To appear in ARTIFICIAL INTELLIGENCE (1993)
(Elsevier Science Publishers)

Book Review

John A. Hertz, Anders S. Krogh, and Richard G. Palmer,

Introduction to the Theory of Neural Computation [1]

Reviewed by: [2]

Andreas S. Weigend
Xerox PARC
3333 Coyote Hill Road
Palo Alto, CA 94304

(Received June 1992; Revised October 1992)

Neural computation, also called connectionism, parallel distributed processing, neural network modeling or brain-style computation, has grown rapidly in the last decade. Despite this explosion, and ultimately because of impressive applications, there has been a dire need for a concise introduction from a theoretical perspective, analyzing the strengths and weaknesses of connectionist approaches and establishing links to other disciplines, such as statistics or control theory.

The *Introduction to the Theory of Neural Computation* by Hertz, Krogh and Palmer (subsequently referred to as HKP) is written from the perspective of physics, the home discipline of the authors. The book fulfills its mission as an introduction for neural network novices, provided that they have some background in calculus, linear algebra, and statistics. It covers a number of models that are often viewed as disjoint. Critical analyses and fruitful comparisons between these models

[1] Addison-Wesley, Redwood City, CA (1991).
Lecture Notes Volume I in the *Santa Fe Institute Studies in the Sciences of Complexity.*
352 pages. US$ 30.25 (paperback). ISBN 0-201-51560-1. (Library of Congress: QA76.5.H475)

[2] Electronic mail: weigend@cs.colorado.edu
Address after June 1993: **Department of Computer Science and Institute of Cognitive Science, University of Colorado, Boulder, CO 80309-0430, USA.**

are made possible by uniform notation and consistent use of symbols throughout. People adverse to formulae might be less happy with some sections; nonetheless, it is well worth working through them. Connectionism is heading in a direction that requires a solid mathematical background, and many of the interesting recent results are theoretical in nature: no longer can people jump into the field without understanding the sort of results presented in this book. Therefore, HKP is not only well written and concise, but also important. In my opinion it is the best book on neural networks since *Parallel Distributed Processing* by Rumelhart and McClelland (1986).

In this review, I will summarize some of the concepts in the field, following the main structure of the book:

- Search and optimization [Chapters 2–4]
- Supervised learning [Chapters 5–7]
- Unsupervised learning [Chapters 8–9].

My critical comments are given in the course of discussing specific chapters. I will also point out some issues that have come to light since the book was written, and I will give some suggestions on using HKP in teaching. The review ends with my overall impressions of the book.

1 Search and Optimization

The two main parts of HKP address two very different problems. Before turning to learning, i.e., the automatic extraction of (usually implicit) rules from examples, the book discusses connectionist networks with previously stored knowledge.[3] Typical applications include retrieving stored images from noisy cues, satisfying simultaneous constraints, and optimizing solutions to tasks such as the traveling salesperson problem. Search and optimization problems are often cast in the framework of statistical mechanics, a branch of physics that describes the macroscopic behavior of large numbers of simple, interacting entities. One model of statistical mechanics whose computational properties are well understood is the *Hopfield model.* After briefly characterizing it and contrasting it to networks for learning, I will present the ideas introduced in Chapters 2 and 3 of HKP that are important for later chapters.

To introduce some basic concepts, let us consider a Hopfield network for the task of retrieving previously stored images. Each pixel of the image corresponds to a *unit* of the network. The gray level of the pixel is represented by the *activation* of the unit. In a Hopfield network, the units are binary: each unit can only take one of two activation values, similar to spins in statistical mechanics that point either up or down. Furthermore, there are no hidden units. All units are

[3] Rumelhart and McClelland (in *Parallel Distributed Processing*, 1986, subsequently abbreviated as PDP) also begin with networks that do not learn (search and constraint satisfaction). They, however, motivate these tasks from a cognitive perspective; for example, motor control (reaching for a cup), perception (trying to make sense of visual displays of words with partially obstructed letters), and language (forming the past tense of verbs in English). Their models successfully account for variations in response times and error rates, typical quantities measured in cognitive psychology to gain some access to the way the mind works.

''visible"; i.e., they all correspond to image pixels. The units are linked through connections. Each connection has a weight associated with it that specifies the strength of the link. In a typical Hopfield network, the weights between units are symmetric: the connection from unit a to unit b is as strong as the connection from b to a. In the no-learning paradigm, the values of the weights are considered as given–for example, through correlations of the patterns to be stored.

For each configuration of activation values, the distance to the closest previously stored image can be computed; the goal is to find a set of activations that is consistent with a stored image. This search takes place through the space of activations, and the search process–the *dynamics* of the Hopfield model–can be prescribed and described on two levels. The rules that determine how to update the activations live on the ''lower" level. These rules are usually local; the decision of whether to flip or not to flip the activation of a unit depends only on the activations of neighboring units and weights near by. On the ''higher" level lives an energy function, a global quantity that depends on the weights and activations of the entire network.[4] These two levels are complementary: the proof that local updates can minimize the global energy is very clearly presented in the book. It is part of connectionist culture.

