et al.*, 2012), but most researchers in computational linguistics and traditional symbolic AI were confident that neural networks would never be able to deal with natural language tasks such as translating between different languages or answering questions. For these tasks, the input is a sequence of symbols and the output is another sequence of symbols, so it is not unreasonable to guess that the computation that converts input to output resembles logic and consists of using symbolic rules that specify how to manipulate strings of symbols to create new strings of symbols.

According to traditional symbolic AI, which dominated the field for half a century, human knowledge consists of symbolic expressions in some unambiguous internal language and communicating knowledge consists of taking a symbolic expression out of my head and putting it into yours. Neural networks implement a radically different and far more successful view of natural language processing and knowledge representation. Understanding a sentence does not consist of translating it into a symbolic expression in some internal language. It consists of associating appropriate feature vectors with each word or word fragment. These feature vectors are context-dependent: the feature vectors of the various words in a context constrain each other. This is what makes it possible to infer the meaning of a novel word from a single example of its use as in, "She scromed him with the frying pan." Knowledge does not consist of symbolic expressions. It consists of weights that specify how features of words should interact in the iterative process of settling on appropriate feature vectors for each word or word fragment. Neural networks do not store any strings of symbols. They use their stored weights to generate strings of symbols when needed. Knowledge is not communicated by copying a symbolic expression from one system to another. Instead, the student system adapts its weights so that it would be more likely to generate the same sequence of words or actions as the teacher.

A very early example of the neural network approach to language was described in the 1986 *Nature* paper on backpropagation (Rumelhart, Hinton, and Williams, 1986). The point of the example was to show that neural nets could discover appropriate feature vectors for words simply by trying to predict the next word in a short sequence. The error in the prediction was backpropagated through the network to get gradients for the weights. This included the weights used

to convert each word symbol to a feature vector and also the weights that allowed the features of the words in the context to predict the features of the next word. The neural net was tiny. The feature vectors for words only had six components and the whole net contained only about a thousand weights. It was trained on sequences of three words that came from a simple domain and it learned very sensible features for the words that captured the underlying structure of the domain and enabled it to predict the third word of sequences it had not been trained on.

The large language models that we have today can be viewed as descendants of this tiny language model from 1986. They use word fragments rather than words and they use 2 orders of magnitude more features in each feature vector. They use much longer contexts to predict the next word fragment and they also use many layers of feature vectors so that the meanings can be progressively refined. The way in which the features of word fragments interact is also a lot more complicated (Vaswani *et al.*, 2017). But, just like the tiny language model from 1986, they learn the meanings of words by trying to predict the next word and backpropagating the prediction error.

IV. HOPFIELD NETS AS A MODEL OF MEMORY

Our main way of understanding mysterious things is by using analogies to things that are more familiar. Most people think that memories are like files in a computer or a filing cabinet. After they have been stored, the files just sit there passively until they are retrieved, though they may fade as time goes by. It is important to realize that this is just an analogy and it may not be a very good one.

There is a very different view of what memories are which emerges when we consider networks of interconnected brain cells. Real brain cells are complicated devices with many properties that we still do not fully understand. If we just ignore a lot of the details of real neurons and model them as relatively simple devices, we discover that networks of these artificial neurons can exhibit very interesting collective behavior that leads to a completely different view of what memories are.[3]

For now, we will only consider binary artificial neurons. At any moment, each binary neuron can be active or inactive. If it is active it sends out a ping which arrives, at the next moment, at all the neurons it is connected to. The only thing a neuron needs to do is decide, at each moment, whether to be active or inactive and it has to base this decision on the pings it receives from other neurons that were active at the previous moment.[4] A few of the neurons might also receive inputs from sensors that are driven by the external world.

Not all pings are equal. When a ping sent by one neuron arrives at another neuron, it makes a contribution to the total input received by that neuron and the size of this contribution depends on the strength of the connection between those two neurons. The connection strengths, which are called "weights," can be positive or negative and can vary in magnitude. If the total input to a neuron is positive the neuron becomes active, otherwise it is inactive. That's it. That is how the artificial neural network works.

Neural networks can behave in many different ways depending on the values of the weights. The activity patterns can oscillate or evolve chaotically or settle into a stable state. In 1982, John Hopfield, inspired by physical systems which settle into states that minimize energy, figured out a simple way of ensuring that neural networks settle to a stable state (Hopfield, 1982). The trick is to make the weights symmetric so that the amount that a ping coming from neuron A contributes to the total input to neuron B is the same as the amount that a ping coming from neuron B contributes to the total input to neuron A. When the weights are symmetric, something special happens. Every pattern of activity of the whole network has an "energy" and every time an individual neuron uses its total input to decide whether or not to be active; it is either decreasing the energy or leaving it the same.[5]

The energy of an activity pattern is determined by the sum of all the weights between neurons that are both active.

$$E = -\sum_{i<j} s_i s_j w_{ij}, \tag{1}$$

where s_i and s_j are the binary states of two neurons and w_{ij} is the weight of the connection between them. Changing the binary state of neuron j from a 0 to a 1 decreases the energy of the whole network by Δ_j, which is just the total input to j from the other neurons:

$$\Delta_j = \sum_i s_i w_{ij}. \tag{2}$$

So, turning on a neuron whose total input is positive or turning off a neuron whose total input is negative is guaranteed to decrease the energy of the whole network. In a low energy pattern, the neurons that are active form a coalition in which each active neuron receives net positive support from other active neurons because the connections between the active neurons have mostly positive weights.

The weights determine which patterns of activity will be stable. These patterns are local energy minima that cannot be improved by turning on or off any single neuron. We can find these local minma by simply running the network from many different starting patterns. The process of running the network takes a starting pattern and repeatedly updates the activities of individual neurons decreasing the energy of the pattern each time a neuron changes its activity state. Hopfield proposed

[3]The idea that memories are energy minima was proposed in 1924 by Ivor Richards (Richards, 2017), who used the analogy of a crystal that can rest on any one of its many faces.

