CHAPTER 28

NEURAL NETWORKS, LEARNING, AND INTELLIGENCE

Bruce J. MacLennan

Crows' intelligence is astonishing. They solve problems, intelligently cache food, make and use tools, possess episodic-like memory, predict the behavior of individual conspecifics, and learn to recognize individual humans (Emery & Clayton, 2004). More generally, an animal's nervous system serves functions essential for its survival and reproduction, including communication of information throughout its body, control of behavior (voluntary and involuntary), perception and cognition, and learning and memory. This raises many questions: How are complex motor programs sequenced and controlled? How are skills learned and refined? How are episodic memories formed and retrieved? How is general information learned? How are mental categories formed? How does reinforcement learning occur? How are associations formed and retrieved? How do sensory systems function? Neural network models are important tools for understanding these and other neurophysiological processes.

Neural network models attempt to capture essential characteristics of neural information processing and control in models that balance simplicity and accuracy. On one hand, they can be used to understand and analyze processes in nervous systems and to suggest neuroscientific hypotheses. On the other hand, neural network models can be applied in artificial intelligence (AI), machine learning, and robotics to serve functions similar to biological neural networks. I can review only a few of the simplest neural network models in this chapter. A comprehensive introduction from a computational perspective can be found in Haykin (2008), and biologically realistic models are applied to cognitive science in O'Reilly, Munakata, Frank, and Hazy (2014).

HISTORY

In this section, I present a brief history of artificial neural networks. Many of the historically important articles can be found in Anderson and Rosenfeld (1988). The earliest artificial neural net model was described by Warren McCulloch and Walter Pitts in 1943. They used simple binary-valued model neurons that were capable of computing logical functions and therefore capable of implementing digital computation. Their use of binary-valued neurons was inspired by the "all-or-nothing" generation of the action potential (neural impulse).

In 1948, Alan Turing also published an artificial neural net model (Copeland & Proudfoot, 1996). In his B-type networks, model neurons were connected in arbitrary networks, with modifiable connections, which were set by means of control fibers. He discussed the possibility of an external teacher training a network by means of the control fibers.

The modern development of neural networks began with the invention of the perceptron by Frank Rosenblatt in 1957. In this model, artificial neurons are connected in feed-forward networks, that is, in layers, in which all the neurons in one layer are connected, via modifiable weights, to all the neurons in the next. Rosenblatt also developed a learning algorithm that allowed a single-layer

http://dx.doi.org/10.1037/0000011-028
APA Handbook of Comparative Psychology: Vol. 1. Basic Concepts, Methods, Neural Substrate, and Behavior, J. Call (Editor-in-Chief)

perceptron to adapt to solve any problem of which it was capable. The problem was that single-layer perceptrons are relatively limited, specifically to linearly separable categories. Multilayer perceptrons are much more powerful, but the development of a multilayer perceptron learning algorithm had to wait until the invention of the back-propagation algorithm by Paul Werbos in 1974 (which was not well-known before the mid-1980s). Other early work in neural networks includes Bernard Widrow and Ted Hoff's development of the Adaptive Linear Neuron (ADALINE) in 1960 and Many ADALINE (MADALINE) in 1962.

Research in artificial neural networks declined through the 1970s and early 1980s. One reason was the publication in 1969 of *Perceptrons* by Marvin Minsky and Seymour Papert, which demonstrated the limitations of the single-layer perceptron and implied that multilayer perceptrons would be similarly limited. Most AI research was devoted to symbolic AI, which focused on language-like representations of knowledge and models of cognition based on logical inference. There was little interest in relating these models of intelligence to the brain, because the brain was supposed to be equivalent, at a deep level, to a universal Turing machine and therefore to a digital computer.

Symbolic AI often adhered to a distinction between what in linguistics is termed *competence* and performance (Chomsky, 1965). *Competence* refers to an agent's abstract, idealized understanding of, for example, grammar, whereas *performance* refers to its ability to use that understanding in actual speech acts. Thus, a language's abstract grammar might permit unlimited nesting, but a human's ability to process nested structures might be strictly limited (e.g., by working memory capacity). Chomsky (1965) argued that the proper subject matter of linguistics was competence, not performance, and therefore that linguistics per se did not need to concern itself with the brain systems serving language.

The theory of universal computation pioneered by Turing was similarly focused on competence (the ability to compute a mathematical function) rather than performance (the real time and memory resources required to compute it). This justified ignoring the neural substrate of intelligence because, from a perspective of competence, all universal computing machines are equivalent.

The reawakening of interest in neural network models resulted in part from recognition of the central importance of performance issues in understanding intelligent behavior. This is illustrated by the "100-step rule": dividing a typical behavioral response time (less than 1 second) by a typical neuron response time (10 milliseconds) implies that there can be at most about 100 sequential processing steps from sensory input to motor output (Feldman & Ballard, 1982). At the very least, this observation implies that brains process information in a very different way from conventional computers: wide and shallow versus narrow and deep. Clearly, intelligence includes the ability to respond in a timely fashion, and so a theory of intelligence needs to be able to account for real-time response by nervous systems.

As progress in symbolic AI stalled around 1980 and as the performance challenges became apparent, cognitive scientists and AI researchers returned to the brain for insights. The new approach is commonly called *connectionism* (because knowledge resides primarily in the connections between neurons) and *(artificial) neural networks*; other terms, such as *neurocomputing* and *neuromorphic computing,* are also used, with slightly different semantic fields. These approaches, which are applied both to modeling biological neural systems and in AI, seek underlying mathematical principles of information processing and control that can be applied in both domains. They have also led to new approaches to traditional philosophical problems, especially in epistemology (e.g., the theory of universals). Many of the seminal articles have been collected in two volumes, Rumelhart and McClelland (1986) and McClelland and Rumelhart (1986).

Improvements in supercomputing technology have made feasible the simulation of neural networks with neuron numbers approaching the size of mammalian brains. The most popular models for these simulations have been spiking neuron models. Combined with increasing knowledge of neural connectivity, these advances will facilitate the simulation of realistic brain systems.

NEURON MODELS

Neural networks can be modeled on different levels depending on the intended application of the model and the issues it is intended to address. Principal among these are extended neuron models, spiking neuron models, and rate-based neuron models.

Extended Neuron Models

Extended neuron models are the most detailed models because they address the detailed morphology of the neuron and its effects on information processing. For example, they take account of the varying diameter and branching structure of dendritic processes, the distribution of ion channels, and the dynamics of ion flows and electrical potentials. These models are commonly compartmental, which means that the neuron is divided into small compartments of simple shape in which the relevant properties are approximately constant. For example, a dendrite can be treated as a number of cylinders of various lengths and diameters, which simplifies simulating dynamical processes within each compartment and combining them to simulate the entire neuron. These models can be computationally demanding.

Extended neuron models are able to address issues such as the filtering of inputs by a neuron's dendritic net and the detailed dynamics of the action potential (neural impulse). For example, dendrites have both electrical resistance and capacitance, which depends on their diameter and other physical parameters. This causes dendrites to act as low-pass filters, which smooth and extend the neural impulses received by the dendrites, which in turn affects the way these signals combine in the neuron soma (cell body) to generate action potentials (Branco, 2011). These models can address the effect on neuron response of the location and timing of neural inputs and the ability of dendritic nets to adapt to match particular input signals (MacLennan, 1994).

Spiking Neuron Models

Spiking neuron models seek a higher level of abstraction than extended models by ignoring the detailed spatial structure of the neuron and the detailed temporal structure of the action potential while retaining the temporal relation between action potentials. These models often incorporate a leaky integrate-and-fire neuron model, which means that arriving impulses each increment somatic membrane potential, which decays in time. If enough impulses arrive in a short enough time to drive the membrane potential over a firing threshold, then the neuron generates an impulse and resets its membrane potential to its resting level.

One advantage of spiking neuron models is that discrete event simulation methods can be used with them. Rather than simulating the network at every time step, these methods update the simulation state only when certain discrete events happen, processing these events successively in time. Therefore, the generation of an impulse is treated as a discrete event that occurs at a specific time and triggers later events at specific times, such as the arrival of the impulse at the soma. This later event triggers the computation of the membrane potential at that time, incorporating the arriving impulse, its comparison with the threshold, and possible generation of another impulse. This compromise between biological accuracy and computational efficiency has permitted some large-scale brain simulations.

