Of Memories, Neurons, and Rank-One Corrections

Kevin G. Kirby

Kevin Kirby's Ph.D. is from Wayne State University (1988); in 1994 he joined the Mathematics and Computer Science Department at Northern Kentucky University. He has taught graduate courses on neural networks, but his main research interests are natural computability theory and the physics of information. His hobbies include piano, taekwondo, and symplectic geometry. One day in 1986 his advisor let him give a crazy guest lecture on neural networks in the Dirac notation; that lecture was the ancestor of this paper.

On this nonlinear tour of one part of the concept of *linearity* we begin with the classical art of memory, cast in a formalism that turns out to be the Dirac notation from quantum mechanics. From this we move to an investigation of how early associationist models of mind found their natural home in a very simple neurophysiological model. This leads us to the hopeful (and hype-ful) contemporary field of *artificial neural networks*, of which we provide a glimpse. A recurrent event on this tour is the construction of linear transformations by the accumulation of many small rank-one adjustments. This places some of the ideas in undergraduate linear algebra in an unusual light.

Places

In classical Greece and Rome the art of memory was an important part of the art of rhetoric. Orators were expected to memorize—sometimes verbatim, sometimes approximately—very long speeches. Distinguishing "natural" from "artificial" memory, numerous *Ars Memorativa* treatises up to the Renaissance elaborated a variety of memorization techniques. Cultivating artificial memory is less prized today, but the mnemonic tricks in popular books on memory improvement for fun and profit are not so far in spirit from those found in these early treatises. In her 1966 study of this tradition, Frances Yates describes the *mnemonic place system* as set forth by the Roman orator Quintilian around A.D. 90. The idea was to memorize a sequence of images by associating them with an easily recoverable sequence of places:

> In order to form a series of places in memory, he [Quintilian] says, a building is to be remembered, as spacious and varied a one as possible, the forecourt, the living room, bedrooms, and parlours, not omitting statues and other ornaments with which the rooms are decorated. The images by which the speech is to be remembered . . . are then placed in imagination on the places which have been memorised in the building. This done, as soon as the memory of the facts requires to be revived, all these places are visited in turn and the various deposits demanded of their custodians. We have to think of the ancient orator as moving in imagination through his memory building whilst he is making his speech, drawing from the memorised places the images he has placed on them. [**13**]

Note that the term *image* is interpreted broadly as something to be evoked by the appropriate mnemonic trigger. This technique is quite effective; often drivers who listen to audio recordings of novels find that specific places continue to evoke the passages previously heard there.

Let's try a medieval exercise in symbolization. It calls for some odd notation which will be explicated later (readers familiar with this notation are requested to forget any prior interpretation).

Quintilian's orator is wandering through a building while working through his speech, and every few steps he notices an item to use as a locus for some image. Here is an urn in the atrium, which we shall denote in this way: $|\text{urn}\rangle$. The image from his speech at this point is a joke about the Senate: $|\text{joke}\rangle$. Now we can write a kind of "product,"

$$|\text{joke}\rangle\langle\text{urn}|,$$

to denote this association, which the orator somehow adds to the store of associations already in his brain. Walking on, he adds $|\text{insult}\rangle\langle\text{pool}|$ and $|\text{apology}\rangle\langle\text{porch}|$. Gradually these associations accumulate into the totality of his memory.

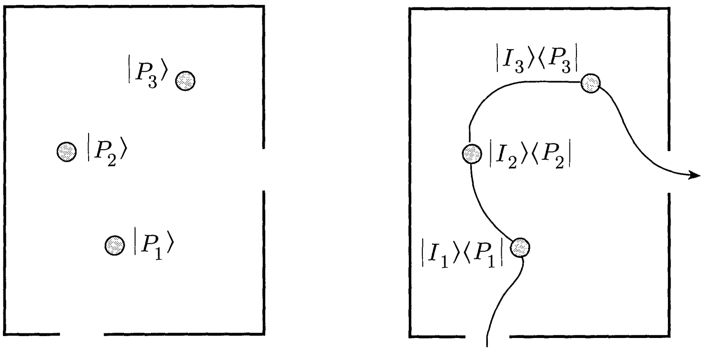
Let $\mathbf{M}$ denote the memory of the orator, viewed as an accumulating store of image and place associations:

$$\mathbf{M} = |\text{joke}\rangle\langle\text{urn}| + |\text{insult}\rangle\langle\text{pool}| + |\text{apology}\rangle\langle\text{porch}| + \cdots .$$

Is the "+" commutative? No; more recent associations might somehow be stronger than older ones. For our purposes, however, we will consider all of these associations to have equal weight, so the addition operator is truly commutative. Figure 1 sketches the procedure: As the orator first walks past places $|P_k\rangle$, $k = 1, 2, \ldots$, he associates to them the images $|I_k\rangle$ and accumulates all these associations:

$$\mathbf{M} = \sum_k |I_k\rangle\langle P_k|.$$

To deliver his speech, the orator imagines himself walking through the building, evoking the places in sequence. Now his memory $\mathbf{M}$ functions as a kind of transformation that assigns to each place $|P_k\rangle$ the corresponding image $|I_k\rangle$. For example, $\mathbf{M}|\text{urn}\rangle = |\text{joke}\rangle$.



$|P_3\rangle$
$|P_2\rangle$
$|P_1\rangle$
$|I_3\rangle\langle P_3|$
$|I_2\rangle\langle P_2|$
$|I_1\rangle\langle P_1|$

Figure 1. Left: A building with three places. Right: A tour to associate the places with images.

There is one more detail to take into account. Quintilian called for a building as "spacious and varied" as possible. Why "varied"? Because if two places were highly similar, like $|\text{urn}\rangle$ and $|\text{vase}\rangle$, the orator could confuse their associated images upon recall. Suppose we can get a real number to indicate the degree of similarity. We might then write the similarity of $|\text{urn}\rangle$ and $|\text{vase}\rangle$ as a product

$$\langle \text{urn}|\text{vase}\rangle$$

that is equal to zero if the two places are "completely different." A good orator would pick a building with a set of places $\{P_i\}$ such that $\langle P_i|P_j\rangle \approx 0$ for all $i \neq j$. Since these places are mental pictures rather than real physical locations, this product could be described as *the shadow cast by one idea on another*. This is an appealing metaphor, one to which we shall return.

Bras and Kets

In 1939 Paul Adrien Maurice Dirac [4] introduced a clever variant of vector notation for use in quantum mechanics. It turns out to be quite useful for our purposes here, so we take a little time to explore it.