The retrieval process begins by presenting the network with a test pattern, usually a partial or noisy version of one of the original images. The hope is that at the end of an iterative process that repeatedly executes the update rules, the activation values of the units settle to those corresponding to the closest pattern in the training set. This process can be pictured as sliding down the basin of attraction (in activation space) to the closest point attractor.[5]

An interesting theoretical question is what *network size* is needed to store a given set of patterns, or conversely, how many patterns can be stored in a network of a given size. This is an area of active research, as we will see in the discussion of effective training set size and effective network size in the section on learning and generalization. A general answer depends on the properties of the pattern set, such as the redundancy within it.

The book calculates the storage capacity only for random patterns. From the perspective of artificial intelligence, excluding all domain structure might seem a bit dull, reminiscent of the drunkard who is looking for his keys not where he lost them but close to the lamp post, because that's where the light is. Another limitation is the requirement of symmetric weights as a sufficient condition for the existence of a global energy function. While this is a natural assumption in physics (it corresponds to Newton's third law), this symmetry requirement distinguishes Hopfield networks from the asymmetric networks used in decision theory (influence diagrams) or in artificial intelligence (such as Bayesian networks or belief networks).

[4]The idea of such a cost function is important in many models, including backpropagation where precisely such a cost is backpropagated. Note, however, that the Hopfield model assumes binary units whereas backpropagation as a gradient descent method requires continuous activations.

[5]These networks are supposed to retrieve one of the stored patterns sharply, even when the test pattern is equally similar to two of the stored patterns. Compare this case to estimating the probabilities that a test pattern was generated by the various stored patterns, a task typical in layered feedforward networks.

So far, we have viewed Hopfield networks as engines that *search* in activation space by performing local updates of the activation values, and we have seen that the similarity to a stored image can be described by a cost function. Minimizing a cost function is a very general method which can also be used for many problems outside pattern retrieval. A standard example of an *optimization* problem that is covered in the book is the traveling salesperson problem, where the length of the path connecting the cities assumes the role of a cost.

Often the goal is to find a good fit rather than the perfect solution–there might not be a perfect solution satisfying all of the constraints. It is useful to distinguish between soft and hard constraints. Soft constraints may be violated, but at a price: a weight in the network reflects the importance of the constraint. Hard constraints must not be violated. They may be coded as logical expressions or have an infinitely large cost assigned if they are violated. Adding a penalty term to the original cost makes it an *effective cost* or effective energy. This idea of an effective cost function will be encountered again both in supervised learning (the effective cost is the trade-off between training error and network complexity) and in unsupervised learning (in clustering, for example, the penalty can correspond to the number of clusters).

Cost surfaces often have several minima. Deterministic update rules end up in the first minimum they find, which might not be the one with the lowest overall value. Introducing some stochasticity in the search process can help escape local minima. *Stochastic update rules* explore the vicinity of the present location: although usually taking a downhill step, an uphill step is allowed with a certain probability. This probability can be varied during the settling process. In analogy to a method used for growing large uniform crystals where the temperature is slowly decreased, this algorithm is called simulated annealing. At the beginning, uphill steps are fairly likely. According to an annealing schedule, their probability is subsequently reduced. The book covers stochastic units and simulated annealing, but comparisons to other optimization techniques are missing. For example, genetic algorithms have proven to be a useful idea for rugged energy landscapes with many minima.

My overall impression of these chapters on search and optimization is mixed. On the one hand, there is not sufficient guidance through the forest of details. The poor novice, craving for more context, background, structure and motivation, feels abandoned in a vacuum shared only with greek letters. On the other hand, these chapters are the best encyclopedic collection of various physics-style results on the Hopfield model that I know.

The most rewarding part was the end of Chapter 4. The authors express neural network equations as electrical circuits and as mechanical systems of springs and masses. After all, the same equations have the same solutions: a quadratic term can always be interpreted as a spring, an integral as a capacitor, and a surface can be minimized by immersing a physical model of the constraints in soap-water. Since people have spent more time obeying the laws of physics than staring at equations, such analogies can evoke intuitions that lead to deeper understanding of otherwise dry formulae. This beautiful collection of insightful physical analogies triggered many ideas and became, quite unexpectedly, my favorite part of the book.

2 Learning and Generalization

To contrast ''learning" without generalization from learning with generalization, let us consider the widely and wildly celebrated fact that neural networks can learn to implement exclusive OR (XOR). But–what kind of learning is this? When four out of four cases are specified, no generalization exists! Learning a truth table is nothing but rote memorization: learning XOR is as interesting as memorizing the phone book. More interesting–and more realistic–are real world problems, such as the prediction of financial data. In forecasting, nobody cares how well a model fits the training data–only the quality of future predictions counts, i.e., the performance on novel data or the *generalization* ability. Learning means extracting regularities from training examples that transfer to new examples.

Learning procedures for connectionist networks are in essence statistical devices for performing inductive inference. There is a tension between two goals. The immediate goal is to fit the training examples, suggesting devices as general as possible so that they can learn a broad range of problems. In connectionism, this recommends large and flexible networks, since networks that are too small might not have the complexity needed to model the data. The ultimate goal of an inductive device is, however, the performance on cases it has not yet seen, i.e., the quality of its predictions outside the training set. This recommends–at least for noisy training data[6]–networks that are not too large since networks with too many high-precision weights will pick out idiosyncrasies of the training set and will not generalize well.

