ARTIFICIAL NEURAL NETWORKS THAT LEARN MANY-BODY PHYSICS

John W. Clark and Srinivas Gazula

Department of Physics and
McDonnell Center for the Space Sciences
Washington University, St. Louis, Missouri 63130

INTRODUCTION

Neural networks are systems of neuron-like units that store information in the connections between the units.[1–4] From the viewpoint of the many-body theorist, the neurons may be thought of as particles, and the weighted connections between units provide the interactions between these particles. The neuronal units exhibit varying degrees of activation, and the activation of a given unit in turn promotes or hinders the activation of a unit to which it extends a connection, depending on whether the connection has a positive or negative weight, respectively. Thus, neurons turn each other on (or off). Since there are in general many pathways by which information can travel through an intricate mesh of connections, processing in a neural network is said to be "massively parallel" as opposed to sequential.

This feature of parallel processing of information in neural networks is reminiscent of the manner in which visual images are processed in the human brain. It may then be no surprise that, with an appropriate choice of wiring diagram or network "architecture" and appropriate "learning rules" for determining the weights of connections, neural nets are very good at pattern recognition. In particular, they are good at classifying input patterns into two or more categories. To achieve this ability with limited resources of neuron-like units and interconnections, they must be good at abstracting the regularities present in the ensemble of input patterns to which they are exposed. All but the simplest classification tasks require neural networks to form internal representations of their environment. They must 'invent' economical rules that describe the correlations inherent in the input patterns. This is done (as we shall see) by the development of neurons with special "receptive fields" which enable them to detect essential regularities and correlations, and then to influence the output categorization decision accordingly. Suppose a network has indeed discovered a working rule that gives a reasonably faithful representation of a given set of patterns (the training set), in the sense that the response of the network to the patterns in this set is nearly always correct. The wider applicability of the rule may then be tested by exposing the network to novel patterns (the test set) and observing its response. In many cases, some useful generalization ability is demonstrated. Such an ability is a sign that the system is not merely using its free parameters to create a lookup table (analogous to rote memorization, and a prominent feature of so-called expert systems), but is actually capturing the most important rules governing the composition of the world of input stimuli.

Condensed Matter Theories, Vol. 6, Edited by S. Fantoni and
S. Rosati, Plenum Press, New York, 1991

It is by now well known that condensed-matter theories are useful in studying the statistical mechanics of neural networks. The most spectacular example of this statement is the exact solution of the equilibrium statistical mechanics of the Hopfield model[5] by means of spin-glass techniques based on mean-field theory.[6,7] We would like to make the case that it is not out of the question to return the favor, so to speak: it may be possible to solve many-body problems using neural networks.

Reconsider the scenario sketched in the second paragraph, but substitute *data* for *patterns*, *model* or *theory* for *rules*, and *prediction* for *response to novel patterns*. It should become apparent that, in the abstract, what the neural net is doing in this scenario is just what a scientist does when he does science!

The idea, then, is to train a neural network to analyze scientific data – in particular, complex data generated by a complicated many-body system – and let it formulate its own laws about the regularities implicit in these data. It may arrive at laws (or merely rules) we already know, or it may represent the data in ways alien to us. There is the tantalizing prospect that the neural net may discover aspects of nature that we have not grasped before. Of course, the 'theory' postulated by a neural network is buried in the connections and their weights – and in practice it may be difficult or impossible to extricate a set of clearly expressed rules from the assemblage of 'learned' neuron-neuron interactions. One is confronted with a new and challenging class of ill-posed inverse problems whose analysis has scarcely begun.

In this contribution, we shall examine two promising scientific applications of neural networks to many-body problems:

- Learning and prediction of protein secondary structure.[8–10]
- Learning and prediction of nuclidic properties.

The first application has reached a respectable level of sophistication within just three years, and has yielded fresh insights into an extraordinarily difficult problem. The second, still in its early stages, is represented here by an original investigation in which a neural net is taught to distinguish between stable and unstable nuclides, based only on the specification of proton and neutron numbers.

An understanding of these applications requires some acquaintance with multilayer, feedforward perceptrons. Accordingly, we shall begin by sketching the necessary background. We introduce the Elementary (two-layer) Perceptron, point out its limitations, state the "credit assignment problem" for nets with hidden neurons (which impeded progress in neural networks for over two decades), outline the backpropagation learning algorithm for solving this problem, and cite prominent demonstrations of the efficacy of backpropagation. For those interested in more details, Refs. 1-4 provide very accessible reviews of the field of neural networks.

THE ELEMENTARY PERCEPTRON

In setting up a neural network model, a primary consideration is architecture: What is the wiring diagram, i.e., which neurons are connected to which? For the many-body theorist this translates to: What is the pattern of neuron-neuron interactions V_{ij}? However, when thinking in many-body terms, a novel feature of neural systems must be kept in mind. The interactions between neurons are *not* necessarily reciprocal. Neuron j can send a connection to neuron i, but i might not send one back to j in return. Even if it does, there is ordinarily no reason to expect the weights or even the signs associated with the "forward" and "backward" connections to be the same. Thus, in general, $V_{ij} \neq V_{ji}$. In other words, we don't have to obey Newton's third law in designing a neural net. What we do have to be concerned with is the information processing task that is to be performed by the network. This task will govern the type of architecture that should be adopted. One should make a "context-dependent" choice of architecture.

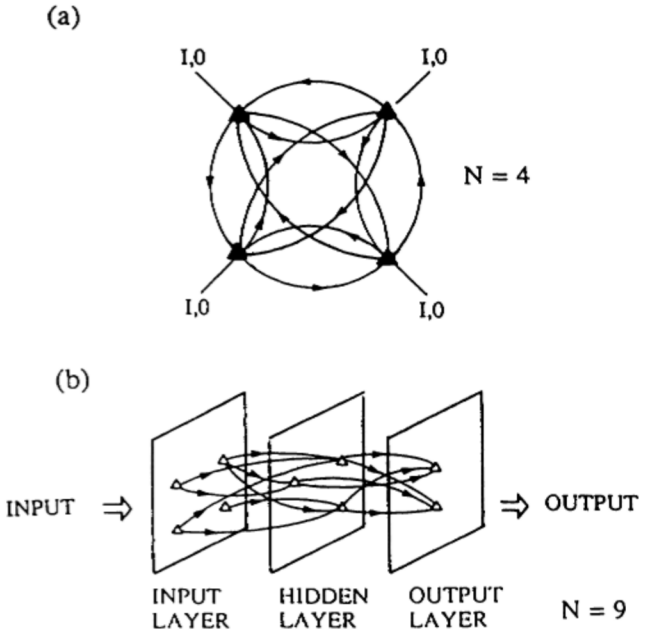


Fig. 1. Extreme architectures for neural network modeling. (a) Fully connected net with symmetrical couplings. (b) Layered, feedforward network. Triangles represent neurons and arrows represent directed couplings.

In current applications of neural nets to abstract, real-world, and scientific problems, two extreme architectures are common (see Fig. 1).

- *Fully connected* with reciprocal or *symmetrical interactions* (thus obeying Newton's third law). This is the architecture of the Hopfield model,[5] which functions as an attractor network[7] and is useful for storing content-addressable memories.
- Processing units arranged in *layers* $L = 1, \ldots K$, connections being exclusively *feedforward*. Layer 1 (layer K) serves as the input (output) interface, and there is zero coupling from unit j to unit i unless j belongs to layer L and i to layer $L+1$. (In some cases, forward connections are allowed to skip layers.) This is the architecture of neural networks of the Perceptron class,[11,12] which learn to classify input patterns.

Here we shall be concerned entirely with networks of the latter type, since the scientific problems to be attacked involve the assignment of input patterns (corresponding to the sequence of amino acids in a protein, or to the numbers of neutrons and protons in a nuclide) to one of a small number of categories (corresponding to secondary structures, and to stability or instability, respectively).

The simplest realization of this class of networks, susceptible to a fairly complete mathematical analysis, is the Elementary Perceptron (Fig. 2). Such a device has only two layers of decision units or model neurons, which serve as input and output interfaces. The *input layer*, or *preprocessing layer*, consists of a set of N_I binary-decision units j whose states ("on" or "off") are determined unambiguously by the stimulus presented to the system. The stimulus might be 'sensory' in nature, as would befit the term Perceptron, but it could, more abstractly, be the output from another neural network. In any case, the

stimulus can be represented as a bit string of length R. To make the setup more concrete, the impressed data can be arranged in a two-dimensional array, or screen, with 1s and 0s making bright and dark spots, respectively. In the original perceptron literature, this screen is regarded as the counterpart of the retina. Continuing the analogy with vision, the units of the preprocessing layer – also called input units – may be regarded as feature analyzers. Each such unit receives input lines from some finite proper subset of the R elements of the screen, i.e. from its assigned receptive field, and generates a signal "0" or "1" depending on the states of these elements. The functions computed by the units of the preprocessing layer are "hard wired" at the outset and are not subject to modification

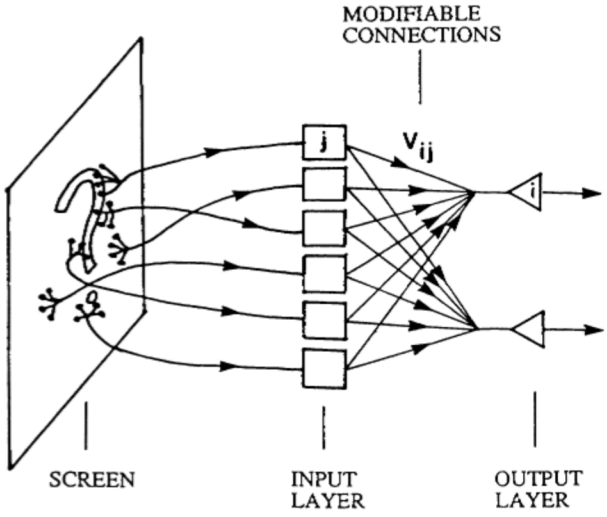


Fig. 2. Architecture of the Elementary Perceptron.

during learning. In the broadest formulation, there is wide latitude in the choice of feature analyzers; indeed, they may compute arbitrary Boolean functions. But without restrictions on this freedom, the conception becomes an empty generality. As a natural option, one might take the input units to be simple McCulloch-Pitts [13] (McP) threshold neurons. An McP neuron goes into a fully activated state (signal value unity) if and only if it receives a total input signal exceeding some sharp threshold; otherwise it remains inactive (signal value 0). Our purposes are best served by the even more primitive specialization to elementary perceptrons of order 1, which means that no feature analyzer is driven by more than one element of the screen, and accordingly that a feature analyzer is "on" if and only if its assigned screen element is "on."

