Neural Networks and Modeling of Neuronal Networks

Bagrat Amirikian

Introduction

The past decades have seen an explosive growth in accumulation of experimental data in neuroscience research. The detailed anatomical and physiological data alone, however, are not enough to understand how the nervous system works. It is the recognition of this fact that makes modeling studies a significant part of mainstream research in neuroscience. The combination of theoretical methods, including mathematical analyses and computer simulations, together with modern experimental techniques has led to the emergence of a new discipline of *computational neuroscience* with the ultimate goal of explaining how neural signals represent and process information in the brain. Modeling of neuronal networks is a powerful tool that enables accomplishment of this goal by understanding how specific parts of the nervous system perform certain operations (for instance, learning specific motor skills, computing the direction of reaching movement, decoding spatial information, etc.) and is complementary to traditional techniques in neuroscience research.

Another field of modern science, often referred to as *neural computation*, is concerned with learning and computing in networks of artificial, neuron-like units. Though closely related to computational neuroscience, the field of neural computation differs fundamentally in its goal. One of the motivations for studying network computations is the fact that in many tasks the human brain outperforms even the fastest supercomputers available today. Inspired by the knowledge from neuroscience, the artificial neural networks realize an alternative computational paradigm to the classical one introduced by von Neumann. Its main concern is what the artificial networks can do to learn and implement a particular task. Thus, utilizing the idea of parallel and distributed processing, which is widely believed to be the way the brain operates, the field of neural computation does not try to be biologically realistic. What is the best hardware that solves the task? That is the question. In contrast, the field of computational neuroscience is interested in how the task is solved by the nervous system, i.e., how the biological hardware solves the task. Despite the difference in goals, the importance of interplay between modeling biologically plausible and artificial neural networks should not be underestimated. The ideas and concepts developed in one field drive the other, and vice versa.

The relationship between theory and experiment plays a particularly crucial role and creates a wide spectrum of approaches in the modeling of neuronal networks (Koch and Segev, 1989; Abeles, 1991; Marder and Abbott, 1995; Marder et al., 1997). Some models are heavily based on the anatomical and electrophysiological properties of the actual bi-

Bagrat Amirikian, Veterans Administration Medical Center, Brain Sciences Center, One Veterans Drive, Minneapolis, MN, 55417, USA (phone: +01-612-725-2282; fax: +01-612-725-2291; e-mail: amiri001@maroon.tc.umn.edu), University of Minnesota Medical School, Department of Physiology, 6–255 Millard Hall, 435 Delaware Street SE, Minneapolis, MN, 55455, USA

ological structures involved. Studies along this line usually proceed from the detailed description of single cells to the behavior of the network. This approach is most useful when accurate experimental data at the spatial and biophysical levels are available, the function of a neuronal network is already known, and the network itself is relatively small. Such models can determine whether existing data are sufficient to explain observed network behavior, and intend to pinpoint drawbacks and missing components in the model. The alternative to this *data-driven/bottom-up approach* is a *theory-driven/top-down approach.* Here, the emphasis shifts to descriptions of higher level functions such as a perceptual ability. Based on the theoretical analysis, an algorithm that performs the desired function is developed first and then embedded into the simplified network while imposing known biological constraints. This kind of approach tends to be more loosely bound to particular experimental data. However, by sacrificing specificity, the theory-driven approach attempts to address fundamental and puzzling questions, and can help in formulating and testing what kind of computational algorithms the brain is using in different tasks. In the long run, this approach is expected to suggest new experiments and research directions. Whereas the data-driven and theory-driven approaches represent two opposite extremes, there are varieties of other approaches that combine different proportions of the "abstract" and "realistic" components of the modeling and fill in the gap between these two extremes.

It is important to realize that any modeling, by definition, accepts *intentional simplification* of a real system. The reason for this is not merely the limit of computational power available today. Suppose that one would be able to explicitly simulate on a computer the nervous system at the level of every single ionic channel. This would not advance, even for an iota, our understanding of how the brain functions. The simplifying models are necessary to find out which properties of the system are crucial for a particular phenomenon and which are not. This could be achieved by incorporating into a model only those features that are most relevant to the phenomenon under investigation and omitting all other details that a modeler believes are less important. Such an approach raises a fundamental question appropriate for any phenomenological modeling in general and modeling of neuronal networks in particular. Namely, what is the relationship between the complexity of a model, i.e., its realism, and the credibility of the model, i.e., its predictive power? In other words, given two models that equally well explain the same data set, which model is preferable, the simple one or the more complex? *Occam's razor*, the *principle of parsimony*, suggests a criterion for selecting the credible model. Though its validity has never been proven in full generality, the parsimony principle has been incorporated into the methodology of science a long time ago and can be formulated as follows. If two explanations conform equally well to past observations, the simpler of the two has a better chance to predict future observations. The application of this general principle has been extremely useful in many areas of science. I strongly believe that the principle of parsimony is also appropriate for neuronal network modeling and should be exercised in order to achieve a reasonable compromise between tractability and realism. Based on this principle, the idealized strategy for developing *biologically plausible models* can be formulated as follows. Design the simplest neural network that incorporates a set of experimental data relevant to a phenomenon under investigation. If the network operation accurately simulates the phenomenon, then it is an appropriate model for studying that phenomenon. Biologically plausible models must not contain all the known features of the target system, they need to include only those features that are necessary to accurately simulate the phenomena under study. The model should be made more complex only when it contradicts new experimental observations related to the phenomena of interest.

The goal of this chapter is not to review all kinds of models currently used in neuroscience research. The significant theoretical work built around biological neural net-

works is so huge that to cover it all is perhaps impossible. Rather, I will concentrate on major concepts and procedures relevant to the modeling of large-scale neuronal networks thus conveying knowledge to a reader on "how to go on". For readers who would be further interested in the theory-driven approaches I recommend the monograph by Hertz et al. (1991) that focuses not so much on the level of detailed modeling as on the level of algorithms and representations. In contrast, the textbook by Anderson (1995) approaches to neural networks from a broad neuroscience perspective, with an emphasis on the biology behind the assumptions of the models, as well as on what the models might be used for. Finally, the link between the theoretical studies and experimental approaches is the main concern of the book by Koch and Segev (1989). It has an excellent collection of papers for those interested in biophysical mechanisms for computation in neurons.