An instructive example is polynomial curve-fitting in the presence of noise. On the one hand, a polynomial of too low an order cannot capture the structure present in the data. On the other hand, a polynomial of too high an order, going through all of the training points and merely interpolating between them, captures the noise as well as the signal and is likely to be a very poor predictor for new cases. This problem of fitting the noise in addition to the signal is called *overfitting.* The discussion of the problem of overfitting in Chapter 6.5 of HKP is very clear.

Given the distinction between generalization and memorization, let us see how the concepts introduced so far have to be modified for networks that can learn. Like optimization, learning can be viewed in terms of a cost function that is iteratively minimized. The difference is that in optimization the function is to be minimized in *activation space,* whereas in learning, the search is through *weight space,* i.e., the weights are adaptive. Another difference is that Hopfield networks for pattern retrieval use the same set of units to represent both input and output. When the goal of the network is to respond to a given input with a certain output, the visible units have to be divided into inputs to the network and outputs from the network. Furthermore, the *binary* units encountered so far are generalized to have *continuous* activation values. Continuous units are particularly suited for probability estimation and regression tasks.

Within learning, several *learning paradigms* can be distinguished, depending on what information

[6]Sources of measurement noise are rounding errors and fluctuations when the data were collected. Another kind of noise is sampling noise due to the finite size of the training set.

is available. If the targets are known exactly, we have the case of supervised learning or learning with a teacher. If the target information is not fully available, we have the continuum ranging from reinforcement learning with a critic–which amounts to ''thumbs up" or ''thumbs down" feedback–to unsupervised learning, where learning means discovering structure in the data.

2.1 Supervised Learning

Supervised learning is at the heart of connectionism: rather than the user guessing the parameters of the network, a learning algorithm determines them from training data. This point is sometimes expressed as ''the network programs itself". The lucid Chapters 5 and 6 of the book discuss supervised learning in neural networks. They begin with the simplest case of networks without hidden units, i.e., networks that consist solely of inputs, outputs and weighted connections between the inputs and outputs. Such networks, sometimes called *simple perceptrons*, are well understood because they are linear systems. Unfortunately, their applications are very limited.

The computational power drastically increases when an intermediate layer of nonlinear units is inserted between inputs and outputs. The example of XOR nicely emphasizes the importance of such *hidden units:* they re-represent the input such that the problem becomes linearly separable. Networks without hidden units cannot learn to memorize XOR, whereas networks with hidden units can implement any Boolean function.[7]

Networks with at least one layer of hidden units (i.e., at least two layers of weights) are called *multi-layer networks.* They have been very successful in applications. Analyzing them, however, is much more difficult than analyzing networks without hidden units. In general, it is important to stretch linear analysis techniques as far as possible–but not further: it is just as stupid to ignore linear results entirely as it is dangerous to apply them blindly to nonlinear cases. A superb example of the fruitful application of linear techniques to the nonlinear case is the analysis of gradient descent in terms of eigenvalues and eigenvectors of the error surface. For linear networks, the optimal value of the learning rate can be related to properties of the training patterns (via the eigenvalues of the covariance matrix). The authors use these linear concepts to discuss methods to accelerate learning in multi-layer networks. Clear figures help visualize the learning.

In addition to the choice of parameters such as the learning rate, further decisions have to be made concerning the *architecture* of the network. An important issue for good generalization performance is the choice of the optimal network size. For multi-layer networks, this means finding the optimal number of hidden units, since inputs and outputs are defined by the problem.

[7] Two classes are linearly separable if there exists a hyperplane that divides all the examples of one class from the examples of the other. XOR is not linearly separable since a straight line cannot divide the four corners of a square such that two diagonally opposite corners lie on one side and the remaining two on the other. Hidden units can re-represent the task by lifting two opposite corners into the third dimension. This allows the two classes to be separated by a hyperplane.

There are some rules of thumb for the network size–which often fail drastically since they ignore both the complexity of the task and the redundancy in the training data. The optimal network size is usually not known in advance.

A straightforward approach to this problem is to hold back some part of the available data, train networks of different architectures on the remaining data and establish their performances on the held-back data. These out-of-sample errors estimate the generalization performances. For sufficiently large networks, the test errors first decrease, then start to increase again. This indicates overfitting: the network at that stage is learning more idiosyncrasies of the training set (i.e., ''noise"-features that do not generalize) than ''signal"-features that do generalize. It is interesting to note that although the number of *available* parameters of the network is fixed, the number of *effective* parameters increases during training. Consider the network at the beginning of the training process when the parameters are initialized with random values. Although all parameters are already present, the number of parameters that are effective for solving the task is zero, since they have not learned to be useful for anything yet. As training proceeds, the number of effective parameters increases, as shown by Weigend and Rumelhart (1991). Training has to stop when the appropriate effective size is reached: the error on the held-back data is monitored during training in order to select the network at the minimum of this error. This method is sometimes called cross-validation.

Although cross-validation is useful in practice, it is difficult to prove theoretical results about it. In order to obtain general results that are independent of a specific input-output mapping and a specific probability distribution of the inputs, the analysis presented in HKP focuses on randomly chosen inputs. Both average and worst case generalization performances are considered as functions of the number of training *examples*. (For each training set size, convergence of the algorithm to the global minimum of the cost function is assumed.) Note the difference between this kind of learning curve and the one described in the previous paragraph dealing with a fixed set of training examples and displaying the estimated generalization error as a function of the number of *iterations* of the algorithm.

