NEURAL NETWORKS FOR DECISION SUPPORT: PROBLEMS AND OPPORTUNITIES

by

Shimon Schocken

Gad Ariav

NEUREAL NETWORKS FOR DECISION SUPPORT: PROBLEMS AND OPPORTUNITIES

by

Shimon Schocken
The Leonard N. Stern School of Business
New York University
New York, NY 10003

and

Gad Ariav
The Leon Recanati Graduate School of Business Administration
Tel Aviv University
Tel Aviv, Israel 69978

November 1991

Center for Research on Information Systems
Information Systems Department
Leonard N. Stern School of Business
New York University

Working Paper Series

STERN IS-91-35

Neural Networks for Decision Support: Problems and Opportunities

Neural networks offer an approach to computing which – unlike conventional programming – does not necessitate a complete algorithmic specification. Furthermore, neural networks provide inductive means for gathering, storing, and using, experiential knowledge. Incidentally, these have also been some of the fundamental motivations for the development of decision support systems in general. Thus, the interest in neural networks for decision support is immediate and obvious. In this paper, we analyze the potential contribution of neural networks for decision support, on one hand, and point out at some inherent constraints that might inhibit their use, on the other. For the sake of completeness and organization, the analysis is carried out in the context of a general-purpose DSS framework that examines all the key factors that come into play in the design of *any* decision support system.

Keywords: Decision Support Systems, Neural Networks, Applications.

1 Introduction

The term "neural networks" means different things to different people. Neural networks are popularly perceived as descriptive models, designed to emulate low-level operations in the human brain. Yet most computer scientists view the similarity of artificial neural architectures to the brain circuitry as no more than a useful analogy. This is because neural networks have proven to be powerful computational models in their own right, *regardless* of their biological justification. They offer novel solutions to many problems which defy standard algorithmic techniques, and they lend themselves nicely to new developments in parallel and optical hardware.

Since the inception of artificial neural networks in the mid 1940's, the area has attracted scores of researchers from computer science, psychology, mathematics, physics, and statistics. During the past decade, in particular, numerous networks were constructed to carry out a wide variety of computational tasks, primarily in scientific and in engineering applications. Most of this research, however, has been largely experimental [1]; It remains to be seen whether or not neural networks will become a widely-used computational paradigm, as some proponents of the area claim.

Several hardware and software vendors now offer off-the-shelf products that enable end-users to develop and test neural networks with minimal programming. Is it possible that these programs will eventually become popular *business tools*, like statistical packages and spreadsheet programs? The answer seems to go far beyond user-interface and performance considerations; It is related to deeper issues regarding the appropriateness of neural computing to the special nature of business applications, in general, and decision support systems, in particular. Hence, we take the position that neural networks offer a great deal of promise, on the one hand, and a great degree of uncertainty, on the other; In this paper, we wish to systematically explore the limitations and potential of this technology to decision support applications.

By and large, there are three different ways to think about neural networks: descriptive, computational, and normative. The *descriptive* line of thought, popular among neuroscientists and psychologists, is concerned with the proximity of the artificial model to biological systems. Indeed, the "hardware" of neural networks was much inspired by the architecture and behavior of the brain circuitry, to the extent that we presently (don't) understand it. For example, a real neuron tends to fire only when its combined input exceeds a certain threshold value. This non-linear firing pattern is simulated in the artificial neuron by a numeric activation function which behaves in a similar way.

The *computational* approach to neural networks, popular among computer scientists, views this model as a novel computational paradigm, akin to a Von-Neumann or a Turing machine. Over the last decade, numerous experiments have indicated that neural "machines" can efficiently solve many problems that defeat standard sequential algorithms. This led to the pragmatic conclusion that neural networks have an important computational merit independent of their controversial biological interpretation. Hence, the computational approach to neural network focuses more on the functionality and limitations of the model, and less on its external validity.

Finally, there is the *normative* view of neural networks, which examines the mathematical and statistical backdrop of neural architectures and "learning" algorithms. This approach attempts to cut through the esoteric terminology of the field, and elucidate what neural networks really do in the way of data-analysis. The normative interpretation of neural networks is critically important; First, it shows where neural networks are isomorphic to related techniques like linear regression and cluster analysis. Second, it puts the finger on where neural networks either violate or extend what can be already done with other, more traditional models.

In this paper we adopt the *computational* view of the field, perceiving neural networks as a *resource* that can be used to augment decision support systems. With this pragmatic perspective, we wish to answer such questions as: how do neural networks affect the structural aspects of a DSS? What kind of decision problems lend themselves to neural networks? What are the contingent relationships between the structure (or lack of) the underlying decision problem and alternative neural architectures?