[4]This is a gross simplification. Neurons are leaky integrators that integrate information over time by accumulating charge. There are also fast, temporary changes to synapse strengths that allow the recent past to influence the current processing.

[5]If we update several different neurons at the same time, it is possible for the energy to increase so, strictly speaking, we need to only update one neuron at each time step, though updating a random subset of the neurons generally works quite well.

that this process is a model of content-addressable memory in which a highly corrupted pattern is transformed back into a familiar, uncorrupted pattern. For example, a pattern could be corrupted by turning off most of the active neurons. If the remaining active neurons represent a few of the features of an entity, the process of running the network will fill in the other features of the entity and settle on a stable pattern that is the network's standard representation of the entity. This model of what it means to retrieve a memory is very different from the model in which retrieval consists of fetching a stored file. The memory is constructed on the fly by using the weights to fill in missing features.

If we are given the weights, we can find the stable patterns of activity by running the network. But what about the inverse problem? If we are given some patterns of activity, how do we find weights that will make these patterns be stable states of the network? How do we implant memories?

Here is a quick and dirty way to implant a desired activity pattern as a memory: Using strong external input to the neurons, force the network to adopt the desired pattern. Then for all connected pairs of neurons that are both active in the desired pattern, slightly increase the weight between them. For all connected pairs of neurons in which one is active and the other is inactive, slightly decrease the weight. The increases will make the neurons in the desired pattern form a stronger coalition and the decreases will help to ensure that this coalition does not turn on other neurons that should be off in the desired pattern.

V. SEARCHING FOR GOOD INTERPRETATIONS OF IMAGES USING NOISY BINARY NEURONS

In the summer of 1982, Hopfield presented his work on what are now called Hopfield nets at a small workshop in Rochester.[6] Terry Sejnowski and I were both there. Terry had recently read a new paper by Kirkpatrick, Gelatt, and Vecchi (1983) on a technique called simulated annealing which was a way of finding better low energy states of a system by using the equivalent of thermal noise to jump out of local energy minima into deeper minima. In real annealing you start with very hot metal and then gradually reduce the temperature to allow the metal to form large regular crystals rather than the small randomly oriented crystals that you get if you cool the metal very fast by quenching it.

I had recently collaborated with Paul Smolensky to implement the backpropagation algorithm using logistic neurons that have a smooth relationship between their total input and their activity rather than the step function used by the neurons in a Hopfield net. During Hopfield's talk, Terry and I realized that you could do simulated annealing in a Hopfield net to find better energy minima and that the way to do this was to use noisy binary neurons which always have a state of 1 or 0 but use the logistic function to make a probabilistic decision about which state to adopt,

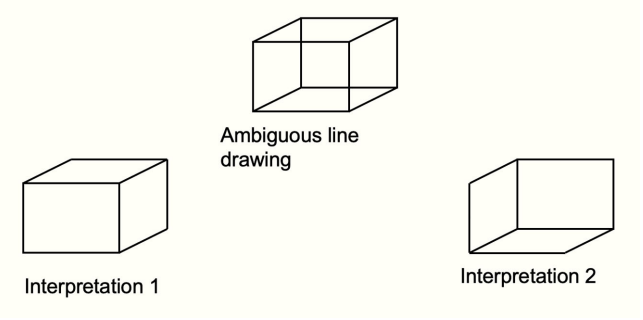

FIG. 1. At the top is a line drawing of an ambiguous figure. At the bottom are two different 3D interpretations of the line drawing. The interpretations differ in the depths and 3D orientations they assign to the lines.

$$p(s_j = 1) = \frac{1}{1 + \exp(-\Delta_j/T)}, \tag{3}$$

where T is a "temperature" that scales the energies. When $T = 0$ we recover the deterministic Hopfield update rule.

Terry and I got very excited about the idea of combining Hopfield nets with simulating annealing. It opened up a lot of new possibilities. Instead of using Hopfield nets to store memories as stable activity patterns, they could be used to solve optimization problems. In particular, they could be used for interpreting visual input.

Visual perception is a good area to study if you want to know how the brain works. We are very visual animals, and about a third of our neocortex is devoted to visual perception. More importantly, the way in which we do vision is very similar to the way monkeys do it and quite similar to the way cats do it, so the enormous effort that neuroscientists have put into investigating the visual systems of monkeys and cats provides a lot of insight into how vision works in our brains.

To explain how an artificial neural network composed of noisy binary neurons could do visual perception, I will describe a nonexistent neural network that is designed to be easy to understand but would be hopelessly impractical.[7] The job of the imaginary neural network is to interpret a class of 2D line drawings as 3D objects, as shown in Fig. 1. Imagine that the 3D object is rectangular with flat faces that join at straight edges. You are looking at it through a window with one eye and you make a 2D line drawing by using a marker pen to draw on the window. Each 3D edge in the object gives rise to a 2D line in the image, but this 2D line has lost some information about the 3D edge. The line does not specify the depth of the two ends of the 3D edge. This means there is a whole family of 3D edges that could have given rise to exactly the same 2D line in the image as shown in Fig. 2.

We will assume that the first stage of our imaginary neural network converts each 2D line in the image into activity in a specific neuron whose only job is to keep going ping

[6]Hopfield nets can use activity states of 1 and −1 or 1 and 0. I have chosen to describe them using states of 1 and 0 because they are more like real neurons and they lead more naturally to Boltzmann machines.

[7]It is very inefficient to use a single neuron to represent a single line in the image. It is much more efficient to use a distributed representation (Hinton, McClelland, and Rumelhart, 1986) in which each neuron represents a whole set of possible lines and a single line is represented by the intersection of many such sets.