The units in the preprocessing layer extend modifiable connections with strengths V_{ij} to the N_O model neurons i of the *output* or *response layer* of the perceptron, which are normally assumed to be of the sharp-threshold, McP type defined above. More specifically, the state of neuron i is computed from

$$S_i = \Theta\left[\sum_{j=1}^{N_I} V_{ij}S_j - V_{io}\right] \quad , \tag{1}$$

where $\Theta(x)$ is the usual step function (taking values 0 and 1 as x takes negative and nonnegative values, respectively), S_j is the state of input unit j, and V_{io} is the threshold assigned to neuron i. The state variables S_i, S_j are restricted to binary values 1,0 corresponding to full activity or total inactivity. The resulting states of the output neurons determine, in turn, the response of the system. The response could, for example, affect 'motor' neurons or some other system external to the perceptron.

In the more elaborate architecture of *multi-layer* perceptrons, additional layers of neuronal units, with additional sets of modifiable connections feeding into them, are interposed between the input and output layers. The neurons of the intermediate layers do not communicate directly with the environment of the system and thus are called *hidden neurons*, in contrast to the units in the input and output layers, which are *visible*.

A perceptron defines a mapping from input to output. For example, one might want the network to respond to an input image (say of a magnet, a horse, a rose, or a triangle) with an output that correctly classifies the image as animal, vegetable, mineral, or "none of the above." In the simplest case the system has a single, "yes-no" output neuron and one asks the device to chose between only two alternatives. The perceptron might be taught to tell us whether an aircraft silhouette, or a voice, belongs to friend or foe; or whether a radiographic scan shows normal or cancerous tissue.

A perceptron is designed to *learn by example*, with the intervention of a 'teacher' that observes the responses of the network and incrementally corrects the weights of the connections between layers to improve the accuracy of response. Thus, in application, the system is first trained on a representative sample of stimulus-response pairs. During this *learning phase*, it is caused to build its own model of the regularities of the training sample, through the alteration of the weights of its connections, i.e. the values of its interactions. (When hidden neurons are present we may imagine that it constructs an internal representation of the environment.) Subsequently, in the *prediction mode*, the network is asked to classify stimuli it has never seen before.

Focusing on the two-layer perceptron, one can easily find a learning rule that works (assuming the perceptron can solve the problem at all).

Perceptron Learning Rule. The perceptron is shown a stimulus from the training set and the teacher inspects the resultant states of all the output units i.

(i) If the response of neuron i is correct, the couplings or V_{ij} affecting it are left unchanged.

(ii) If neuron i should be active but is not, the connections from *active* input units are strengthened: $\Delta V_{ij} = \eta S_j$, where $0 < \eta < 1$.

(iii) If neuron i is active but should not be, the connections from *active* input units are weakened: $\Delta V_{ij} = -\eta S_j$.

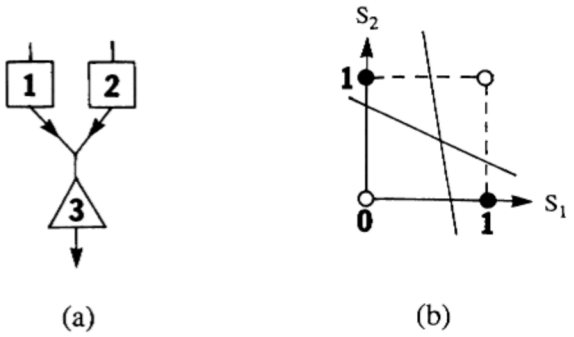
This rule also provides for appropriate changes in threshold, if we implement a simple reinterpretation of thresholds in terms of *biases*. Each neuron i is assigned an extra "true unit" that is always active, and sends a connection to i having weight (bias) $-V_{io}$.

It can be rigorously proven[11,12] that *if* there exists a solution V_{ij} , V_{io} for the couplings and thresholds permitting the system to accomplish its classification task, the perceptron learning rule will converge to *some* solution of the problem in a finite number of steps for any initial choice of couplings (*Perceptron Convergence Theorem*).

The *if* part of this statement greatly limits the practical value of the theorem. There is a very large class of tasks that the Elementary Perceptron is unable to perform, since it must base its decisions entirely on linear combinations of its inputs. To understand this statement more fully, we note that the states of the input layer may be represented by points in an N_I-dimensional Euclidean space. An elementary perceptron works by finding

a hyperplane $\Sigma_j V_{ij}S_j = V_{io}$ in this space that separates input states into two groups, so that a clean decision can be made whether some particular feature of the stimulus is present or absent. But it is obvious that not every two groups of points is separable by a linear manifold. Indeed, among the $2^{(2^n)}$ Boolean functions of n binary inputs, the fraction belonging to the linearly separable category shrinks with extreme rapidity as n increases. Already at $n = 6$, the fraction is less than 10^{-12}.

A telling failure of the Elementary Perceptron – in the form specified above – is its inability to compute the logical function exclusive-OR (also known as x OR ELSE y, and commonly denoted XOR). The XOR function, with arguments x and y, is true if and only if *exactly one* of its arguments is true. If *both* x and y are true, or if neither is true, the XOR function is *false*. Truth has a value 1, and falsity, 0. Thus the perceptron of Fig. 3(a), with two input units (labeled 1 and 2) and one output neuron (labeled 3), is asked to



(a) (b)

Fig. 3. An elementary perceptron (a) and its failure (b) in the XOR problem.

respond to the stimulus patterns (0,1) and (1,0) with an output $S_3 = 1$, and to (0,0) and (1,1) with output $S_3 = 0$. There is no combination of the weights V_{31} and V_{32} and the bias $-V_{3o}$ that will produce the correct behavior. To see why, plot the four input states (0,0), (0,1), (1,0), (1,1) in the relevant 2-space (Fig. 3(b)), and label them with the desired outputs. It is not possible to find a straight line $V_{31}S_1 + V_{32}S_2 = V_{3o}$ that separates the two inputs associated with output 0 from the two associated with output 1. However, if we are permitted to go beyond the Elementary Perceptron and introduce an intermediate (or hidden) layer of neurons, this simple problem may be solved quite readily. For instance, note that the combined states of three hidden neurons form the vertices of a three-dimensional hypercube, and that having constructed a suitable mapping from states of the input layer to states of the hidden layer, it is possible to find a plane that separates the vertices on the 3D cube corresponding to "true" and "false" responses.[3]

The Elementary Perceptron is incapable of solving problems with *predicate order* exceeding one, where predicate order[14] means, roughly, the minimum number of preprocessor units needed to give any useful information relevant to the desired output. The XOR problem is equivalent to addition modulo 2 and to determination of the parity of a bit string of length 2 (the parity of a bit string being even [odd] if it contains an even [odd] number of ones). It is a second-order problem, in that the answer depends on pairwise correlations between screen elements. In this sense, the problem of assigning parity to a bit string of length R is of Rth order, since its solution involves the detection of correlations among R screen elements. By contrast, the problem of computing the ordinary OR function is of first order – the information that either of two screen elements is "on"

(irrespective of what the other is doing) is enough to decide the correct output. Another interesting problem that an elementary perceptron cannot solve is the "T-C" problem,[14] in which the network must learn to distinguish between templates forming the letters "T" and "C," located anywhere on a grid and with possible orientations of 0°, 90°, 180°, and 270°. As usually formulated, this problem is of order three.

In fact, the elementary, two-layer perceptron *can* solve the parity problem (and similarly, other higher-order problems), provided the scope of the feature analyzers in its input layer is extended to allow at least one of them to look at *all* elements of the incoming bit string (or screen). The strategy is thus to increase the complexity of the system's preprocessing units, replacing a perceptron of order $k = 1$ by a perceptron of order $k = n$ (Ref. 14). In addition, the available solutions call for a number of preprocessors that grows exponentially in R (say, as 2^R, so as to encompass all possible combinations of activity on the stimulus screen), and involve weights with an uncomfortably large range (the largest requiring, say, at least 2^R learning instances!). This discouraging scenario is typical of attempts to compensate for the limitations of the two-layer perceptron. Such strategies are self-defeating, since they make insatiable demands on computer resources and/or hardware in large-scale problems of practical interest.