Following Ariav and Ginzberg [6], we observe that the building blocks of any DSS are *environment, components*, and *resources*. This "DSS framework" provides the applied context in which the above questions will be answered. In particular, the plan of the paper is as follows. Section 2 provides a brief overview of key neural network concepts. Section 3 describes the DSS framework. This sets the stage for the next three sections, which survey the typical *environ-*

ment of neural network applications (section 4), the *components* that make up a neural networks-based DSS, (section 5), and the software and hardware *resources* which are presently available to designers of neural networks (section 6). Section 7 discusses several decision support themes which are relevant to neural networks, and section 8 provides concluding remarks. Readers who are not interested in the technical aspects of neural networks may skip sections 2.1-2.3 without losing the main thread of the paper. All the same, we feel that this material is quite readable to those who are willing to endure some minimal mathematical notation.

2 An Overview of Neural Networks

The computer science literature offers several comprehensive reviews of neural networks, and the interested reader is referred to Lippmann [43], Wasserman [61], and Feldman et al [19] for a sample of highly readable surveys/tutorials of the subject. This section attempts to pack such a survey into a few pages, with the objective of equipping the reader with some essential terminology and conceptual understanding of how neural networks are constructed, trained, and used, in the field. In particular, we'll focus on *feedforward* neural architectures and learning algorithms, as they unfold in the context of a typical business *classification* problem. For the sake of clarity, we'll begin with the simplest neural model - consisting of a single neuron - and gradually extend it to a multi-layered, feedforward network of neurons. This model is by far the most widely-used neural architecture in business applications. A neural network is a collection of many independent processing elements, also called "neurons" or "units." Each unit (except those at the network's boundary) is linked to a set of input-units, from one end, and to a single output-unit, from the other. The inter-unit connections are parameterized by a set of numeric weights which modulate the intensity of the messages that go through the network. When a unit receives a set of messages (encoded as numeric values) via its incoming connections, it multiplies them by their respective weights, sums up the result, and passes it to a sigmoidal *activation function* which determines the unit's output. The simplest activation function is the signum function, which acts as follows. If the weighted sum exceeds a certain *threshold value*, the function outputs 1. Otherwise, it outputs -1. This non-linear transformation serves to either cancel out a weak message (if it falls short of the threshold), or, alternatively, amplify it to an upper-bound (if it exceeds the threshold). The resulting message is transmitted via the unit's outgoing connection to another unit, and so on and so forth.

Neural networks are an old emerging technology. Versions of the basic model – also called the *perceptron machine* – were first articulated by McCulloch and Pitts [45] and by Widrow and Hoff [63]. Rosenblatt developed the first perceptron *learning algorithm* [49], and Minsky and Pappert provided a brilliant analysis of their mathematical properties [46]. This early work evolved into a variety of neural network "sub-species" like backpropagation networks [51], counterpropagation networks [39] [29], Hopfield networks [32], associative memories [41], and adaptive resonance networks [11], to mention some leading paradigms. These models differ from each other in terms of their connectivity pattern (e.g., full or partial), topology (free-form or layered), data-flow (cyclical or acyclical), data-type (discrete, binary, or real), output representation (localized or distributed), and learning algorithms (autonomous or feedback). In spite of these differences, though, all neural models are basically different variations on the same theme: a connected collection of many independent processors, working in tandem to carry out a global computational task.

With that in mind, it's important to understand that different neural models don't compete with each other; Rather, they represent different specializations designed to solve different types of problems. In this paper we focus on one specific model which lurks behind most if not all business applications of neural computing: *feedforward networks* (also called *backpropagation networks*). This model, which resulted from the seminal work of the PDP Research Group [51], is widely-used in practice for a number of reasons. First, many important business problems can be cast in terms of *classification*, a generic task which lends itself nicely to feedforward networks. Second, the training algorithm of feedforward networks, known as *backpropagation learning*, is relatively established and well-understood. Finally, and perhaps most importantly, there exist by now many software shells that enable users to build and train feedforward networks with minimal programming.

