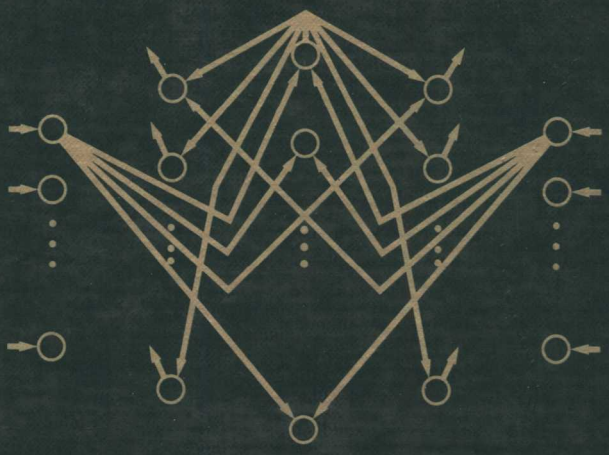
Neurocomputing



Robert Hecht-Nielsen

NEUROCOMPUTING

Robert Hecht-Nielsen

HNC, Inc. and University of California, San Diego

Addison-Wesley Publishing Company

Reading, Massachusetts • Menlo Park, California • New York
Don Mills, Ontario • Wokingham, England • Amsterdam • Bonn
Sydney • Singapore • Tokyo • Madrid • San Juan

Library of Congress Cataloging-in-Publication Data

Hecht-Nielsen, Robert.
Neurocomputing / Robert Hecht-Nielsen.
p. cm.
Includes bibliographical references.
ISBN 0-201-09355-3
1. Neural computers. I. Title.
QA76.5.H4442 1989
006.3--dc20 89-18261
CIP

ABCDEFGHIJ-(DO)-943210

Preface

Neurocomputing is a subject that has captivated the interest of thousands of technologists, scientists, and mathematicians. The idea of training a system to carry out an information processing function (instead of programming it) has intrinsic appeal, perhaps because of our personal familiarity with training as an easy and natural way to acquire new information processing capabilities. Neurocomputing systems are often endowed with a "look and feel" vaguely reminiscent of animals. Like a pet, this is a technology that is easy to fall in love with. I did, 22 years ago.

I hope you find this book useful and enjoyable.

About This Book

The author of this book is an industrialist, an adjunct academic, and a philanthropist without financial portfolio. Activities of these sorts heighten one's awareness of the need for technology textbooks to have industrially oriented material sprinkled in with the purely academic grist. This book is designed with this idea in mind. It contains auxiliary material on subjects as diverse as business plan development, venture capital, proposal writing, and development project planning and management. Discussions about the history of neurocomputing and historical vignettes summarizing the stories of pioneering neurocomputing companies are also presented. The observations that have led to the decision to include this material are sketched below.

Until about 1960, the primary focus of technology was the near-term betterment of the life of the average human being (with the development of weaponry running a close second). However, over the past 30 years some countries (such

as the United States and the Soviet Union — but not, for example, Japan) have built up large technology research infrastructures (separate from the science and mathematics infrastructures) that are, to a distressing extent, disconnected from the practical civilian and military applications of technology (both short-term and long-term). These infrastructures have diverted large numbers of highly talented technologists from useful work. A concomitant of the rise of these infrastructures has been the emergence of a reprehensible snobbery that places all technological efforts into one of two categories: high-tech or I'd-rather-die-than-do-that. The social damage caused by these aberrations has been enormous.

It is now time for technology to regain its traditional primary focus. The formula is the same one that has always worked. Namely: young technologists should work on design, project management, product development, and/or manufacturing. Those of supreme talent and/or pedagogic ability who are appropriately motivated *and* who have proven themselves in practical work can, later in their careers, carry out research and/or teach technology in universities. By addressing some of the issues surrounding the practice of technology in industry, as part of the presentation of a popular and potentially important new information processing technology, perhaps this book can, in some small way, serve to help bring about this renaissance.