The inability of the Elementary ($k = 1$) Perceptron – and of individual McP neurons – to realize x OR ELSE y assumes larger significance when it is recognized that the logical function NOT (x OR ELSE y) is *computationally universal*, in that every other logical function, no matter how complicated, can be expressed as a combination of such functions. Thus, although single McP neurons can compute simple logical functions like x AND y, x OR y, NOT x, and x AND NOT y by suitable choices of couplings and biases, an elementary, two-layer perceptron with McP neurons as output units is not computationally universal.[2] On the other hand, suitably constructed nets of McP neurons can compute NOT (x OR ELSE y) and hence McP *networks* are computationally universal. Indeed, a three-layer perceptron built with neuron-like units can compute any Boolean function.[15] The three-layer architecture suffices to form arbitrarily complex decision regions – convex, concave, multiply connected and disconnected, along with the simple hyperplane-bounded regions of the elementary perceptron.[16] (For precise statements, see Refs. 15,16.)

THE CREDIT ASSIGNMENT PROBLEM

Although multi-layer feedforward nets are vastly more capable than elementary perceptrons, they involve hidden units, and we come face-to-face with the *credit assignment* problem. It was easy enough to see how to modify the connection weights of the two-layer perceptron to improve its performance in tasks within its abilities, since the faulty connections directly affect the output. But when hidden units are present, or when there are feedback loops in the system, it is not at all obvious how to identify the connections that are contributing to incorrect [correct] behavior and determine their degree of responsibility, so that they can be punished [rewarded] accordingly. Another way of stating the problem is to observe that the hidden units must be turned into feature detectors which effect an appropriate mapping between configurations of input and output units; the difficulty is to figure out what features they should detect. Several workable solutions of this problem were offered in the 80s, although convergence theorems comparable to that stated above for the two-layer perceptron are still lacking.

Current neural-network approaches to higher-order categorization tasks include the Boltzmann machine,[17] backpropagation of error signals,[18–21] higher-order nets involving nonlinear discriminant functions,[22] and genetic neural networks.[23]

The Boltzmann machine has special appeal to condensed-matter theorists because it is based on familiar ideas from statistical mechanics. However, it does not fit comfortably into the framework of the present discussion, since the architecture and dynamics are more

like those of a Hopfield net at finite temperature. Although full connectivity is not imposed, the couplings are symmetrical, which has the consequence that the system will always relax to a condition of Boltzmann equilibrium. It is a remarkable property of Boltzmann equilibrium that a simple learning rule can be devised that overcomes the credit assignment problem and tells how to adjust all the weights, including those for connections to the hidden units, on the basis of information available locally. While theoretically elegant, this approach is inefficient in practice, because of the excessive amount of time required to attain equilibrium in large systems.

LEARNING BY BACKPROPAGATION OF ERRORS

Currently, the most popular method for dealing with hidden units is backpropagation ("backprop") or some variant thereof. In contrast to the Boltzmann-machine learning algorithm, backprop is designed for layered, feedforward, perceptron-like systems. But unlike traditional perceptrons, the neural network trained with the backprop algorithm must be made up of *analog* neuronal units having graded, i.e., continuous, state variables or output signals, since derivatives of their response must be calculated. To see how the scheme works, let us consider a three-layer net and denote the states of input, intermediate (hidden), and output layers by $I_k = \{S_k\}$, $H_j = \{S_j\}$, and $O_i = \{S_i\}$, respectively, reserving the indices i, j, and k, respectively, for output, hidden, and input units. The goal is to adjust the couplings V_{jk} (from input neurons to hidden neurons) and V_{ij} (from hidden neurons to output neurons) to achieve a prescribed input-output mapping

$$I_k^\mu \rightarrow O_i^\mu = T_i^\mu \quad , \tag{2}$$

i.e., the stimulus, represented by I_k^μ, should lead to an output O_i^μ which coincides with the desired target T_i^μ. For example, the mapping might be an association of authors with prose, jobs with workers, parities with bit strings, connectivity with figures, etc. The output of a given neuronal element l is assumed to obey

$$S_l = g(F_l) \quad , \tag{3}$$

where the response function $g(u)$ is chosen to be of sigmoidal or soft-threshold form and hence differentiable. A typical sigmoid (or "squashing") function is $g(u) = (1 + e^{-u})^{-1}$; generically, such a function is positive and monotonic and has asymptotes $g(-\infty) = 0$ and $g(\infty) = 1$. Around $u = 0$ there is a transition region corresponding to the soft-threshold response of the unit. The argument F_l entering (3) is either an external stimulus (if l is a neuron in the input layer) or is of the form $\Sigma_m V_{lm} - V_{lo}$ (for neurons l in hidden and output layers). The state variables S_l now range over the continuum [0,1]. A neuron l is considered to be "on" (or "off") when S_l is *nearly* 1 (or *nearly* 0). More generally, since a given neuron l will be specialized to detect particular features in the impressed patterns (through its incoming connections and their weights, which define the receptive field of the neuron), we may regard the value of its output S_l as a measure of the confidence with which such features are judged to be present.

Backpropagation is a supervised learning procedure in which a cost function

$$C[V] \equiv \sum_{i,\mu} (O_i^\mu - T_i^\mu)^2 \quad , \tag{4}$$

measuring the deviation of the actual outputs from the desired targets, is minimized by gradient descent. The process is an iterative one in which, at each step, the connection strengths are improved incrementally by altering them in a way which produces a decrease of $C[V]$. This behavior is assured if the change ΔV_{lm} of V_{lm} (where $lm = ij$ or jk) is taken to have the form

$$\Delta V_{lm} = -\eta \frac{\partial C}{\partial V_{lm}} \qquad (\eta > 0) \quad . \tag{5}$$

The required derivatives can be computed from

$$O_i^\mu = g\left[\sum_j V_{ij} S_j^\mu\right] = g\left[\sum_j V_{ij} g\left[\sum_k V_{jk} S_k^\mu\right]\right] \quad , \tag{6}$$

suppressing thresholds for economy. Working backward from the output units to the hidden layer, we find

$$\Delta V_{ij} = \eta \sum_\mu [T_i^\mu - g(F_i^\mu)] g'(F_i^\mu) S_j^\mu \equiv \eta \sum_\mu \delta_i^\mu S_j^\mu \quad , \tag{7}$$

with $F_i^\mu = \Sigma_j V_{ij} S_j^\mu$. The correction is thus governed by the *error signal* $\delta_i^\mu = [T_i^\mu - g(F_i^\mu)] g'(F_i^\mu)$. This part of the problem was already solved in the 50s by Rosenblatt and coworkers[11,12] and by Widrow and Hoff,[24] dealing with two-layer nets. The "hard" part of the three-layer problem is to correct the couplings from the input neurons to the hidden neurons, since it is not clear how the hidden neurons should behave to produce the required behavior of the output units. While this problem is difficult conceptually, backpropagation offers a starkly mechanical solution: knowing the error signals δ_i^μ, one just uses the chain rule of partial differentiation to calculate the derivative of O_i^μ of (6), and hence the derivative of C of (4), with respect to the couplings V_{jk}. The result for the correction (5), with $lm = ij$, is

$$\Delta V_{jk} = \eta \sum_{i,\mu} \delta_i^\mu V_{ij} g'(F_j^\mu) S_k^\mu \equiv \sum_\mu \delta_j^\mu S_k^\mu \quad , \tag{8}$$

with $F_j^\mu = \Sigma_k V_{jk} S_k^\mu$. The recipes (7) and (8) are often called the "generalized delta rule," for historical reasons. Compared with earlier training rules, the important new feature is that the error signals δ_j^μ appearing in (8) are obtained by a *recursive* computation in which the error signals δ_i^μ for the *following* layer (reckoned in the direction from input to output) are first determined. Extension of this procedure to more layers is straightforward, weight modifications being implemented using a series of error corrections δ_i^μ, $\delta_{j_1}^\mu$, $\delta_{j_2}^\mu$, ... The δs for the couplings to the Lth layer are evaluated from those for the couplings to the $L+1$th layer just as was done above for $L = 2$. Thus one can say that the error correction – 'the medicine' – is propagated backward from the output to the preceding layers, the 'doses' to the individual connections being apportioned in a manner that assures steady improvement of the performance of the system in the sense of a decrease of the cost function C toward a local minimum. For each presentation of μ, the dose ΔV_{lm}^μ to a given connection – or "synapse" – is proportional to the output S_m of the "presynaptic" unit m and the error δ_l^μ attributed to the "postsynaptic" unit l.

Having chosen a suitable architecture (number of layers, numbers of neurons in each layer, distribution of connections between layers) and a suitable coding of stimuli and responses, learning by example with backpropagation proceeds as follows. The system is exposed to a stimulus μ from the training set. A *forward* computation is carried out to determine the response of the network, which is in turn fed into a *backward* computation that generates a set of small corrections ΔV_{lm} to the couplings. The members of the training set are presented, in random order, as many times as are needed to reach an acceptable level of performance for these inputs. (If the corrections are made after each stimulus presentation, there will be some deviation from true gradient descent in C; however, the difference is inconsequential for small η.) The performance of the system is then tested for inputs outside the training set.

This approach is not without its limitations and pitfalls. In general, there is no guarantee that the system will respond satisfactorily when shown novel stimuli. Moreover, since the method is based on gradient descent, there is the possibility of getting stuck in a local minimum of C, which may be far above the global minimum corresponding to the "best" internal representation of the regularities of the stimulus-response ensemble. (We hasten to add that in a broad range of nontrivial applications this was not found to be a

serious problem.[20]) Perhaps most important, it is not clear that the backpropagation algorithm will scale economically when confronted with large-scale, real-life problems.

To avoid a possible misunderstanding, we should stress that the backprop algorithm has no basis in neurobiological fact. It would appear to require antidromic transmission of information, backward along axon fibers, and the existence of units that compare the output of each neuron of the learning assembly with signals from a 'teacher' (itself an unknown element without clear biological counterpart) (cf. Ref. 25). The biological relevance of the neural networks under discussion here is remote at best. Their purpose is not to mimic biology in any serious detail, but rather to carry out useful information-processing tasks, based on the neurobiological paradigm that knowledge of the world may be embodied in the connections between relatively simple processing units.