Consider a finite-dimensional real inner product space X. We denote the vectors in X by symbols called *kets*: $|x\rangle$, $|x'\rangle$, and so on. Dual vectors in X^* are denoted by symbols called *bras*: $\langle x|$, $\langle x'|$. Remember that dual vectors are linear functions from X to the real numbers $\mathbb{R}$. Applying a dual vector to a vector gives the inner product, which in Dirac's bra-ket notation is

$$\langle x| : X \to \mathbb{R} : |x'\rangle \mapsto \langle x|x'\rangle.$$

Now, the Dirac notation is particularly valuable because it works to defer matters involving components of vectors. The term *vector* can refer to an element of a set with a vector space structure on it, or it can refer to an n-tuple that is subject to addition and scaling. These two notions are easily conflated when we use a notation like $\mathbf{x}$, which seems to abbreviate "the vector $(x_1, x_2, \ldots, x_n)$." The Dirac notation brings home the idea of a vector as *an intrinsic object that has components only with regard to a particular basis*. For instance, let $\{|1\rangle, |2\rangle, |3\rangle, \ldots, |n\rangle\}$ be an orthonormal basis for X. (In quantum mechanics one honors an important basis by naming it with integers.) Instead of using subscript notation, as in $x_1 = 3$, Dirac used the less compact but mathematically more precise notation $\langle 1|x\rangle = 3$, which spells out what "component" means: $\langle 1|x\rangle$ is the orthogonal projection of $|x\rangle$ onto the first basis vector.

When we turn to linear transformations, the standard terminology is not ambiguous. Instead of using a single term for both the object and its representation, we use two: the *linear transformation* and the *matrix* that represents it in the bases chosen for its domain and range. To see this, let X and Y be two (finite-dimensional real inner product) spaces, possibly of different dimension. The entries in a matrix representing a linear transformation $\mathbf{M} : X \to Y$ are usually referred to by using subscripts: m_{ij}. In the Dirac notation, however, this matrix entry is $\langle i|\mathbf{M}|j\rangle$. This exposes the real meaning: You take a basis vector $|j\rangle$ in X and apply the transformation $\mathbf{M}$ to it to get $\mathbf{M}|j\rangle$; then you find the magnitude of the component of this vector in the $|i\rangle$ direction in Y, which is $\langle i|\mathbf{M}|j\rangle$.

There is another kind of product we can define, which is central to the theme of this paper. The *outer product* of $|y\rangle$ with $|x\rangle$, denoted $|y\rangle\langle x|$ is defined as the following function from X to Y:

$$|y\rangle\langle x| : X \to Y : |x'\rangle \mapsto |y\rangle\big(\langle x|x'\rangle\big).$$

We can drop the parentheses and write $|y\rangle\langle x|x'\rangle$ or $\langle x|x'\rangle|y\rangle$ without any ambiguity; it is the vector $|y\rangle$ scaled by the real number $\langle x|x'\rangle$. It is easy to show that $|y\rangle\langle x|$ is a linear transformation. It is a very special linear transformation, however, since *every* vector $|x'\rangle$ in the space X is mapped onto the $|y\rangle$ direction. All vectors orthogonal to $|x\rangle$ are mapped to 0. This means

$$\begin{aligned} \text{im } |y\rangle\langle x| &= \text{span } \{|y\rangle\} \\ \ker |y\rangle\langle x| &= \text{span } \{|x\rangle\}^{\perp}. \end{aligned}$$

Since the range im $|y\rangle\langle x|$ is one-dimensional, the outer product yields a linear transformation of rank one. Moreover, any rank-one linear transformation can be written as an outer product.

The definition of outer product does not use any kind of basis. Nevertheless, the matrix of an outer product in orthonormal bases for X and Y is simple. If $\mathbf{M} = |y\rangle\langle x|$, the ijth entry is $\langle i|\mathbf{M}|j\rangle = \langle i|y\rangle\langle x|j\rangle = \langle i|y\rangle\langle j|x\rangle$, using the fact that the inner product on real vector spaces is commutative; in subscript notation, $m_{ij} = y_i x_j$. Fixing these bases reveals how we can use the familiar matrix multiplication rule if we place the components of $|y\rangle$ in a column vector and the components of $\langle x|$ in a row vector. For example,

$$\begin{bmatrix} 2 \\ 3 \\ 4 \end{bmatrix} \begin{bmatrix} 10 & 100 \end{bmatrix} = \begin{bmatrix} 20 & 200 \\ 30 & 300 \\ 40 & 400 \end{bmatrix}$$

is a matrix that represents a rank-one transformation from $X = \mathbb{R}^2$ to $Y = \mathbb{R}^3$.

Starting with $\mathbf{M} = 0$ (which has zero rank, of course), we can build up a higher-rank linear transformation by adding outer products to it. Take the two-term sum

$$\mathbf{M} = |y\rangle\langle x| + |y'\rangle\langle x'|.$$

If $\{|y\rangle, |y'\rangle\}$ and $\{|x\rangle, |x'\rangle\}$ are both linearly independent sets, then we are now at rank two: $\mathbf{M}$ maps all vectors in X to the two-dimensional subspace of Y spanned by $|y\rangle$ and $|y'\rangle$. In fact, using the singular value decomposition,[1] any rank-r linear transformation $\mathbf{M} : X \to Y$ can be written as a weighted sum of outer products of vectors in Y and X,

$$\mathbf{M} = \sum_{k=1}^{r} \sigma_k |y_k\rangle\langle x_k|$$

where the σ_k's are nonnegative numbers and $\{|y_k\rangle\}$ and $\{|x_k\rangle\}$ are both orthonormal sets. Computing such a decomposition can be very involved. Fortunately, our goal is not to decompose but to compose, which is easy.

The following expansion of the identity operator in an orthonormal basis is often used in calculations:

$$\mathbf{I} = \sum_{j=1}^{n} |j\rangle\langle j|. \tag{1}$$

[1]Dan Kalman [5] has a good introduction to this decomposition and its applications.

What does this mean? Applying both sides to an arbitrary vector $|x\rangle$, we get

$$|x\rangle = \sum_{j=1}^{n} |j\rangle\langle j|x\rangle = \sum_{j=1}^{n} x_j |j\rangle,$$

which echoes our definition of x_j as the component of $|x\rangle$ in direction $|j\rangle$.

To see how the Dirac notation is used in contemporary quantum mechanics, I recommend Albert's informal, elementary exposition [**1**] or Sudbery's intermediate-level survey [**10**]. With this notation in hand, we return to our formalization of the mnemonic place system.

Associations

Our orator memorized his speech by walking through a building while associating places (a specific urn, pool, porch) with parts ("images") of his speech (a specific joke, insult, apology). Suppose the place memories are represented by vectors $|\text{urn}\rangle$, $|\text{pool}\rangle$, $|\text{porch}\rangle$ in some vector space X. (Yes, this is still quite abstract—since we cannot yet imagine how to represent these vectors in terms of components—but it's in the nature of linear algebra that we can defer this.) Even better, we can represent the places by *directions*, since $c|\text{urn}\rangle$ should represent the same memory as $|\text{urn}\rangle$ for any real $c \neq 0$. We therefore may pick representatives that are normalized:

$$\langle\text{urn}|\text{urn}\rangle = \langle\text{porch}|\text{porch}\rangle = \langle\text{pool}|\text{pool}\rangle = 1,$$

and, following Quintilian's recommendation of variety, we pick them so that they are orthogonal to each other:

$$\langle\text{urn}|\text{porch}\rangle = \langle\text{porch}|\text{pool}\rangle = \langle\text{pool}|\text{urn}\rangle = 0.$$

Likewise, the images from the speech can be represented by vectors: $|\text{joke}\rangle$, $|\text{insult}\rangle$, $|\text{apology}\rangle$, this time in a different vector space, Y. To consider them distinct, we require that they be linearly independent.