Spiking neuron models permit addressing phase relations among action potentials, which may be important in primary visual cortex (Montemurro, Rasch, Murayama, Logothetis, & Panzeri, 2008) and coordinating neural communication (Fries, 2005).

Rate-Based Neuron Models

The remainder of this chapter focuses on rate-based models, which abstract away from the detailed timing of the neural impulses in favor of the instantaneous impulse rate, which seems to bear most of the information in many neural systems. One argument in favor of these models is that there are many stochastic factors in the generation of an action potential and in its transmission across a synapse and that these factors blur or obliterate any information that could be conveyed by temporal relations among impulses. Others have argued that the most appropriate unit for analyzing neural information representation and processing is the cortical minicolumn,

which consists of about 100 neurons (in humans). From this perspective, we should be looking at the inputs and outputs of minicolumns rather than individual neurons. Therefore, in discussing rate-based models, I refer to neural units and leave it open whether these are individual neurons or small assemblies, such as minicolumns, of interconnected neurons with similar inputs and outputs.

Because the firing rate of a biological neuron can vary continuously within limits, it is common in rate-based models to treat the activity of a unit as a bounded real number. The specific bounds used can vary depending on the neural network model, the neural systems being modeled, or simply mathematical or computational convenience. For example, activities can be constrained to real numbers between 0 and 1, with 0 representing the minimum rate and 1 the maximum rate. Or they can be constrained to real numbers between −1 and 1, with 0 representing the resting impulse rate. In some cases, we are concerned with only two levels of activity, low and high, which can be represented as binary activities, 0 and 1. Sometimes it is mathematically more convenient to represent low and high activity with bipolar values: −1 and +1.

The activity of a unit can be analyzed into two components, its net input and its activation function. The net input of a unit is the total stimulation it is receiving at a time, and it is analogous to the somatic membrane potential of a neuron. Excitation is modeled by positive numbers and inhibition by negative numbers. The activation function translates the net input into the output activity range of the neuron; this mapping is monotonic, that is, higher net inputs lead to higher activities and vice versa. If the unit is binary or bipolar, then the activation is a threshold or step function: For inputs above the threshold, it produces 1; for those below the threshold, it produces 0 (binary) or −1 (bipolar). For real-valued units, the activation function is a continuous "squashing function" that squeezes the output values into the required range ([0, 1] or [−1, +1]). Sometimes the activation function is omitted (i.e., it is an identity function), in which case the output of the neuron is its net input.

Biological neurons are connected by synapses, which modulate the influence of one neuron on another. They are excitatory if they tend to move the membrane potential closer to its firing threshold, and inhibitory if they have the opposite tendency. In rate-based models, we have the analogous concept of the connection weight from one unit to another. It can be interpreted as the average influence (excitatory or inhibitory) of the synapses connecting neurons corresponding to the first unit to those corresponding to the second. Therefore, weights are represented by real numbers, which may be positive (relatively more excitatory) or negative (relatively more inhibitory).

Suppose that $x_1, x_2, \ldots, x_n$ are the activities of n units providing inputs to some unit of interest. Suppose also that $w_1, w_2, \ldots, w_n$ are the corresponding connection weights from those units (see Figure 28.1). Then the simplest model of the net input y is the weighted sum of these activities:

$$y = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n = \sum_{j=1}^{n} w_j x_j.$$

If f is the activation function, then $f(y)$ would be the activity of the unit.

Neurons differ in the amount of excitation they require to raise their firing rates and in the amount of inhibition to decrease it. This can be modeled

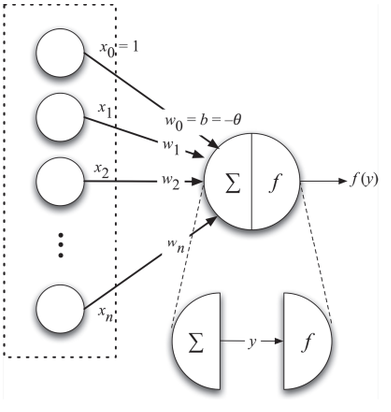

FIGURE 28.1. Typical artificial neural unit. The x_j are the activities of other neural units providing input to the large unit, and the w_j are the connection weights into this neuron. The weighted sum y is passed optionally through a nonlinear activation function f to yield the activity $f(y)$ of the neural unit. The x_0 input is a constant 1, which applies a bias b to the sum, which is equivalent to a threshold θ.

with neural units that have the same activation function but different thresholds. By subtracting the threshold θ from the net input, $f(y-\theta)$, the unit's response is made relative to the threshold. Sometimes we talk in terms of a bias *b* added to the net input rather than a threshold subtracted, for example, $f(y+b)$; it is just a difference in sign. For mathematical convenience, we frequently treat the bias as a "0th weight" that is wired to a unit providing a constant 1 input; that is, set $x_0 = 1$, $w_0 = -\theta$, and let

$$y = \sum_{j=0}^{n} w_j x_j.$$

It might seem counterintuitive at first, but it is convenient to treat the activities of groups of units and their weights as vectors in *n*-dimensional space. This permits geometrical visualizations that can illuminate neural information representation and processing. It is customary to use *n*-dimensional column vectors ($n \times 1$ matrices). Therefore, the connection weights to a unit can be treated as its weight vector $\mathrm{w} = (w_1, w_2, \ldots, w_n)^{\mathrm{T}}$ and its inputs as an input activity vector $\mathrm{x} = (x_1, x_2, \ldots, x_n)^{\mathrm{T}}$. Then the net input can be written as an inner product between the (transposed) weight vector and the input vector,

$$y = \mathbf{w}^{\mathrm{T}}\mathbf{x} = \sum_{j=0}^{n} w_j x_j.$$

(This is also known as a *scalar product* or *dot product* and written $\mathbf{w} \cdot \mathbf{x}$. Note that the product of a $1 \times n$ matrix $\mathbf{w}^{\mathrm{T}}$ and a $n \times 1$ matrix x is a 1×1 matrix, or scalar.)

If we think of these vectors as arrows pointing into *n*-dimensional space, then their length (or *Euclidean norm*) is given by $||\mathbf{x}|| = \sqrt{\mathbf{x}^{\mathrm{T}}\mathbf{x}}$, which is just the square root of the sum of the squares of the components of x. If we think of the weight and input vectors as arrows pointing into *n*-dimensional space, then we can express the inner product (and hence the net input) in terms of the angle φ between the vectors: $y = \mathbf{w}^{\mathrm{T}}\mathbf{x} = ||\mathbf{w}|| \cdot ||\mathbf{x}|| \cdot \cos\varphi$.

Because the cosine takes on its maximum value when $\varphi = 0^{\circ}$, this formula tells us that (other things being equal), the unit's net input will be maximized when the weight and input vectors are aligned. It also tells us that the net input will be negative to the extent that the weight and input vectors point in opposite directions ($90^{\circ} < \varphi \leq 180^{\circ}$). If they are orthogonal (at right angles), then ~~when~~ the cosine is 0, the net input is 0; the significance of this fact is addressed later. This geometric interpretation shows us that the weight vector is a sort of pattern and that the net input is greater to the extent that the input matches this pattern. Thus, these units (artificial neurons) act as simple pattern detectors.

Typically, we are interested in neural networks in which groups of units provide inputs to other groups of units. Therefore, let $\mathrm{x} = (x_1, x_2, \ldots, x_n)^{\mathrm{T}}$ be the activities of the sending units and $\mathrm{y} = (y_1, y_2, \ldots, y_m)^{\mathrm{T}}$ be the net inputs of the receiving units. Moreover, let w_{ij} be the weight of the connection to unit *i* from unit *j*. We have seen that the net inputs are expressed

$$y_i = \sum_{j=1}^{n} W_{ij} x_j,$$

which can be expressed compactly as the product of a weight matrix $\mathbf{W} = \left(W_{ij}\right)$ and the input vector x, that is, $\mathbf{y} = \mathbf{W}\mathbf{x}$. Expressing neural networks in terms of vectors and matrices allows us to apply many powerful mathematical tools, to use computers optimized for these operations, and to understand neural information processing in geometric terms.

ADAPTATION AND LEARNING

Neuroplasticity is a fundamental characteristic of nervous systems that allows animals to change their behavior as a result of experience. This section explains several basic neural network adaptation and learning mechanisms that illuminate both animal and machine learning.

Learning Paradigms

There are three primary means by which artificial neural nets adapt to perform some function, all inspired by neurological models: supervised (or error-driven) learning, reinforcement learning, and unsupervised (or self-organized) learning.