Indeed, within the computational context, the backpropagation algorithm (with variation and refinement as appropriate) has seen some extraordinary successes, and it is definitely the "industry standard." Beyond the standard test cases of the XOR and parity problems, the encoding problem, symmetry problems, addition, negation, and the "T-C" problem,[20] a short list of some of its more interesting applications includes: (a) extraction of family-tree relationships,[26] (b) discrimination between noisy speech patterns,[27] (c) learning to read English text aloud[28], (d) use of self-supervised backpropagation to compress images[29] and speech waves,[30] (e) determination of the shape of an object from its shading,[31], and (f) recognition of handwritten zip codes.[32] Implementation in the commercial world is becoming commonplace.

Application (c) – known as NETtalk – has received considerable attention as dramatic evidence of the power of neural networks. Sejnowski and Rosenberg[28] have devised a model which demonstrates that a relatively small feedforward network containing one hidden layer, taught by backpropagation, is able to capture the most important regularities of English and also to grasp many of its irregularities. The model nominally consists of 7×29 input units, 80 hidden units and 26 output units. Each input unit extends connections to every hidden unit, and in turn each hidden unit sends a connection to every output unit. To each character of written text and its context there corresponds an appropriate verbal response – an appropriate *phoneme*, or contrastive speech sound. There are 29 possible text characters: the 26 English letters plus two punctuation symbols and a "space." The seven groups of input units are thus sufficient for a *local* encoding of a target letter (one unit of the middle group being "on" in correspondence to the given letter, the other 28 being "off"), plus similar local encodings of three additional characters on either side of the target letter to provide a partial context. The output coding is *distributed* instead of local. Each of 51 phonemes is represented as a combination of (a few of) 23 articulatory features (e.g. "voiced," "high," "glottal," "labial," "liquid," "nasal," etc.). Of the 26 output units, 23 are dedicated to these articulatory features and 3 are used to denote stress and syllable boundaries. Thus, any given output unit generally contributes to the encoding of several phonemes. The window of seven characters is moved over the text, one letter at a time. Two texts were used as sources in the training and subsequent testing of the network, namely phonetic transcriptions from the informal, continuous speech of a child and a 20,112-word corpus from a dictionary. In one of the experiments, a stream of 1024 words from the informal speech text was used to train the network, a number of passes through this segment being required to achieve satisfactory performance. After presentation of 50,000 words (and a corresponding number of learning steps), the percentage of correct "best guesses" for the target phoneme had reached 95%. (The best guess is that phoneme which, considered as a vector with components along the set of articulatory features, makes the smallest angle with the actual output vector.) The corresponding figure for "perfect matches" was 55%. (To qualify as a perfect match, the strength of each articulatory feature in the output vector must be within 0.1 of its correct value.) Upon testing the system, without further training, on a 439-word continuation of speech

from the same speaker, the performance was 78% in correct best guesses and 35% in perfect matches. A pre-processor is needed to identify characters, and a post-processor to convert phonemes to sounds. When the pitch of the DECtalk speech synthesizer used for the latter is set to mimic the voice of a female child, and the record of a training session is run at normal speed, the impression is striking. Early in the training, one hears an incoherent babble, then pseudowords with recognizable boundaries, then recognizable words, and finally a fluent (if not perfect), understandable speech. The learning rate and accuracy of response may be improved by increasing the number of hidden units, but, with other aspects of the architecture kept as before, the performance measures appear to saturate at about 120 hidden neurons. On training with the 1000 most common English words, the best performance achieved with 120 hidden units was 98% accuracy in best guesses; in subsequent testing on the randomized dictionary of 20,012 words, the average performance was 77% in best guesses, 28% in perfect matches. Such performance is comparable to that of standard computer-based approaches (e.g. DECtalk) which use a lookup table supplemented by a large set of hand-programmed phonological rules. Once a neural network has been taught to convert speech to text in a digital simulation, the backpropagation solution for the connections may be extracted for the design of a hardware version that is efficient enough for practical uses in real time.

PREDICTION OF PROTEIN SECONDARY STRUCTURE

We are now prepared to discuss applications of neural networks to scientific pattern-recognition problems, and particularly to the discovery of rules implicit in data gathered on many-body systems.

A protein is a polymer of α-amino acids (of which there are 20 different types), joined in a peptide linkage to form a polypeptide chain (or chains). The *primary structure* of a protein is defined by the sequence of amino acid residues $R_1, R_2, R_3, \ldots$ and the number of chains. The *secondary structure* of the protein is the putatively regular configuration that tends to be assumed by its polypeptide backbone. It is determined by various noncovalent bonding mechanisms (notably hydrogen bonds, electrostatic interactions, and van der Waals forces), including interactions between residues and interactions of the residues with the surrounding medium. Prominent types of secondary structures are (a) helices produced by hydrogen bonding between amino-acid side groups on the same chain (the most important being Linus Pauling's α-helix) and (b) sheet-like structures formed by hydrogen bonds between groups on different, fully extended polypeptide chains (a so-called β configuration).

The actual three-dimensional "native" form assumed in the natural environment is termed the *tertiary* structure. It is often very irregular, due to various factors that disrupt the generally regular influence of hydrogen bonding on the backbone. A crucial factor is the diverse chemistry of the amino acid side groups under exposure to the ambient medium (solubility vs. insolubility in water, etc.). The *tertiary* structure is that which yields the maximum number of favorable atomic contacts and produces the lowest free energy. The protein folding problem is one of the most challenging in science.

The function of a given protein – as an enzyme, a messenger, or a structural component – usually depends critically on its exact arrangement in space. Accordingly, the prediction of secondary and tertiary structures is of immense importance in molecular biology and biotechnology. From a first-principles, many-body point of view, both of these problems are formidable, and reliable *ab initio* treatments based on quantum chemistry and statistical mechanics require prodigious computational resources.

Recently, several groups[8–10,33] have exploited the learning, pattern recognition, and generalization abilities of neural networks to develop a new and very different approach to these problems. Our discussion is focused on the work of Bohr *et al.*,[9] in which three-

layer networks have been taught by backpropagation to associate primary and secondary structure – to classify amino-acid residues into two categories for each of three types of secondary feature: α-helix (or not), β-sheet (or not), random coil (or not). This approach to the prediction of protein secondary structure offers an alternative to the current generation of *ab initio* calculations taking account of interactions of residues with one another and with the environment[34] and to statistical approaches.[35] Statistical methods are capable of correctly predicting the associated secondary structure for up to 50% of a protein's residues.

The problem has much in common with that of teaching a network to pronounce written text: A single character does not correspond uniquely to a given phoneme (compare the "a" in "fat" with that in "eat"); rather, the correct pronunciation depends on context, i.e., on correlations with other nearby characters. Likewise the residue-to-secondary-unit mapping depends, in a nontrivial way, on preceding and successive residues in the series. Thus, in an arrangement[9] similar to that of NETtalk, the input layer of the network views a range of 51 successive residues, the center residue being the target and the 25 on either side providing its context. Each of the 51 places in the "window" may be occupied by any of the 20 possible individual amino acids. A sparse, local representation was adopted. Each residue was encoded as 19 zeros and 1 one (the latter appearing in a different position for each different amino acid) – resulting in an input layer of 1020 neurons. The hidden layer contained 40 units. The output layer consisted of 2 neurons, the sum of their activities being unity, so that in any particular case one or the other was dominant, yielding an exclusive indication of whether or not the target residue participates in the secondary feature being investigated. Separate networks were constructed to predict the presence or absence of the three secondary features α-helix, β-sheet, random coil. Training and testing was based on a sample of 56 proteins, selected from the Brookhaven Protein Data Bank, for which the amino-acid → secondary structure mapping has been determined by *X*-ray diffraction. A net was trained on a subset of n proteins from the sample; the remaining $56-n$ were reserved to test performance. For the network that learns to detect α-helix structure, the percentage of correctly predicted associations approached 73% as the training set was enlarged stepwise to include the full sample. A detailed comparison with the results of the Chao-Fasman statistical analysis in the case of the membrane-bound rhodopsin protein shows the superior discriminating ability of the neural net in the assignment of α-helix structure to segments of the protein molecule. The correlation between actual and target responses of the trained network declined when the window size was taken smaller than 51, but did not increase appreciably for larger sizes.

The novelty of this application is that a neural network is being used to carry out an empirical analysis of the properties of an immensely complicated collection of natural systems. The network builds an (internal) model that captures the regularities in a data set and makes predictions based on this model. We have here (together with Ref. 8) the first example of the use of an artificial neural net to do science. The drawback of this kind of approach is that the 'theory' developed by the net is buried, perhaps inaccessibly, in the myriad connections (some 40,000, in the configuration used by Bohr *et al.*). But, from a pragmatic viewpoint, it is hard to argue with success.