Network Architecture and Operation

Units and Connections

Usually the first step in modeling a neuronal network is the definition of the network architecture. Depending on the complexity of a target system and the desired level of realism, each unit of the model network may simulate either a single neuron or a set of similar neurons with coherent functions and properties. The communication of activity from one cell to another is modeled by means of a connection between a corresponding pair of model units. The entire set of units and the pattern of connections between them define the architecture of the network. The realistic design of architecture requires knowledge of the underlying neural structure from neuroanatomical studies. When such data are not available, which is often the case, an *educated guess* based on other indirect electrophysiological studies could be useful.

Types of Units

Each unit can be classified as input, output, or hidden depending on the role that it plays in the operation of the network. *Input units* receive external signals that may represent sensory signals, signals from other networks, or some events in the external environment of an organism. The stimulation of input units may then change the activities of *hidden units* that they are connected to. This perturbation may further propagate across the whole network through connections to other hidden units and reach, ultimately, the *output units*. The role of the latter is to provide an input to another network or simulate events at the behavioral level, for example, a motor action. Depending on the network architecture and its interpretation the input, hidden, and output units may partially or completely overlap.

Types of Architecture

There are two prevailing network architectures that have been used in the theory and modeling of neural networks. Figure 1A shows an example of layered *feed-forward architecture*. The role of the input units is to feed external signals to the rest of the network. All connections are feed-forward. There are no connections between units in the same layer. Units in the intermediate layers are considered as hidden units, whereas the last layer represents the output units. These types of architecture are also known as *perceptrons* (Rosenblatt, 1962).

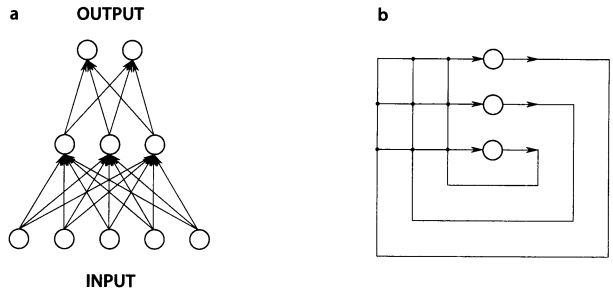

Fig. 1. Network architecture. Circles represent units whereas lines show connections between them. The arrow indicates the direction of a connection. *A*: Feed-forward network with one hidden layer. *B*: Fully recurrent network with three units.

Networks that are not strictly feed-forward, but include feedback connections are called *recurrent networks*. An example of a fully recurrent network is presented in Fig. 1B. In this architecture each cell receives inputs from, and sends its output to all other cells in the network. Unlike the feed-forward architecture, there is no explicit distinction between input, hidden and output units. Rather, in the framework of this architecture, each unit may play a single (input, hidden, or output), dual (input-hidden or hidden-output), or even triple (input-hidden-output) role.

Dynamical Rules

The specification of architecture, model neurons, and connections between them are necessary but not sufficient for the complete definition of network operation. The missing component that must also be provided is the dynamical rule that stipulates how and when the state of each neuron is updated. Once the network is fully specified its operation is to transform the input signals into output ones.

Model Neurons, Connections and Network Dynamics

McCulloch-Pitts Model

One of the first attempts to understand how the brain works can be traced back to Aristotle, the ancient Greek philosopher who lived more than 2000 years ago. However, the first mathematical models of a neuron, the elementary processing unit of the brain, were proposed relatively recently. The model suggested in the 1940s by Warren S. McCulloch, a neurophysiologist, and Walter Pitts, a mathematician, played a particularly critical role, and is often considered as the ultimate ancestor of all artificial neural networks. In the framework of their model (McCulloch and Pitts, 1943), the neuron is a simple *binary threshold element* operating in a *discrete time scale*, t=0, 1, 2,..., spending one time unit per processing step. The basic idea is that each neuron computes a weighted sum of activities of other units that have synaptic connections to it. The neuron then updates its output to either active or inactive state according to whether the sum is above or below a certain threshold. Formally, if at time t the activity of the jth neuron is $V_j(t)$, then the output of neuron i one time step later, $V_i(t+1)$, is given by

$$V_i(t+1) = H\left(\sum_j w_{ij} V_j(t) - \theta_i\right). \quad (1)$$

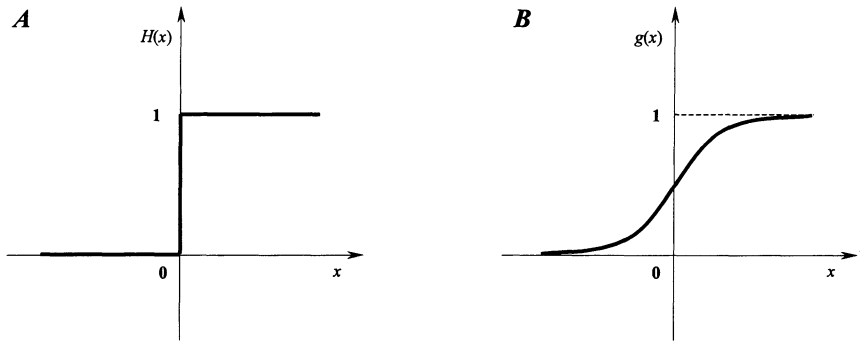

Fig. 2. Activation functions. *A*: The unit (Heaviside) step function. *B*: Sigmoid function.

Here the weight w_{ij} stands for the *strength of the synapse* connecting neuron j to neuron i. A positive w_{ij} corresponds to an *excitatory synapse* whereas a negative one to an *inhibitory synapse*. If there is no synaptic connection between neuron j and i, w_{ij} is set to zero. The parameter θ_i is a threshold value of ith neuron. The function is the unit (Heaviside) step function:

$$H(x) = \begin{cases} 1 & \text{if } x \geq 0; \\ 0 & \text{otherwise} \end{cases}, \tag{2}$$

see Fig. 2A. From Eqs. 1 and 2 it follows that the variable V_i can be either 1 or 0, thus representing the state of neuron i as active (V_i=1) or inactive (V_i=0).