The extension of feedforward networks from a single-neuron machine to a multi-layered architecture is outlined in figure 1. The simplest model consists of a single neuron designed to classify n-dimensional objects into 2 classes. This basic model can be extended in two different ways. First, in order to clas-

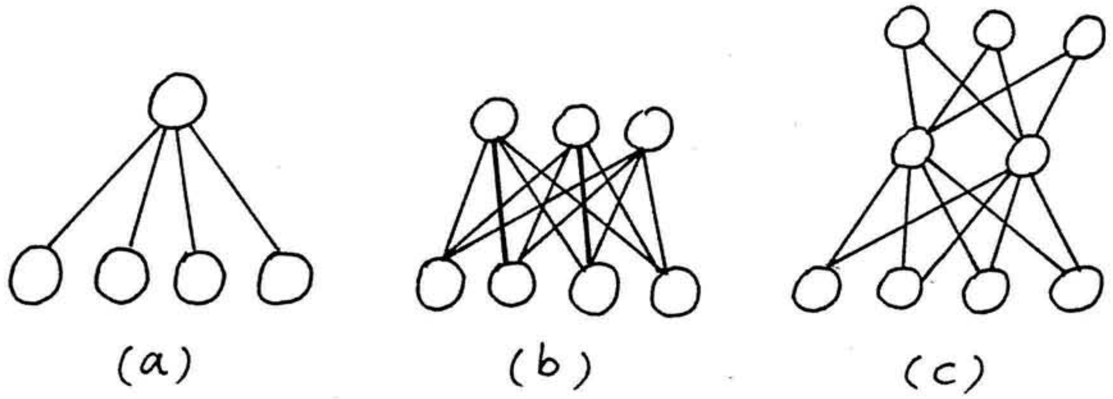

Figure 1: Three successive architectures of feedforward networks

sify objects into $m > 2$ classes, one adds more neurons to the network, each representing a different class (Figure 1-b). Second, in order to classify objects which are not linearly separable, one adds more layers of neurons between the n-ary input-layer and the m-ary output-layer (Figure 1-c). The "hidden layers" are designed to model non-linear boundaries in the objects-space, as will be explained shortly.

Hence, feedforward networks represent an elegant ascent from simple to complex architectures. The extensions from one architecture to another are straightforward, a complex network being a union of simpler networks, all the way down to the level of individual neurons. In other words, all feedforward networks are made up of the same atomic material – independent neurons – arranged in different patterns of connectivity. Therefore, the computational behavior of the *single neuron* holds the key to the behavior of the entire network.

2.1 The Single-Neuron Model

Consider the following problem, taken from the domain of direct-mail marketing. A consumer-products company plans to promote a new product through a fancy (and expensive) mailing kit. In order to cut cost and maximize yield, the company seeks to approach only those customers who are likely to purchase the new product. Strictly speaking, the company wishes to partition its customers-base into two categories: "targets" (c_1) and "non-targets" (c_2). This, of course, is a generic classification problem: the objective is to sort n-dimensional entities (customers) into m classes (here $m = 2$).

We assume that the company maintains a `customers` file and a `transactions` file. The first file keeps track of n customer attributes, denoted $X_1, \ldots, X_n$, e.g. `age`, `income`, `zip_code`, etc. With this notation, the universe of all possible customers is the cartesian product $X = X_1 \times \ldots \times X_n$, a specific customer is denoted $\mathbf{x} =< x_1, \ldots, x_n >\in X$, and the company's `customers` file is denoted $X_f \subset X$[1]. The `transactions` file specifies which customer purchased what product, and when. We also assume that there is a certain metric or a domain expert who can partition the company's products into two sets: products that are "related" to the promoted product, and products that are "unrelated." Based on these assumptions, the company's files can be reprocessed to generate two data-sets, as follows: $X_1 = \{\mathbf{x}|\mathbf{x} \in X_f \texttt{ purchased a related product}\}$, and $X_2 = X_f \backslash X_1$. The first set contains examples of customers who might be interested in the promoted product, whereas the second set contains all the other customers. To complete the problem's setting, suppose now that the company has access to a mailing list, denoted $X_3 \subset X$, consisting of potential adopters of the product whose buying behavior is unknown. Can we use X_1 and X_2 and a neural network to predict which X_3 customers are likely to purchase the new product?

We begin with a simple network, consisting of n input-units and a single processing-unit (figure 2). The input-units are connected to the processing-unit by n "wires" whose "widths" are represented by a set of numeric weights, denoted $w_1, \ldots w_n$, or $\mathbf{w}$. The input-units store the descriptor values (customer attributes), denoted $x_1, \ldots x_n$, or $\mathbf{x}$. The processing-unit is characterized by two mathematical operations: a fixed weighted-sum operator which computes the inner-product $\mathbf{wx} = \sum_i^n w_i x_i$, and a fixed activation function, denoted $g(\cdot)$, which maps $\mathbf{wx}$ on $[-1, 1]$. If the network "thinks" that the customer in question should be classified as "target," it will output 1. Otherwise, it will output -1.

[1]Throughout this section, slanted letters (x), bold-face letters ($\mathbf{x}$), and upper-case letters (X) represent scalars, vectors, and sets of vectors, respectively.