In supervised or error-driven learning, a neural network is trained to perform some function by presenting it with a sequence of desired input–output pairs. The network compares its actual behavior for a given input with the desired (or target) behavior for that input and then uses the difference between the two (the error) to guide the modification of its weights to decrease the error. This process can be compared with learning a motor skill, in which perceived errors in execution guide adaptation to decrease the error. Normally, in error-driven or supervised learning, the network is expected to be able to generalize in a useful way to novel inputs.

In reinforcement learning, a neural network is trained to perform some function by presenting it with a series of inputs, to which it responds. In each case, it is told whether the response was correct or not, but if it responded incorrectly, it is not told the correct response. Therefore, there is no explicit error signal to guide learning. This is analogous to reinforcement learning in animals, in which responses elicit a punishing or a rewarding stimulus (or, rather, an unexpected negative or positive stimulus). Again, the network is expected to generalize reasonably to novel inputs.

In unsupervised or self-organized learning, a neural network adapts its behavior without any explicit reinforcement or target information. Therefore, it is responding to the statistics of its input, for example by clustering inputs into salient groups. Such networks can model the self-organization of low-level feature detectors in primary sensory cortices, but also higher level representations of statistical structure (see the "Deep Belief Networks" section later in this chapter). I consider several examples of each learning paradigm.

Hebbian Learning

The most basic rule of neural network learning was proposed by Donald O. Hebb:

> When an axon of cell *A* is near enough to excite a cell *B* and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that *A*'s efficiency, as one of the cells firing *B*, is increased. (Hebb, 1949/2002, p. 62)

We now know that the process is more complicated than this, but Hebb's rule remains fundamental to neural network learning. One of the important discoveries was spike-timing-dependent plasticity, by which a synapse strengthens if the receiving neuron fires soon after the sending neuron (long-term potentiation) but weakens if it fires shortly before the sender (long-term depression; Bi & Poo, 1998). The effect is to strengthen synapses that are likely to have a causal role in firing the receiving neuron (see Chapter 25, this volume).

In the context of rate-based models, the weight connecting two neural units increases if their short-term joint activity is above a threshold and weakens it if it below the threshold. The threshold adapts over a longer period to balance the activity and inactivity of the receiver. Learning rules of this kind include the Bienenstock, Cooper, and Munro (1982) model and the eXtended Contrastive Attractor Learning model (O'Reilly, Munakata, et al., 2014), both of which are consistent with more detailed models (Urakubo, Honda, Froemke, & Kuroda, 2008).

A simple mathematical model of (supervised) Hebbian learning adjusts an interconnection weight based on the joint activity of the units it connects. This is expressed $\Delta w = \eta y x$, where x is the activity of the sending unit, y is the activity of the receiving unit, and η is the learning rate. Therefore, the weight will increase if the neurons are simultaneously active ($x, y > 0$) or simultaneously inactive ($x, y < 0$). Conversely, it will decrease if their activities are inversely related (i.e., one more active and the other less active). Intuitively, over time, the weight will adapt to reflect the correlation (positive, negative, or zero) between the sending and receiving units.

We can extend the foregoing observations to the connections from a group of sending units with activities $x_1, \ldots, x_n$ to a group of receiving units with activities $y_1, \ldots, y_m$ (see Figure 28.2). The change in the weight W_{ij} to unit i from unit j is $\Delta W_{ij} = y_i x$.

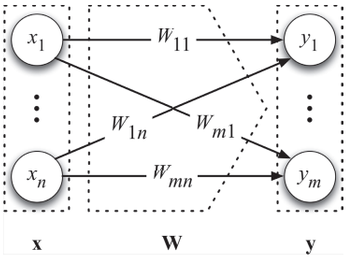

FIGURE 28.2. Linear associator. The input layer is fully connected to the output layer, which means that each input unit is connected to every output unit, but generally by different weights. W_{ij} is the weight to output unit *i* from input unit *j*. The activities y_i of the output units are weighted sums of the input unit activities x_j. In vector terms, y = Wx.

Therefore, the change in the weight matrix $\mathbf{W} = \left(W_{ij}\right)$ is given by the outer product $\Delta\mathbf{W} = \eta\mathbf{y}\mathbf{x}^{\mathrm{T}}$. (Note that the product of an $m \times 1$ matrix y by a $1 \times n$ matrix $\mathbf{x}^{\mathrm{T}}$ is an $m \times n$ matrix.) Therefore, the elements of W come to reflect the correlation of activity between corresponding sending and receiving units.

Suppose that a sequence of *P* pairs of input–output patterns $(x_1, y_1), (x_2, y_2), \ldots, (x_P, y_P)$ are successively imprinted on a weight matrix that is initially 0. Then we have the outer product rule for computing a weight matrix,

$$\mathbf{W} = \eta \sum_{p=1}^{P} \mathbf{y}_p \mathbf{x}_p^{\mathrm{T}}.$$

Clearly, W_{ij} will be proportional to the average value, over all the patterns, of the joint receiving and sending activities, $W_{ij} \propto \left\langle y_i x_j \right\rangle$, where the brackets denote the average.

If the mean activities of the units are 0 (i.e., $\left\langle x_i \right\rangle = \left\langle y_j \right\rangle = 0$), then the weight matrix will approximate the covariance matrix of the activities of the receiving and sending units, $\mathbf{W} \to \left\langle \mathbf{y}\mathbf{x}^{\mathrm{T}} \right\rangle = \mathrm{cov}(\mathbf{y}, \mathbf{x})$. Neural thresholds adapt to balance their average level of activity (O'Reilly, Munakata, et al., 2014), so it is reasonable to assume that their mean activity is zero.

The outer product rule implements a simple form of associative memory, known as a linear associator (see Figure 28.2). (If thresholds or biases are included in the weight matrix, then it is more correct to call it an affine associator.) To see this, suppose that $\mathbf{x}_k$ is one of the imprinted input patterns. If the activity of the sending units is $\mathbf{x}_k$, then we can compute the activity of the receiving units by matrix multiplication (setting $\eta = 1$ for convenience):

$$\mathbf{W}\mathbf{x}_k = \left(\sum_{p=1}^{P} \mathbf{y}_p \mathbf{x}_p^{\mathrm{T}} \right) \mathbf{x}_k = \sum_{p=1}^{P} \mathbf{y}_p \left(\mathbf{x}_p^{\mathrm{T}} \mathbf{x}_k \right).$$

That is, the resulting activity will be superposition of the imprinted output patterns $\mathbf{y}_p$, each weighted by the inner product of the corresponding imprinted input pattern $\mathbf{x}_p$ with the actual input pattern $\mathbf{x}_k$. That is, the strength of $\mathbf{y}_p$ in the result will be proportional to the similarity of $\mathbf{x}_p$ and $\mathbf{x}_k$. For convenience, suppose that the pattern vectors are normalized, $1 = \left\| \mathbf{x}_p \right\|^2 = \mathbf{x}_p^{\mathrm{T}} \mathbf{x}_p$. Then we can separate the effect on the receiving units due to $\mathbf{x}_k$ from that due to the other imprinted patterns:

$$\mathbf{W}\mathbf{x}_k = \mathbf{y}_k + \sum_{p \neq k} \mathbf{y}_p \left(\mathbf{x}_p^{\mathrm{T}} \mathbf{x}_k \right).$$

This shows us that, under the conditions specified, the linear associator will retrieve the pattern $\mathbf{y}_k$ associated with the input $\mathbf{x}_k$, possibly contaminated with crosstalk from the other patterns, which is represented by the summation. Retrieval will be perfect if $\mathbf{x}_p^{\mathrm{T}} \mathbf{x}_k = 0$ for $p \neq k$, that is, if the patterns are orthogonal. This condition may be easy to achieve in the high-dimensional spaces characteristic of biological neural networks (see the "Blessing of Dimensionality" section later in this chapter).

If the linear associator is presented with a pattern that has not been imprinted, then it will respond weakly, a result of the average crosstalk between the input and the imprinted patterns. Therefore, the linear associator can report whether a pattern is stored in the memory by responding strongly or whether it is novel by responding weakly. Because it keeps memory traces distinct, it is sometimes said to have separator dynamics, such as is characteristic of the dentate gyrus of the hippocampus (O'Reilly, Bhattacharyya, Howard, & Ketz, 2014; see also Chapter 25, this volume).