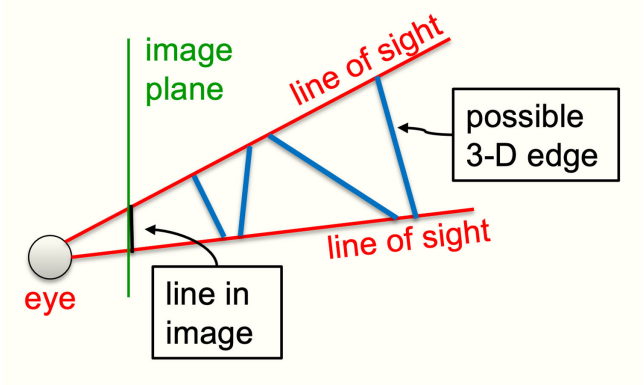

FIG. 2. A line in the image plane could depict many different 3D edges. All these edges have ends that lie on the two lines of sight, but they differ in the depths they assign to the ends of the line. Only one of these edges can be seen at a time.

whenever that particular line is present. We would need a very large number of such neurons, and only a very few of them would be active for each image. Each of these 2D line neurons has connections with positive weights to a large set of 3D edge neurons that represent all the possible 3D edges that could have given rise to that specific 2D line. Whenever the 2D line neuron is active it means that one of these 3D edges must be present in the object, but we do not know which one because the image has lost the information about the depths of the two ends of the 3D edge. Since we could only see one of the 3D edges in this set at a time, we interconnect them with weights with negative connections so that they compete with each other to be active as shown in Fig. 3. How can a neural network magically recover the lost depths? To put it another way, how can the neural network decide which one of the 3D edge neurons in each set to activate?

Let's start with a simpler example of missing information. If you ask a mathematician to solve the equation $x + y = 384$, they will tell you it is impossible. If you ask a computer scientist, they will tell you that x and y are most likely to be 256 and 128 but they don't know which is which. Computer scientists spend a lot of their time dealing with powers of 2 so, for them, powers of 2 are more probable than other numbers and if they realize that 384 is just the sum of two powers of 2 that seems like a good bet for the answer.[8]

Our visual systems have learned some important principles that allow them to find good interpretations of images. First, if two lines join in the 2D image, it is reasonable to assume that the 3D edges they depict have exactly the same depth at the point where their images join. It is possible that they do not, but this would mean we were viewing the 3D edges from a very special viewpoint where their ends just happened to line up even though they are at different depths. This is very unlikely, so our visual systems assume it does not happen. We can wire this assumption into the network by putting connections with positive weights between all pairs of 3D edge neurons that join in depth as shown in Fig. 3.

A second assumption that we make when interpreting line drawings is that 3D edges often join at right angles in 3D even

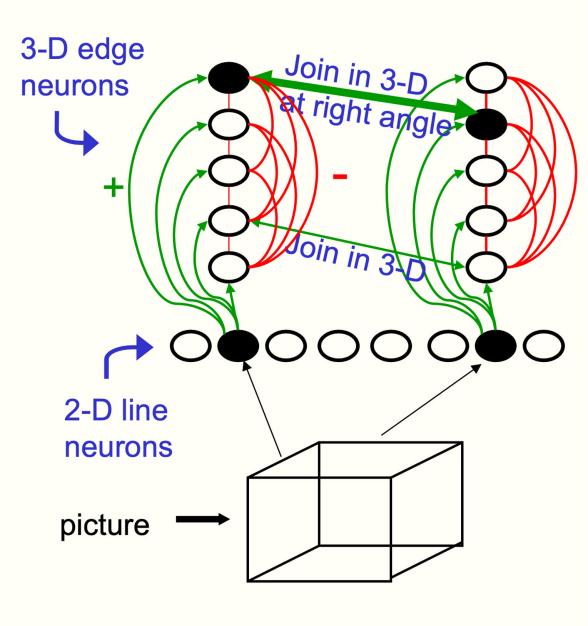

FIG. 3. An imaginary neural network for interpreting line drawings. Each 2D line in the drawing activates a specific 2D line neuron. Each 2D line neuron is directly connected to a whole set of 3D edge neurons that represent all the possible 3D edges that could have given rise to that line. The line neuron has connections with positive weights that try to activate the 3D edge neurons, but the 3D edge neurons in this set all try to suppress each other using connections with negative weights. There are also connections with positive weights between 3D edge neurons in different sets if the two 3D edge neurons represent edges that join in depth. The weights are larger if the edges join at right angles.

though the 2D lines that depict them do not join at right angles in 2D. We can build this assumption into the neural net by using large positive weights on the connections between 3D edge neurons that represent 3D edges that join at right angles.

With all of these suitably weighted connections in place, we hope that the neural network will settle into an interpretation in which each 2D line neuron activates a single 3D edge neuron, and the 3D edges that are represented by these active neurons join at right angles.[9] For this to work, we need to solve two problems. First, the positive and negative weights on the connections have to be just the right size so that good interpretations correspond to global energy mimima. Learning the sizes of the weights is a very tricky problem with a beautifully simple solution that will be described later. For now, we will assume that good interpretations correspond to global energy minima. This still leaves the search problem of ensuring that the network settles on a coalition of active 3D edge neurons that minimizes the energy. It must not get jammed in a frustrated activity pattern that consists of several small, internally consistent coalitions that do not support each other. If this happens, the individual active neurons in each coalition will be unwilling to change states because that would raise the energy contributed by the local coalition.

[8]I verified this by asking Jeff Dean to solve the equation.

[9]There are many interpretations of this form because the 2D image would be unchanged if the object were bigger and further away.

The solution to the search problem is to use noisy neurons that occasionally adopt states that raise the energy of the whole pattern of activity. This allows the neural network to escape from poor energy minima. In simulated annealing, we start with very noisy neurons and gradually make them less noisy, eventually settling on a single minimum. Miraculously, if we reduce the noise level slowly enough, the system will almost certainly settle on the globally best minimum (Geman and Geman, 1984) (or pick one of the best minima if there are many that have equal energy). Unmiraculously, this could be a glacially slow process.