It should be noted that the work of Bohr *et al.* has recently been extended with some success to the prediction of the three-dimensional (tertiary) structure of a class of proteins that are functionally but not structurally homologous.[33] At its output, the more elaborate network must decide between three secondary structures (helix; sheet; coil), but it must also generate information about tertiary structure in terms of binary distance constraints on the C_α atoms in the protein backbone. Thus, the system is asked to do more, but in the learning phase it is also taught more. It is perhaps not surprising that – along with encouraging results on the protein-folding problem – the modeling of secondary structure is significantly improved over that for the simpler network of Ref. 9. A further noteworthy

recent application of the neural network approach is the analysis of the secondary structure of certain proteins found in the human AIDS virus.[36]

TEACHING NUCLEAR PHYSICS TO NEURAL NETS

The nuclear many-body problem provides a rich collection of phenomena and properties which may also be suited to representation and analysis using neural network techniques.[37] To test the utility of this idea, we have begun to teach nuclear systematics to layered feedforward systems, using the standard backpropagation algorithm modified with a momentum term.[20] The essential inputs to the net are the proton and neutron numbers Z, N of a given nuclide ${}^{A}Z$. We would like to teach a network the information contained in the Nuclidic Chart, without giving it so many resources (neurons and connections) that it merely makes a lookup table and develops no predictive ability. Our approach is first to investigate the simplest aspects of the nuclear world, using the simplest of architectures and forgoing context-specific refinements of the basic learning rule. If the results are encouraging, we can proceed with some confidence to more sophisticated structures and procedures, and to a wider selection of nuclear properties. The tentative agenda is:

- *First step*: Train a network to discriminate correctly between *stable* and *unstable* nuclides, given only Z and N.
- *Next*: Introduce degrees of stability of stable nuclides, in terms of binding energies and/or abundances; and degrees and kinds of instability in terms of half lives and branching ratios for the various possible decay channels. The additional information is to be provided at the output level, through the training process, i.e., the system must learn to associate the additional properties with the given (Z,N). (An intermediate step might involve the prediction of the dominant decay modes of nuclides – deciding first between (1) stability and (2) instability and in the latter case deciding further whether the nuclide prefers to transform via (2a) β^- emission, (2b) positron emission or electron capture, (2c) α decay, (2d) spontaneous fission, (2e) proton emission, (2f) neutron emission, etc.)
- *Supplemental*: Teach the net to identify other properties with specified (Z,N), mainly for the ground states of stable nuclides, including nucleon separation energies,[37] neutron cross sections, spins, magnetic moments, quadrupole moments, etc.

In the longer term, one might envision a neural network that analyzes nuclear spectra (coded as input using some appropriate scheme). This would come closer to the modern conception of nuclear modeling. At a very simple level, the net might be asked to determine whether the nuclide in question is spherical or deformed. At a more ambitious stage, aiming toward a symbiotic connection with microscopic many-body theories, one might try to train a neural network to correlate quantitative features of spectra with parameters entering a proposed model Hamiltonian or bare interaction, and to make subsequent predictions based on the rules of correlation established in the learning process.

Here we shall only report some progress on the first step of the above program. In judging the success of teaching nuclear systematics, as epitomized by the Nuclidic Chart, a number of questions naturally arise: Does the neural net recognize the regularities that are so familiar to us: the existence of a valley of β-stability; the favorability of even Z and even N for stability (attributed to a pairing energy for like particles); the energetic significance of ''magic numbers'' of neutrons and protons (attributed to shell closure); the regions of the (Z,N) plane associated with β^- decay, positron emission, α instability, and spontaneous fission; and so on? Can a neural network 'discover' the liquid-drop model and the shell model (or at least ingredients thereof), and other well-known nuclear models? Does it see nuclear physics in ways that surprise us? Is there a possibility of learning new nuclear physics from neural nets?

Of course, the acid test of such teaching efforts is: How good is a trained network at generalization to novel stimuli – i.e., does it give reliable, or at least 'sensible,' responses for nuclides outside the training set? Does the system have any real predictive power? This is the logical equivalent of testing the theory of a human nuclear theorist by comparing his predictions with experiment.

Architecture, Training Set, and Teaching Procedure

The goal of our initial investigations is to assess the extent to which the simplest feedforward nets can learn the stable-unstable dichotomy. We adopt the bare-bones *architecture* shown in Fig. 4. The three layers are characterized as follows.

- The *input layer* consists of $N_I = 16$ analog neurons, numbered from left to right and arranged in two groups of 8. The first 8 are clamped "on" or "off" to encode Z as a binary number; the second 8 serve for a similar encoding of N. (In practice the leftmost neuron of the Z array is never active, since Z remains below 128.)
- The *hidden layer* contains H analog neurons, numbered left to right. The cases $H = 0$, 10, 19, and 24 were considered (the greatest effort being expended on $H = 19$ as a "standard"). Each hidden neuron receives a connection from each input neuron.
- In the *output layer* there is a *single* output neuron, which receives a connection from each hidden neuron. The analog state variable $S_i \in (0,1)$ of the output neuron measures the confidence with which the net judges the input nuclide (Z,N) to be stable. This is the quantity used to determine the error signal δ_i^μ at the output layer, corresponding to presentation μ.

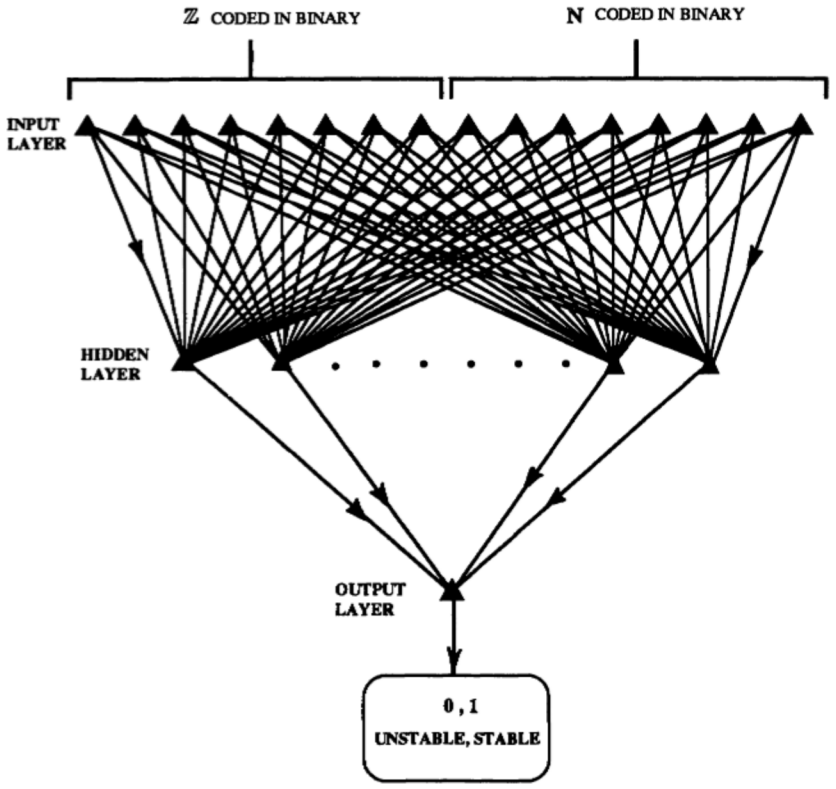


Fig. 4. Architecture of a neural network that is taught to distinguish between stable and unstable nuclides. Target outputs are indicated in the window at the bottom.

The *training set* (for most runs) is the entire collection of nuclides for which entries exist in the General Electric Chart of the Nuclides[38] (1984). Stable nuclides are taken as those on the chart bearing grey patches. The composition of the training set is (in part) as follows:

Total number of nuclides: 2226

Total number of stable nuclides: 249

Number of stable even Z - even N nuclides (''even-even''): 160

Number of stable even Z - odd N nuclides (''even-odd''): 37

Number of stable odd Z - even N nuclides (''odd-even''): 45

Number of stable odd Z - odd N nuclides (''odd-odd''): 7

The *training procedure* is based on minimization of the cost function (4) in weight (coupling) space, through backpropagation of error corrections. Following Ref. 20, the backprop algorithm is modified to include a momentum term, the change in weight V_{lm} being computed from

$$\Delta V_{lm}(n+1) = \eta \delta_l^{\mu} g(F_m^{\mu}) + \alpha \Delta V_{lm}(n) \quad , \tag{9}$$

where n is the presentation number and α is a momentum parameter embodying the effect of earlier weight changes on the current direction of motion in weight space. As indicated in (9), *weights are updated after the presentation of each pattern* (each (Z,N) pair). The momentum term acts to filter out high-frequency variations of the error surface in weight space. In particular, it improves learning in cases where there are deep, narrow ravines with a gently sloping floor, cases where sizeable steps (sizeable learning rate η) would otherwise lead to wild oscillations of the cost function and small steps would entail very slow progress in lowering C.

Initial weights and biases for a given teaching run were chosen ''randomly,'' by sampling a uniform distribution on the interval $[-0.5,+0.5]$. The learning rate was taken as $\eta = 0.05$ and the momentum parameter as $\alpha = 0.9$, the default values in the software provided by Ref. 39. No upper limit was set on the final weights or biases.

The basic unit of training time is the *epoch*. In one epoch, all 2226 training patterns are presented in random sequence, and an equal number of weight corrections is made. As is typical of applications of the backprop algorithm, many epochs must pass before the system may be said to have learned the data set, or before it reaches its asymptotic level of performance (as measured, for example, by the cost function C). In our experiments, performance did not improve significantly beyond about 400 epochs.

After each 25 epochs (a period we call an *eon*), learning is halted momentarily and the accuracy of response to each pattern is checked serially, thus yielding a value of the cost function – the total sum of squares of errors – at that particular stage in the process.

At least two independent runs were carried out for each choice of H (three for the case $H = 19$), starting from different random weights and biases by choosing different seeds for the random-number generator.

The preceding description of procedure refers to experiments in which the system is taught with the full data base. Performance in the *predictive mode* is studied by training a network on a subset of the data base and evaluating the accuracy of its response for the remaining subset of nuclides. Care is exercised to ensure that there are adequate stable and unstable test samples of each of the categories that were identified above: even-even, even-odd, odd-even, and odd-odd. Otherwise, the selection is roughly random. For example, if (nominally) 75% of the nuclides are used for training, every fourth stable even-even nuclide and every fourth unstable even-even nuclide are reserved for the test pool – and likewise for the other cateories.