Sometimes separator dynamics is not what is required; rather, we want the linear associator to approximate some general input–output relationship represented by a set of training pairs $(x_1, t_1), (x_2, t_2), \ldots, (x_P, t_P)$. For example, an animal might be learning some specific sensory–motor correspondence. In this case, the performance of the linear associator can be improved by an error-driven learning algorithm called the *delta rule*. The idea is to adjust the weights in such as way as to minimize the sum of the differences between the target patterns and the corresponding retrieved patterns,

$$\sum_{p=1}^{P} \left\| \mathbf{t}_p - \mathbf{y}_p \right\|,$$

where $\mathbf{y}_p = \mathbf{W}\mathbf{x}_p$. It is most convenient to think of the sum-of-squares error as a function of the weights,

$$E(\mathbf{W}) = \sum_{p=1}^{P} \left\| \mathbf{t}_p - \mathbf{y}_p \right\|^2.$$

Then the weights can be changed in the direction of steepest descent of the error function, which is given by the negative gradient of the error with respect to the weights: $\Delta\mathbf{W} = -\eta\nabla\mathbf{E}(\mathbf{W})$. When the mathematics is worked out, the learning algorithm takes a very simple form: $\Delta W_{ij} = \eta(t_{pi} - y_{pi})x_{pj}$, or in matrix terms, $\Delta\mathbf{W} = \eta(\mathbf{t}_p - \mathbf{y}_p)\mathbf{x}_p^{\mathrm{T}}$. This rule adjusts the weights to make the actual outputs $\mathbf{y}_p$ closer to the corresponding targets $\mathbf{t}_p$ and leads to the best least-squares linear approximations $\mathbf{W}\mathbf{x}_p$ to the $\mathbf{t}_p$ (i.e., linear regression). Adaptive processes such as the delta rule may implement error-driven learning in the cerebellum.

Back-Propagation Learning

The linear associator is limited in its computational abilities (linear approximations or linearly separable classes). Therefore, algorithms have been developed for supervised training of more complex, multilayer neural networks, in which each layer provides input to the next and neural units have nonlinear activation functions. The first, and still very popular, algorithm is called *back-propagation* or the *generalized delta rule*; it is, in effect, the multilayer perceptron learning algorithm. This algorithm operates by propagating error estimates backward from the output layer, where the errors are measured, into earlier layers so that their weights can be adjusted. As with the delta rule, back-propagation is an example of a gradient (or steepest) descent learning algorithm. This means that partial derivatives of the error with respect to each of the individual weights are estimated, and these derivatives are used to adjust the weights in the direction of steepest descent. Backward propagation is required because the derivatives of later layers are needed to compute the derivatives of earlier layers. Although there is no evidence for back-propagation in biological neural networks, bidirectional connectivity between neural areas results in formally related learning processes (O'Reilly, Munakata, et al., 2014).

Competitive Learning

Competitive networks can implement self-organized adaptation. A typical competitive network has two layers: a first layer of feature detectors and second layer of mutually competitive units, corresponding, perhaps, to mutually inhibitory minicolumns within a single cortical macrocolumn (see Figure 28.3). Let

$$y_i = f\left(\sum_j W_{ij} x_j\right)$$

be the activity of a unit in the first layer, which reflects how well the input matches the pattern encoded in its weights. The first-layer units provide excitatory inputs to corresponding units in the second layer, which are self-exciting and mutually inhibitory. Therefore, whichever second-layer unit is most strongly activated will suppress the others and win the competition.

Adaptation occurs when the winning first-layer unit *k* adjusts its weights to more closely match the input that caused it to win the competition: $\Delta W_{kj} = \eta(x_j - W_{kj})$. (In biological neurons, adaptation could be triggered by backward propagation of the action potential into the dendritic tree.) Over time, units will divide the inputs into related clusters, with each competitive unit capturing one

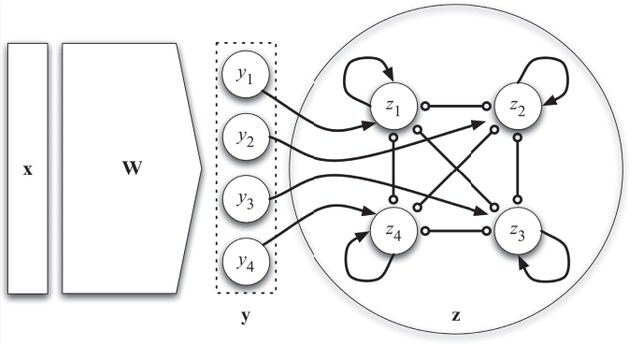

FIGURE 28.3. Competitive neural network. A competitive neural network classifies an input into one of several categories (four in this example). The first layer of connection weights W implements feature detectors corresponding to the categories. The activities y_j represent how well the input x matches each category. These units excite corresponding units z_i in the competitive layer, which are self-exciting and mutually inhibitory. As a result, one unit wins the competition, going to its maximum activity, and the other units are completely inactive. Through proper choice of parameters, it is possible to have *K* winners among the competitive units, rather than just one.

cluster and the unit's weights moving to the center of that cluster. Therefore, competitive adaptation is a kind of unsupervised clustering algorithm. It allocates neural units to represent its input space in a way that respects its statistics; in effect, the units compete to represent the input domain. This is one mechanism by which low-level feature detectors, such as orientation columns in primary visual cortex, can self-organize.

I have described a winner-takes-all form of competition. By adjusting the parameters, a specific number of winners to each competition can be arranged ("*K* winners take all"), which is useful for self-organized sparse representations of input and is more biologically realistic (see the "Sparse Distributed Representation" section later in this chapter).

Radial Basis Function Networks

Radial basis function networks have considerable biological relevance (Broomhead & Lowe, 1988). They are two-layer networks with a layer of feature detectors followed by a layer that computes a linear combination of their outputs. Each neural unit in the first layer computes a radial basis function of the inputs, that is, a function that responds maximally to a particular input pattern, and responds progressively less for inputs increasingly distant from this pattern. The usual artificial neuron model $y = f(\mathbf{w}^{\mathrm{T}}\mathbf{x})$ has this behavior and leads to a cosine-shaped tuning curve or receptive field, such as observed in many sensory and motor systems (e.g., Georgopoulos, Kalaska, Caminiti, & Massey, 1982; Salinas & Abbott, 1994). That

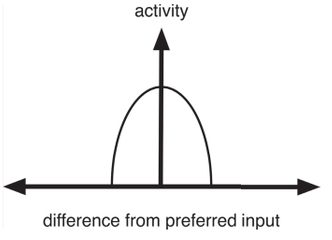

FIGURE 28.4. Example of cosine tuning curve. The horizontal axis represents the difference between the neuron's actual input and its preferred input. The neuron exhibits maximal activity for its preferred input, and its activity falls off approximately as the cosine of the difference between its actual and preferred input.

is, if a neuron has a preferred input to which it responds, then its response to other inputs tends to fall off with the cosine of its difference from the preferred input (see Figure 28.4). To perform adequately, the receptive fields of all the radial basis function units need to cover the input space adequately, which can be accomplished by competitive learning and similar self-organized adaptive processes.

The second layer of the radial basis function net is a linear associator applied to the first layer. The delta rule can be used for supervised training of the network to produce desired outputs. In effect, the first layer rerepresents the input in another space in which the desired behavior has a good linear approximation. This is a rapid, Hebbian process, which is capable of approximating any input–output relationship (Haykin, 2008).

Attractor Networks

Attractor networks model processes that complete patterns and satisfy constraints; the simplest such network is the bipolar Hopfield network (Hopfield, 1982). It consists of *n* neural units, each of which is bidirectionally connected to all the others (see Figure 28.5). That is, the connection weight from unit *i* to unit *j* is equal to that from *j* to *i*; that is, the weight matrix is symmetric ($W_{ij} = W_{ji}$). However, there is no "self action," that is, units do not provide inputs to themselves ($W_{ii} = 0$). The states of units are bipolar ($x_i \in \{-1, +1\}$). The units change state one at a time (typically in random order), and the new state x_i' of a unit is determined by applying a step function to its net input:

$$x_i' = \begin{cases} +1 & \text{if } y_i > 0 \\ -1 & \text{if } y_i < 0 \end{cases},$$

where $\mathbf{y} = \mathbf{W}\mathbf{x}$. If the net input is zero, it is conventional to leave the state unchanged, $x_i' = x_i$, but it can also be treated like a positive or negative input, without important differences.