Other ideas for arriving at networks of appropriate size include *(i)* penalizing network complexity by adding a term to the error function, *(ii)* removing unimportant weights with the help of the Hessian (the second derivative of the error with respect to the weights), and *(iii)* expressing the problem in a Bayesian framework. Understanding the relationships between these ideas and developing new approaches is an area of active research.

Apart from network size, further design choices concern the error measure and the activation function for the output units. An important theoretical framework is emerging that was not available when HKP was written. The *error measure* reflects the assumptions about the noise in the data. The usual sum of squared errors assumes that the measurement errors are distributed Gaussian and are independent of each other. For binary targets, a binomial distribution of the errors is often more appropriate. It is implemented by evaluating the errors with cross-entropy, $-\sum_i t_i \log y_i + (1-t_i)\log(1-y_i)$, where t_i denotes the target and y_i the output. The subscript i enumerates the patterns (and, if the output is a vector, also the components of the output).

The range of the *output activation function* should match the range required by the problem. Linear units are often appropriate in regression, where the range of the target is not constrained. In classification, however, there are constraints since probabilities only take values between zero and one. Such constraints can be incorporated by an appropriate choice of the output activation function. The table below lists some specific choices that give particularly simple update rules for the weights. Derivations, justifications and underlying assumptions are described by Bridle (1990), Buntine and Weigend (1991), and Rumelhart *et al.* (1993).

Combinations of activation functions and error measures that give simple weight update rules and allow the output values to be interpreted as probabilities.

task	regression	classification *1 of N*	classification *1 of 2* and *k of N*
activation function	linear	normalized exponential	0–1 sigmoid
error measure	squared error	cross-entropy	cross-entropy

Recurrent Networks

So far, we have only considered feedforward networks, i.e., networks that implement a mapping from a set of inputs to a set of outputs. Most connectionists shy away from *recurrent networks*, i.e., networks that include feedback loops. The state of recurrent networks reminds them of early biological classification of species, subspecies, and subsubspecies. Chapter 7 of HKP is suited to dispel that fear in the community. It superbly introduces recurrent backpropagation,[8] giving a lucid overview that is neither overly simplistic nor too technical for an introduction.

A good problem domain for the comparison of feedforward with recurrent networks is *time series prediction*.[9] A simple, non-recurrent approach uses feedforward networks where the inputs correspond to a delay line: the temporal structure is mapped onto a spatial structure, reducing prediction to interpolation. This static approach has two major drawbacks: the maximum size of the temporal window required by the problem needs to be known in advance, and the number of

[8]Reflecting the larger structure of the book, the chapter starts with Hopfield-style networks. With hidden units and a learning algorithm, they are called *Boltzmann machines*. Given the importance of Boltzmann machines for VLSI implementations, more emphasis on applications would have been desirable. In terms of the clarity of the presentation of the learning algorithm for Boltzmann machines, I prefer the description by Hinton and Sejnowski (1986).

[9]Time series prediction is also a good example of the importance of true generalization since it highlights the difference between predicting the future and memorizing the past. In the recent *Time Series Prediction and Analysis Competition at the Santa Fe Institute*, several temporal sequences were made generally available. They ranged from tick-by-tick financial data to chaotic data from a physics laboratory experiment. The task was to predict their continuations. (The true continuations were kept secret until the competition deadline at the beginning of 1992.) Comparing different techniques applied to the same data sets had a surprising result: connectionist networks outperformed all other methods submitted (Weigend and Gershenfeld, 1993).

connections grows with window size, requiring more and more training samples for larger input windows.[10] Recurrent networks avoid some of these problems but are harder to train. Even once a network has learned to emulate the behavior of the system that produced the time series, it is a challenge to extract from the network solution properties that characterize the system, such as the number of degrees of freedom, the degree of nonlinearity, or the amount of noise. Mozer (1993) gives an overview that unifies connectionist approaches to temporal sequence processing.

2.2 Reinforcement Learning

In some cases of learning, there is no teacher who can provide all of the desired output values. This forces us to leave the fairly safe ground of supervised learning. In *reinforcement learning*, a critic provides only one scalar value of evaluation, depending on the correctness of the entire output. Can we invent an explicit teacher signal in order to reduce reinforcement learning to supervised learning? Yes–but assumptions have to be made since one value is going to be ''expanded" into many. For example, when the critic gives positive feedback, one possible assumption is that the present output is exactly the desired output. In this case, the network is rewarded by increasing the magnitude of all weights. When the critic signals incorrect, there are two choices: penalty (the A_{RP} or reward-penalty rule), or inaction (the A_{RI} or reward-inaction rule). These schemes are described well in the book.[11] A distinguishing feature of all reinforcement learning is trial-and-error, or stochastic, search that explores the output space in order to find responses that maximize the reward. I would like to add here some recent developments.

First, critics often give more information than just one value. Consider a basketball player trying to shoot a basket. Although it is certainly true that no direct teacher signal is present for each individual neuron, there is more information available than just whether it was a hit or a miss–for example, information about *how* the ball missed the hoop. This class of problems of a *distal teacher*, or a teacher at a distance, is discussed by Jordan and Rumelhart (1992).