What if we use a fixed noise level and just keep running the network at that noise level? The network will never settle on a stable pattern. The activity pattern will keep changing forever. But there is something else, something more abstract, that will settle down even though the pattern keeps changing. The probability distribution over activity patterns will settle to the Boltzmann distribution.

Imagine that we have made a huge number of copies of exactly the same neural network. The weights on the connections in each copy are identical. We activate exactly the same set of 2D line neurons in each copy and then we run all of the copies in parallel, updating the states of the 3D edge neurons but allowing each copy to make its own noisy decisions about whether or not to activate a neuron given the total input it is receiving. Even if we start off with exactly the same activity pattern in each copy, the patterns will soon diverge because the copies will make different noisy decisions. We now have a complicated mess with each copy doing its own thing. How could order emerge from this randomness?

Once we have fixed the activity states of the 2D line neurons, the number of possible activity patterns is exponential in the number of 3D edge neurons. If there are n 3D edge neurons there are 2^n possible activity patterns. Assuming that we made a number of copies of the neural network that is far larger than the 2^n possible patterns of activity, we could keep track of what fraction of the copies have each particular activity pattern, as we are running all these copies in parallel.

To begin with, all the copies might have the same activity pattern because we initialized all the copies the same way. But after a while, every pattern of activity will be present in some fraction of the copies and these fractions settle down. Even though every individual copy keeps jumping around between different activity patterns, the fraction of the copies that have a particular activity pattern will stabilize. This is called thermal equilibrium. At each time step, the number of copies that leave any particular activity pattern will be equal to the number of copies that enter that activity pattern. The fraction of the copies that have a particular activity pattern is called the probability of that activity pattern.

If we only have one copy of the neural network and we run it for a sufficiently long time before we look at its activity pattern, we cannot predict what the pattern will be because the neurons have been making noisy decisions. But we can say what the probability of each pattern will be. It will be equal to the fraction of the copies that would have that pattern if we had run a huge number of copies of the same network in parallel.

Once the network has reached thermal equilibrium, the probability of an activity pattern depends only on its energy relative to the energies of other patterns. Assuming a fixed temperature of 1, the probability of a pattern consisting of a vector $\mathbf{v}$ over the visible neurons and a vector $\mathbf{h}$ over the hidden neurons is determined by the energy $E(\mathbf{v},\mathbf{h})$ of that global activity pattern relative to the energies of all possible global activity patterns:

$$p(\mathbf{v},\mathbf{h}) = \frac{\exp[-E(\mathbf{v},\mathbf{h})]}{\sum_{\mathbf{u},\mathbf{g}}\exp[-E(\mathbf{u},\mathbf{g})]}. \tag{4}$$

This is the Boltzmann distribution.

Terry and I spent the second half of 1982 playing around with relatively simple neural networks in which we set the weights by hand. We managed to show that noisy neurons helped the network solve simple search problems, like finding short paths through little mazes, but this did not impress anyone.

We realized that we needed a learning procedure for setting the weights on the connections of a "Boltzmann machine"—a neural network that uses symmetric connections and binary, noisy neurons. The reason for thinking that there might be a neat learning procedure was the simple relationship between the probabilities of patterns of activation and their energies when the network has reached thermal equilibrium. The energy of a pattern can be easily manipulated by changing the weights on the connections between active neurons and this allows the probabilities of patterns to be manipulated in a controlled way.

VI. THE BOLTZMANN MACHINE LEARNING PROCEDURE

Let's return to the idea of implanting certain patterns of activity as memories. To do this, we need these patterns to have low energy and all the other patterns to have significantly higher energy. It is easy to make patterns have low energy: we just increase the connection weights between pairs of neurons that are both active in that pattern. But how should we make all the other patterns have higher energy? Raising the energy of patterns that already have high energy is a waste of time since they are not the rivals that are competing with the memories we are trying to implant. What we need to do is find patterns of low energy that are not the desired memories and suppress them.

Terry and I were aware of recent work by Crick and Mitchison (1983) in which they proposed that the function of dreams was to get rid of things you were inclined to believe but were not real. Their idea was that during dreams we were doing unlearning: the sign of the changes to the weights was reversed. But they did not have a good theory of how much unlearning you should do because they did not have a mathematical model of what the unlearning achieved. Terry and I realized that the simple relationship between the energy of a pattern and its probability of occurring when sampling from the equilibrium distribution might allow us to come up with a precise mathematical justification for unlearning, and we were delighted when the mathematics led to a beautifully simple learning procedure for implanting memories.

The learning procedure has two phases. In the "wake" phase you keep picking patterns that you want to implant as memories and you use external input to force the network to adopt the chosen pattern of activity. Then you add a small amount to the weight of all the connections between neurons that are both active. In the sleep phase you run the neural network until it has reached its equilibrium distribution and then you sample a pattern of activity and subtract a small amount from the weight of all the connections between neurons that are both active. It's as simple as that.

In the sleep phase, the process of first settling to thermal equilibrium and then doing unlearning finds and suppresses the low energy patterns that are the main rivals of the patterns being implanted as memories. Of course, the sleep phase will occasionally find one of the memories and suppress it, but this is OK. Eventually, the sleep phase will find the same low energy activity patterns as are being imposed on the network in the wake phase and when this happens the increments to the weights during wake and the decrements during sleep will cancel out and the learning will stop. At this point, the dreams of the network will look just like the activity patterns being imposed on it when it is awake: the weights in the network will be a good model of reality because they will allow the network to generate correct memories.

Our new learning procedure was good for implanting memories but Terry and I wanted to solve a far more ambitious problem. In a Hopfield net, a memory consists of an activity pattern of *all* of the neurons. This means that when you try to implant a memory, you know the desired binary states of all the neurons. Now suppose that you have some neurons whose activities represent which of the many possible 2D lines are present in a line drawing. You would like the patterns of activity of these 2D line neurons to have low energy when they correspond to a line drawing of a rectangular 3D object and high energy when they do not. You could try to achieve the desired low and high energies by setting the weights between pairs of 2D line neurons, but this won't work.