The Hopfield network can be understood as a soft constraint satisfaction system. To see this, suppose that the units represent *n* interdependent yes–no (+1/−1) decisions and that the weights represent soft constraints between the decisions. That is, if a weight is positive, then it constrains the units it connects to make the same decision (to agree); if negative, to disagree. The magnitude of the weight represents the strength of the constraint, that is, the importance of satisfying it. Many cognitive tasks can be understood as soft-constraint satisfaction problems, including the inference of depth from binocular images, perceptual interpretation tasks, and memory retrieval.

The dynamics of the network causes a unit to change its state, if necessary, to satisfy the soft constraints, in effect decreasing the "tension" or "frustration" experienced by the unit. This idea can be made more precise. For a unit in state x_i experiencing net input y_i , we can quantify its tension by $-x_i y_i / 2$. (The one-half factor is for mathematical convenience and simply changes the units of tension.) This quantity will be positive (high tension) if the unit and its input have opposite signs (are in disagreement), and it will be negative (low tension) if they have the same sign (are in agreement). Then the total tension or *energy* in the network is

$$E = -\tfrac{1}{2}\sum_i x_i y_i = -\tfrac{1}{2}\mathbf{x}^{\mathrm{T}}\mathbf{y} = -\tfrac{1}{2}\mathbf{x}^{\mathrm{T}}\mathbf{W}\mathbf{x}.$$

Sometimes it is simpler to think in terms of the coherence or harmony of the network, which is simply the negative of its energy, $H = -E$.

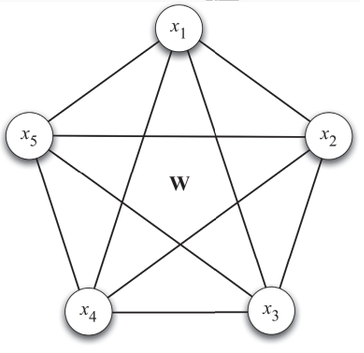

FIGURE 28.5. Hopfield network. In this (unrealistically simple) example, the network has five neural units, which are bidirectionally connected ($W_{ij} = W_{ji}$) with no self action ($W_{ii} = 0$).

It is easy to show that the energy of a network cannot increase ($\Delta E \leq 0$) and, in fact, that it will decrease with each state change. That is, each state change improves coherence by better satisfying all the constraints. In other words, the individual units are making microdecisions that increase the overall coherence of the network state (a macrodecision). Because energy cannot decrease forever, the network state must eventually stabilize in a local energy minimum (state of maximum local coherence). However, this need not be a global minimum energy state (absolute maximum coherence). These stable patterns of activity are called *attractors,* and therefore each attractor is surrounded by a basin of attraction, that is, a set of similar patterns that will settle into that attractor.

The Hopfield network can function as an associative memory. A pattern x can be imprinted on the memory by Hebbian learning, $\Delta \mathbf{W} = \eta \mathbf{x}\mathbf{x}^{\mathrm{T}}$. A series of patterns can be imprinted sequentially, or as a batch, using the sum-of-outer-products rule, $\mathbf{W} = \sum_{p=1}^{P} \mathbf{x}_p \mathbf{x}_p^{\mathrm{T}}$. The weights W_{ij} encode the correlated activity of the units *i* and *j* over all the imprinted patterns. If the number of imprints $P << n / 20$, then the imprinted patterns will be global energy minima and therefore attractors with relatively large basins of attraction (Amit, 1989).

If an imprinted Hopfield network is initialized to a pattern of activity, then its activity will evolve to the attractor in whose basin it was initialized. Therefore, the imprinted pattern acts as a prototype that represents all the patterns in its basin, with each basin corresponding to a distinct category. As a consequence, a Hopfield network can be used for pattern restoration; that is, if initialized to a degraded version of an imprinted pattern, it will evolve to the undegraded original. Missing or incorrect parts of the pattern are reconstructed from their correlations with other parts as reflected in the weights. More generally, the Hopfield memory can be used for pattern completion, the restoration of a whole pattern from some of its parts. Therefore, it can also be used for pattern association by imprinting it with composite patterns, because if it is initialized to a part of a composite pattern, it will evolve to the complete pattern. Attractor dynamics of this sort may be operating in area CA3 of the hippocampus, allowing a cue to settle into a complete episodic memory trace (see Chapter 25, this volume and Volume 2, Chapter 11, this handbook), which is then passed to CA1 (Knierim & Zhang, 2012; O'Reilly, Bhattacharyya, et al., 2014). Attractor networks have also been applied to modeling grid, place, and head-direction cells in the rat spatial navigation limbic system (Knierim & Zhang, 2012).

Imprinting patterns on a Hopfield network generates undesirable spurious attractors, which are linear combinations of the imprinted patterns. They can interfere with its operation because they have their own basins of attraction, which steal from the intended basins and therefore decrease the network's ability to complete or associate patterns. If the load factor, $\alpha = P/n$, on the memory is kept sufficiently low ($\alpha << 0.05$), then the spurious attractors will have shallower basins (higher energy) than the imprinted patterns and be less likely to interfere.

Operation of the Hopfield network can be improved by introducing a certain amount of indeterminacy into its operation: the stochastic Hopfield network. Instead of a unit's input absolutely determining its future state, it determines it probabilistically. That is, more strongly positive net inputs will make it more likely to go into the +1 state and more strongly negative inputs will make the −1 state more likely. This can be accomplished by "squashing" the net input *y* into the range [0, 1] so that it can be used as a probability, $\sigma(2y/T)$, where $\sigma(u) = 1/[1+\exp(-u)]$ is the logistic sigmoid function. This function has the value ½ at $y = 0$, approaches 1 as *y* becomes more positive, and approaches 0 as *y* becomes more negative. Thus, it can be used as the probability for a unit changing to the +1 state. The *T* parameter, which is called *pseudo-temperature* or *computational temperature,* controls the degree of random behavior. At $T = 0$, the behavior is completely deterministic, as before. At high *T* values, behavior is almost completely random, that is, activity has a 50–50 chance of being +1 or −1, regardless of the unit's input.

Intuitively, a higher pseudotemperature makes it more likely that a unit will make the "wrong" microdecision (go against its net input) and thus climb a little up the energy surface. This gives the network some possibility of climbing out of the basins of spurious attractors and finding its way into the larger, deeper basin of an imprinted pattern. The optimization process called *simulated annealing* starts at a high computational temperature, thus sampling the solution space in a relatively unbiased manner, and then gradually decreases the temperature, freezing the state into a global minimum with very high probability (Kirkpatrick, Gelatt, & Vecchi, 1983). A similar, gradual decrease of pseudo-temperature improves Hopfield memory retrieval. Stochastic behavior of this kind is to be expected in biological neural networks.

Deep Belief Networks

The Restricted Boltzmann Machine (RBM) is a neural network for the unsupervised discovery of an internal representation or model of some input domain (Hinton, 2010). A useful representation captures salient aspects of the statistics of the represented patterns. Therefore, one way to test and improve a representation is to see how well it does at reproducing the input statistics from the representation. An RBM has two groups of neural units, *m* for the internal representation (often called the *hidden units*) and *n* for the input to be represented (often called the *visible units*; see Figure 28.6). Each unit in one group is bidirectionally connected to every unit in the other group. That is, W_{ij} represents the connection weight to hidden unit *i* from input unit *j* and also the weight to input unit *j* from hidden unit *i*. (There are no connections among the input units or among the hidden units.) In addition, each unit has a bias, which is treated as a 0th weight connected to a constant 1 unit. The weights are initialized to small random values.

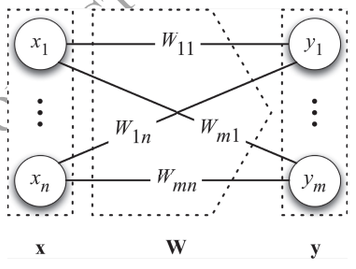

FIGURE 28.6. Restricted Boltzmann machine. The visible units x_j are bidirectionally connected ($W_{ij} = W_{ji}$) to the hidden units y_j, with no connections among the visible units or among the hidden units.

In the following, we use x_j for the activity of an input unit and y_i for the activity of a hidden unit. In the simplest RBM, the neurons are binary valued ($x_j, y_i \in \{0,1\}$), representing low or high levels of activity. The units are stochastic, as in the stochastic Hopfield network.