Second, critics often wait for a while until they criticize. In *delayed reinforcement learning*, evaluation is provided only occasionally. The learning system passively observes a whole sequence of inputs and receives the reinforcement signal only at the end. This problem of *temporal credit assignment* is hard, but many complex tasks fall into this category. Tesauro (1992) finds that his temporal difference program learns to play backgammon significantly better than the previous world-champion (Neurogammon, a supervised version of connectionist backgammon) which had already defeated all other AI techniques at the *Computer Olympiad* in London in 1989.

[10] Time series that contain information on several time scales are particularly difficult for feedforward networks. The frequency spectrum of such series is often proportional to the inverse of the frequency. In music, for instance, this 1/f behavior holds over six orders of magnitude, as shown for example in the analysis of Bach's *Kunst der Fuge* by Dirst and Weigend (1993). Mozer (1992) designs networks to specifically cope with multiscale temporal structure and applies them to music composition.

[11] The currently best-understood and most widely-used reinforcement learning algorithm, *Q-learning*, is, however, missing from HKP. Q-learning amounts to an incremental method for dynamic programming. Watkins and Dayan (1992) prove its convergence.

2.3 Unsupervised Learning

After search and supervised learning, the last major topic in the book is unsupervised learning. Teacher and critic are gone; only the data are left. Unsupervised learning is about discovering structure in the data; the term 'learning' no longer refers to an input-output map. Chapter 8 begins with an excellent overview of unsupervised tasks that emphasizes that there is no unsupervised learning without regularities in the data.

Before turning to the analysis given in HKP, let us consider the simple concept of a *pattern matrix*, where each column gives all the measurements of a single variable, and each row gives the whole set of variables measured at, for the sake of being specific, one point in time. We introduce this pattern matrix to clarify two types of questions:

- Can the number of *columns* of this matrix be reduced? For example, if it turns out that one of the measured variables is just the product of two others, this redundancy can be exploited to reduce the number of variables by one, without loss of information.
- Can the number of *rows* of this matrix be reduced? Let us assume that the rows are repetitions of only a few prototypes, and let us picture each row as a single point in the space of the variables. Once all the points are plotted, clusters emerge that correspond to the prototypes. The data can be summarized by giving the centroids of the clusters, or by naming the cluster corresponding to each column.

Variable Reduction

To clarify the first point of reducing the number of variables, assume that n variables are recorded (i.e., each row of the data matrix has n columns), and assume that–due to some limitation–only $h < n$ variables can be transmitted for each observation. How can a network learn a representation that minimizes the loss of information resulting from transmitting fewer variables than observed? Molding it into the framework of error backpropagation, the same trick is used as in reinforcement learning: a teacher, i.e., a set of target values, is invented that allows the generation of an error signal. Specifically, the target is chosen to be identical to the input: the network is given n input units, h hidden units, and again n output units. In learning, the network finds a representation for the hidden units that allows it to reproduce as much of the input patterns at the output as possible (in a sum squared error sense), after piping the signal through the bottleneck of hidden units. This architecture is called an *auto-associator.*

The history of auto-associator networks is a nice example of demystification by theoretical understanding. Initially, when this architecture appeared to work well for image compression, the feeling was that the network seemed to somehow manage to magically re-encode the inputs. Subsequent analysis clarified that an auto-associator with h linear hidden units simply projects the input onto the subspace of the largest h principal components, i.e., the space that maximizes the variance of the signal in h dimensions. Since variance is a quadratic measure, and since this auto-associator minimizes the squared error, both methods project onto the same subspace.

The sole difference is that the principal components are mutually orthogonal (i.e., the first one takes as much variance as possible, the next takes as much of the remaining variance, etc.), whereas a simple auto-associator provides no incentive for the hidden units to form an orthogonal representation. In fact, the variance tends to be spread evenly across the hidden units.

There are two connectionist learning rules that directly project onto this space of the largest h principal components without using an auto-associator. Sanger's rule produces outputs that are orthogonal and correspond individually to the principal component directions. It is a nice connectionist implementation of a principal component computation. Oja's rule gives outputs that have on average the same variance, similar to the hidden units in the auto-associator. In all cases, however, the ensemble of units spans the same space.

As before, it is important to be aware of the limitations of the linear approach and to consider nonlinear generalizations. Principal components only capture *linear two-point correlations*; i.e., information on how each column of the pattern matrix co-varies with each other in a linear way. If higher than second order moments are important (such as in the extreme case of shift detection, where all of the first and second order moments vanish), more powerful concepts than linear correlation are needed. For example, Becker and Hinton (1992) use mutual information to discover surfaces in random dot stereograms.

Clustering

So far, the discussion has focused on the relation between the columns of the pattern matrix. We now turn to similarities between the rows of the pattern matrix. (It is worth keeping in mind that a simple matrix transposition transforms one problem into the other.) Clustering similar rows together means finding similarities between observations.

One way to achieve clustering within a connectionist framework is through *unsupervised competitive learning.* Several outputs compete for input patterns in the following way. A pattern is presented to the inputs, and a feedforward pass determines the response of the network. One of the output units–the one with the largest response–will be the winner. Then, the update rule of competitive learning moves the weight vector of the winner towards the input vector. Thus, when this pattern is presented the next time, the response of the outputs will be more distinct. This process is repeated over and over again for all of the patterns in the training set.