Now, as described earlier, each new association is a "rank-one" update to the orator's memory,

$$\mathbf{M} = |\text{joke}\rangle\langle\text{urn}| + |\text{insult}\rangle\langle\text{pool}| + |\text{apology}\rangle\langle\text{porch}|.$$

Thus we can verify that the memory transformation evokes the right images in response to the right places. When the speaker recalls the urn, for example, his memory is applied to $|\text{urn}\rangle$ to yield:

$$\begin{aligned}\mathbf{M}|\text{urn}\rangle &= |\text{joke}\rangle\langle\text{urn}|\text{urn}\rangle + |\text{insult}\rangle\langle\text{pool}|\text{urn}\rangle + |\text{apology}\rangle\langle\text{porch}|\text{urn}\rangle \\ &= |\text{joke}\rangle\end{aligned}$$

using the orthonormality requirements from above.

It works, but unfortunately over time our orator's memory begins to fade, so he remembers the shape of the urn's belly, but not how its handles look. Is it enough to remember only the container? Will just part of the urn suffice to evoke the joke?

Let's break the urn into two independent components, the "container" portion and the handles:

$$|\text{urn}\rangle = |\text{container}\rangle + |\text{handles}\rangle$$

where $\langle \text{container}|\text{handles}\rangle = 0$. This "+" can also be written as $\oplus$, meaning *orthogonal sum*. The container and handles must be sufficiently dissimilar from the other places (porch, pool) that their respective inner products are also zero. If $|\text{urn}\rangle$ is normalized to have unit length, then its part $|\text{container}\rangle$ must have some positive length $c < 1$. As our orator now mentally revisits the place $|\text{container}\rangle$, let's see what is evoked:

$$\begin{aligned}
\mathbf{M}|\text{container}\rangle &= \big(|\text{joke}\rangle\langle\text{urn}| + |\text{insult}\rangle\langle\text{pool}| + |\text{apology}\rangle\langle\text{porch}|\big)|\text{container}\rangle \\
&= |\text{joke}\rangle\langle\text{urn}|\text{container}\rangle + |\text{insult}\rangle\langle\text{pool}|\text{container}\rangle \\
&\quad + |\text{apology}\rangle\langle\text{porch}|\text{container}\rangle \\
&= |\text{joke}\rangle\langle\text{urn}|\text{container}\rangle + 0 + 0 \\
&= |\text{joke}\rangle\big(\langle\text{container}| + \langle\text{handles}|\big)|\text{container}\rangle \\
&= |\text{joke}\rangle\big(\langle\text{container}|\text{container}\rangle + \langle\text{handles}|\text{container}\rangle\big) \\
&= |\text{joke}\rangle\big(c^2 + 0\big) \\
&= c^2|\text{joke}\rangle.
\end{aligned}$$

Since the scale factor c^2 does not change the vector's direction, this fragmentary recall works too. Memory is *so* associative that images can be correctly evoked even when prompted with only partial information. On the other hand, because $c^2 < 1$, the image has indeed grown fainter.

All the orthogonality conditions constrain the memory capacity of this system—a finite-dimensional space has only so many orthogonal vectors! Suppose the vector space X of places has dimension n, and we keep adding associations. Beyond n associations, we can no longer keep the places distinct; our *overloaded* memory starts getting confused. This problem deserves attention, but we need not explore it in the abstract setting of vector spaces. Let's turn to a concrete model: neurophysiology.

Neurons

Association seems a reasonable basis for the art of memory. Memories are "sticky" things; a scent or a song attached to an event can evoke memories of the event for a lifetime. Is this association a sophisticated, high-level cognitive process? Hardly; rats learn to press bars for biscuits. Such a simple phenomenon seems basic, a building block for more complicated cognitive processes.

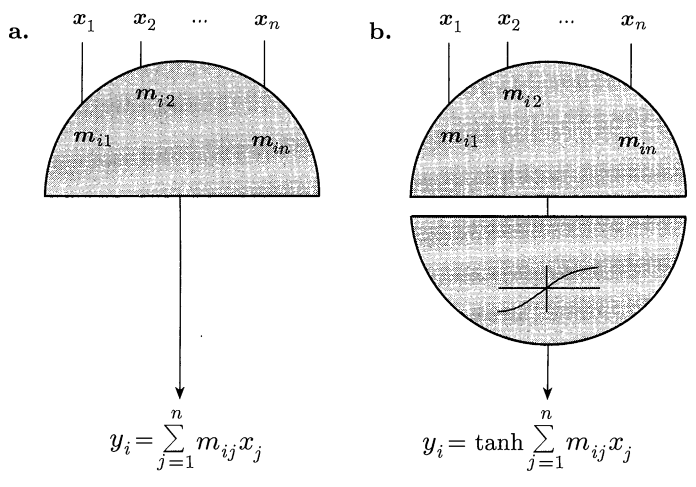
Associationist theories of thinking and behavior were advocated by the eighteenth-century philosophers Berkeley and Hume. For Hume, in fact, the "self" was no more than a bundle of associations. His view was reified when the field of neurophysiology began to advance in the nineteenth century. In Herbert Spencer's 1850 book *Principles of Psychology* we read about networks of connections in the nervous system. The key move here is the interpretation of the vague and abstract notion of *association* of ideas as a physical *connection* between neural configurations. An "idea" is somehow a pattern of activity in part of a neural net. If it is associated with another idea, its activity pattern propagates and triggers the activity corresponding to that other idea. The construction of an association becomes a physical process, broadly construed. For many psychologists in the nineteenth and twentieth centuries, this reification seemed natural.

In the following sections we will interpret our formalism for associative memory in "neural connectionist" terms, just as the early neuropsychologists did. The pictures *we* draw (Figures 2 and 3, for example) are very nearly the pictures *they* drew (see [**11**]). We will think of neurons as little hardware elements that process input signals and produce output signals. Note carefully: The word "neuron" here is used in the same fashion as "virus" is used in the world of computers. A computer virus is not a model of a real virus; rather, a real virus serves as a vivid metaphor that helps us describe the nature of a complex fragment of software. So, too, by using the colorful term "neuron" we do not mean to imply that we are doing any kind of physiological modeling.

Figure 2a shows a *linear neuron*. It has n input lines carrying real numbers x_j, and it produces a single real-valued output y_i, which is computed as a weighted sum of the inputs:

$$y_i = m_{i1}x_1 + m_{i2}x_2 + \cdots + m_{in}x_n.$$

The *synaptic weights*, m_{ij}, are also real numbers. Their name comes from the *synapse*, which is the subcellular structure that mediates signal transmission between nerve cells. A synaptic weight $m_{ij} = 0$ means that the neuron ignores the jth input x_j completely. As m_{ij} increases, the input has a stronger excitatory effect, producing a larger output y_i. A negative m_{ij} means the input has an inhibitory effect.



a.
x_1 x_2 $\cdots$ x_n
m_{i1}
m_{i2}
m_{in}
$y_i = \sum_{j=1}^{n} m_{ij}x_j$
b.
x_1 x_2 $\cdots$ x_n
m_{i1}
m_{i2}
m_{in}
$y_i = \tanh \sum_{j=1}^{n} m_{ij}x_j$

Figure 2. A linear (a) and a nonlinear (b) neuron.