Despite the simplicity of a single McCulloch-Pitts neuron, a network assembled from these units is computationally very powerful. McCulloch and Pitts showed that synchronous operation of a sufficiently large number of such units, when the connections between them are suitably chosen, could in principle perform any desired computation.

Hopfield (1982) suggested a similar binary threshold unit model of a neuron. However, the dynamical rule of network operation is different. Unlike the McCulloch-Pitts model, each Hopfield neuron updates its state asynchronously, that is at a random time, independently of any other neuron. Another difference is the interpretation of the binary states. According to Hopfield, the active state corresponds to "firing at maximum rate" whereas inactive to "not firing". In contrast, McCulloch and Pitts interpret the active state as the event of firing of a single spike rather than a prolonged firing.

Leaky Integrator Model

While discrete-time, binary-threshold models are still widely used in the theory of neural computation, they no longer play an active role in the field of computational neuroscience. Most of the accumulated experimental data demonstrate that average firing rate, and not individual spike times, correlate with external stimuli or behavioral variables ("rate code"). These observations justified the modeling of biological neurons as units that respond to their inputs in a continuous fashion by gradually changing their average firing rate. A simple way to incorporate this feature into the binary unit models discussed above is to substitute the step function $H(x)$ in Eq. 1 by a continuous nonlinear function $g(x)$, usually called the *activation function*. Because the firing rate of real neurons is bounded, the activation function usually has a sigmoid form, see Fig. 2B. These types of units are often called *graded response neurons* as opposed to binary units. The commonly used activation functions include $g(x) = 1/(1+\exp(-x))$ and $g(x) = \tanh(x)$. The

former varies between 0 and 1, whereas the latter between −1 and +1. Mathematical convenience rather than biological reasoning usually dictates the choice of a particular function. In any case, the lower and upper bounds of $g(x)$ may be offset and re-scaled to bring them into correspondence with the lower and upper bounds of the firing range of a given cell.

The graded response neurons, like binary-threshold units, may operate in a discrete time scale updating their states either synchronously or asynchronously. However, there is also a new possibility for operation in a *continuous time scale* (Cohen and Grossberg, 1983; Hopfield, 1984). Namely, all neurons continuously and simultaneously change their states according to dynamical rules given by a set of differential equations. The best-known example of such a model is Hopfield's (1984) network of graded response neurons. The basic idea is to represent each neuron in terms of its highly simplified equivalent electrical circuit. Then the dynamical rules that govern the time evolution of the network of such neurons are given by a set of resistance-capacitance charging equations, which can be transformed to the following system of coupled differential equations:

$$\begin{aligned} \tau_i \frac{du_i}{dt} &= -u_i(t) + \sum_j w_{ij} V_j(t) + u_i^0 \\ V_i(t) &= g(u_i(t)) \end{aligned} \quad . \tag{3}$$

Here, $u_i(t)$ is an internal state variable conceptually representing the cell's membrane potential, whereas $V_i(t)$ corresponds to the output activity of the cell in terms of the average firing rate. The time constant $\tau_i = R_i C_i$ depends on the resistance, R_i, and capacitance, C_i, properties of the cell membrane, and defines the time scale of the network dynamics. The constant u_i^0 is included for generality and represents a fixed external current to neuron i expressed in units of the potential. Neurons obeying dynamical rules described by Eq. 3 are known as *leaky-integrator neurons*. Such a name serves to emphasize the opposite contribution of the first, $-u_i(t)$, and the second, $\sum_j w_{ij} V_j(t)$, term on the right side of Eq. 3. If only the latter would be present, then the neuron would simply integrate its inputs:

$$u_i(t) = u_i(0) + \frac{1}{\tau_i} \int_0^t \sum_j w_{ij} V_j(x) dx. \tag{4}$$

The term $-u_i(t)$ provides a "leakage" of the potential, thus opposing the integration. At steady state, when the increase of the potential is compensated by its leakage, $u_i(t)$ ceases to change so $du_i/dt = 0$ for all i. The solution of Eq. 3 in this case gives the steady state output activity of cell i:

$$V_i = g\left(\sum_j w_{ij} V_j + u_i^0 \right). \tag{5}$$

Equation 5 shows that at steady state the relationship between the firing rate of the leaky integrator neuron and its net input is similar to the corresponding relationship for the McCulloch-Pitts neuron (cf. Eq. 1).

Integrate-and-Fire Model

Recent experimental investigations led to a view of neural coding that is quite distinct from the classical one based exclusively on average firing rates. They provide supporting evidence for the "temporal coding" based on the precise timing of single spikes fired by

a group of neurons in the same or different cortical areas (Fetz, 1997; Gerstner et al., 1997; and references in them). This controversy concerning neural coding represents one of the hottest topics in current neuroscience research. The adequate neuron model for investigating this issue would be the one that produces action potentials rather than continuously varying firing rate.

The simplest model that generates action potentials can be obtained by coupling the leaky integrator neuron to a firing threshold. The basic idea is to divide the operation of the neuron into two qualitatively distinct modes. First, the model neuron builds up its potential starting from a specific value u_i^{rst}, called the reset potential, by temporally integrating its inputs. This mode is described by an ordinary differential equation similar to Eq. 3:

$$\tau_i \frac{du_i}{dt} = -u_i(t) + U_i(t). \tag{6}$$

Here, $U_i(t)=R_i I_i(t)$, where $I_i(t)$ is the total synaptic current charging the spike emitting part of the cell, soma. Second, once the soma potential $u_i(t)$ reaches a specific threshold value u_i^{thr}, the cell instantaneously fires a spike and resets its potential to u_i^{rst}. After an absolute refractory period, during which the cell cannot fire spikes, the neuron restarts its operation in the first mode. Thus the outcome of the model is the alteration of the prolonged period of integration and instantaneous firing. Note that the action potentials have no structure in this model. Therefore the output train of cell i is completely described by the sequence of times $\{t_i^k, k=1,2,...\}$ at which spikes occur. This model, known in the literature as *integrate-and-fire neuron*, was introduced by Lapicque (1907), a neurophysiologist who first employed it in the calculation of firing times.