In Chapter 9, the update rule approach is followed by a cost function formulation. As in previous chapters, this cost function contains an error term (such as the distances to the closest center, summed over the training patterns) and a complexity term. The sum of these two terms reflects the trade-off between trying to have many clusters in order to minimize the within-cluster spread, and trying to keep the number of clusters small in order to have less complexity. The authors apply competitive learning to some of the optimization problems introduced earlier. In graph bipartitioning, for example, competitive learning usually clusters the graph into two parts of almost equal size with as few links as possible between them.

Up to now, the ordering of the output units has been random, determined by the initial values

of the weights–the spatial arrangement of the output units was thus meaningless. The final part of the book brilliantly introduces *self-organizing feature maps.* The key idea of self-organizing feature maps is that not only are the weights of the winner increased, but also the weights of its neighbors, although to a lesser degree. This beautifully introduces some topology in the output space: patterns that are close in input space activate output units that are close together in output space, and input patterns far apart in input space activate output units distant in output space.

The output units can be spatially arranged in any way. This gives self-organizing feature maps the freedom of mapping a set of observations (often of many variables) into essentially any user-defined space. Since the output space is usually low-dimensional, self-organizing feature maps are, like multi-dimensional scaling, a technique to reduce dimensionality while trying to preserve the relationships between the observations. The visualizations produced for the book are beautiful and demonstrate the concepts crisply. I do not know of any other description of feature maps that is as easy to understand.

The closing Chapter 10 presents formal statistical mechanics of neural networks. It revisits the Hopfield model by presenting further capacity calculations, but they are ''not for everyone", as the authors freely admit. An important part of the book, however, is its excellent 26-page *bibliography*,[12] impressive in breadth and completeness. I expect that most readers will find new references there to interest them.

3 Use of HKP in Teaching

After this guided tour of the book, I now turn to its usefulness in teaching, based in part on my experiences of using HKP to accompany a course on connectionism and artificial intelligence for computer science seniors and beginning graduate students. After highlighting some of the recurring themes that emerged during the semester, I will give some pointers to connectionist simulators: in my opinion, exploring neural networks with computer simulations is as important as being comfortable with a mathematical/statistical description–these are the two legs of productive connectionist research. I will then mention a few good collections of connectionist research, ranging in scope from tutorial papers to recent results, and close with some general remarks.

Recurring Themes

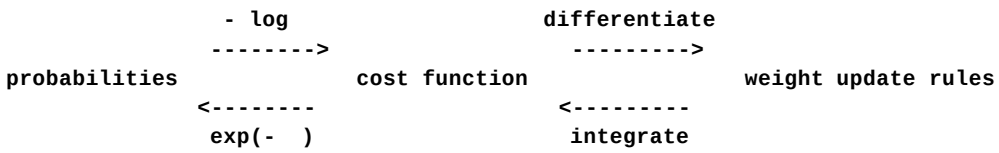
Conciseness, clarity, breadth and objectivity make HKP a superb textbook. In any text (to be precise, in any linear text as opposed to a hypertext), it is hard to transmit ideas on a variety of scales: just as the big picture remains fuzzy if the details are not understood, the details are no fun if the larger goals do not shine through. I would like to bring together here some of the recurring patterns that appear throughout the book.

[12] The bibliography was made publicly available by the authors and translated into BIBTEX format by Nici Schraudolph at the University of California at San Diego. It can be obtained electronically via anonymous ftp at archive.cis.ohio-state.edu (or 128.146.8.52) in pub/neuroprose as hertz.refs.bib.Z .

- *Transition from linear networks to nonlinear networks.* Pattern retrieval starts with the linear associator (Hebb rule) and then introduces the pseudo-inverse for correlated patterns. Supervised learning starts with the Hebb rule, turns to the delta rule (for networks without hidden units), and generalizes it to backpropagation (for networks with hidden units). Unsupervised learning starts with the Hebb rule and generalizes it to nonlinear methods. The first of the general patterns is thus to start with a linear model that can be analyzed with standard methods, and then to introduce nonlinearities.
- *Three levels of interpretation.* The second general theme concerns the advantage of having three levels of description at hand:

  1. Tinkering with (local) update rules
  2. Introducing a (global) cost function
  3. Giving the network a probabilistic interpretation.

  Each level has its advantage. The update rule level is closest to the actual implementation, the cost function level most clearly exhibits the trade-off between training error and model complexity, and the probability level often helps uncover the underlying assumptions. The transitions between the levels can be summarized in a diagram:



- *Express non-supervised learning as supervised learning.* Once additional assumptions make possible the construction of a target that can in turn be used to generate an error signal, one algorithm (error backpropagation) can serve a variety of learning paradigms. For example, reinforcement learning invents an explicit target output that transforms the critic into a teacher. Even completely unsupervised learning can be reformulated as self-supervised learning by again inventing a target output, in this case identical to the input, that allows the generation of an error signal for backpropagation.

Neural Networks and Statistics

One emphasis of this review is the relation between connectionism and statistics. In the discussion of the individual chapters above I stress similarities. Let me focus here on some of the differences.

- Statistical models usually limit the *order of interactions* (such as products between two inputs, or up to three inputs), whereas neural networks tend to limit the *number of features.* The number of features is controlled by the number of hidden units; each feature may represent high order interactions. This assumption is well suited in domains such as optical character recognition, where a feature might require information from a large number of different input pixels.