Use of this Book in a Course

This text is designed to provide technologists, scientists, and mathematicians with an introduction to the field of neurocomputing. It is intended for use at the graduate level, although seniors would typically have all of the required background. The text is written to support a year-long course. It can also be used for shorter courses if some of the material is skipped or skimmed.

A year-long course can go through the whole book, in order. This would afford an opportunity to explore all three aspects of the subject (theory, implementation, and applications). Instructors can expand or contract the various topics to fit their personal preferences.

A semester-long course can probably still cover the whole book. Chapters 1, 2, 3, 4, 5, and 9 can be covered in detail, and the rest of the book can be skimmed. A quarter-long course can cover most of Chapters 1, 2, 3, 4, and 5, along with a small amount of material selected by the instructor from other chapters.

Concurrent Laboratory Projects

One of the realities of neurocomputing is that hands-on practice is as important as classroom learning. Thus, students learning neurocomputing should, if possible, be involved in laboratory projects concurrent with their classroom instruction. This activity can be done on a group basis or, preferably, on an

individual basis. To help in the planning of a laboratory section to complement a course based on this book, the following suggestions for projects are offered:

- Develop a backpropagation image compression system based on the example in Section 9.2.2. Images of the members of the class can be used for training and testing, with images of automobiles used to demonstrate the problems caused by data that are not statistically consistent with those used during training.
- Simulate a broomstick-balancing system based upon visual feedback, as discussed in Section 9.3.1.
- Write a neurosoftware description (in AXON, for example) of a famous neural network architecture, such as the Boltzmann Machine or GMDH and build a demonstration program that illustrates the architecture's capabilities.
- Solve a "toy problem" using multiple neural networks. The group can first define a suitable toy problem, and then solve it.
- Carry out experimental validation of theoretically derived results from Chapters 3 through 6.
- Develop variants of some of the simpler neural networks described in Chapters 4, 5, and 6.
- Define and solve a simple application problem from start to finish.

It is suggested that projects be organized via the writing of a brief (one or two page) development plan in accordance with the planning approach presented in the Appendix. Projects can last from 1 week to 4 weeks and can be concluded with a detailed report describing the development plan, the work carried out, the results, the software and neurosoftware developed (with listings and permanent archival diskette provided), and recommendations for future work.

Acknowledgments

Writing a textbook involves a major commitment of time and energy and a commensurate loss of availability of that time and energy for other uses. Robert L. North and Gerald I. Farmer (President and Executive Vice President of HNC, Inc.) have been extremely generous in supporting the development of this book during the past 3 years. Thanks are also due to the 100+ students who have patiently sat through an entire year of ECE-270 A/B/C (*Neurocomputing*) in the Electrical and Computer Engineering Department of the University of California, San Diego in 1986/1987, 1987/1988, and 1988/1989. They have been both the guinea pigs for the material of the text and the most helpful assistants in making improvements. The class of 1989/1990 has been helpful in doing final proofreading. Special thanks are also due Lawrence Milstein and Manuel Rotenberg, the successive Chairs of the UCSD ECE Department over the past 4 years, for their unwavering support. Also, the lively interdisciplinary interaction in neural networks at UCSD (across 14 academic departments and the Salk Institute) has been of great value in this project.

The process of synthesizing and refining the material for this book was aided significantly by the following people, whose help is greatly appreciated: Karen Haines (teaching and laboratory assistant for ECE-270 during 1987/1988 and 1988/89), Barton Addis, Timur Ash, Dale Barbour, Steve Biafore, Groff Bittner, Richard Bocker, Dale Bryan, Bill Caid, Maureen Caudill, Subhasis Chaudhuri, Jen Chou, Brad Coté, David DeMers, Duane Desieno, Andrew Diamond, Michael Fennel, Alexander Glockner, Joel Gross, Takeo Hamada, Adam Harris, Hillary Heinmets, David Holden, Geoffrey Hueter, William Jasper, Richard Kasbo, Fouad Kiamilev, John Kim, Myung Soo Kim, Paul Klimowitch, Dennis Kocher, Hari Kuchibhotla, Markham Lasher, Yun-Parn Thomas Lee, Harry Luithardt, Stephen Luse, Tracy Mansfield, Harold McBeth, John McDonnell, John McInerney, Martin McNeill, Robert Means, Phillippe Mercier, Hoa Nguyen, David Olsen, Robert Osborne, Frank Overton, Mark Plutowski, Carl Rindfleisch, John Robinson, John Sabin, Robert Sasseen, Valery Secarea, Holly Shen, Vincent Stuart, Robert Tekolste, Max S. Tomlinson, Anthony Weathers, Eric Wolin, Barnes Woodhall, and Hedong Yang. Comments by Stephen Grossberg and David Casasent were of great value. The author's numerous conversations about the contents of this book with Duane Desieno, Robert Sasseen, and Geoffrey Hueter have been both enjoyable and helpful. Harley Hahn's assistance is also appreciated.