Self-organization proceeds as follows. As patterns of activity develop over the input units, they stimulate the hidden units, with a hidden unit's net input being given by

$$p_i = \sigma\left(\sum_j W_{ij} x_j\right),$$

where σ is the logistic sigmoid function, $\sigma(u) = 1/[1+\exp(-u)]$. The hidden unit will become active ($y_i = 1$) with probability p_i and will be inactive otherwise. Therefore, the input pattern probabilistically generates a pattern of activity y_i over the hidden units, which is a potential representation of the input. The activity of the hidden units then generates a reciprocal pattern of activity x'_j on the input units, which we can think of as an imagined input reconstructed from the internal representation. The probability that $x'_j = 1$ is given by

$$q_j = \sigma\left(\sum_i W_{ij} y_i\right).$$

The reconstructed input then generates another internal representation in which $y'_i = 1$ with probability

$$p'_i = \sigma\left(\sum_j W_{ij} x'_j\right).$$

The goal of the adaptive process is for the model statistics to agree with the input statistics, in particular for the model-driven correlations $\langle y'_i x'_j \rangle$ to agree with the input-driven correlations $\langle y_i x_j \rangle$. This is accomplished by updating the weights to bring the model closer to the correct statistics:

$\Delta W_{ij} = \eta\left(\langle y_i x_j \rangle - \langle y_i' x_j' \rangle\right)$. Generally, several cycles of input-driven and model-driven comparisons contribute to the averages (i.e., the weights adapt slowly). Self-organization stabilizes when the model-driven statistics agree with the input-driven statistics. A process such as this may be occurring in the bidirectional connections between the entorhinal cortex and CA1 in the hippocampus, since CA1 is developing a distinct memory trace by which it can reproduce a pattern of activity in entorhinal cortex. Theta waves may clock the alternation between input- and model-driven phases (O'Reilly, Bhattacharyya, et al., 2014).

Deep belief networks, which are inspired by the hierarchies of representations in mammalian sensory systems, have been successful in many application areas (Hinton, Osindero, & Teh, 2006). These are networks with multiple hidden layers between the input and output layers, which represent the input at successively more abstract, but task-relevant, levels (see Figure 28.7). RBM training is used at each level to develop the representations. For example, suppose a network has an input layer *I*, hidden layers $H_1, \ldots, H_N$, and output layer *O*. Inputs are applied to *I,* and RBM training is applied to *I* and H_1 to generate a model in their interconnecting weights and a higher order representation in H_1. Next, inputs are applied and passed through to H_1 and then to H_2. RBM training is applied, treating H_1 as the input layer and H_2 as the hidden layer, to generate a higher order representation in H_2. This process is continued until all the hidden layers are trained. Supervised training can be used to adjust the weights from the last hidden layer to the output layer to accomplish the network's purpose.

Deep belief network training can be used in conjunction with other deep (i.e., multilayer) networks. For example, back-propagation training of a deep network can be slow if all the weights are initialized randomly. The error has to propagate backward through many layers, its weights might have far to go in "weight space," and there are often many equally good sets of weights in different directions, so the network might have trouble committing to one direction in the early stages of training. For this reason, it can be advantageous to use deep belief network training to initialize the weights, which are then fine-tuned by back-propagation.

Learning Generals and Particulars

Two broad, complementary categories of adaptation are distinguished on the basis of their functions and dynamical properties (O'Reilly, Bhattacharyya, et al., 2014).

The first adaptive mechanism learns generalities and provides the basis for semantic memory. This is an integrative process of gradually learning statistical regularities in experience. Some of the processes are self-organized, such as the development of visual feature detectors in early vision areas. Neurons competitively organize to detect salient low-level statistical regularities in the environment. Higher order neurons adapt to regularities in the activities of lower level ones, leading to representations at successively higher levels of abstraction. These are slow, gradual processes to acquire statistically representative samples. These adaptive processes are modeled by neural networks such as competitive networks and reduced Boltzmann machines (see the "Competitive Learning" and

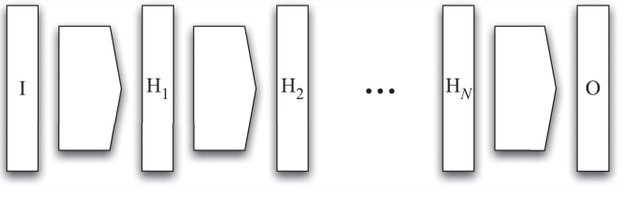

FIGURE 28.7. Deep belief network. A deep belief network is a cascade of restricted Boltzmann machines, which are trained in succession from the input layer I to the output layer O.

"Deep Belief Networks" sections earlier in this chapter).

Continuing experience, including positive and negative reinforcement, modifies the patterns of interconnection developed through self-organization. Sometimes more specific information than reinforcement is available for error correction, which allows directed adaptation by error-driven processes analogous (though not identical) to backpropagation (O'Reilly, Munakata, et al., 2014). These are the adaptive processes that allow us to refine perceptual and motor skills over time. Self-organized and error-driven learning of generalities seems to be the principal adaptive mechanism of the neocortex.

The second adaptive mechanism is memory for particulars, including episodic (or episodic-like) memory (memory of individual events), and memory for individual objects and places (see Volume 2, Chapters 11 and 21, this handbook). The principal characteristic of these separator memories is that individual memory traces are distinct and noninterfering; they are well separated in the space of possible traces. An additional characteristic is that these traces are imprinted quickly, often after only a single exposure, in contrast to semantic memory, which adapts gradually. We have seen that approximately orthogonal vectors have the characteristic of being distinct and noninterfering and that they can be imprinted quickly by Hebbian mechanisms. Moreover, because particulars differ from each other in many specifics, their representations are quite random and therefore approximately orthogonal (see the "Blessing of Dimensionality" section later in this chapter). Memory for particulars seems to be a principal function of the hippocampus (see Chapter 25, this volume and Volume 2, Chapter 21, this handbook).

Because of their distance, memory traces in a separator memory can have large basins of attraction, which means these memories can accomplish pattern completion. This is important for episodic memory, for example, because it means that a cue that activates part of a trace will allow the memory state to converge to the complete trace (i.e., allow the memory to be recalled).

A memory for particular items is capable of encoding quite arbitrary sequences of discrete states. Suppose that $e_1, e_2, \ldots, e_N$ is a sequence of (approximately orthogonal) neural state vectors representing distinct items (e.g., events, locations, or specific motor actions). Then the sequence can be encoded by associating each vector with its successor by Hebbian learning, for example,

$$\sum_{k=1}^{N-1} \mathbf{e}_{k+1}\mathbf{e}_k^{\mathrm{T}}.$$

If the items repeat in the sequence, then they can be encoded in pairs or larger subsequences. That is, the encoded pair $(\mathbf{e}_{k-1}, \mathbf{e}_k)$ associates with the pair $(\mathbf{e}_k, \mathbf{e}_{k+1})$, and so forth (Kanerva, 2009). The cerebellum may use a mechanism such as this to generate complex motor sequences. Error-driven learning can refine the timing and component actions of such sequences.

REPRESENTATION

A central issue in computing is how information is represented in memory. Similarly, the representation of information in nervous systems is critical to their ability to respond effectively in real time. Therefore, artificial neural network models are both inspired by neural representations in brains and in turn provide models for suggesting and testing hypotheses about natural neural information processing.

Coarse Coding

Neurons in the brain are generally quite broadly tuned; that is, although they respond maximally to certain input patterns, their response falls off gradually as the input diverges from this optimal input. Cosine-shaped tuning curves are often observed, which is to be expected if neuron assemblies are computing an inner product between their synaptic weight vectors and their inputs (see the "Rate-Based Neuron Models" section earlier in this chapter). As a consequence, a specific stimulus value is represented by activity in a large number of neural units, which individually do not determine the value

precisely, but collectively do. This is called *coarse coding*.

Sparse Distributed Representation

Moreover, mammalian brains seem to make extensive use of sparse distributed representations. They are distributed in that a stimulus may lead to widely scattered activity in a cortical area; information is not in general represented by the activity of one or a few neurons. However, high levels of inhibitory competition lead to representations that are sparse in that a stimulus activates a relatively small percentage of the neurons (15%–25%: O'Reilly, Munakata, et al., 2014). Because the number of neurons is large, there tends to be little overlap in activity for unrelated patterns; this is especially the case if individual neurons are computing essentially random conjunctions between the neurons providing their inputs. Random vectors in these high-dimensional spaces are approximately orthogonal (see the "Blessing of Dimensionality" section later in this chapter).