- Update rules in neural networks are usually *local*; i.e., updating a weight often requires information from the error of a single pattern and from neighboring activations only. (Historically, locality has been an important goal in the search leading to the backpropagation rule.) The flip-side of locality is that both search (settling into activations) and learning (settling into weights) are carried out iteratively. Statisticians usually neither focus on locality nor emphasize the emergence of a solution as a function of the number of iterations.
- Connectionists are usually more generous than statisticians with the *number of parameters* of their models. This often makes interpretation more difficult, and also increases the danger of overfitting. Stopping training early can sometimes alleviate overfitting, since the number of effective parameters steadily increases with the number of iterations.

Connectionists have–not astonishingly–rediscovered problems already well known in statistics. The language game of connectionism, however, often in conjunction with fairly simple algorithms, enables neural networkers to think about old problems differently and often come up with new solutions–and new problems.

In my course, less formally inclined students sometimes complained about the somewhat mathematical approach presented in HKP. However, by the end of the semester most students had learned to appreciate this perspective and realized how easy it is to deceive oneself by believing that connectionism can be fully understood by reducing it to a few metaphors. Yet, my students' favorite part of the whole book was Section 6.3, a good summary of instructive examples and important applications of backpropagation. In teaching and in understanding connectionism, it is important to find a balance between theoretical issues and practical applications.

Simulators and Problem Sets

One of the strengths of the connectionist approach to artificial intelligence and cognitive science is that the model-builder cannot avoid making the assumptions of the model explicit in order to simulate it on the computer. Being forced to be explicit often exposes weak parts in one's own understanding. Interacting with simulators, forming one's own hypotheses, trying them out and modifying them is essential. Learning from computer simulations is to studying a book as experiencing a delicious meal is to studying a restaurant review.

The book does not come with a simulator. Let me give here two suggestions; further information can be found in Skrzypek (1993). A simulator that includes most of the models discussed in HKP and that is publicly available is called PlaNet.[13] In my course, I used the simulator distributed with *Explorations in Parallel Distributed Processing* by McClelland and Rumelhart (1988). This software package is well documented and easy to use. Although it covers only some of the models discussed in HKP, it includes other models that complement the book nicely. Since

[13] PlaNet includes backpropagation, competitive learning, Boltzmann machines, mean field networks, cascade correlation, quickprop, temporal difference learning, sparse distributed memory, recursive tensor product, feature maps and other models. It is written mainly by Yoshiro Miyata and is available via anonymous ftp from tutserver.tut.ac.jp (or 133.15.240.3) in pub/misc .

HKP does not contain any exercises, I assigned some of the problems given in McClelland and Rumelhart (1988). In addition, ''non-neural" problems can help clarify some of the statistical concepts that underly connectionism.[14]

Further Reading

HKP is the best connectionist introduction I know. For further reading, I would like to point out two proceedings series and two edited books. Many of the results given in HKP were presented at the annual conferences *Neural Information Processing Systems* in Denver, Colorado. The proceedings of these conferences (volume 4 is edited by Moody *et al.*, 1992) are important collections of excellent, rigorously refereed papers of high quality. Also good in their wide-ranging and balanced perspective on the state of the art are the proceedings of the *Connectionist Models Summer Schools* (volume 2 is edited by Touretzky *et al.*, 1990, and reviewed by Ahmad, 1993).[15]

The principles and theory behind backpropagation are analyzed in *Backpropagation: Theory, Architectures and Applications* (edited by Chauvin and Rumelhart, 1993). The book offers a number of perspectives including statistics, machine learning and dynamical systems, and presents applications to several fields related to the cognitive sciences, such as control, speech recognition, robotics and image processing.

The increasingly major role that mathematical analysis plays as the field matures in the 1990s is recognized in *Mathematical Perspectives on Neural Networks* (edited by Smolensky *et al.*, 1993). That book provides clear introductions to the many relevant branches of mathematics, accounts of the advances that they have brought in the understanding of neural networks, and discussions of major open problems.

Outlook

Hertz, Krogh and Palmer do a first-class job of collecting, organizing and presenting results that originate in the physics community, such as calculating the storage capacity for random patterns in a Hopfield network. They also present other standard techniques in the same notation and framework, allowing the reader to recognize similarities of approaches from different scientific

[14] At present, we are collecting electronic copies of homework sets and lecture notes of courses taught with HKP. Please contact me if you are willing to contribute problem sets and/or lecture notes. We will make them available via anonymous ftp.

[15] Previous Summer Schools were held at Carnegie Mellon University in 1986 and 1988 and at the University of California San Diego in 1990. The next summer school will be held at the University of Colorado at Boulder in 1993. An important side effect of the summer schools is the creation of the connectionists electronic mailing list. It has grown to include several thousand subscribers and is now a primary vehicle for communication in the field. It is neither a strongly edited forum like the Neuron Digest nor a newsgroup like comp.ai.neural-nets, but somewhere in between, relying mainly on the self-restraint of its subscribers. Membership is restricted to persons actively involved in connectionist research. Requests for addition to the list can be sent to Connectionists-Request@cs.cmu.edu .

communities. An example is the assumption of many simple processing elements with nonlinear interactions, typical of models both in solid state physics and in cognitive and brain sciences.