The greatest thanks are owed to Carol Bonomo. She did the majority of the typing, many of the corrections, the initial versions of the figures, and the initial proofreading. She also functioned as a critic, promoting a number of important improvements in the book. Without Carol's contributions this book would not exist. Anna Ewers' help with finishing the last draft of the book has also been invaluable.

The love, patience, understanding, and encouragement of my wife Judi Hecht-Nielsen and our son Andy Hecht-Nielsen are greatly appreciated. Also reflected in this book are the early influences of my mother, Elisabeth Kost; father, Robert Hecht-Nielsen; and grandmother, Jessie Laing Wilson; and the later influences of friends Roy McAlister, David Hestenes, and Alan Wang.

This is a perfect opportunity to thank all of those individuals and organizations that so generously supported my neurocomputing activities when the subject was not fashionable. These include (in chronological order, from 1968 through 1986) the following individuals and organizations: Rennie Molumby, Roy McAlister, Jim Torbert, Hap James, Sid Swanson, Russ Yost, Jerald Bauck, Don Spencer, Todd Gutschow, Robert L. North, Ira Skurnick, Charles Kellum, Robert Launer, Jagdish Chandra, Hugo Poza, Richard Booton, Terry Dolan, Paul Glenn, A. T. LaPrade, Mark Collins, and Gordon Davidson; Arizona State Hospital, Arizona State University, TransEnergy Corporation, Jet Propulsion Laboratory, Chandler Flyers, Ramada Laboratories, Advance Aviation, Talley Industries, TAD Tech, Motorola, and TRW. The encouragement and generous support of DARPA over the years 1982 through 1986 is especially appreciated.

Special thanks go to my friend and business partner Todd Gutshow. He has been a constant source of new insights and ideas, as well as a superb critic and

evaluator. During our years at TRW, Todd and I accumulated neurocomputing applications skills and advanced the state of the art in implementation technology. These experiences became the launching pad for the development of a new generation of approaches at HNC. His help has been an essential ingredient for this book.

After the initial draft of the book was completed it was reviewed by Harold K. Brown, David Casasent, Hans Peter Graf, Stephen Grossberg, Karen Haines, David Hestenes, Stefan Shrier, and Carme Torras. Their many comments and criticisms were of enormous value. Many thanks to them for their generous help.

Finally, I want to thank Tom Robbins of Addison-Wesley for his encouragement, patience, and wise counsel. It was almost axiomatic that a publisher with a hyphenated name would turn out to be a good choice.

Contents

1

Introduction: What Is Neurocomputing?

Neurocomputing is the technological discipline concerned with information processing systems (for example, *neural networks*) that autonomously develop operational capabilities in adaptive response to an information environment. Neurocomputing is a fundamentally new and different approach to information processing. It is the first alternative to *programmed computing*, which has dominated information processing for the last 45 years. This book provides a graduate-level introduction to neurocomputing, including theory, implementation, and applications.

This chapter begins with an overview of neurocomputing. The structure of the field of neurocomputing is then discussed, followed by a discussion of the relationship between neurocomputing and neuroscience. The history of the subject is then surveyed, and finally, a brief overview of the structure of the rest of the book is presented.