Although single-cell neural recordings can exhibit high specificity, such as responding to a specific person (e.g., Quiroga, Reddy, Kreiman, Koch, & Fried, 2005), this does not necessarily imply that such specific stimuli are represented by single neurons (so-called "grandmother cells"). On one hand, many other neurons are activated besides those being recorded. On the other hand, such recordings demonstrate specificity only among the stimuli presented, which for complex stimuli (such as people's faces) can represent only a small sample of possible stimuli.

Cortical Maps

Topographic maps are frequently used to represent information in brains. In these maps, dimensions of a sensory input space or a motor output space are systematically mapped to neural locations, so that nearby neurons represent nearby points in these spaces. For example, in retinotopic maps nearby neurons respond to nearby locations on the retina. In tonotopic maps, nearby auditory neurons respond to similar pitches, and neurons in somatotopic maps are organized according to bodily location. Topographic organization seems to be one of the principal means of information representation in nervous systems.

In contrast, in the neural networks discussed earlier in this chapter, the neural units have no significant spatial relations. In some networks, such as Hopfield networks, every unit is connected to every other unit (although the strength of connections varies and can in fact be 0, which is equivalent to no connection). In others, such as back-propagation and other deep networks, each unit in one layer is connected to every unit in the next layer. In general, there is no defined spatial relation among the neurons in any one layer.

There are, however, some neural networks that make use of the spatial organization of the units. By analogy with retinotopic maps in the brain, these networks are often applied to image processing, because it is usually significant if units are responding to nearby regions in an image. For example, a convolutional network applies the same transformation at every point in a two-dimensional image and produces a two-dimensional map of the results. Thus, by analogy with the retina, a convolutional network might apply an edge-detecting kernel at each point of an image to produce an image with enhanced edges. In contrast, a Gaussian kernel would blur the image. Convolutional neural networks are a way of extracting features from every location of an image while retaining their spatial relationships. Let the matrix K represent the kernel, where $K_{d,e}$ is the weight applied at a displacement (d,e) from the central point where the kernel is applied. Then a convolutional neural network applied to an image I computes an image J by

$$J_{x,y} = \sum_{d,e} K_{d,e} I_{x+d,y+e}.$$

Convolutional networks and other topographically organized neural networks can be hard wired, but they can also self-organize by mechanisms analogous to developmental processes in the brain. For example, a self-organized feature map can be implemented by combining a competitive mechanism with spatially localized learning (Kohonen,

1982). When a unit wins the competition and its weights adapt, spatially nearby units will also adapt, but to a lesser degree. The result will be that the competitive units self-organize into a map in which nearby units respond to nearby points in the input space (as determined by the distance measure). The weights can be further tuned by supervised learning, if necessary.

Nonlinear Computation in Topographic Maps

Topographic maps permit the computation of arbitrary transformations, subject to the resolution of the map (i.e., the number of neurons in it). To illustrate the process, begin with an unrealistically simple example. Suppose that a neural network needs to compute a function $v_k = F(u_k)$ for a finite number of possible inputs $u_1, \ldots, u_N$. If a single neuron in the input area responds to each possible input, then that neuron can project to a single neuron in the output area representing the corresponding output. Mathematically, let $\mathbf{u}_k$ be a vector of neural activities in which only the neuron representing u_k is active and likewise let $\mathbf{v}_k$ be a vector in which only the neuron representing v_k is active. Then the connection weights implementing F are computed by the sum of outer products

$$\mathbf{W} = \sum_{j=1}^{N} \mathbf{v}_j \mathbf{u}_j^{\mathrm{T}},$$

and we have $\mathbf{v}_k = \mathbf{W}\mathbf{u}_k$; this works because the vectors $\mathbf{u}_k$ are orthogonal.

More realistically, as in a radial basis function network, neurons are broadly tuned, and therefore a neuron will respond to a range of inputs, some more strongly, some less. As a consequence, a particular input will be represented by a population code, with individual neurons responding in accordance with how closely they are tuned to the input. Therefore, a particular input u will generate a pattern of activity x in which each x_k represents how strongly unit k responds to u, and x will be a weighted linear combination of the unit vectors:

$$\mathbf{x} = \sum_{k=1}^{N} x_k \mathbf{u}_k.$$

These weights will be passed through the linear associator and will weight the corresponding output values:

$$\mathbf{W}\mathbf{x} = \left(\sum_{j=1}^{N} \mathbf{v}_j \mathbf{u}_j^{\mathrm{T}} \right) \left(\sum_{k=1}^{N} x_k \mathbf{u}_k \right) = \sum_{j=1}^{N} x_j \mathbf{v}_j.$$

More generally, because the transformation is represented by the pattern of input–output connections, the activity of the neurons can represent pragmatic factors, such as the importance, urgency, or reliability of the input information, which is then transferred to the output representation (MacLennan, 1999).

BLESSING OF DIMENSIONALITY

Richard Bellman (1961) coined term the "curse of dimensionality" to refer to the computational problems arising from the exponential increase in the volume of space as the dimension of data increases. In the context of neural networks, more neural units mean more weights to be adjusted, and therefore slower learning. However, there is a complementary "blessing of dimensionality" (Donoho, 2000) in that certain things become easier in the very-high-dimensional spaces typical of biological nervous systems, which have been called *hyperdimensional spaces* (Kanerva, 2009). Hyperdimensional spaces have some unintuitive properties, to which I turn.

An unusual and paradoxical property of high-dimensional spaces is that as the dimension increases, an increasingly large fraction of the volume of an object is concentrated near its surface. Consider the example of an n-dimensional cube of width d; its volume is d^n. Next consider a shell of thickness $\varepsilon/2$ within this cube; the volume of the cube within the shell is $(d-\varepsilon)^n$. The fraction of the cube's volume that is inside the shell itself is then

$$\frac{d^n - (d-\varepsilon)^n}{d^n} = 1 - (1-\varepsilon/d)^n,$$

which approaches 1 as n increases; that is, the volume is concentrated in the shell, namely, within â/2

of the surface. Therefore, if we pick a random point uniformly within the volume, it is very likely to be near to the surface, and that probability increases with dimension. This property is not peculiar to *n*-dimensional cubes but also applies to *n*-dimensional spheres and other shapes.

The significance of the foregoing observations for neural computation is that if a vector is drawn randomly from a bounded, high-dimensional volume, then it is very likely to be near the surface of that volume. For example, if weight vectors are bounded in magnitude (e.g., $||\mathbf{w}|| \leq 1$), which is a reasonable assumption, then for large *n*, randomly chosen weight vectors will be approximately normalized (e.g., $||\mathbf{w}|| \approx 1$). Many neural network algorithms work better with approximately normalized vectors, which is therefore a reasonable assumption when the number of neurons is large and the vectors are random. This is one of the blessings of dimensionality.

We have seen that memory traces are less likely to interfere with each other if they are nearly orthogonal, that is, their inner products are small. However, with increasing dimension randomly chosen vectors are increasingly likely to be nearly orthogonal. I illustrate this in the case of bipolar vectors, but it is true more generally. Let u and v be two *n*-dimensional random bipolar vectors. Then their inner product

$$\mathbf{u}^{\mathrm{T}}\mathbf{v} = \sum_{i=1}^{n} u_i v_i$$

is a sum of *n* random ± 1s. This is like tossing a fair coin *n* times; the result is a binomial distribution with 0 mean and variance $\sigma^2 = n$. For large *n*, this is closely approximated by a Gaussian distribution with the same mean and variance. Because, for example, with 99.99% probability, the inner product will be within 4σ of the mean, with this probability we have $\mathbf{u}^{\mathrm{T}}\mathbf{v} < 4\sqrt{n}$, that is, $||\mathbf{u}||||\mathbf{v}||\cos\varphi < 4\sqrt{n}$. Because for bipolar vectors $||\mathbf{u}|| = ||\mathbf{v}|| = \sqrt{n}$, we conclude that with 99.99% probability the cosine of the angle between the vectors φ is less than $4/\sqrt{n}$, which decreases with increasing *n*. For example, with $n = 10{,}000$ (a small number of neurons in biological terms), the cosine will be less than 0.04 with 99.99% probability. In general terms, the probability that the cosine is less than ε is given by

$$\Pr\{|\cos\varphi| < \varepsilon\} = \mathrm{erfc}\left(\varepsilon\sqrt{n}/\sqrt{2}\right) \approx \tfrac{1}{6}\exp(-\varepsilon^2 n/2) + \tfrac{1}{2}\exp(-2\varepsilon^2 n/3)$$

(using the approximation to erfc from Chiani, Dardari, & Simon, 2003). Therefore, the probability of the vectors deviating from orthogonality by more than ε decreases exponentially with *n*, or, equivalently, the probability of any required degree of approximate orthogonality increases exponentially with dimension.