The rapid growth of connectionism would not have been possible without the general availability of powerful computers for the simulation of the models. This approach is not unique to connectionism; other fields such as artificial life and distributed computational economies also simulate complex ensembles of simple computational agents.

The book would have profited from more links to related disciplines such as statistics and machine learning. On a larger scale, it remains to be seen to what degree a satisfying perspective will emerge that might embrace areas from neuroscience and cognitive psychology to control theory and physical realizations. At present, these disciplines use different methods to study different objects. The research ranges from dissecting lobsters in order to model their stomatogastric ganglion, to using holograms for optical implementations of connectionist models.

Whatever the grand picture will be, neural networks will find their place as another tool in the toolbox. Just as there are no departments of differential equations at our universities now, we will probably not end up with departments of neural networks. But then, just as students now study differential equations, students in the future might study neural networks–hopefully guided by results and understanding rather than by mysticism or dogma. A book like HKP, collecting important theoretical results and making them accessible, is an important step in the right direction.

Acknowledgments

I thank David Rumelhart for always sharing his ideas and intuitions so openly; many of the thoughts expressed in this review ultimately originate in discussions with him. I thank the students at Chulalongkorn University for their feedback during my sabbatical, in particular Patchrawat Uthaisombut and Kitinon Wangpattanamongkol. I also thank Subutai Ahmad (Siemens, München), Stephen Smoliar (National University of Singapore), Mark Stefik (Xerox PARC) and Fu-Sheng Tsung (University of California at San Diego) for their helpful comments. And, last but not least, I thank John Hertz, Anders Krogh and Richard Palmer for this great book.

References

AHMAD, Subutai **Book review of the Proceedings of the 1990 Connectionist Models Summer School**. *Artificial Intelligence*, this volume (1993).

BECKER, Suzanna and HINTON, Geoffrey E. **A self-organizing neural network that discovers surfaces in random-dot stereograms**. *Nature*, 355:161–163 (1992).

BRIDLE, John S. **Probabilistic interpretation of feedforward classification network outputs**. In Fogelman-Soulié, F. and Hérault, J., eds. *Neurocomputing: Algorithms, Architectures and Applications*, p. 223–236. Springer (1990).

BUNTINE, Wray L. and WEIGEND, Andreas S. **Bayesian backpropagation**. *Complex Systems*, 5:603–643 (1991).

CHAUVIN, Yves and RUMELHART, David E., eds. *Backpropagation: Theory, Architectures and Applications*. Lawrence Erlbaum (1993).

DIRST, Matthew and WEIGEND, Andreas S. **On the completion of J. S. Bach's last fugue**. In Weigend and Gershenfeld, eds. (1993).

HINTON, Geoffrey E. and SEJNOWSKI, Terrence J. **Learning and Relearning in Boltzmann Machines**. In Rumelhart and McClelland, eds., vol. 1, p. 282–317 (1986).

JORDAN, Michael I. and RUMELHART, David E. **Forward models: Supervised learning with a distal teacher**. *Cognitive Science*, 16:315–355 (1992).

MOODY, John E., HANSON, Steven J., and LIPPMAN, Richard P., eds. *Advances in Neural Information Processing Systems*, vol. 4 (*NIPS*4*). Morgan Kaufmann (1992).

MOZER, Michael C. **The induction of multiscale temporal structure**. In Moody *et al.*, eds., p. 275–282 (1992).

MOZER, Michael C. **Neural net architectures for temporal sequence processing**. In Weigend and Gershenfeld, eds. (1993).

MCCLELLAND, James L. and RUMELHART, David E. *Explorations in Parallel Distributed Processing*. MIT Press (1988).

RUMELHART, David E., DURBIN, Richard M., GOLDEN, Richard M., and CHAUVIN, Yves. **Backpropagation: Theoretical foundations**. In Chauvin and Rumelhart, eds. (1993).

RUMELHART, David E., MCCLELLAND, James L., and the PDP Research Group, eds. *Parallel Distributed Processing*. MIT Press (1986).

SKRZYPEK, Josef, ed. *Neural Networks Simulation Environments.* Kluwer (1993).

SMOLENSKY, Paul, MOZER, Michael C., and RUMELHART, David E., eds. *Mathematical Perspectives on Neural Networks.* Lawrence Erlbaum (1993).

TESAURO, Gerald **Practical issues in temporal difference learning**. *Machine Learning*, 8:257–277 (1992).

TOURETZKY, David S., ELMAN, Jeffrey L., SEJNOWSKI, Terrence J., and HINTON, Geoffrey E., eds. *Proceedings of the 1990 Connectionist Models Summer School*. Morgan Kaufmann (1990).

WATKINS, Christopher J.C.H. and DAYAN, Peter **Q-learning**. *Machine Learning*, 8:279–292 (1992).

WEIGEND, Andreas S. and GERSHENFELD, Neil A., eds. *Predicting the Future and Understanding the Past: a Comparison of Approaches.* Santa Fe Institute Studies in the Sciences of Complexity, Proc. Vol. XVII. Addison-Wesley (1993).

WEIGEND, Andreas S. and RUMELHART, David E. **Generalization through minimal networks with application to forecasting**. In Keramidas, E., ed. *INTERFACE'91–23rd Symposium on the Interface: Computing Science and Statistics*, p. 362–370. Interface Foundation (1991).