One extension of this simple definition of a neuron is worth mentioning. In many neural network architectures, it is convenient if a neuron's output signal is bounded—commonly being "clipped" to the interval $(-1, 1)$. In many algorithms the linear neurons acquire this nonlinear property via a hyperbolic tangent function applied to their output, as shown in Figure 2b. We have $\tanh u \approx u$ for u near 0, and $\tanh u \approx \operatorname{sgn} u$ for large $|u|$. An engineer or statistician who "uses neural nets" nowadays is almost certainly employing the latter kind of neurons. We, however, shall stick to linear neurons (or, equivalently, tanh neurons operating near 0).

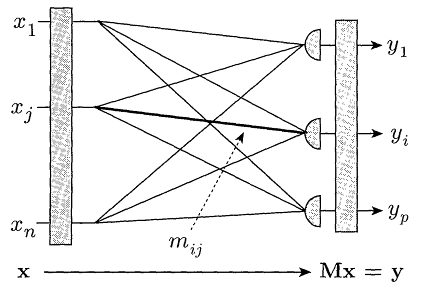


Figure 3. One-layer network of linear neurons; i.e., a matrix-vector multiplier.

Neural associations occur within a *layered network* of neurons, where each of the n inputs is distributed to p linear neurons, with input j to neuron i being weighted by the real number m_{ij}. Figure 3 depicts this synaptic weight as attached to an arc from input j to neuron i. (The value $m_{ij} = 0$ would be the special case of no connection.)

Figure 3 reveals a striking underlying simplicity: This device just computes a matrix-vector product. When we pile the inputs into a column vector $\mathbf{x} = [x_1, \dots, x_n]^T$, pile the outputs into a column vector $\mathbf{y} = [y_1, \dots, y_p]^T$, and make m_{ij} an entry in a $p \times n$ matrix $\mathbf{M}$, our neural net implements $\mathbf{y} = \mathbf{Mx}$.

What can we do with neural nets? That's about like asking, "What can we do with matrix multiplication?"—a yawningly broad question. Applied mathematicians, electrical and optical engineers, statisticians, physicists, computer scientists, and psychologists have all found uses in their fields. The variety of applications is exemplified by the astounding scope of the recent *Handbook of Brain Theory and Neural Networks* [**2**]. We will concentrate only on some key ideas behind *linear* neural associative memory, the precursor of today's most famous neural algorithms.

Holographic Memory

One early idea about the neural mechanisms of memory came from the observation that repetition strengthens an association. William James in the late nineteenth century and, more explicitly, Donald Hebb in the 1940s suggested a physical basis: One strengthens *associations* by strengthening *neural connections*. How does this scheme fit the layered neural network of Figure 3?

Assume for the moment that inputs and outputs are both binary: $x_j, y_i \in \{0, 1\}$ for all i, j. This all-or-nothing setup means either an input signal is there or it isn't, and either the neuron "fires" in response or it is silent. When an input $\mathbf{x}$ is presented to the net, it elicits the output $\mathbf{y} = \mathbf{Mx}$. Now, if a nonzero input signal x_j contributes to a nonzero output signal y_i, we imagine that the weight connecting them increases by some fixed amount—say weight m_{ij} becomes $m_{ij} + 1$. Otherwise (for zero input, zero output, or both), m_{ij} is unchanged. In short, if input signal x_j contributes to output y_i, then the neural network modifies itself this way:

$$m_{ij} \to m_{ij} + y_i x_j. \tag{2}$$

In the binary case, the rightmost term is 0 unless $x_j = y_i = 1$. This rule captures the intuition that a frequently used connection is reinforced. By extension, we can suppose that connections strengthen in this way for *all* real-valued inputs and outputs. Our neural net could become a kind of forced dynamical system, with a weight matrix $\mathbf{M}(t)$ that evolves over time as inputs $\mathbf{x}(t)$ arrive and outputs $\mathbf{y}(t)$ are emitted. We will use this update rule to get a formula for storing associations in the network.

First, let's ask what kinds of things can be associated. Patterns, obviously—but what are patterns? In the neural computation game we are playing, one assumes that *patterns are vectors*. To associate a sign of the zodiac, ♋, with a face, 😐, we would want the neural network to produce the output vector $|♋\rangle$ when presented with the input vector $|😐\rangle$. In other words, the system should implement a transformation $\mathbf{M}$ such that $|♋\rangle = \mathbf{M}|😐\rangle$.

To realize this transformation concretely (albeit crudely), suppose we digitize a photograph of the face into a 30×30 gray-level image, representing this digital image as a 900-tuple of real numbers. The value of the jth entry ($j = 1, 2, \ldots, 900$) holds the gray-level of the jth pixel in the image (e.g., -1 for black, $+1$ for white, with a continuum of grays in between). We can view this 900-tuple of real numbers as the components of the vector $|😐\rangle$ in some basis. If the basis is labeled $\{|1\rangle, |2\rangle, \ldots, |900\rangle\}$, the pixels are $\langle 1|😐\rangle, \langle 2|😐\rangle, \ldots, \langle 900|😐\rangle$. In this representation, a scaled version of the image (e.g., $1.5|😐\rangle$) is merely the image with its contrast adjusted, and $-|😐\rangle$ is its photographic negative. Accordingly, it is safe to identify the pattern with any nonzero vector lying along the $|😐\rangle$ direction. How do we present this face as input to a neural network? Simply put these values on input lines $x_1 = \langle 1|😐\rangle$, $x_2 = \langle 2|😐\rangle$, and so on. See Figure 4.

The output of the neural net will be a linear transformation of the input:

$$y_i = \sum_{j=1}^{n} m_{ij} x_j .$$

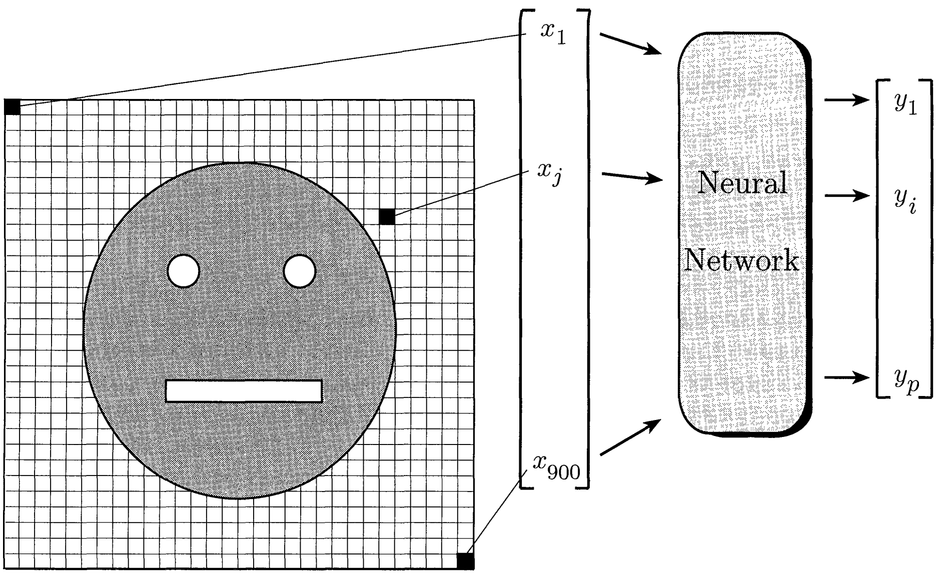


Figure 4. Making a vector out of a black-and-white photo of a face. A grid is overlaid and digitizes the face into a 30×30 gray-level image. The vector is fed into a neural network and evokes an output pattern.