In the framework of integrate-and-fire models, there are several approaches to estimate the effective synaptic current $I_i(t)$ charging the soma. The dynamics of $I_i(t)$ depends on a set of synaptic time constants $\{\tau_{ij}^{syn}\}$. Each τ_{ij}^{syn} characterizes the temporal variation of the synaptic conductance of neuron i invoked by arriving spikes fired by neuron i. In the approximation $\tau^{syn} \ll \tau$, i.e., when the characteristic time of the synaptic current changes is much shorter than that of the charging of the soma, the effective total current $I_i(t)$ is represented by a sum of elementary contributions made at the time of arrival of individual spikes (Frolov and Medvedev, 1986; Amit and Tsodyks, 1991):

$$I_i(t) = \tau_i \sum_j w_{ij} \sum_k \delta(t - t_j^k - \Delta_{ij}), \tag{7}$$

where Δ_{ij} is the delay in the arriving time at synapse i of spikes fired by neuron j, $\delta(x)$ is the Dirac delta function. The strength of synaptic connection, w_{ij}, is expressed in units of the current.

Conductance-Based Models

A more realistic approach in representing the effective charging current $I_i(t)$ utilizes a conductance-based model that accounts for a variety of transmembrane ionic currents. In the framework of this approach, Eq. 6 is usually given in the following equivalent form:

$$C_i \frac{du_i}{dt} + I_i^{ion}(t) = 0. \tag{8}$$

Here, $I_i^{ion}(t)$ designates the net transmembrane ionic current, including the leakage. It is assumed that all ionic current flow occurs through membrane channels and the in-

stantaneous voltage-current relationship obeys Ohm's law. The ionic current through channels of a particular type is then given by a linear expression:

$$I_i^{chn}(t) = g^{chn}\left(u_i(t) - E^{chn}\right), \tag{9}$$

whereas the net ionic current $I_i^{ion}(t)$ is a simple sum of the currents through different types of channels: $I_i^{ion}(t) = \sum_{chn} I_i^{chn}(t)$. Here, g^{chn} is the conductance associated with a specific type of channel. The sign of the expression in Eq. 9, which indicates whether the current is outward or inward, depends on whether the membrane potential $u_i(t)$ is above or below the channel reversal potential, E^{chn}. It is usually assumed that E^{chn} does not explicitly depend on time or potential. The known ion channels can be divided into three distinct types: passive or leak, synaptic, and active. Depending on the type of the channel, the corresponding conductance g^{chn} may have a mathematical description that ranges from very simple to very complex. For example, the passive channels are represented by a constant (time- and voltage-independent) conductance. Other channels, such as those located at synapses, change their conductance to certain ions when the appropriate chemical agents (e.g., neurotransmitters or second messengers) bind to their receptors. Because the release of chemical agents is triggered by a presynaptic action potential, the conductance of the synaptic channels is modeled as a time-dependent but voltage-independent function that has a sharp peak at the spike arrival time. The active channels, which are the most complex from a modeling viewpoint, have conductances that are both voltage- and time-dependent. The model neuron that incorporates these types of nonlinear channels may produce membrane responses that mimic not only a subthreshold mode but also the generation of action potentials. Unlike the integrate-and-fire model, in which spikes are unstructured and discontinuous in time, here the spike generation occurs in a continuous-time fashion. Therefore, the model neuron of this type, often referred to as a *biophysical model*, may produce action potentials that have a shape similar to those observed in experiments. The best-known example of such a model, which played a crucial role for the development of biophysics of nerve cells, is Hodgkin and Huxley's (1952) description of initiation and propagation of action potentials in the squid giant axon.

Compartmental Approach and Realistic Modeling

The models we have considered so far are *single-point models* that disregard the underlying spatial structure of the neuron. The application of cable theory to nerve axons (Hodgkin and Rushton, 1946) and dendrites (Rall, 1959), as well as the introduction of a compartmental approach (Rall, 1964), made it possible to develop increasingly realistic models of a single neuron (cf. Chapter 8). Advanced biophysical models of this kind, which are trying to incorporate as much morphological and physiological data as possible, represent a neuron as a set of electrically coupled isopotential compartments. The basic assumption is that the continuously distributed system can be divided into small segments, called compartments. The geometry of compartments is modeled as an ellipsoid (soma) or cylinder (dendrites, axon and its branches) of various sizes. Electrically, each compartment is considered as isopotential and modeled as a resistance-capacitance pair. Adjacent compartments are connected by series resistances. It is assumed that nonuniformity in physical properties of the neuron (i.e., geometry, specific electric characteristics) and differences in potential occur between compartments rather than within them. From a modeling perspective, however, one must be aware that detailed biophysical models incorporate a vast number of adjustable parameters. While these models are adequate for studying in detail the behavior of a single neuron or a small

neural circuitry consisting of a few cells, their application to a large-scale network may be inappropriate. Such a network would be so complex that it would be impractical to simulate and analyze its behavior. By choosing a simpler model of a single neuron one may reach a reasonable (in the context of phenomenon of interest) compromise between tractability and realism.

Learning and Generalization

Basics

Any model, including a neural network model, has a set of *adjustable parameters*. Their values are usually determined by bringing the performance of the model into correspondence with a particular set of experimental data related to a phenomenon under study. The performance of a neural network, that is, the relation between the input signals and the activities of output units produced by the network, depends on several factors such as the network architecture, the number of model neurons, the strengths of synaptic connections between them, etc. Traditionally, in the field of neural networks, the adjustable parameters are associated with the strengths of synaptic connections, $\mathbf{w}=\{w_{ij}\}$, whereas other parameters are kept constant. This approach is consonant to numerous experimental observations which indicate that during a relatively short time period, a key mechanism by which biological neuronal networks change their behavior is the modification of the conductivity of preexisting synapses (i.e., modification of strengths of synaptic connections) rather than the variation of the number of neurons or formation of new connections between them (i.e., modification of the network size and architecture, respectively).