1.1 Introduction

1.1.1 Overview of Neurocomputing

From the advent of the first useful electronic digital computer (ENIAC) in 1946 [23] until the late 1980s, essentially all information processing applications used a single basic approach: *programmed computing*. Solving a problem using programmed computing involves devising an algorithm and/or a set of rules for solving the problem and then correctly coding these in software (and making necessary revisions and improvements).

Clearly, programmed computing can be used in only those cases where the processing to be accomplished can be described in terms of a known procedure

or a known set of rules. If the required algorithmic procedure and/or set of rules are not known, then they must be developed — an undertaking that, in general, has been found to be costly and time consuming. In fact, if the algorithm required is not simple (which is frequently the case with the most desirable capabilities), the development process may have to await a flash of insight (or several flashes of insight). Obviously, such an innovation process cannot be accurately planned or controlled. Even when the required algorithm or rule set can be devised, the problem of software development still must be faced.

Because current computers operate on a totally logical basis, software must be virtually perfect if it is to work. The exhaustive design, testing, and iterative improvement that software development demands makes it a lengthy and expensive process.

A new approach to information processing that does not require algorithm or rule development and that often significantly reduces the quantity of software that must be developed has recently become available. This approach, called *neurocomputing*, allows, for some types of problems (typically in areas such as sensor processing, pattern recognition, data analysis, and control), the development of information processing capabilities for which the algorithms or rules are not known (or where they might be known, but where the software to implement them would be too expensive, time consuming, or inconvenient to develop). For those information processing operations amenable to neurocomputing implementation, the software that must be developed is typically for relatively straightforward operations such as data file input and output, peripheral device interface, preprocessing, and postprocessing. The Computer Aided Software Engineering (CASE) tools often used with neurocomputing systems can frequently be utilized to build these routine software modules in a few hours. These properties make neurocomputing an interesting alternative to programmed computing, at least in those areas where it is applicable.

Formally, *neurocomputing* is the technological discipline concerned with parallel, distributed, adaptive information processing systems that develop information processing capabilities in response to exposure to an information environment. The primary information processing structures of interest in neurocomputing are *neural networks* (although other classes of adaptive information processing structures are sometimes also considered, such as learning automata, genetic learning systems, data-adaptive content addressable memories, simulated annealing systems, associative memories, and fuzzy learning systems). The formal definition of a neural network follows.

■ ***DEFINITION 1.1.1*** *A* neural network *is a parallel, distributed information processing structure consisting of* processing elements *(which can possess a local memory and can carry out localized information processing operations) interconnected via unidirectional signal channels called* connections. *Each processing element has a single output connection that branches ("fans out") into as many collateral connections as desired; each carries the same signal*

— *the* processing element output signal. *The processing element output signal can be of any mathematical type desired. The information processing that goes on within each processing element can be defined arbitrarily with the restriction that it must be completely local; that is, it must depend only on the current values of the input signals arriving at the processing element via impinging connections and on values stored in the processing element's local memory.* ■

One might wonder why this particular type of Multiple Instruction Multiple Data (*MIMD*) parallel processing architecture should be worthy of such concentrated attention (beyond the obvious fact that biological neuron networks seem to be neural networks in the above sense). In particular, why not simply study some of the more general MIMD architectures (such as dataflow architectures [57]) which contain neural networks as a subclass? Surprisingly, no completely satisfactory answer to this question is yet known. However, it seems very likely to me that the neural network definition will someday be shown to be a particularly good compromise that allows substantial information processing capability while at the same time providing sufficient structure to allow the development of efficient general-purpose implementations (methods for efficiently implementing arbitrary general MIMD architectures are not known and may not exist). All we can say for sure now is that the neural network definition does produce a class of powerful and potentially useful information processing structures that lend themselves to efficient implementation by general-purpose neurocomputers. It is these consequences of the definition that we will discuss in this book.

To illustrate the nature of neural networks we shall describe briefly a classical neural network architecture known as the *perceptron.* Because it has been largely superceded by more powerful neural networks (for example, some of those discussed in Chapters 3, 4, 5, and 6), the perceptron is primarily of historical interest, although it is still occasionally used.