Translating into the Dirac notation and using equation (1), we have

$$\langle i|y\rangle = \sum_{j=1}^{n} \langle i|\mathbf{M}|j\rangle\langle j|x\rangle = \langle i|\mathbf{M}\left(\sum_{j=1}^{n} |j\rangle\langle j|\right)|x\rangle = \langle i|\mathbf{M}|x\rangle$$

for $i = 1, 2, \ldots, n$. In short, $|y\rangle = \mathbf{M}|x\rangle$ in any basis.

We interrupt this exercise in linear algebra to point out a conceptual limitation: *Patterns are not really vectors*. Some features of patterns, such as their complexity, cannot be captured by vectors. For example, a solid black image is not recognizably more "complicated" than a high-resolution photo of a face if we consider them merely as vectors. In some sense, any nonzero vector in a vector space is just as complex as any other nonzero vector! Some measures that might distinguish between simple and complex patterns, such as the entropy

$$H = -\sum_{j=1}^{n} \langle j|\text{☺}\rangle \log\langle j|\text{☺}\rangle,$$

depend on the basis chosen, and thus fall outside the domain of linear algebra. But within these constraints, it is surprising what we can do.

We now return to linearity. To get the neural network of Figure 3 to "store" the association of $|\text{☺}\rangle$ with $|\text{♋}\rangle$ we rewrite the James/Hebb update rule (2) for adjusting connection weights:

$$\langle i|\mathbf{M}|j\rangle \to \langle i|\mathbf{M}|j\rangle + \langle i|\text{♋}\rangle\langle\text{☺}|j\rangle$$

for all i, j, or simply

$$\mathbf{M} \to \mathbf{M} + |\text{♋}\rangle\langle\text{☺}|. \qquad (3)$$

In a nutshell: *To store an association in a linear neural network, make a rank-one correction to its linear transformation by adding the outer product of the patterns to be associated.* This amounts to strengthening or weakening each synaptic weight in the network in proportion to the product of the corresponding components of the two patterns.

Now, back to our orator memorizing his speech. Starting with $\mathbf{M} = 0$, we store two associations:

$$\mathbf{M} = |\text{joke}\rangle\langle\text{urn}| + |\text{insult}\rangle\langle\text{pool}|.$$

An arbitrary input pattern $|x\rangle$ would evoke the output pattern

$$|y\rangle = \mathbf{M}|x\rangle = |\text{joke}\rangle\langle\text{urn}|x\rangle + |\text{insult}\rangle\langle\text{pool}|x\rangle.$$

Geometrically, this means that the output $|y\rangle$ lies in the space spanned by the two memories. If $|x\rangle = \text{urn}\rangle$ and the triggers of the associations are orthogonal, then our earlier calculations show that the output is just the pattern $|\text{joke}\rangle$.

We now have a recipe for making a neural associative memory, and it is the same one we informally sketched at the outset. Take a table of patterns to be associated:

Input pattern	*Output pattern*
$\lvert \mathrm{in}_1\rangle$	$\lvert \mathrm{out}_1\rangle$
$\lvert \mathrm{in}_2\rangle$	$\lvert \mathrm{out}_2\rangle$
$\vdots$	$\vdots$
$\lvert \mathrm{in}_N\rangle$	$\lvert \mathrm{out}_N\rangle$

To build a one-layer linear neural network (Figure 3), use the weight transformation $\mathbf{M}$ given by the formula

$$\mathbf{M} = \sum_{k=1}^{N} \lvert \mathrm{out}_k\rangle\langle \mathrm{in}_k\rvert. \tag{4}$$

If the input patterns are orthogonal, then the input $\lvert \mathrm{in}_k\rangle$ produces the output $\lvert \mathrm{out}_k\rangle$. Furthermore, as we saw earlier with our "$\lvert \mathrm{urn}\rangle = \lvert \mathrm{container}\rangle + \lvert \mathrm{handles}\rangle$" example, even if part of the input is corrupted, it can still be reliably associated with the desired output. And indeed *that is the point*. If we didn't need our associative memory to function with incomplete inputs (and weren't interested in neural network hardware for engineering reasons), we could just use a simple look-up table.

There is another reason why this architecture is interesting. What is fascinating about the "accumulation of rank-one transformations" approach is the nature of the neural net that it builds. After storing the associations, suppose we inflict some "brain damage" on the network. That is, we cut some connections: set $m_{35} = m_{94} = 0$, say. What happens? Not too much. The network's accuracy degrades only slightly. Did we corrupt any one particular stored association? No. Instead of having one memory affected, *all* memories deteriorate very slightly. This property is called a *holographic* property. If you break off a small piece of an optical holographic plate, you do not lose a specific piece of the image; instead, the entire hologram becomes a little bit fuzzier. The same thing happens in our neural network. A specific memory is stored in a distributed fashion, spread throughout the network. It is not stored at a specific locus. In this way our algorithm is even more reminiscent of natural neural systems. And this holographic property is a direct consequence of our accumulation of outer products.

Beyond Orthogonality

Our associative memory algorithm is appealing, but requiring orthogonality for the input patterns seems unrealistic. Here is a standard way of getting around this problem: Make the dimension n of the input space X much larger than the number N of patterns stored, and encode the inputs so they appear arbitrary or random. This dodge presupposes that N randomly chosen vectors in a space of dimension $n >> N$ will be approximately orthogonal. For example, we could represent the pattern $\lvert \mathrm{dog}\rangle$ as the word "dog" represented as a 21-component vector corresponding to the $3 \times 7 = 21$ bits in the ASCII representation of the 3-character string "dog." (ASCII is a 7-bit character encoding used by older computer operating systems.) Represent the one bits by 1.0 and the zero bits by -1.0, and then normalize the vector. Our "dog" would be represented by

$$\frac{1}{\sqrt{21}}\,[+1,-1,-1,-1,+1,-1,-1,\ \ +1,-1,-1,+1,+1,+1,+1,$$
$$+1,-1,-1,-1,+1,+1,+1]^T.$$

If this is not arbitrary, what is?