A fixed set of connection weights $\mathbf{w}$ corresponds to a specific input-to-output transformation task implemented by the network. As the values of the weights change, the same signals acting on the input units generate different activities of the output units. Therefore, by varying the w_{ij}'s one can implement different transformation tasks. This also means that the network *memorizes* the task that it implements in the set of synaptic connections $\mathbf{w}$. A key question in the theory of neural computation is concerned with the problem of *learning*: "How do we choose the connection weights so the network implements a specific task of interest?" The process of a systematic adjustment of the connection weights, the goal of which is to find a solution to this problem, is called training or learning and is described by a corresponding *learning algorithm*.

Generalization by Induction

The most common approach to network learning, which dates back to the pioneering work on perceptrons (Rosenblatt, 1962), takes the following form. The known examples of a particular transformation task to be learned are divided into two subsets: the training set and the testing set. The former is used to train the network to produce an appropriate output (in terms of the transformation task under consideration) for each example in the set by applying a specific learning algorithm. It is expected that after training, the network would generate correct (or nearly correct) responses to all examples in the training set. Next, one would like to check whether the network indeed has learned the transformation task or whether it has simply memorized examples in the training set. For that purpose, examples from the testing set are presented to the network. If the responses to the novel examples of the same task are correct, then it is said that a *generalization* has taken place. If, however, the number of correct responses is at a chance level, then there is no generalization.

It is the ability of neural networks to generalize by induction that originated much of the excitement about them. As was illustrated in numerous papers, the networks, indeed, could be astonishingly successful at generalization. However, this is not always the case. The key question is then: "Which properties of the networks and the tasks they are trained on determine the success of generalization?" A theoretical framework for generalization is usually formulated in terms of the probability that the trained network generates a correct output for a novel input as a function of the number of examples in the training set (Denker et al., 1987; Carnevali and Pattarnello, 1987; Anshelevich et al., 1989). In the framework of this consideration some general analytical results can be obtained. They provide an estimate of either the average probability (over all possible sets of connection weights that are consistent with the training examples; e.g., Schwartz et al., 1990; Levin et al., 1990) or the worst-case bounds of the probability (among those sets of connection weights that are consistent with the training examples; e.g., Blumer et al., 1989; Baum and Haussler, 1989). These kinds of theoretical analyses are important especially in the field of neural computation as they might provide general guidelines and tips for designing the network architecture optimal not only for generalization, but also for training time, cost of computations, etc. For example, it is widely believed that by limiting the number of units and the number of adjustable connections (thus embedding into the network as much information about the learned task as possible) one can reduce not only computational costs and possibly training time, but also improve generalization ability. It is important to realize, however, that these are merely general recommendations, not strict rules that guarantee a success in training a network on examples of a specific task. In fact, Amirikian and Nishimura (1994) demonstrated that appropriate rules for selecting the optimal architecture can be drastically different depending on what particular task has to be learned. Specifically, the authors addressed the question of how the generalization ability depends on the network size. It turned out that as the number of units in the network considered in Amirikian and Nishimura (1994) increases, the generalization of some tasks worsens whereas generalization of others improves. Therefore, the answer to the question of which network, small or large, is good for generalization will depend on the particular task to be learned.

Type of Learning Paradigms

Two general classes of learning paradigms are commonly distinguished: supervised and unsupervised. *Supervised learning* requires knowledge of correct answers to all examples in the training set. In this approach, which is also known as *learning with a teacher*, a direct comparison of the produced outputs with known answers provides a feedback to the network about any errors. The comparison is done in terms of an *error function* $E(\mathbf{w})$ (sometimes called *cost function* or *objective function*) that tells us how well the network performs on examples from the training set. Although there are many functions suitable for that purpose, there is one that is most commonly used. It is a simple quadratic function of the differences between the produced outputs and corresponding correct answers:

$$E(\mathbf{w}) = \frac{1}{2}\sum_{\mu}\sum_{i}\left(O_i^{\mu}(\mathbf{w}) - A_i^{\mu}\right)^2 \tag{10}$$

Here $O_i^{\mu}(\mathbf{w})$ is the value of the output variable i when example μ is presented to the network with a set of synaptic connections $\mathbf{w}$, and A_i^{μ} is the known correct value of the same variable. It is important to notice that, in the context of computational neuroscience, the output variables O_i^{μ} and their correct values A_i^{μ} given in the answers could

be given at the neuronal level (i.e., in terms of activities of the output units) or behavioral level (e.g., direction of motor action). The learning algorithm is an iterative procedure that adjusts synaptic connections based on the feedback error $E(\mathbf{w})$. Its ultimate goal is to find such a set of connections $\mathbf{w}$ that minimizes the error function. Thus, once a particular form of the $E(\mathbf{w})$ is chosen, the issue of supervised learning merely reduces to the solving of an optimization problem.

In practice, information supplied by the correct answers may not always be complete, that is, not all of the correct values of the output variables might be known. In the extreme case there is just a single bit of information specifying whether the output is right or wrong. Because in this case the feedback is only evaluative, it is often called *reinforcement learning* as opposed to *instructed learning*, which specifies what the correct output is.

Sometimes knowledge of correct answers to specific examples is not available at all and the learning goal is not explicitly defined. In such cases *unsupervised learning* algorithms, which do not require any feedback from the environment about the performance, could be useful. In the course of unsupervised learning the network is expected to find out by itself correlations, features, or regularities in the input data and represent them in the output in some appropriate way. In unsupervised learning, changes in synaptic connections are influenced only by local events (i.e., strength of a connection and activities of a pair of units that it links) whereas in supervised learning, due to the global character of the feedback error, the learning algorithm is affected by remote events (i.e., activities of the output units or behavioral consequences of that activity). There are two other points that are noteworthy. First, unsupervised learning can be useful only when examples in the training set are *redundant*. For example, if a network is required to learn similarities between presented examples and categorize them accordingly, the examples must indeed contain similar (though not the same) cases, i.e., be redundant. And second, unsupervised learning may be useful even when supervised learning would be possible. In particular, supervised learning, because it responds to remote events, sometimes could be extremely slow, while unsupervised learning that is, in contrast, affected by local events might be faster.