The 3D edges of rectangular objects give rise to 2D lines that can join at more or less any angle in the image—it just depends on your viewpoint. The underlying regularity in a big set of 2D line drawings of 3D rectangular objects is that the 3D edges join at right angles. This regularity cannot be expressed by learning weights between neurons that represent 2D lines. To give low energy to line drawings of 3D rectangular objects, we need neurons whose activities represent the presence of particular 3D edges. Then we can learn big positive weights between pairs of neurons that represent 3D edges that join at right angles.

If our training data just consist of patterns of activity of the 2D line neurons, neurons that represent 3D edges would be called hidden neurons because their states are not directly specified by the training data. Conversely, the neurons whose states are directly specified by the training data—the 2D line neurons—are called visible neurons. The states of the hidden neurons have to be inferred from the pattern of activity of the visible neurons. This inference problem would be hard enough even if we hand wired all of the weights between the visible 2D line neurons and the hidden 3D edge neurons and all of the weights between pairs of hidden neurons as shown in Fig. 3. But what if the neural net just starts off with a pool of uncommitted hidden neurons that have randomly chosen weights on the connections between pairs of hidden neurons and the connections between the 2D line neurons and the hidden neurons? It seems outrageously ambitious to expect that a simple learning procedure could discover weights that would make the hidden neurons represent 3D edges. Remember, nobody is telling these hidden neurons what they ought to be representing. They have to discover what they need to represent in order for the underlying regularities in the data to be expressible by the weights between hidden neurons.

To our surprise, a simple extension of our learning procedure for implanting memories does the job, at least theoretically (Ackley, Hinton, and Sejnowski, 1985). In the wake phase, you take one of the line drawings in the training data and clamp the corresponding pattern of activity on the 2D line neurons. Then you keep updating the hidden neurons until you have reached thermal equilibrium. Then you sample a pattern of activity of all the neurons and add a small amount to the weight between pairs of neurons that are both active. This includes pairs of active hidden neurons but also pairs consisting of an active hidden neuron and an active 2D line neuron. In the sleep phase you do not provide any external input to the 2D line neurons. You simply run the neural network, updating the states of both the hidden and the visible neurons until it has reached its equilibrium distribution and then you sample a pattern of activity and subtract a small amount from the weight between neurons that are both active. This is the Boltzmann machine learning procedure.

The mathematics says that, on average, the Boltzmann machine learning procedure does exactly the right thing. It changes the weights in just the right way to increase the probability that the patterns of activity on the visible neurons when the network is dreaming look like the patterns of activity it observes when it is awake. This means there is a simple way for a Boltzmann machine to learn to represent the hidden underlying causes of the sensory data. That is the holy grail of perceptual learning.

Assuming that we have reached thermal equilibrium at a temperature of 1, the probability of a visible vector $\mathbf{v}$ is obtained by summing the joint probabilities over all possible hidden vectors,

$$p(\mathbf{v}) = \frac{\sum_{\mathbf{h}} \exp[-E(\mathbf{v},\mathbf{h})]}{\sum_{\mathbf{u},\mathbf{g}} - \exp[E(\mathbf{u},\mathbf{g})]}. \tag{5}$$

It can be shown that the derivative of the log probability of $\mathbf{v}$ with respect to a weight, $w_{i,j}$, is given by

$$\frac{\partial \log p(\mathbf{v})}{\partial w_{ij}} = \langle s_i s_j \rangle_{\mathbf{v}} - \langle s_i s_j \rangle_{\text{model}}, \tag{6}$$

where the first correlation is measured at thermal equilibrium when vector $\mathbf{v}$ is clamped on the visible neurons and the second correlation is measured when the network is running freely with no neurons clamped.

If we average over all the visible vectors in the training data, we get the Boltzmann machine learning rule:

$$\Delta w_{i,j} \propto \langle s_i s_j \rangle_{\text{data}} - \langle s_i s_j \rangle_{\text{model}}. \tag{7}$$

Terry and I got very excited when the mathematics said that our simple learning procedure would do the right thing. Our excitement gradually dissipated over the next few years when it became apparent that our wonderful learning procedure was hopelessly impractical. Mathematical results always depend on assumptions and those assumptions can be unrealistic. In the case of the Boltzmann machine learning procedure, the mathematics hinges on the network reaching thermal equilibrium because it is only at thermal equilibrium that there is a simple mathematical relationship between the energy of a pattern of activity and the probability that the network will adopt that pattern of activity. The problem is that a large network, especially one with large weights, can take an extremely long time to reach thermal equilibrium.

For a while, we hoped that we could find a way to learn weights that allowed the network to reach thermal equilibrium rapidly, but we failed. This meant that our beautiful learning procedure was hopelessly impractical for large networks. It was an interesting application of statistical physics to neural networks, but it was not the answer to how the brain learns.

VII. RESTRICTED BOLTZMANN MACHINES

Seventeen years after we first discovered the Boltzmann machine learning procedure, I finally found an efficient learning procedure for a simplified form of the Boltzmann machine, which I called the restricted Boltzmann machine (RBM), that had been studied by Paul Smolensky (Smolensky, 1986) and Yoav Freund (Freund and Haussler, 1991) in the 1980s. An RBM has visible neurons that are connected to hidden neurons, but there are no connections between the hidden neurons. This throws away almost all of the representational power of a Boltzmann machine. For example, it is the connections between pairs of hidden neurons that capture the fact that 3D edges should join at right angles. But this loss of representational power is accompanied by a huge gain in computational efficiency. During the wake phase, when the states of the visible neurons are determined by the sensory data, the hidden neurons reach thermal equilibrium after a single update of the state of each hidden neuron, and all these updates can be done in parallel because, given the states of the visible neurons, the states of the hidden neurons are independent of one another. To put it another way, if you fix the states of the visible neurons and repeatedly update the states of the hidden neurons, there will be no correlations between the fluctuations in the states of the hidden neurons.