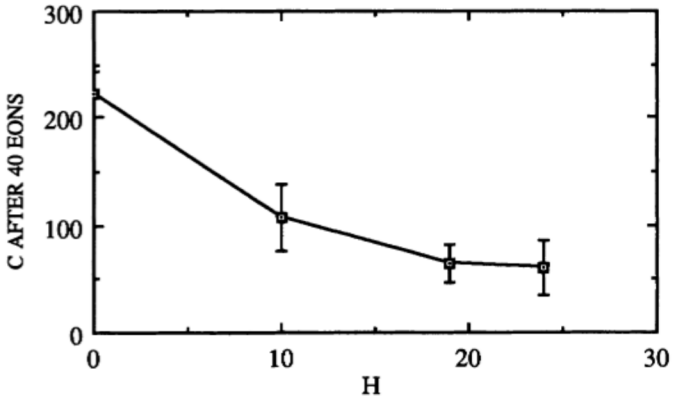


Fig. 5. Cost function C, evaluated at 40 eons, versus number of hidden neurons H.

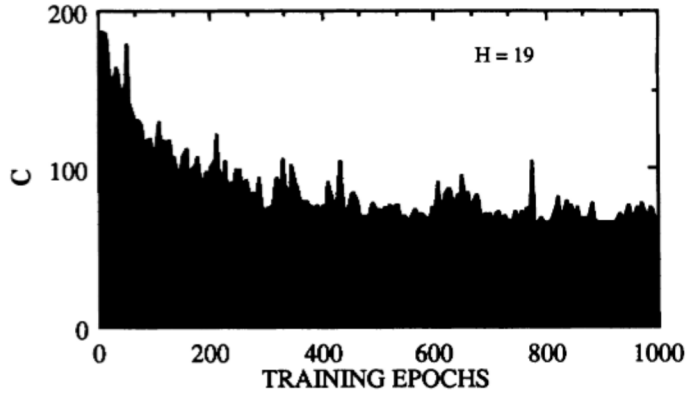


Fig. 6. Cost function C versus number of training epochs, for $H = 19$ hidden units (Run 3).

Results

First, let us consider how well the networks learn when trained with the full collection of nuclides from the chart. Some data on the dependence of the cost function of the trained net on the number of hidden units is collected in Fig. 5. The cost C is quoted after teaching for 40 eons. (The results would not be materially different for a cutoff of, say, 16 eons.) The error bars are not true statistical measures; they merely show the extremes obtained in the limited number of runs that has been performed. However, the trends seen in the plot are surely genuine: the introduction of a hidden layer substantially improves the performance on the training set, although the improvement tends to saturate as H is increased beyond 10 or 20 neuronal units. If the response of the net were governed entirely by chance, C would be $(1/4)\times 2226 = 556.5$. The nets with $H = 19$ and 24 operate at about a tenth of this level, so the performance of these systems does not achieve perfection. On the other hand, it is clear that the nets generated in this study do not simply memorize the data. In particular, they do not have enough units and connections to assign to each (Z,N) a hidden neuron that turns the output "on" if the nuclide is stable and "off" if it is not. The trained nets must have formulated rules which utilize their limited

resources more economically. Some explicit evidence for this statement will be presented below.

Fig. 6 demonstrates the overall saturation of performance on the training set, as the number of epochs is increased beyond 300-400. Note, however, that there are substantial fluctuations on an apparent baseline, continuing out to the termination of the run at 1000 epochs. In this run, the initial cost (not visible in the figure) was close to the chance value, although in some others (see below) it was much worse, giving such nets a poor starting point in weight space. Nevertheless, the "asymptotic" behavior, including the "asymptotic" value of C and the distribution of the correct and incorrect responses (see upcoming discussion of Fig. 7), was similar in the various runs for the same H.

We have monitored the 'student nets' during learning and have compiled a large store of information on the evolution of performance and on the development of weights and biases of the neuronal units. The predominance of unstable nuclides over stables means that the output neuron must be "off" much more often than it is "on." This has the consequence that, under backprop, the system will normally wind up at a local minimum in weight space characterized by many inhibitory (negative) couplings, many of which are quite large in magnitude. Indeed, final weights and biases, whether positive or negative, tend to be much larger in size than their initial values. We should also mention that, in many cases, the biases of the units (including that of the output neuron) will be *positive*, corresponding to negative threshold. (This occurs in part because of the prescription of a random starting point in weight-bias space.) Thus, we must cope with the counterintuitive situation that many units stay "on," unless they are turned off by sufficient inhibition.

The receptive fields of the hidden units (i.e., the patterns of the weights associated with their incoming connections) determine the features which are being detected by these units, and accordingly the representation of the environment that has been created in the hidden layer. Before discussing the nature of these receptive fields (and the patterns of weights and biases more generally), it will be illuminating first to look at some learning curves. A learning curve is a plot of the percentage of correct responses, versus the number of presentations of the training set, i.e., versus the number of epochs. Here, a "correct" response is taken to mean that the output neuron has an activity $S_i \geq 0.5$ if the given nuclide is stable and $S_i < 0.5$ if it is unstable. Thus the learning curves displayed below may be interpreted as measuring the percentage of correct "best guesses" as a function of training time (cf. Ref. 28).

An overall learning curve, for the full data base, would not be very informative, since the unstable nuclides numerically overwhelm the stables (by 9 to 1), and all nets quickly learn this "first order" feature of nuclear phenomenology. A much more revealing view of what is happening when layered nets learn about nuclear stability can be obtained from the individual learning curves for the following classes of *stable* nuclides: even-even, even-odd, odd-even, odd-odd, magic-Z, and magic-N. (The magic nuclides, with Z or N equal to one of the shell closure numbers 2, 8, 20, 28, 50, 82, 126, are more likely to be stable than other nuclides with neighboring Z or N.)

Fig. 7 provides composite plots of class-specific learning curves for runs with $H = 0$ and 19 hidden units. The two-layer net of Fig. 7(a) learns that most nuclides appearing in the chart are unstable, and of course this knowledge dominates its mature behavior. Lacking hidden units, it learns little more, although it does seem to discern the special character of magic and even-even nuclides. Obviously, its performance on the stable odd-odd nuclides should be (and is) very poor, since this system is not going to waste its scant resources on such a small class of inputs.

The introduction of hidden units in substantial number leads to dramatic improvements, as seen in Fig. 7(b). The accuracy for even-even and magic-Z stables is particularly impressive – exceeding the 90% level – and the performance for magic-N stables is also

quite respectable. By contrast, the accuracy of response for the even-odd and odd-even classes appears to be at the chance level or below. (However, this is somewhat misleading, since, once a stable nuclide is 'learned' (once the net tends to respond correctly to it), this knowledge is usually not 'lost' again. Also, the response to a stable nuclide that has been learned is often near the saturation value (i.e. S_i is near 1), whereas incorrect responses are often near the dividing line of 0.5.) Again, as expected, the network does not grasp the details of odd-odd nuclides; generically, it 'writes these off' as unstable, con-

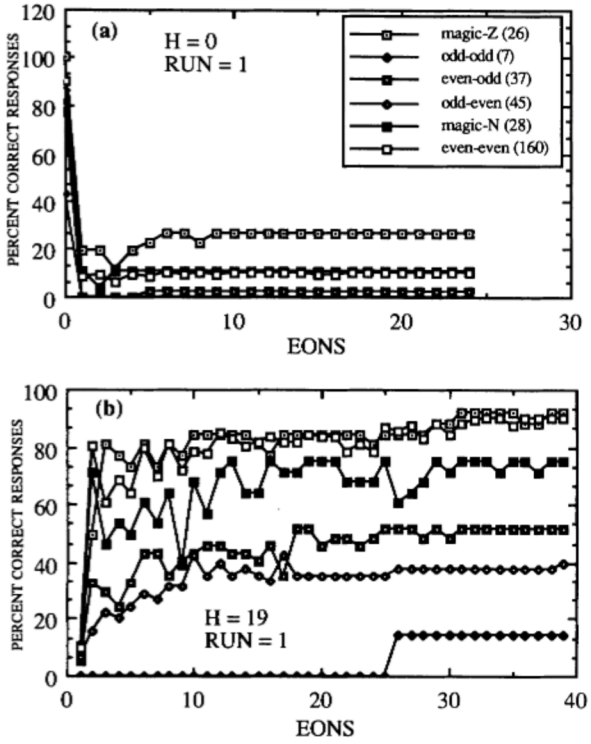


Fig. 7. Learning curves for a network with (a) $H = 0$ and (b) $H = 19$ hidden neurons, for specific classes of stable nuclides (see key).

serving its resources for the treatment of aspects of nuclear stability that have more influence on the cost function. For the run in question, the overall accuracy is about 70% for stable nuclides and of course extremely good for the unstables (better than 90%). The specific performance levels for the various classes of unstable nuclides (that is, for even-even, even-odd, etc. subsets of the unstables) display the expected anti-correlations with the corresponding results for the stables. Perhaps the most striking feature apparent in the learning curves is that the network clearly recognizes magic numbers and their importance, as well as the enhanced likelihood of stability of even-even nuclei.