The import of the foregoing is that neural systems whose function requires keeping memory traces distinct and noninterfering (such as episodic memory) can do this by, in effect, assigning random codes in a hyperdimensional space. Random codes can result from neural representations composed of many random conjunctions of features. Such traces can be imprinted quickly by Hebbian learning and have large basins of attraction. Therefore, presentation of a partial trace can cue retrieval of the entire trace by pattern completion.

CONCLUSIONS

Biological neural networks can be modeled in various ways depending on the sorts of questions the models are intended to answer. In particular, rate-based neuron models can illuminate processes, including pattern recognition and categorization, pattern completion and association, and soft constraint satisfaction. Neural network models are also capable of various kinds of adaptation, including unsupervised learning, reinforcement learning, and error-driven learning. These models, which have varying degrees of fidelity to established neurophysiological processes, can facilitate understanding perception, learning, and motor control in nervous systems. They also illuminate principles of neural information representation and processing, including coarse coding, sparse distributed representation, topographic cortical maps, generalizing versus separating memories, and the blessing of dimensionality. Artificial neural network models also can be applied

to practical machine learning and to psychological data processing (e.g., clustering, pattern classification, regression).

References

Amit, D. J. (1989). *Modeling brain function: The world of attractor neural networks.* http://dx.doi.org/10.1017/CBO9780511623257

Anderson, J. A., & Rosenfeld, E. (1988). *Neurocomputing: Foundations of research*. Cambridge, MA: MIT Press.

Bellman, R. (1961). *Adaptive control processes: A guided tour*. http://dx.doi.org/10.1515/9781400874668

Bi, G. Q., & Poo, M. M. (1998). Synaptic modifications in cultured hippocampal neurons: Dependence on spike timing, synaptic strength, and postsynaptic cell type. *Journal of Neuroscience*, *18,* 10464–10472.

Bienenstock, E. L., Cooper, L. N., & Munro, P. W. (1982). Theory for the development of neuron selectivity: Orientation specificity and binocular interaction in visual cortex. *Journal of Neuroscience*, *2,* 32–48.

Branco, T. (2011). The language of dendrites. *Science*, *334,* 615–616. http://dx.doi.org/10.1126/science.1215079

Broomhead, D. S., & Lowe, D. (1988). Multivariable functional interpolation and adaptive networks. *Complex Systems*, *2,* 321–355.

Chiani, M., Dardari, D., & Simon, M. K. (2003). New exponential bounds and approximations for the computation of error probability in fading channels. *IEEE Transactions on Wireless Communications*, *4,* 840–845. http://dx.doi.org/10.1109/TWC.2003.814350

Chomsky, N. (1965). *Aspects of the theory of syntax*. Cambridge, MA: MIT Press.

Copeland, J., & Proudfoot, D. (1996). On Alan Turing's anticipation of connectionism. *Synthese*, *108,* 361–377. http://dx.doi.org/10.1007/BF00413694

Donoho, D. L. (2000). Aide-Memoire. *High-dimensional data analysis: The curses and blessings of dimensionality*. Palo Alto, CA: Department of Statistics, Stanford University. Retrieved from http://statweb.stanford.edu/~donoho/Lectures/AMS2000/Curses.pdf

Emery, N. J., & Clayton, N. S. (2004). The mentality of crows: Convergent evolution of intelligence in corvids and apes. *Science*, *306,* 1903–1907. http://dx.doi.org/10.1126/science.1098410

Feldman, J. A., & Ballard, D. H. (1982). Connectionist models and their properties. *Cognitive Science*, *6,* 205–254. http://dx.doi.org/10.1207/s15516709cog0603_1

Fries, P. (2005). A mechanism for cognitive dynamics: Neuronal communication through neuronal coherence. *Trends in Cognitive Sciences*, *9,* 474–480. http://dx.doi.org/10.1016/j.tics.2005.08.011

Georgopoulos, A. P., Kalaska, J. F., Caminiti, R., & Massey, J. T. (1982). On the relations between the direction of two-dimensional arm movements and cell discharge in primate motor cortex. *Journal of Neuroscience*, *2,* 1527–1537.

Haykin, S. O. (2008). *Neural networks and learning machines* (3rd ed.). Upper Saddle River, NJ: Prentice-Hall.

Hebb, D. O. (2002). *Organization of behavior*. Mahwah, NJ: Erlbaum. (Original work published 1949)

Hinton, G. (2010). *A practical guide to training restricted Boltzmann machines, version 1* (UTML TR 2010-003). Toronto, Ontario, Canada: Department of Computer Science, University of Toronto.

Hinton, G. E., Osindero, S., & Teh, Y. W. (2006). A fast learning algorithm for deep belief nets. *Neural Computation*, *18,* 1527–1554. http://dx.doi.org/10.1162/neco.2006.18.7.1527

Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences, USA*, *79,* 2554–2558. http://dx.doi.org/10.1073/pnas.79.8.2554

Kanerva, P. (2009). Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors. *Cognitive Computation*, *1,* 139–159. http://dx.doi.org/10.1007/s12559-009-9009-8

Kirkpatrick, S., Gelatt, C. D., Jr., & Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, *220,* 671–680. http://dx.doi.org/10.1126/science.220.4598.671

Knierim, J. J., & Zhang, K. (2012). Attractor dynamics of spatially correlated neural activity in the limbic system. *Annual Review of Neuroscience*, *35,* 267–285. http://dx.doi.org/10.1146/annurev-neuro-062111-150351

Kohonen, T. (1982). Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, *43,* 59–69. http://dx.doi.org/10.1007/BF00337288

MacLennan, B. J. (1994). Continuous computation and the emergence of the discrete. In K. H. Pribram (Ed.), *Origins: Brain and self-organization* (pp. 121–151). Hillsdale, NJ: Erlbaum.

MacLennan, B. J. (1999). Field computation in natural and artificial intelligence. *Information Sciences*, *119,* 73–89. http://dx.doi.org/10.1016/S0020-0255(99)00053-5

McClelland, J. L., & Rumelhart, D. E.; PDP Research Group. (1986). *Parallel distributed processing: Explorations in the microstructure of cognition: Vol. 2. Psychological and biological models*. Cambridge, MA: MIT Press.

Minsky, M., & Papert, S. (1969). *Perceptrons: An introduction to computational geometry*. Cambridge, MA: MIT Press.

Montemurro, M. A., Rasch, M. J., Murayama, Y., Logothetis, N. K., & Panzeri, S. (2008). Phase-of-firing coding of natural visual stimuli in primary visual cortex. *Current Biology*, *18,* 375–380. http://dx.doi.org/10.1016/j.cub.2008.02.023

O'Reilly, R. C., Bhattacharyya, R., Howard, M. D., & Ketz, N. (2014). Complementary learning systems. *Cognitive Science*, *38,* 1229–1248. http://dx.doi.org/10.1111/j.1551-6709.2011.01214.x

O'Reilly, R. C., Munakata, Y., Frank, M. J., & Hazy, T. E. (Eds.). (2014). *Computational cognitive neuroscience* (2nd ed.). Retrieved from http://ccnbook.colorado.edu

Quiroga, R. Q., Reddy, L., Kreiman, G., Koch, C., & Fried, I. (2005). Invariant visual representation by single neurons in the human brain. *Nature*, *435,* 1102–1107. http://dx.doi.org/10.1038/nature03687

Rumelhart, D. E., & McClelland, J. L.; PDP Research Group. (1986). *Parallel distributed processing: Explorations in the microstructure of cognition: Vol. 1. Foundations*. Cambridge, MA: MIT Press.

Salinas, E., & Abbott, L. F. (1994). Vector reconstruction from firing rates. *Journal of Computational Neuroscience*, *1,* 89–107. http://dx.doi.org/10.1007/BF00962720

Urakubo, H., Honda, M., Froemke, R. C., & Kuroda, S. (2008). Requirement of an allosteric kinetics of NMDA receptors for spike timing-dependent plasticity. *Journal of Neuroscience*, *28,* 3310–3323. http://dx.doi.org/10.1523/JNEUROSCI.0303-08.2008