The wake phase of an RBM is computationally very efficient, but this still leaves the sleep phase. For the whole network to reach thermal equilibrium when there is no external input to determine the states of the visible neurons, it is necessary to alternate many times between updating all of the visible neurons in parallel and updating all of the hidden neurons in parallel.

Figuring out mathematically how many alternating updates are required to get close to thermal equilibrium is very difficult because the weights are not random. They are learned and have a lot of nonrandom structure which determines how fast the network approaches thermal equilibrium. So I decided to do computer simulations to see how many alternations were required in order for the Boltzmann machine learning procedure to work properly. The task was to learn the weights between visible neurons that represented the intensity of the pixels in a binary image of a handwritten digit and hidden neurons that learned to represent features of the digits such as strokes and loops and cusps.

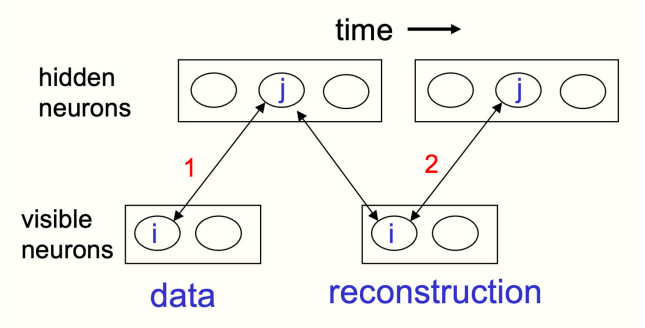

FIG. 4. Illustration of the contrastive divergence learning procedure. The correlation of the activities of visible neuron i and hidden neuron j is measured after the first update of the hidden neurons and again after their second update. The difference between these correlations is the signal used for learning.

I started by using a lot of alternations between updating the hidden neurons and updating the visible neurons. The learning worked well, so I assumed this meant that the sleep phase had reached thermal equilibrium. But as I reduced the number of alternations, I was astonished to see that the learning still worked even when the network could not possibly have approached thermal equilibrium. I had made a really dumb logical error. If the network approaches thermal equilibrium, the learning will work. But this does not imply that if the learning works the network must have approached thermal equilibrium. I had accidentally discovered a different and very efficient learning procedure which I called contrastive divergence because, to a first approximation, it minimizes the difference between two Kullback-Leibler divergences (Hinton, 2002).

The contrastive divergence procedure works like this: Start by initializing the binary states of the visible neurons to one of the binary training images. Then stochastically update the binary states of all the hidden neurons in parallel. Given the new states of the hidden neurons, stochastically update the binary states of the visible neurons in parallel to get a binary image that is called a "reconstruction" because, as the network learns, the reconstruction will come to look more and more like the initial image. Finally, stochastically update the states of the hidden neurons again. That's it. Instead of alternating for a long time between updating the hidden neurons and updating the visible neurons, it is sufficient to just update the hidden neurons, update the visible neurons, and update the hidden neurons once more as shown in Fig. 4.

The wake phase of the contrastive divergence learning procedure consists of adding a small amount to the weights between visible neurons that are active in the initial image and hidden neurons that are active after the first hidden update. The sleep phase consists of subtracting a small amount from the weight between visible neurons that are active in the reconstruction and hidden neurons that are active after the second hidden update.

Here is a way of visualizing what the contrastive divergence learning procedure is doing. You initialize the network with a pattern of activity on the visible neurons that you would like to be at an energy minimum. At the beginning of learning it is very unlikely to be at an energy minimum, so the reconstruction you get after first updating the hidden neurons and then updating the visible neurons will have wandered away from the initial image. To stop it from wandering away, you decrease the energy of the initial image in the wake phase of learning and you increase the energy of the reconstruction that it wandered to in the sleep phase.

Before we had computers, the police would construct images of faces by using an "identikit." The kit consisted of a large set of images of parts of faces. Parts might include a nose that was long and narrow and a nose that was short and wide or a pair of eyes that were small and close together and a pair of eyes that were large and far apart. By selecting the right parts, it was possible to produce an image that matched a witness's recollection of a face. As an RBM is learning to fit a set of training images, its hidden neurons are creating an identikit—they are learning to detect combinations of active and inactive pixels that occur frequently in the training images. These combinations are called "features." If the training images are handwritten digits, the features might be parts of strokes or loops or cusps. Different hidden neurons learn to detect different features and, once learning is complete, the active hidden neurons contain enough information about what is going on in the image to allow the image to be reconstructed from the features.

Contrastive divergence learning in an RBM does not have the nice mathematical guarantees of the original Boltzmann machine learning procedure, but it is practical and can be applied to big data. It was one of the methods used by the team that won the million dollar prize offered by Netflix for any system that could significantly outperform their existing system at predicting which movies people would like. But the lack of connections between hidden neurons seems like a very severe limitation on the representational power of RBMs. Fortunately, there is a way round this limitation that does not abandon the efficiency of contrastive divergence learning.

VIII. STACKING RESTRICTED BOLTZMANN MACHINES

After an RBM has been trained, you can show it an image and it will activate some of its hidden neurons. Each active hidden neuron will represent a particular combination of active pixels that occurs frequently. If we show the RBM lots of different images of the type that it has been trained on, there will be patterns among the active hidden neurons. For example, there may be three hidden neurons that typically all come on at the same time. This type of structure among the activities of hidden neurons is just like the structure among the activities of pixels and we can model it by using a second RBM that treats the hidden activities of the first RBM as its visible data and uses a layer of hidden neurons to model the patterns of activity in these "data." There is no need to write any more computer code. After training the first RBM using binary images as the data, we simply define the activities of its hidden neurons, when it is looking at images, as the data for