Still, this representation has problems. While the inner product $\langle \text{dog}|\text{cat}\rangle$ in this representation is 0.14, the inner product $\langle \text{rat}|\text{cat}\rangle$ is 0.80. Consequently, if we use rule (4) to store the associations

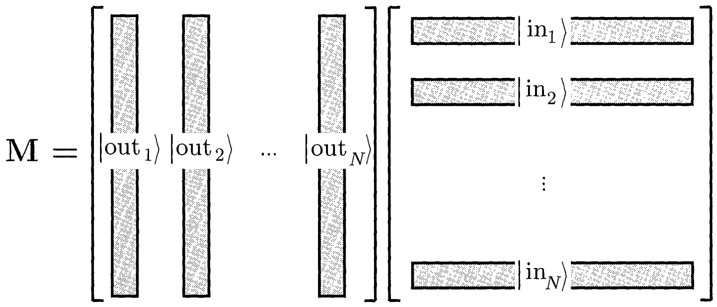
$$|\text{cat}\rangle \to |\text{felix}\rangle \qquad |\text{rat}\rangle \to |\text{templeton}\rangle \qquad |\text{dog}\rangle \to |\text{fido}\rangle,$$

presenting the input pattern $|\text{cat}\rangle$ will evoke the output

$$\mathbf{M}|\text{cat}\rangle = 1.0|\text{felix}\rangle + 0.80|\text{templeton}\rangle + 0.14|\text{fido}\rangle.$$

So we see that the response of the neural net is somewhat muddled, with pieces of the wrong answer superimposed on the right answer. The output patterns overlap to same the extent as the input patterns. Now, some superposition of memories may be a good idea in real brains (one can envision its role in creativity, for example), but in an associative memory architecture it needs to be controlled. We present some techniques to control this superposition, techniques that remain in the realm of the linear. More sophisticated nonlinear methods are beyond the scope of this article.

First, piling our input and output vectors into "vectors of vectors" gives us a new way to display rule (4) for building a neural network weight transformation from a sum of outer products:



If we treat the two factors on the right side as matrices $\mathbf{Y}$ and $\mathbf{X}^T$, our memory recipe (4) amounts to

$$\mathbf{M} = \mathbf{Y}\mathbf{X}^T. \tag{5}$$

We require $\mathbf{M}$ to transform each input pattern into its corresponding output pattern, a requirement we can now write as $\mathbf{MX} = \mathbf{Y}$. Does recipe (5) meet this requirement? Yes; $\mathbf{MX} = \mathbf{YX}^T\mathbf{X} = \mathbf{Y}$ as desired, since the condition that the input patterns be orthonormal amounts to the condition $\mathbf{X}^T\mathbf{X} = \mathbf{I}$.

Now, what if we assume the inputs $|\text{in}_k\rangle$ are linearly independent but *not necessarily* orthonormal? We wish to map an input $|\text{in}_k\rangle$ as nearly as possible to the output $|\text{out}_k\rangle$, for each k. In matrix form this says $\mathbf{MX} \approx \mathbf{Y}$. Although $\mathbf{X}^T\mathbf{X} = \mathbf{I}$ is no longer true, the columns of $\mathbf{X}$ are linearly independent so it still has full rank; thus $\mathbf{X}^T\mathbf{X}$ is invertible. Accordingly, it is possible to define $\mathbf{X}^+ \equiv (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T$. This $\mathbf{X}^+$ is actually a *left* inverse, since multiplying this definition by $\mathbf{X}$ on the right reveals that $\mathbf{X}^+\mathbf{X} = \mathbf{I}$. This $\mathbf{X}^+$ is a special case of the *pseudoinverse* [**5**] of a matrix $\mathbf{X}$. Therefore, changing the associative memory storage rule (5) to

$$\mathbf{M} = \mathbf{YX}^+ \tag{6}$$

gives our neural net perfect recall: $\mathbf{MX} = \mathbf{YX}^+\mathbf{X} = \mathbf{Y}$, *despite* the nonorthogonality of the input vectors.

Where does this leave us? Apparently with a less confused neural net. This network is less likely to get muddled when learning a set of associations many of whose inputs are similar. Our dog/cat/rat example would have perfect recall, despite the patterns' lack of orthogonality. Unfortunately, however, the rule for modifying the synaptic weights is no longer local. The advantage of the James/Hebb rule for strengthening a connection was that the weight on a "wire" increased in proportion to its activity (real or expected) at both ends. This localization of the modification algorithm seemed plausible biologically, whereas the pseudoinverse approach appears less natural—less easy for either nature or engineers to implement.

Descent

Can we learn nonorthogonal associations in a more local, more natural way? Yes. To do so, we turn to one of the most basic search algorithms in all of applied mathematics: *steepest descent*. Suppose we want to store N associations indexed by $k = 1 \ldots N$. This requires constructing a linear transformation $\mathbf{M}$ such that $\mathbf{M}|\text{in}_k\rangle \approx |\text{out}_k\rangle$ for each k. We start by examining the following real-valued function E on the set of all linear transformations from $\mathbb{R}^n$ to $\mathbb{R}^p$:

$$E(\mathbf{M}) = \frac{1}{2}\sum_{k=1}^{N} ||\mathbf{M}|\text{in}_k\rangle - |\text{out}_k\rangle||^2$$

(the factor of 1/2 simplifies some later calculations). Note that this function does not depend on any basis. The value $E(\mathbf{M})$ is a nonnegative number that measures the *error* made by the neural net when tested on all N associations. We have $E(\mathbf{M}) = 0$ if and only if the net has stored all the associations perfectly. The domain of E is sometimes called "weight space," since each point $\mathbf{M}$ represents one assignment of weights to the synaptic connections.

To start our search for such a perfect $\mathbf{M}$, we begin with no connections at all: $\mathbf{M} = 0$. At this point $E(\mathbf{M})$ is certainly nonzero. Now we execute an iterative algorithm that modifies $\mathbf{M}$ in such a way as to minimize $E(\mathbf{M})$. Imagine walking around weight space searching for the point $\mathbf{M}$ that has the lowest error possible. To help us with our search, we compute the gradient $\nabla E(\mathbf{M})$:

$$\left[\nabla E(\mathbf{M})\right]_{ij} \equiv \frac{\partial}{\partial m_{ij}} E(\mathbf{M}).$$

We regard this gradient as a vector in $\mathbb{R}^{pn}$, which we identify with $\mathbb{R}^{p\times n}$, the set of all $p \times n$ matrices (which in turn represents all linear transformations from $\mathbb{R}^n$ to $\mathbb{R}^p$).

The gradient vector points "straight uphill," in the direction of the most rapid increase in $E(\mathbf{M})$. Since E is smooth, it decreases most rapidly in the opposite

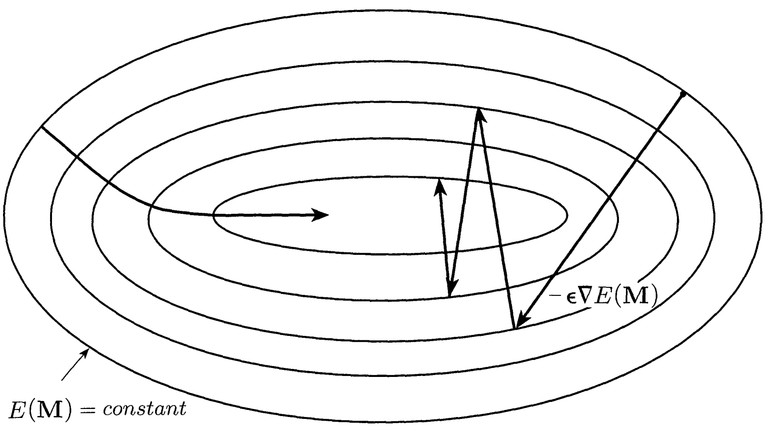


Figure 5. A walk in weight space, using steepest descent in the function E. The arrows, perpendicular to the contours, are $-\epsilon\nabla E(\mathbf{M})$ evaluated at various points $\mathbf{M}$.

direction—which is where we need to go: straight downhill, along $-\nabla E(\mathbf{M})$. The steepest descent algorithm requires us to take a *small* step downhill:

$$\mathbf{M} \to \mathbf{M} - \epsilon\nabla E(\mathbf{M}) \tag{7}$$

where $0 < \epsilon < 1$. The ϵ ensures a small step size, which is essential. As Figure 5 shows, each discrete step starts out perpendicular to the constant-E contours. Big steps would leave us zigzagging forever, never approaching the minimum. Alternatively, were ϵ infinitesimal our path would *always* be perpendicular to the contours: rolling smoothly and precisely downhill to the minimum—but taking an infinite number of steps.