The run involved in Fig. 7(b) is, in one way, a "worst-case" scenario: by the luck of the toss, the initial weights were unduly biased toward instability. For these initial weights, the net judged some 80-90% of the stable nuclides to be unstable, and the decisions for the unstables were also well on the unstable side of chance. This bad starting point is rapidly corrected, as seen in the figure. There follows a phase characterized by marked oscillations in performance, most prominent for the stable magic-N category. We can interpret this behavior as follows: The net quickly discovers some general rules that work reasonably well, but must then deal with a residue of detailed contradictions, by iterative refinement or modification of the working rules. At later stages, there is a (generally) slow increase to a saturation level of performance. (The odd-odd case is peculiar because of the small sample: the rise around 600 epochs reflects the fact that the net has finally learned *one* example of a stable odd-odd nucleus, namely the deuteron.) In spite of the unlucky choice of initial weights, the ultimate character of this network is rather typical, as is its quantitative performance on the stable-unstable dichotomy.

Next, let us look at some of the receptive fields developed by hidden neurons. (This is like opening up a black box to see what makes it tick.) As expected, these fields show complex structure and it is difficult to ascertain what individual tasks the various hidden neurons are performing. Normally, a given neuron is effective in detecting more than a single simple feature; moreover, it collaborates (or competes) with other hiddens in analyzing the input and contributing to the overall decision of the network. Nevertheless, it is sometimes possible to discern some of the functions of a hidden unit by inspection of the weights of its connections from the input units, the bias of the hidden unit, and the weight of its connection to the output unit. Four interesting examples are presented in Fig. 8. The weights of the 16 incoming connections from the input units to a given hidden unit,

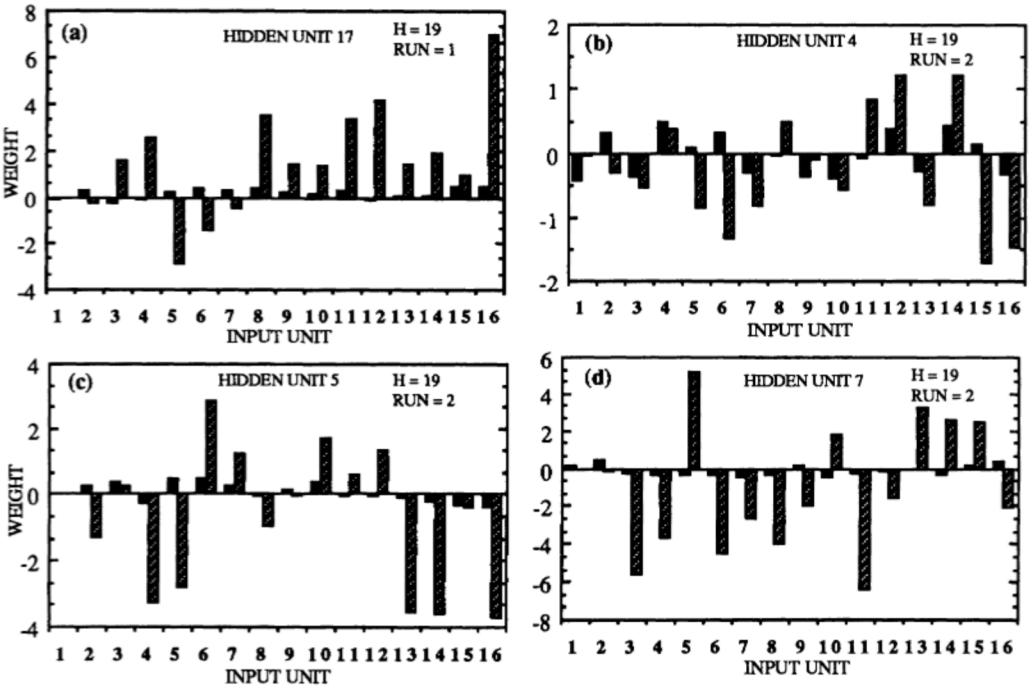


Fig. 8. Sample receptive fields. See text for discussion.

respectively before and after training, are depicted by the solid and cross-hatched bars; however, the magnitudes of the final weights are *a factor 10 larger* than indicated in the figure. In analyzing such data, it is helpful to have on hand the binary codes of the magic numbers: 2 = (10), 8 = (1000), 20 = (10100), 28 = (11100), 50 = (110010), 82 = 1010010, and 126 = (1111110). The final biases and outgoing weights $\{V_{jo}, V_{ij}\}$ of the hidden units j involved in cases (a)-(d) of Fig. 8 are, respectively, $\{-89, -5.1\}$, $\{-23, 12\}$, $\{-30, 27\}$, and $\{-32, 18\}$. The final biases of the output units in Runs 1 and 2 for $H = 19$ are, respectively, 3.0 (counterintuitive) and −4.7.

Consider each example in turn. *Fig. 8(a)*: Hidden neuron 17 of Run 1 for $H = 19$ is evidently a good detector of nuclides with odd Z or odd N. Upon seeing such a nuclide, this neuron (which has a negative connection to the output) tends to vote more strongly against stability. Yet it is obviously analyzing other aspects as well, since most of its incoming connections have developed substantial weights. In particular, activation of input neuron 16 by itself (corresponding to an odd-N nuclide) is not sufficient to produce a strong activity of hidden unit 17, because of the large threshold (89) developed by this unit. One or more of the other input units with positive weights to the hidden neuron must also be "on" – e.g. input neuron 8 (corresponding to an odd-Z nuclide). *Fig. 8(b)*: Hidden neuron 4 of Run 2 for $H = 19$, having a positive connection to the output neuron, votes for stability. The strength of this vote is increased when $N = 20$ is presented. Thus we say that the unit "supports" the magic number 20. *Fig. 8(c)*: Similarly, the hidden neuron involved in this case supports the magic neutron number $N = 82$. (We also see a preference for $Z = 6$.) *Fig. 8(d)*: Hidden neuron 7 of Run 2, $H = 19$, supports the proton magic number 8 (in fact, hidden units that detect magic number 8 appear to be fairly common), but again the complexity of the receptive field implies that this function must be balanced against others that are not so obvious.

The available data on the *predictive ability* of our networks is consistent with general expectations. Two experiments were carried out for the case of $H = 19$ hidden neurons, in which approximately 24% and approximately 14% of the nuclides were deleted from the training set (by the pseudorandom procedure indicated earlier) and used to test the predictive accuracy of the trained networks on unfamiliar inputs. As in the assessments of learning performance, an output $S_i \geq 0.5$ is interpreted as a categorical decision in favor of stability and $S_i < 0.5$ as a categorical decision against. A correct decision is called a 'hit' and an incorrect decision is 'miss.' Results are first quoted for the experiment with the smaller training set, followed by the corresponding results for the larger training set, enclosed in square brackets.

Total number of nuclides in test sample: 537 [317], containing 62 [35] stable and 475 [282] unstable nuclides.

Stable test sample. Ratios of *hits* for each class to number from each class present in sample: *even-even* 20/40 [15/22], *even-odd* 3/9 [1/6], *odd-even* 2/11 [1/6], *odd-odd* 0/2 [1/1], *magic-Z* 4/6 [3/4], *magic-N* 2/4 [3/3]. Percentage of unfamiliar stables correctly predicted: 40% [51%].

Unstable test sample. Ratios of *misses* for each class to number from each class present in sample: *even-even* 16/103 [10/58], *even-odd* 8/111 [1/75], *odd-even* 2/125 [2/72], *odd-odd* 1/136 [0/77]. Percentage of unfamiliar unstables correctly predicted: 94% [95%].

Overall accuracy of prediction: 88% [91%].

As expected, performance in the predictive mode is noticeably better when the larger training set is used. The superior performance on unstables (in both tests) is obviously due to their dominant weight in the teaching process. The results of additional experiments with a larger number of hidden units ($H = 24$) show a marked deterioration in predictive power. It would appear that the strategy adopted by the network, when provided with greater computational resources and taught with abbreviated training sets, has changed drastically – a considerably lower cost function being achieved at the sacrifice of generalization ability.

This kind of tradeoff is a common feature of other applications of the backpropagation algorithm (cf. Ref. 8).

Conclusions

We have made a first attempt to teach nuclear physics to neural nets. By intent, the task, architecture, encoding scheme, and teaching algorithm were all kept as simple as possible. Even so, some remarkable results were obtained, which encourage a further exploration of the potential of this new way of looking at the nuclear world. The behavior of layered, feedforward nets containing substantial, but not large, numbers of hidden neurons is summarized below. These nets were trained to distinguish between stable and unstable nuclides, using large ($\geq 75\%$) and reasonably homogeneous portions of the chosen nuclear data base.

- The dominant feature learned by the 'student' nets is that most nuclides are unstable. A net can achieve an 89% success rate merely by classifying all nuclides in the Nuclidic Chart as unstable. However, the networks created in our experiments did not settle into this trivial approximate solution but instead went on to learn some well-known features of the valley of β stability. To capture these less trivial features at a reasonable level of reliability, the hidden layer is necessary.

* Among other secondary features, the trained nets recognize that even-even nuclides are more likely to be stable than nuclei with odd Z and/or odd N. Hidden neurons that detect odd Z or odd N are not uncommon, and these usually vote against stability when activated.

* The trained networks recognize that magic numbers of protons or neutrons enhance the likelihood of stability. As illustrated by Fig. 8, there arise hidden neurons whose receptive fields and output weights tend to support specific magic numbers.

* The networks recognize, roughly, the outlines of the valley of β stability, with definite regions of more or fewer stable nuclides. The very heavy nuclei are 'written off' as unstable, at a very early stage of training.

- The networks really learn; they do not guess (in the ordinary meaning of that term). Once knowledge of the stability or instability of a particular nuclide is acquired (as manifested by a reliable high-level or low-level output, respectively), it is rarely lost as the teaching process continues. Thus, learning is generally cumulative.

- Learning of a particular nucleus is often very sudden (as reflected by an abrupt increase or decrease of the corresponding output activity).

- Learning is rapid for light nuclei, and very slow for the stable heavy nuclides.