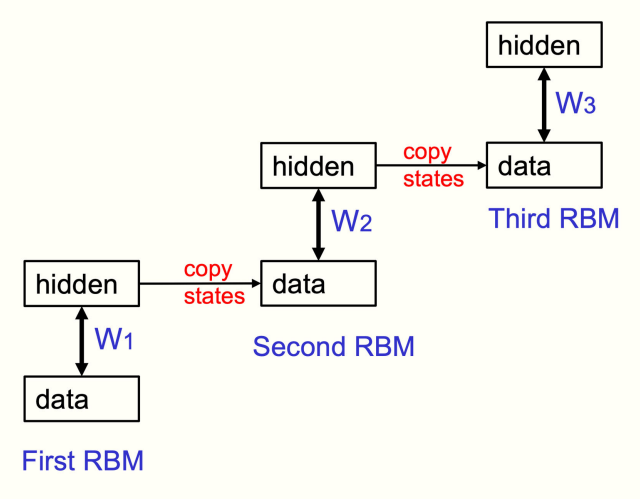

FIG. 5. A stack of separate RBMs can be trained by using the hidden states of one RBM as the data for training the next RBM.

training the second RBM. The hidden neurons of the second RBM learn features of features—patterns of activity of the hidden neurons of the first RBM that occur more frequently than would be expected if those hidden neurons behaved independently of each other.

It is fairly obvious that we could treat the patterns of activity of the hidden neurons of the second RBM as the data for training a third RBM that would learn features of features of features as shown in Fig. 5. So just by looking at images and training a stack of RBMs we can learn many layers of progressively more abstract features. This turns out to be a very effective way to initialize a deep neural network that has many layers of hidden neurons followed by a layer of output neurons that represent different class labels (Hinton and Salakhutdinov, 2006). This kind of network is used for classifying images—identifying the class label of the most salient object in an image. The standard "supervised" way to train these nets is backpropagation. But if there are many hidden layers and we start with small random weights, backpropagation is slow to get off the ground. If we start by ignoring the class labels and training the hidden layers one at a time as a stack of RBMs, we will learn a smorgasbord of complicated features. Many of these will be irrelevant for distinguishing between the class labels we are interested in, but many others will be very helpful.

After learning a stack of RBMs we just add a layer of output neurons on top as shown in Fig. 6 and use backpropagation to train the incoming weights of the output neurons and also to fine-tune all of the other layers of weights so that the features extracted by those layers are more useful for getting the class labels correct. Initializing the weights of a feedforward neural net in this way makes learning much faster. It also improves generalization because the information in the labels is only used to fine-tune the feature detectors already discovered by the RBMs. Between 2006 and 2011, several of the major breakthroughs in neural networks involved pretraining a deep neural net as a stack of RBMs and then adding a final output layer of class labels and fine-tuning all the weights with backpropagation (Hinton *et al.*, 2012). Eventually, researchers

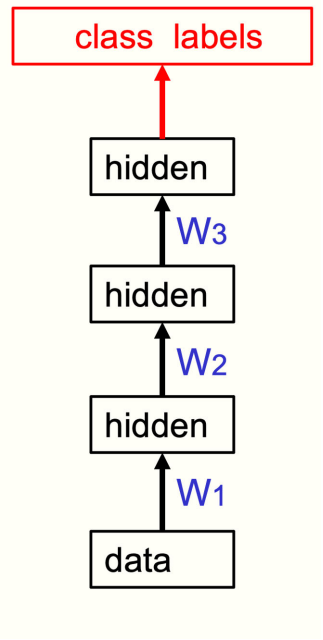

FIG. 6. After training a stack of RBMs, the weights learned by each RBM can be used to initialize a feedforward network. A final layer of output neurons that represent class labels can then be added and the whole network can be fine-tuned using backpropagation.

found other, simpler ways of initializing the weights of deep neural nets, and RBMs are no longer used.

IX. DEEP BOLTZMANN MACHINES

A stack of RBMs can also be used to initialize the weights of a deep Boltzmann machine that has multiple hidden layers, but no connections within a layer. If we abandon all pretense to be modeling the brain or explaining ideas to the uninitiated, we can combine this initialization with two other tricks to learn the weights of a deep Boltzmann machine (Salakhutdinov and Hinton, 2009). In the wake phase, we clamp a data vector on the visible units and instead of updating the multiple layers of binary hidden units stochastically, we use a real value to represent the probability of a hidden unit being on. It is these probabilities that are communicated between hidden units. If we repeatedly update the hidden probabilities by applying the logistic function to the total weighted input received by each neuron, the network will settle to a stable state that can be viewed as a "mean field" approximation to the true Boltzmann distribution. The mean field free energy is an upper bound on the true free energy, so when we change the weights in the wake phase to reduce the mean field free energy of a data vector, we will be pushing down on the true free energy and improving the mean field approximation by changing the weights in a direction that tends to reduce the gap between the true distribution and the mean field approximation.

In the sleep phase, we cannot use the mean field approximation because the sleep phase is trying to raise the energy and raising an upper bound tends to make the bound looser without any need to pull up the bounded quantity. This is probably why early attempts to use the mean field approximation for Boltzmann machine learning did not work very well (Anderson and Peterson, 1987).

In the sleep phase, we need to sample many different free energy minima corresponding to many different parts of the data space and it seems very improbable that we could get a good estimate of the correlations by just having one binary pattern of activity that we keep stochastically updating by first updating the odd-numbered layers and then updating the even-numbered layers. But this actually works remarkably well for a subtle reason. The weight updates in the sleep phase raise the energy of the current activity pattern so it quickly gets pushed away from its current location to a different part of the space. This effect can be enhanced by learning an overlay of "fast weights" that adapt rapidly but also decay rapidly. The total weight is just the sum of the fast and slow weights. The point of the fast weights is to cause rapid exploration of the space without having to wait for changes in the slow weights to repel the global state from its current position (Tieleman and Hinton, 2009).