Supervised Learning

In supervised learning, the error function $E(\mathbf{w})$ can be regarded as a surface in a multidimensional space of connection weights w with a landscape consisting of some wells and hills. Figure3 illustrates this idea schematically. The goal of a supervised learning algorithm is then to find the deepest well on the *error landscape*. In contrast to all other wells that are called *local minima*, the deepest well is called a *global minimum*. As the number of connection weights in networks is usually large, the search for the global minimum, as a rule, is a hard computational problem. The point is that due to the highly nonlinear character of neural networks, the error landscape usually consists of many local minima. Though there are numerous optimization algorithms that find minima, detecting the absolutely best solution among them could be extremely time-consuming. In many applications, however, the practical difference between a very good solution (i.e., a very deep local minimum) and the absolutely best solution (i.e., the global minimum) is small. Therefore, a trade-off goodness of solution against computational costs makes practical sense.

Concerning the algorithm that finds minima, it could be either a general optimization algorithm or one that is specially dedicated to a specific architecture. *Simulated annealing* (Kirkpatrick et al., 1983) is an example of a general algorithm that is commonly used in supervised learning. It is based on ideas taken from statistical physics. In the framework of this approach, a fixed set of synaptic connections $\mathbf{w}$ is treated as the

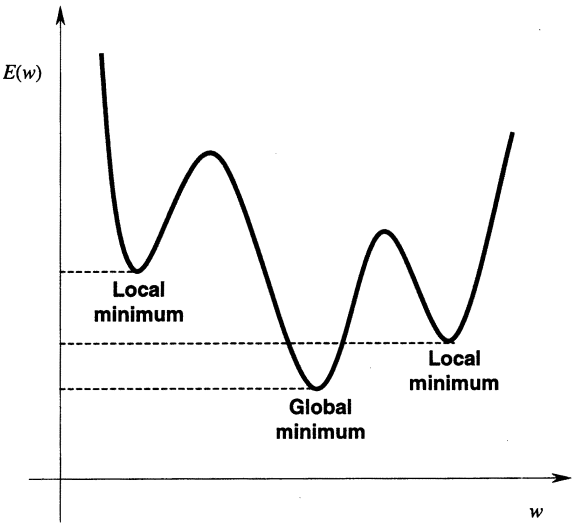

Fig. 3. Schematic illustration of an error landscape. In this case, among three minima only one is global.

"state" of the system while the error function $E(\mathbf{w})$ is treated as the "energy" of the system in that state. In these terms, our goal is to find the "ground state" of the system, that is, the lowest energy state. Search for the ground state is realized by means of an iterative procedure that adjusts synaptic weights. A typical simulated *annealing protocol* can be described as follows. Suppose the network is in a state $\mathbf{w}_1$ with energy $E_1=E(\mathbf{w}_1)$. During each cycle of iteration, a new state $\mathbf{w}_2$ of the system is picked up at random and its energy $E_2=E(\mathbf{w}_2)$ is computed. The energy change between these two states is given by $\Delta E=E_2-E_1$. If the new state has lower energy, i.e., when $\Delta E<0$, then the system unconditionally moves to the new state $\mathbf{w}_2$. If, however, the new state has higher energy, i.e., when $\Delta E>0$, then the system moves to the new state with the probability $\exp(-\Delta E/T)$ and remains in its previous state with the probability $1-\exp(-\Delta E/T)$. (In other words, the algorithm accepts with some probability even those changes in the synaptic connections that worsen performance of the network.) Here the parameter T has a meaning of "temperature". The probability of transition from one state to another is chosen such that the system eventually (after many cycles of iteration) reaches its equilibrium and obeys the Boltzmann distribution at temperature T. The key idea of this method, however, is not to keep the system at a constant temperature but rather gradually cool the system down. The simulated annealing procedure is initialized at a sufficiently high temperature, at which states with higher energies are easily accepted. As the temperature falls, the system state will be more and more likely to be in the lower energy states. If the cooling is slow enough for equilibrium to be established at each temperature, the ground state is reached in the limit of $T=0$. In practical application of simulated annealing, a major challenge is to determine the best *annealing schedule*. If the cooling rate is too fast, the system may be trapped and then freeze out in one of its local minima with relatively higher energy states. If, on the other hand, the annealing schedule is too slow, vast computational resources can be wasted. To achieve reasonable results in practice, a good deal of experimentation should be involved. An exponential schedule $T_k=T_0\exp(-\alpha k)$, where T_0 is an initial temperature, T_k is a temperature on the kth cycle of iteration, and α is a positive scaling constant, could be a good starting point. Another practical issue is how to choose a new state of the system. Although many different schemes are possible, the most commonly used and perhaps the simplest one is to select at random a connection weight and assign to it some new value.

Another well-known and widely used supervised learning algorithm is *back-propagation*. As often happens in science, it was invented independently several times (Wer-

bos, 1974; Parker, 1985; Le Cun 1985). However, the publication by Rumelhart et al. (1986) in the journal *Nature* had perhaps the strongest impact on the field of neural computation. Back-propagation belongs to a broader family of optimization algorithms known as *gradient descent* and is specifically dedicated to networks with feed-forward architecture consisting of graded-response neurons with differentiable activation functions. Later, the back-propagation approach was extended also to recurrent networks (e.g., Pineda, 1987).