- The networks have difficulty learning stable nuclides that are isolated from other stables in the Nuclidic Chart (and similarly for isolated unstables, though to a lesser extent). Conversely, groups of stable isotopes in a given row of the chart, or isotones in a given column, tend to support each other – they are learned rapidly as a group, with modest oscillations. The Sn and Xe isotopes are prominent examples.

- Certain nuclei are recognized as something special. Examples: among the light nuclei, ^{4}He is learned very quickly in all nets and produces a saturated output throughout the remainder of the training process; and the nets eventually learn that the doubly magic nuclide ^{208}Pb is stable, in spite of the strong generalization that such heavies are unstable.

- The 'student' nets are *not* making lookup tables. They are trying to accommodate a large number of constraints at the smallest possible cost, where cost is measured by the sum of squared errors for all of the input patterns, each pattern (stable or unstable) being given equal weight. They do this by making up rules that approximately characterize the data base and especially its chief regularities. Although the receptive fields developed by hidden neurons sometimes show features that are easily

interpreted in terms of even, odd, or magic values of Z and N, the field patterns are generally complex, indicating that rather complex processing is taking place in the hidden layer.

We close by identifying some refinements or extensions of the current scheme that promise substantial improvements, especially in the predictive power of the trained networks.

- One might ask for improved accuracy in learning and prediction of the stability of the stable subset of nuclides. This may be accomplished simply by presenting the stable data sample to the network more frequently than the unstable sample (say 9 times as often), during the training period. There will, naturally, be a concomitant decrease in the accuracy of the treatment of unstables. (Based on preliminary runs, it appears that that this strategy works very well, at least for the learning task.) Alternatively, one may redefine the cost function (4), including a coefficient a_μ which amplifies the importance of the stable patterns.
- The current study is based on a distributed, binary encoding of the input (Z,N) "patterns." This choice has the virtue of economy, but it introduces spurious algebraic relations between the coded forms of the patterns. A local encoding, in which all the input patterns are orthogonal, should lead to networks that show greater precision in the assigned task.[8,9] If the maximum proton and neutron numbers to be considered are $Z_{\max}$ and $N_{\max}$, we may implement such a code with an input layer consisting of $Z_{\max} + N_{\max}$ neurons. When the pattern (Z,N) is presented to the network, neuron Z and neuron $Z + N$ in this input bank are to be clamped "on," and all the others are to be clamped "off." It is expected that the increased complexity (in the receptive fields of the hidden neurons) will be compensated by enhanced performance.
- Proceeding to the next step in our program, nuclei are to be further differentiated by *degrees* of stability or instability. This will entail greater complexity in the output layer, with a set of "neuron bins" to represent (say) abundances, lifetimes, and principal decay modes. Asked to do more, the nuclear neural net should actually perform better – since, in the training phase, it must necessarily be given a more complete picture of the correlations between nuclear properties.

ACKNOWLEDGMENTS

The original research described herein was sponsored by the Condensed Matter Theory Program of the Division of Materials Research, and by the Nuclear Theory Program of the Physics Division of the National Science Foundation, under Grant No. DMR-9002863. S. G. acknowledges summer support from an A. L. Hughes Fellowship. J. W. C. expresses his gratitude to the European Research Office of the United States Army for travel support. We thank H. Bohr and J. S. Prater for stimulating discussions.

This paper is dedicated to the memory of Theodore Sturgeon.

REFERENCES

1. R. P. Lippmann, *IEEE ASSP Mag.* **4(2)**, 4-22 (1987).
2. J. D. Cowan and D. H. Sharp, *Quarterly Reviews of Biophysics* **21**, 365-427 (1988).
3. E. Domany, *J. Stat. Phys.* **51**, 743-775 (1988).
4. J. W. Clark, J. Rafelski, and J. V. Winston, *Physics Reports* **123**(4) 215-273; J. W. Clark, *Physics Reports* **158**(2) 91-157; J. W. Clark, in *Nonlinear Phenomena in Complex Systems*, edited by A. N. Proto (Elsevier, Amsterdam, 1989), pp. 1-102.
5. J. J. Hopfield, *Proc. Natl Acad. Sci. USA* **79**, 2554-2558 (1982).
6. D. J. Amit, H. Gutfreund, and H. Sompolinsky, *Ann. Phys. (NY)* **173**, 30-67 (1987).

7. D. J. Amit, *Modeling Brain Function: The World of Attractor Neural Networks* (Cambridge University Press, Cambridge, 1989).
8. N. Qian and T. J. Sejnowski, *J. Mol. Biol.* **202**, 865-884 (1988).
9. H. Bohr, J. Bohr, S. Brunak, R. M. J. Cotterill, B. Lautrup, L. Nøskov, O. H. Olsen, and S. B. Petersen, *FEBS Letters* **241**, 223-228 (1988).
10. L. Holly and M. Karplus, *Proc. Natl Acad. Sci. USA* **86**, 152-156 (1989).
11. F. Rosenblatt, *Psychological Review* **65**, 386-408 (1958).
12. H. D. Block, *Rev. Mod. Phys.* **34**, 123-135 (1962).
13. W. S. McCulloch and W. Pitts, *Bull. Math. Biophys.* **5**, 115-137 (1943).
14. M. Minsky and S. Papert, *Perceptrons: An Introduction to Computational Geometry* (MIT Press, Cambridge, MA, 1969).
15. J. Denker, D. Schwartz, B. Wittner, S. Solla, J. Hopfield, R. Howard, and L. Jackel, *Complex Systems* **1**, 877-922 (1987).
16. A. K. Hornik, A. M. Stinchcombe, and A. H. White, *Neural Networks* **2**, 359-366 (1989).
17. D. H. Ackley, G. E. Hinton, and T. J. Sejnowski, *Cognitive Science* **9**, 147-169 (1985); G. E. Hinton and T. J. Sejnowski, in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Vol. 1, edited by D. E. Rumelhart, J. L. McClelland, and the PDP Research Group (MIT Press, Cambridge, MA, 1986), pp. 282-317.
18. P. J. Werbos, *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences* Ph.D. Thesis (Harvard University, Cambridge, MA, 1974).
19. Y. Le Cun, *Proc. Cognitiva* **85**, 599-604 (1985).
20. D. E. Rumelhart, G. E. Hinton, and R. J. Williams, in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Vol. 1, edited by D. E. Rumelhart, J. L. McClelland, and the PDP Research Group (MIT Press, Cambridge, MA, 1986), pp. 282-317.
21. D. B. Parker, in *Proc. Conf. Neural Networks for Computing, AIP Conference Proceedings* **151**, edited by J. S. Denker (American Institute of Physics, New York, 1986), pp. 327-332.
22. Y. C. Lee, G. Doolen, H. H. Chen, G. Z. Sun, T. Maxwell, H. Y. Lee, and C. L. Giles, *Physica* **D22**, 276-289 (1986); T. Maxwell, C. L. Giles, Y. C. Lee, and H. H. Chen, in *Proc. Conf. Neural Networks for Computing, AIP Conference Proceedings* **151**, edited by J. S. Denker (American Institute of Physics, New York, 1986), pp. 299-304.
23. E. Mjolsness, D. H. Sharp, and B. K. Alpert, *Advances in Applied Mathematics* **10**, 137-163 (1989).
24. B. Widrow and M. E. Hoff, in *Institute of Radio Engineers, Western Electronic Show and Convention, Convention Record, Part 4* (1960), pp. 96-104.
25. F. Crick, *Nature* **337**, 129-132 (1989).
26. G. E. Hinton, in *Proceedings of the Eighth Annual Conference of the Cognitive Science Society* (Amherst, Massachusetts, 1986)
27. D. C. Plaut, S. J. Nowlan, and G. E. Hinton, *Carnegie-Mellon University Computer Science Technical Report* **CMU-CS-86-126** (1986).
28. T. J. Sejnowski and C. R. Rosenberg, *Complex Systems* **1**, 145-168 (1987).
29. G. W. Cottrell, P. Munro, and D. Zipser, in *Proceedings of the Ninth Annual Conference of the Cognitive Science Society* (Seattle, Washington, 1987), pp. 461-473.
30. J. L. Elman and D. Zipser, in *Technical Report No. 8701, Institute for Cognitive Science, University of California, San Diego* (1987).
31. S. R. Lehky and T. J. Sejnowski, *Nature* **333**, 452-454 (1988).
32. Y. Le Cun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, in *Neural Information Processing Systems*, Vol. 2, edited by D. Touretzky (Denver: Morgan Kaufman, 1990).
33. H. Bohr, J. Bohr, S. Brunak, R. M. J. Cotterill, H. Fredholm, B. Lautrup, and S. B. Petersen, *FEBS Letters* **261**, 43-46 (1990).

34. G. H. Paine and H. A. Scheraga, *Biopolymers* **26**, 1125-1162 (1987).

35. P. Y. Chou and G. D. Fasman, *Ann. Rev. Biochem.* **47**, 251-276 (1978).

36. H. Andreassen, H. Bohr, J. Bohr, S. Brunak, T. Bugge, R. M. J. Cotterill, C. Jacobsen, P. Kusk, B. Lautrup, S. B. Petersen, T. Saermark, and K. Ulrich, *Journal of Acquired Immune Deficiency Syndromes (AIDS)* **3**, 615-622 (1990).

37. H. Bohr, private communication.

38. *Chart of the Nuclides*, revised 1983 by F. W. Walker, D. G. Miller, and F. Feiner (General Electric, San Jose, CA, 1984).

39. J. L. McClelland and D. E. Rumelhart, *Explorations in Parallel Distributed Processing* (MIT Press, Cambridge, MA, 1988).