X. CONCLUSION

Boltzmann machines are no longer used, but they were historically important for two reasons. In the 1980s, they demonstrated that it was possible to learn appropriate weights for hidden neurons using only locally available information without requiring a biologically implausible backward pass. In the 2000s, initializing feedforward nets using stacked restricted Boltzmann machines made backpropagation work better, so RBMs were a useful tool that acted like an enzyme, easing the transition to deep learning.

ACKNOWLEDGMENTS

Many people contributed to the ideas described in this paper. My main collaborators were Terry Sejnowski and David Rumelhart, but Ron Williams, Jay McClelland, Ilya Sutskever, Alex Krizhevsky, David Ackley, Ruslan Salakhutdinov, Nitish Srivastava, and Tijmen Tieleman also made major contributions.

REFERENCES

Ackley, D. H., G. E. Hinton, and T. J. Sejnowski, 1985, "A learning algorithm for Boltzmann machines," Cognit. Sci. **9**, 147–169.

Amari, S.-i., 1993, "Backpropagation and stochastic gradient descent method," Neurocomputing **5**, 185–196.

Anderson, J. R., and C. Peterson, 1987, "A mean field theory learning algorithm for neural networks," Complex Syst. **1**, 995–1019, https://content.wolfram.com/sites/13/2018/02/01-5-6.pdf.

Crick, F., and G. Mitchison, 1983, "The function of dream sleep," Nature (London) **304**, 111–114.

Deng, J., W. Dong, R. Socher, L.-J. Li, K. Li, and L. FeiFei, 2009, "Imagenet: A large-scale hierarchical image database," in *Proceedings of the 2009 IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2009), Miami, 2009*, pp. 248–255 (IEEE, New York).

Freund, Y., and D. Haussler, 1991, "Unsupervised learning of distributions on binary vectors using two layer networks,"

in *Advances in Neural Information Processing Systems*, Vol. 4, edited by J. Moody, S. Hanson, and R. P. Lippmann (Morgan Kaufmann, Burlington, MA).

Geman, S., and D. Geman, 1984, "Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images," IEEE Trans. Pattern Anal. Mach. Intell. **PAMI-6**, 721–741.

Hinton, G. E., 2002, "Training products of experts by minimizing contrastive divergence," Neural Comput. **14**, 1771–1800.

Hinton, G. E., J. L. McClelland, and D. E. Rumelhart, 1986, "Distributed representations," in *Parallel Distributed Processing*, Vol. 1, edited by D. E. Rumelhart and J. L. McClelland (MIT Press, Cambridge, MA), pp. 77–109.

Hinton, G. E., and R. R. Salakhutdinov, 2006, "Reducing the dimensionality of data with neural networks," Science **313**, 504–507.

Hinton, G. E., *et al.*, 2012, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," IEEE Signal Process. Mag. **29**, 82–97.

Hopfield, J. J., 1982, "Neural networks and physical systems with emergent collective computational abilities," Proc. Natl. Acad. Sci. U.S.A. **79**, 2554–2558.

Kirkpatrick, S., C. D. Gelatt, and M. P. Vecchi, 1983, "Optimization by simulated annealing," Science **220**, 671–680.

Krizhevsky, A., I. Sutskever, and G. E. Hinton, 2012, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, Vol. 25, edited by F. Pereira, C. J. Burges, L. Bottou, and K. Q. Weinberger (Morgan Kaufmann, Burlington, MA).

LeCun, Y., 1985, "Une procedure d'apprentissage pour reseau a seuil asymetrique [A learning scheme for asymmetric threshold networks]," in *Proceedings of Cognitiva 85, Paris, 1985*, pp. 599–604.

LeCun, Y., 1989, "Generalization and network design strategies," in *Connectionism in Perspective*, edited by R. Pfeifer, Z. Schreter, F. Fogelman-Soulié, and L. Steels (North-Holland, Amsterdam), pp. 143–155.

Linnainmaa, S., 1970, "The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors," master's thesis (University of Helsinki).

Nair, V., and G. E. Hinton, 2010, "Rectified linear units improve restricted Boltzmann machines," in *Proceedings of the 27th International Conference on Machine Learning (ICML-10), Haifa, Israel, 2010*, edited by J. Fürnkranz and T. Joachims (Omnipress, Madison, WI), pp. 807–814.

Parker, D. B., 1985, "Learning-logic," MIT Sloan School of Management Technical Report No. 47.

Richards, I. A., 2017. *Principles of Literary Criticism* (Routledge, Abingdon, England).

Rumelhart, D. E., G. E. Hinton, and R. J. Williams, 1986, "Learning representations by back-propagating errors," Nature (London) **323**, 533–536.

Salakhutdinov, R., and G. Hinton, 2009, "Deep Boltzmann machines," in *Proceedings of Machine Learning Research: Artificial Intelligence and Statistics (AISTATS 2009), Clearwater Beach, FL, 2009* (PMLR), edited by D. van Dyk and M. Welling, pp. 448–455.

Smolensky, P., 1986, "Information processing in dynamical systems: Foundations of harmony theory, in *Parallel Distributed Processing*, Vol. 1, edited by D. E. Rumelhart and J. L. McClelland (MIT Press, Cambridge, MA), pp. 194–281.

Srivastava, N., G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, 2014, "Dropout: A simple way to prevent neural networks from overfitting," J. Mach. Learn. Res. **15**, 1929–1958, https://www.jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf.

Tieleman, T., and G. Hinton, 2009, "Using fast weights to improve persistent contrastive divergence," in *Proceedings of the 26th Annual International Conference on Machine Learning, Montreal, 2009*, edited by A. Danyluk, L. Bottou, and M. Littman (Association for Computing Machinery, New York), pp. 1033–1040.

Vaswani, A., N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, 2017, "Attention is all you need," in *Advances in Neural Information Processing Systems*, Vol. 30, edited by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Curran Associates, Red Hook, NY), pp. 5998–6008.

Werbos, P., 1974, "Beyond regression: New tools for prediction and analysis in the behavioral sciences," Ph.D. thesis (Harvard University Committee on Applied Mathematics).