The idea of gradient descent is conceptually much simpler than that of simulated annealing. The straightforward way to find a minimum on the error landscape is to check very many points $\mathbf{w}$ in the multi-dimensional space of connection weights and pick up the one that has the lowest value of the error function $E(\mathbf{w})$. In practice, this method is rarely used because the number of points in the space is usually astronomically large and the power of existing computers is far from being enough to solve the problem by this brute-force approach. Instead of exploring the weight space globally, a smarter algorithm would explore it locally, for example, by making small steps in several different directions and selecting the one that goes downhill on the error landscape. Iterative application of this procedure will eventually bring us to one of the nearest (with respect to the starting point) local minima in the error landscape. The gradient descent algorithm is even more clever because it says in what specific direction to move that will make the largest reduction in error at each iteration cycle. The gradient of the error function $E(\mathbf{w})$, often denoted $\nabla E(\mathbf{w})$, is the vector in the multi-dimensional weight space that points in the direction of the steepest ascent on the error landscape at the point $\mathbf{w}$. Correspondingly, the opposite vector $-\nabla E(\mathbf{w})$ points in the direction of the steepest descent. At the point of a minimum or maximum $\nabla E(\mathbf{w}) = 0$. The standard gradient-descent algorithm corresponds to sliding downhill along the direction of the largest decrease of the error function and suggests a simple update rule for the connection weights at the k th iteration cycle:

$$\mathbf{w}^{k+1} = \mathbf{w}^k - \eta \nabla E(\mathbf{w}^k), \tag{11}$$

Here η is a positive parameter often called *learning rate*. When a minimum is reached, $\nabla E(\mathbf{w}) = 0$ and connection weights $\mathbf{w}$ cease to change. The learning rate η together with the magnitude of $\nabla E(\mathbf{w})$ determines the length of the step along the gradient direction. The choice of η is critical to the operation of the method. If it is too small, the downhill sliding on the error landscape may be unacceptably slow. If it is too large, however, the minimum could be overshot.

Gradient-descent methods may be useful only when the cost of calculation of the gradient vector at each iteration cycle is relatively low. Utilizing architectural constraints of feed-forward networks, back-propagation scheme, as was suggested in Rumelhart et al. (1986), provides an elegant and cost-effective way to calculate all the components of the vector $\nabla E(\mathbf{w})$. The weights $\mathbf{w}$ are updated in a sequential procedure that starts from the connections feeding the top output layer and proceeds down, layer by layer, until the bottom input layer is reached. Thus, while the connections of the network propagate signals forwards (see Fig. 1A), the corrections of the weights caused by the output error $E(\mathbf{w})$ propagate backwards. Hence the name – "error back-propagation" or simply "back-propagation". Although the rediscovery of back-propagation in 1986 provoked an explosion in neural network studies, the algorithm itself is not the ultimate solution to neural network training. Perhaps one of the most serious difficulties is its speed. Despite the fact that calculation of the gradient is relatively fast, in practical applications the rate of convergence toward a minimum is exceedingly slow. Furthermore, like any other gradient-descent algorithm, the standard back-propagation gets stuck in one of the local minima. Therefore, a problem is usually solved many times from random starting weights until a satisfactory solution is found. These and some other difficulties of

gradient-descent methods, in general, and back-propagation, in particular, are considered in Anderson (1995). It is worth noticing, however, that back-propagation has been extensively studied in the past decade, and many variations and modifications of the standard algorithm have been suggested to cope with some of these difficulties (see, e.g., Hertz et al., 1991).

Unsupervised Learning

In unsupervised learning there is no teacher. The information available to a synaptic connection during the training phase is local and very limited. Specifically, it knows only its own state, i.e., the strength of connection w_{ij}, and the states of the two neurons that it links, i.e., their activities V_i and V_j. Therefore, the unsupervised learning algorithm, which modifies the strength of synaptic connections, can be expressed, in a very general form, as

$$\frac{dw_{ij}}{dt} = f\left(w_{ij}, V_i, V_j\right), \tag{12}$$

where f is an arbitrary function. In a typical unsupervised learning protocol, an example is drawn from the training set and presented to the network. The synaptic connections are then adjusted according to the learning algorithm given by Eq. 12. This procedure is repeated until all examples from the training set are shown. In practice, the time t in Eq. 12 is a discrete rather than continuous variable and corresponds to an interval in the training phase when an individual example is presented. Accordingly, Eq. 12 takes a form that specifies the changes Δw_{ij}, often called *learning rules*, that occur during that interval.

The choice of learning algorithm f is usually based on intuitively plausible suggestions. Consider a simple case in which the synaptic connection is changing only as a function of its own state:

$$\frac{dw_{ij}}{dt} = -\alpha w_{ij}. \tag{13}$$

Solution of Eq. 13 is given by $w_{ij}(t)=w_0\exp(-\alpha t)$, where w_0 is a value of the synaptic weight at time $t=0$, and α is a positive parameter. This, in fact, represents the case of an exponential decay of memory in the absence of adjacent neuronal activity or, in other words, a *forgetting*. The rate of this process is determined by the parameter α.

In order to learn something useful, the exponential decay must be overridden by some supportive signal that may come from either of the neurons participating in the synaptic connection. A simple intuition, based on the life experience of "strengthening by use", suggests reinforcing the synapse when the pre-synaptic neuron is active. The main drawback of this approach, however, is its lack of selectivity: the change in the connection will occur whatever the activity of the post-synaptic neuron is. Therefore another intuitive idea, "strengthening by coincidence", seems more appropriate. The first person who explicitly phrased that learning involves the activities of both connected neurons was Hebb (1949). In particular, he hypothesized that concurrent firing in the pre- and post-synaptic neurons strengthens a synaptic connection. This idea, known as *Hebbian learning*, can be formalized by adding a second term in Eq. 13 proportional to the product of the activities of the involved neurons:

$$\frac{dw_{ij}}{dt} = -\alpha w_{ij} + \eta V_i V_j. \tag{14}$$

Here π is a positive parameter that controls, as previously, the learning rate. It is worth noting that most of the unsupervised learning algorithms are based on Hebbian learning or use its various modifications.