As we apply the update (7) over and over to refine the synaptic connection matrix $\mathbf{M}$, the neural net comes closer and closer to perfectly storing the associations. To get the algorithm explicitly, we must evaluate the gradient. Write the error as a sum,

$$E(\mathbf{M}) = \sum_{k=1}^{N} E_k(\mathbf{M}),$$

where $E_k(\mathbf{M}) \equiv \frac{1}{2}||\mathbf{M}|\text{in}_k\rangle - |\text{out}_k\rangle||^2$. Since the ∇ operator is linear,

$$\nabla E(\mathbf{M}) = \sum_{k=1}^{N} \nabla E_k(\mathbf{M}).$$

To streamline the calculation, we temporarily write equations in the neuron basis, where $\mathbf{x}$ represents $|\text{in}_k\rangle$ and $\mathbf{y}$ represents $|\text{out}_k\rangle$:

$$\begin{aligned}[\nabla E_k(\mathbf{M})]_{ij} &= \frac{\partial}{\partial m_{ij}}\frac{1}{2}||\mathbf{Mx}-\mathbf{y}||^2 = \frac{\partial}{\partial m_{ij}}\left[\frac{1}{2}\mathbf{x}^T\mathbf{M}^T\mathbf{Mx} - \mathbf{y}^T\mathbf{Mx} - \frac{1}{2}\mathbf{y}^T\mathbf{y}\right] \\ &= \left(\sum_{r=1}^{n} m_{ir}x_r - y_i\right)x_j = \left[(\mathbf{Mx}-\mathbf{y})\mathbf{x}^T\right]_{ij}.\end{aligned}$$

Since the basis was arbitrary, we have

$$\nabla E_k(\mathbf{M}) = \left[\mathbf{M}|\mathrm{in}_k\rangle - |\mathrm{out}_k\rangle\right]\langle \mathrm{in}_k|,$$

so that the explicit form of the steepest descent modification step (7) is

$$\mathbf{M} \to \mathbf{M} - \sum_{k=1}^{N} \epsilon \left[\mathbf{M}|\mathrm{in}_k\rangle - |\mathrm{out}_k\rangle\right]\langle \mathrm{in}_k|. \tag{8}$$

It is natural to call the vector $\mathbf{M}|\mathrm{in}_k\rangle - |\mathrm{out}_k\rangle$ the *error vector* for pattern k, denoted by $|\mathrm{error}_k\rangle$. Then *our algorithm for storing the associations* $\{|in_k\rangle \to |out_k\rangle\}$ *can be* summarized as follows:

```
$\mathbf{M} := 0$
repeat
    for $k := 1$ to $N$ do begin
        $|\,\mathrm{error}_k\rangle := \mathbf{M}\,|\,\mathrm{in}_k\rangle - |\,\mathrm{out}_k\rangle$
        $\mathbf{M} := \mathbf{M} - \epsilon\,|\,\mathrm{error}_k\rangle\langle\mathrm{in}_k|$
    end
until ( $\mathbf{M}$ stops changing)
```

Try experimenting with this simple algorithm on a small set of low-dimensional patterns. For example, use *Mathematica* on the pattern matrices

$$\mathbf{X} = \begin{bmatrix} 3 & 0 & 1 \\ -1 & 2 & 0 \\ 2 & 1 & 5 \\ 0 & 5 & -3 \end{bmatrix} \qquad \mathbf{Y} = \begin{bmatrix} 1 & 0 & -1 \\ -1 & 1 & 0 \\ 1 & -1 & 0 \\ 0 & 1 & -1 \\ 1 & 0 & 1 \end{bmatrix}.$$

When will the iteration converge? Will it converge to the matrix that we would have obtained by the pseudoinverse technique (6)?

What is amazing in (8) is how the outer product has reappeared: As in the James/Hebb rule (4), we are making rank-one corrections to the neural network. Whereas before we implemented an association by adding $|\mathrm{out}_k\rangle\langle\mathrm{in}_k|$ in one fell swoop, now we subtract a small part of the correction $|\mathrm{error}_k\rangle\langle\mathrm{in}_k|$ over many little steps. It is as if we were training the neural network to associate inputs with (negative) error! The steepest descent algorithm serves as an iterative alternative to the pseudoinverse recipe (6). As with James/Hebb, the iterative accumulation of outer products gives us a local algorithm, so it has the advantage of being easily implemented in hardware. Since this is the main reason artificial neural networks are worth studying, we shall spell this out in agonizing detail.

For convenience, refer again to Figure 3. The weight $m_{ij} = \langle i|\mathbf{M}|j\rangle$ connects input line j to output line i. When we apply $|\mathrm{in}_k\rangle$ to the neural net, the jth input line carries the real number $x_j \equiv \langle j|\mathrm{in}_k\rangle$. In response, the neural net emits an output pattern $\mathbf{M}|\mathrm{in}_k\rangle$. Let $y_i^* \equiv \langle i|\mathbf{M}|\mathrm{in}_k\rangle$; this is the real number that is the ith neuron's response to input $|\mathrm{in}_k\rangle$. Now, had the net responded correctly, it would have emitted $|\mathrm{out}_k\rangle$, giving the ith output line the value $y_i \equiv \langle i|\mathrm{out}_k\rangle$ instead of y_i^*. The two assignment statements in the steepest descent algorithm above can be written in this basis as:

$$\langle i|\text{error}_k\rangle := \langle i|\mathbf{M}|\text{in}_k\rangle - \langle i|\text{out}_k\rangle$$
$$\langle i|\mathbf{M}|j\rangle := \langle i|\mathbf{M}j\rangle - \epsilon\langle i|\text{error}_k\rangle\langle \text{in}_k|j\rangle$$

Substituting the second into the first and using m_{ij}, x_j, y_i^*, and y_i, we have

$$m_{ij} := m_{ij} - \epsilon(y_i^* - y_i)x_j.$$

This means that when the neural net adjusts itself on the association $|\text{in}_k\rangle \rightarrow |\text{out}_k\rangle$, each connection modifies itself concurrently, based on the value of the signals (actual and desired) at both of its ends. It is precisely this feature that allows efficient hardware implementations: each connection can be given its own modification circuitry, operating independently of the other connections.

Since the net modifies itself by accumulating many small corrections, we could say that neurons in the neural network are *learning from experience*. Each time the net errs on an association, it revises its connection weights slightly to do better next time. In short, rank-one transformations embody the lessons of experience.