The perceptron is a neural network that consists of one or more of the processing elements shown in Figure 1.1 (which are themselves also referred to individually as perceptrons). For simplicity, we shall concentrate on the operation of a single perceptron processing element.

The goal of the perceptron is illustrated in Figure 1.2. Here we see two classes of *patterns* (class 0 and class 1). A pattern is simply a point in n-dimensional space (the coordinates of the point represent *attributes* or *features* of the object to be classified, such as weight, height, density, or frequency). In the case illustrated in Figure 1.2 (which is the situation of interest relative to the perceptron), the two classes can be separated from each other by a simple linear hyperplane (in 2-dimensional space a hyperplane is a line, in 3-dimensional space it is an ordinary plane, and in n-dimensional space it is an $(n-1)$-dimensional flat surface). Classes that have this property are termed *linearly separable.* The goal is to find a set of *weights* or *adaptive coefficients* $w_0, w_1, \ldots, w_n$ (which, it turns out, determine a unique hyperplane — as will

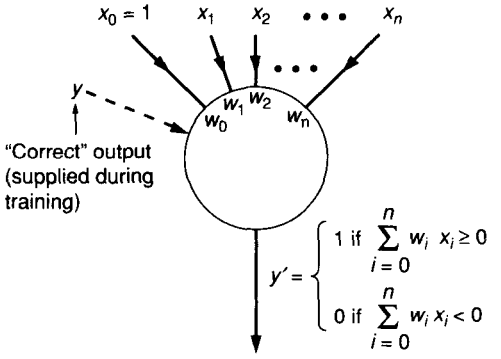


Fig. 1.1. • The perceptron. The perceptron has an input consisting of an $(n + 1)$-dimensional vector $\mathbf{x} = (x_0, x_1, x_2, \ldots, x_n)$, where x_0 is permanently set to 1 (this is called a *bias input*). The output of the perceptron is 1 if the weighted input sum $x_0 w_0 + x_1 w_1 + \cdots + x_n w_n$ is greater than or equal to zero; the output is 0 if this weighted input sum is less than zero.

be discussed in Chapter 3) such that the output of the perceptron is 1 if the input pattern vector $(x_1, x_2, \ldots, x_n)$ belongs to class 1, and 0 if the pattern vector belongs to class 0.

The weights are stored within the processing element and are automatically modified by the processing element itself in accordance with the *perceptron learning law*. This learning law operates during a training process where the perceptron is shown a sequence of randomly selected $\mathbf{x}$ pattern vectors (one at a time). Each time an $\mathbf{x}$ example is presented to the perceptron (as part of a *training trial*), the system is also told to which class (0 or 1) the example belongs. On each training trial, the learning law modifies the weight vector $\mathbf{w}$ in accordance with the equation

$$\mathbf{w}^{\text{new}} = \mathbf{w}^{\text{old}} + (y - y')\,\mathbf{x}, \tag{1.1}$$

where y is the correct class number of the input pattern $\mathbf{x}$ (which is supplied, along with $\mathbf{x}$, on each training trial), and y' is the output of the perceptron. The idea of this learning law is that, if the perceptron makes an error $(y - y')$ in its output, this error indicates a need to reorient the $\mathbf{w}$ hyperplane so that the perceptron will tend not make an error on this particular $\mathbf{x}$ vector (or any other vector near it) again. Note that the output error $(y - y')$ will be 0 if the output of the perceptron is correct. In this situation the weight will not change. If the output is wrong, then $(y - y')$ will be either $+1$ or -1, and $\mathbf{w}$ will be modified appropriately (so that the perceptron will do better in the future).

The perceptron was invented in 1957 by Frank Rosenblatt [199] (who also wrote *Principles of Neurodynamics*, one of the two early books on neurocomputing [198] — the other being *Automat und Mensch* by Karl Steinbuch [216]). Following his invention of the perceptron, Rosenblatt proved that, given linearly separable classes, a perceptron will, in a *finite number* of training trials, develop