So far we have considered networks in which multiple output units can be active together. In the context of some tasks, however, only one output unit should be active at a time. The units compete for being the one to fire, and are therefore often called *winner-take-all units*, whereas the learning algorithm that ensures such a behavior is called *competitive learning* and takes the following form:

$$\frac{dw_{ij}}{dt} = \eta V_i \left(V_j - w_{ij}\right). \tag{15}$$

Interestingly, the competitive learning algorithm given by Eq. 15 can be obtained from Hebbian learning with decay by a formal substitution $\alpha = \eta V_i$ (cf. Eq. 14). At first glance, it is not obvious why Eq. 15 corresponds to a competitive process. The underlying idea is that if the ith unit is a winner, then its activity $V_i \approx 1$. If, however, it is a loser, $V_i \approx 0$. Therefore, the learning algorithm given by Eq. 15 changes connections to the winner and does not change connections to losers.

Finally, unlike supervised learning algorithms, unsupervised learning does not perform any explicit optimization. In some cases, however, there is a well-defined quantity (e.g., *information content* or variance of the output) that is being maximized indirectly while a learning algorithm specified in the form of Eq. 12 is applied (for some examples see Hertz et al., 1991).

Acknowledgement: I would like to thank Apostolos Georgopoulos for critically reading the manuscript and for suggestions. Without Dr. Georgopoulos' help and encouragement this work would have been impossible. This work was supported by the American Legion Brain Sciences Chair and the Department of Veterans' Affairs.

References

Abeles M (1991) Corticonics: Neural circuits of the cerebral cortex. Cambridge University Press, Cambridge New York Port Chester Melbourne Sydney

Amirikian B, Nishimura H (1994) What size network is good for generalization of a specific task of interest? Neural Networks 7:321–329

Amit DJ, Tsodyks MV (1991) Quantitative study of attractor neural network retrieving at low spike rates I: Substrate – spikes, rates and neuronal gain. Netw Comput Neural Syst 2:259–273

Anderson JA (1995) An introduction to neural networks. MIT Press, Cambridge London

Anshelevich VV, Amirikian BR, Lukashin AV, Frank-Kamenetskii MD (1989) On the ability of neural networks to perform generalization by induction. Biol Cybern 61:125–128

Baum EB, Haussler D (1989) What size net gives valid generalization? Neural Comp 1:151–160

Blumer A, Ehrenfeucht A, Haussler D, Warmuth M (1989) Learnability and the Vapnik-Chervonenkis dimension. J ACM 36:929–965

Carnevali P, Patarnello S (1987) Exhaustive thermodynamical analysis of boolean learning networks. Europhys Lett 4:1199–1204

Cohen MA, Grossberg S (1983) Absolute stability of global pattern formation and parallel memory storage by competitive neural networks. IEEE Trans Syst Man Cyber SMC-13:815–826

Denker J, Schwartz D, Wittner B, Solla S, Howard R, Jackel L, Hopfield J (1987) Large automatic learning, rule extraction, and generalization. Complex Syst 1:877–922

Fetz EE (1997) Temporal coding in neural populations? Science 278:1901–1902

Frolov AA, Medvedev AV (1986) Substantiation of the "point approximation" for describing the total electrical activity of the brain with use of a simulation model. Biophys 31:332–337

Gerstner W, Kreiter AK, Markram H, Herz AVM (1997) Neural codes: Firing rates and beyond. Proc Natl Acad Sci USA 94:12740–12741

Hebb DO (1949) The organization of behavior. Wiley, New York
Hertz J, Krogh A, Palmer RG (1991) Introduction to the theory of neural computation. Addison-Wesley, Redwood City
Hodgkin AL, Huxley AF (1952) A quantitative description of membrane current and its application to conduction and excitation in nerve J Physiol Lond 117:500–544
Hodgkin AL, Rushton WAH (1946) The electrical constants of crustacean nerve fiber. Proc Roy Soc Lond B 133:444–479
Hopfield JJ (1982) Neural networks and physical systems with emergent collective computational abilities. Proc Natl Acad Sci USA 79:2554–2558
Hopfield JJ (1984) Neurons with graded response have collective computational properties like those of two-state neurons. Proc Natl Acad Sci USA 81:3088–3092
Kirkpatrick S, Gelatt CD, Vecchi MP (1983) Optimization by simulated annealing. Science 220:671–680
Koch C, Segev I (1989) Methods in neuronal modeling: From synapses to networks. MIT Press, Cambridge London
Lapicque L (1907) Recherches quantitatifs sur l'excitation electrique des nerfs traitee comme une polarizsation. J Physiol Pathol Gen Paris 9:620–635
Le Cun Y (1985) Une procédure d'apprentissage pour réseau à seuil assymétrique. In: Cognitiva 85: A la frontière de l'intelligence artificielle des sciences de la connaissance des neurosciences. CESTA, Paris, pp 599–604
Levin E, Tishby N, Solla S (1990) A statistical approach to learning and generalization in layered neural networks. Proc IEEE 78:1568–1574
Marder E, Abbott LF (1995) Theory in motion. Curr Opin Neurobiol 5:832–840
Marder E, Kopell N, Sigvardt K (1997) How computation aids in understanding biological networks. In: Stein PSG, Grillner S, Selverston AI, Stuart DG (eds) Neurons, networks, and motor behavior. MIT Press, Cambridge London, pp 139–149
McCulloch WS, Pitts W (1943) A logical calculus of ideas immanent in nervous activity. Bull Math Biophys 5:115–133
Parker DB (1985) Learning logic. Technical report TR-47, Center for computational research in economics and magnetic science, MIT, Cambridge
Pineda FJ (1987) Generalization of back-propagation to recurrent neural networks. Phys Rev Lett 59:2229–2232
Rall W (1959) Branching dendritic trees and motoneuron membrane resistivity. Exp Neurol 2:503–532
Rall W (1964) Theoretical significance of dendritic tree for input-output relation. In: Reiss RF (ed) Neural theory and modeling. Stanford University Press, Stanford, pp 73–97
Rosenblatt F (1962) Principles of neurodynamics. Spartan, New York
Rumelhart DE, Hinton GE, Williams RJ (1986) Learning representations by back-propagating errors. Nature 323:533–536
Schwartz DB, Samalam VK, Solla SA, Denker JS (1990) Exhaustive learning. Neural Comp 2:371–382
Werbos P (1974) Beyond regression: new tools for prediction and analysis in the behavioral sciences. Ph.D. thesis, Harvard University