De Umbris Idearum

Here is the story so far. Based on intuitions gained from a simple vector-space formulation of associative memory, we constructed a hardware device that implements an associative memory. Although this hardware device is nothing but a matrix-vector multiplier, an irresistible analogy to neurophysiology compels us to call it a neural network (specifically, a *single-layer, linear* neural network). We can store associations whose input vectors are linearly independent, but not necessarily orthogonal, using an iterative algorithm. Iterative algorithms are not uncommon in numerical linear algebra, but here they invite another irresistible analogy: The net gradually *learns* from its experience, which consists of being corrected when it has not got an association quite right. Finally, each step of this algorithm involves local changes to connections, suggesting literal hardware implementations.

The number of researchers using association in neural networks as technology has exploded in recent years. The most popular algorithms nowadays work in *multilayer* networks of *non*linear neurons like the tanh neuron of Figure 2b; they are used to perform nonparametric regression in high dimensions [**2**]. Nevertheless, the key update steps in these algorithms are still the familiar rank-one corrections. For example, the *backpropagation algorithm* pushes an input pattern forward through a multilayered net, computes the output error, pushes the error signal *backward* through the network (multiplying errors by the transposes of the weight matrices), and adds small rank-one updates to each layer's weight matrix. This turns out to be just another version of steepest descent in error.

Programmed on traditional computers, these algorithms are computationally expensive. However, their simple architecture allows them to be implemented in a literal way with silicon or optical devices. On special-purpose hardware, hundreds of millions of weights can be updated in one second with the rank-one correction rule. Even in software simulations, neural networks enjoy the advantage of simplicity. Unlike artificial intelligence applications based on symbolic logic, which use an arcane repertoire of programming techniques—constraint logic, Horn clause logic, resolution/unification, and message-passing, to name a few—a neural network program can be written in any programming language by anyone with a semester of programming training. The most complicated data structures required are two-dimensional arrays.

Neural network algorithms have been employed in systems that learn to read handwriting, steer trucks, recognize the gender of human faces, pronounce English, diagnose heart attacks, and so on. Despite the ease with which a neural network program can be written, actually getting a neural network to "learn" something can be exasperating. It requires an engineer's knack for clever encoding of the input and output patterns, as well as the patience to experiment with different network parameters. References [**2**], [**3**], and [**8**] give an overview of the field of neural networks. The book by Kohonen [**6**] addresses neural associative memory more specifically.

One final topic deserves mention. Much recent research in artificial intelligence addresses using the low-level abilities of neural networks to represent higher-level hierarchical structures. Associations of associations, for instance, can be created by using pairings that are similar to tensor products. For example, the tensor product of two n-dimensional vectors $|\mathrm{house}\rangle$ and $|\mathrm{dog}\rangle$, written

$$|\mathrm{house}\rangle|\mathrm{dog}\rangle$$

is a vector in n^2-dimensional space. Suppose $\{|1\rangle, |2\rangle, \ldots, |n\rangle\}$ is a basis for the n-dimensional space. Then $\{|1\rangle|1\rangle, |1\rangle|2\rangle, \ldots, |1\rangle|n\rangle, \ldots, |n\rangle|1\rangle, |n\rangle|2\rangle, \ldots, |n\rangle|n\rangle\}$ is a basis for the tensor product space. We can view the tensor product as a kind of concatenation. This allows us to make associations of associations, using terms of the form

$$|\mathrm{dog}\rangle\langle\mathrm{fido}|\langle\mathrm{pet}|$$

in which the pattern $|\mathrm{pet}\rangle$ is associated with the association $|\mathrm{fido}\rangle \to |\mathrm{dog}\rangle$. These associations of associations can continue, building a hierarchy of concepts. The superposition property that arises from linearity can be used constructively, to trigger similar associations. Our use of the Dirac notation gives us a syntax for these superpositional representations. (The tensor product idea comes from Smolensky [**9**]. Plate [**7**] provides a clear summary and some extension of this idea.)

As noted earlier, the classical art of memory was elaborated throughout the Middle Ages and into the Renaissance. The repertoire of tricks for getting humans to memorize things remained small, however; there was scant progress in mnemonic technology (indeed, current self-help books on memory say little that was not known two thousand years ago). The art of memory became increasingly associated with religion and, later, the occult. By the sixteenth century, we find Giordano Bruno's book on memory, *De Umbris Idearum* ("On the Shadows of Ideas") situated at the cusp of magic and science. For Bruno, the places in the memorization algorithm are celestial, such as places on the zodiac, and they function as magical images to be imprinted on memory:

> By using magical or talismanic images as memory-images, the Magus hoped to acquire universal knowledge, and also powers, obtaining through the magical organisation of the imagination a magically powerful personality, tuned in, as it were, to the powers of the cosmos. [**12**]

In the past decade artificial neural networks themselves have been said to have near-magical powers. They are systems that "learn from experience." They are computers that "program themselves." They have been marketed as the answer to the prayers of frustrated workers in artificial intelligence who were overwhelmed by having to handcraft vast databases of everyday knowledge in order to build intelligent machines. The celerity with which the field has grown is astounding. But if we take Occam's Chainsaw to the wild growth of artificial neural network technology, we find one key concept at its core: *association as rank-one correction*. This humble notion sits today at the confluence of major streams of thought in psychology and technology—powerful, simple, and, above all, linear.

References

1. David Z. Albert, *Quantum Mechanics and Experience*, Harvard University Press, 1992.
2. James Anderson, ed., *Neurocomputing: Foundations of Research*, MIT Press, 1988.
3. Michael A. Arbib, ed., *The Handbook of Brain Theory and Neural Networks*, MIT Press, 1995.
4. Paul A. M. Dirac, A new notation for quantum mechanics, *Proceedings of the Cambridge Philosophical Society* 35 (1939) 416–418.
5. Dan Kalman, A singularly valuable decomposition, *College Mathematics Journal* 27:1 (January 1996) 2–23.
6. Teuvo Kohonen, *Self-Organization and Associative Memory*, Springer-Verlag, New York, 1984.
7. Tony Plate, Holographic reduced representations, *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, Morgan-Kaufmann, 1991, 30–35.
8. David Rumelhart, Bernard Widrow, and Michael Lehr, The basic ideas in neural networks, *Communications of the ACM* 37:3 (1994) 87–92.
9. Paul Smolensky, Tensor product variable binding and the representation of symbolic structures in connectionist systems, *Artificial Intelligence* 46:1/2 (1990) 159–216.
10. Anthony Sudbery, *Quantum Mechanics and the Particles of Nature: An Outline for Mathematicians*, Cambridge University Press, 1986.
11. Stephen F. Walker, A brief history of connectionism and its psychological implications, in A. Clark and R. Lutz, eds., *Connectionism in Context*, Springer-Verlag, New York, 1992.
12. Frances A. Yates, *Giordano Bruno and the Hermetic Tradition*, University of Chicago Press, 1964; p. 192.
13. ———, *The Art of Memory*, University of Chicago Press, 1966; p. 3.

———— o ————

A Postprandial Epiphany

I had a feeling once about Mathematics—that I saw it all. Depth beyond Depth was revealed to me—the Byss and the Abyss. I saw—as one might see the transit of Venus or even the Lord Mayor's Show—a quantity passing through infinity and changing its sign from plus to minus. I saw exactly how it happened and why the tergiversation was inevitable—but it was after dinner and I let it go.

Winston Churchill, *My Early Life*.
Contributed by Lynne Houston and Steve Rodi, Austin Community College.