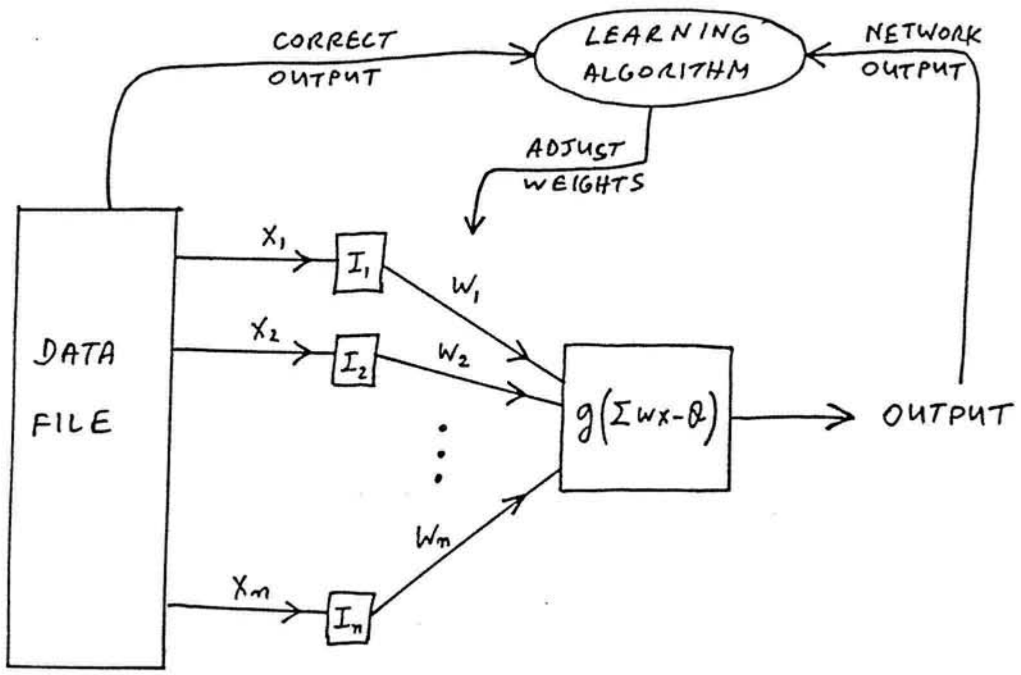

Figure 2: The generic structure of a *neuron*

The training data consists of the sets X_1 and X_2, whose member vectors are known ex-post to be targets and non-targets, respectively. The training phase is a cyclical process in which a "teacher" (which is actually a computer program) repetitively samples objects form X_1 and X_2 and hands them over to the network for classification. After comparing the network's response to the correct classification, an error-minimizing procedure adjusts the network's weights in an attempt to correct wrong classification decisions. Next, the network is fed with another object, and the process continues. The details are as follows:

1. Initialize $\mathbf{w}$ and θ to small random values.
2. Select at random one of the training sets X_1 or X_2 and sample an object $\mathbf{x}$ from it. If $\mathbf{x} \in X_1$, set $d = 1$. Otherwise, set $d = -1$.
3. Compute the classification of $\mathbf{x}$ as follows:
   If $g(\mathbf{wx} - \theta) > 0$, set y=1. Otherwise, set y=-1.
4. If $\mathbf{x}$ was misclassified, compute a new set of weights $\mathbf{w}'$ as follows:
   $w_i' = w_i + \eta \cdot (d - y) \cdot x_i$.
5. Set $\mathbf{w} = \mathbf{w}'$ and go to 2.

The output of the model, which is determined in (3), is denoted y. The parameter η in (4) is a gain-factor between 0 and 1 which determines the rate at which the model converges to a stable set of weights. The *threshold* value θ appears to be fixed in (1-5), but this is done only for the sake of clear exposition. In most models the threshold is made a learnable parameter through the following modification of (1-5): (a) add a new input-unit x_0 to the model and clamp its value to -1; and (b) replace $g(\mathbf{wx} - \theta)$ in step 3 with $g(\mathbf{wx})$ ($\mathbf{w}$ and $\mathbf{x}$ are now $(n+1)$-ary vectors); This way, θ becomes yet another weight (w_0) which is adjusted by (1-5) like all the other weights in the model.

In general, the model is a symmetric implementation of Hebb's rule of learning [28]: *a connection between two neurons should be strengthened whenever both neurons fire.* In the present model, when $\mathbf{x}$ is misclassified, step (4) serves to either increment, or decrement, the weights along the active connections (where $x_i \neq 0$), according to the direction of the classification error. Hence, the weights are continuously modified to promote improved network performance. In fact, the data-driven weights constitute the only "moving parts" of the network's machinery; All the other features of the network, namely the units topology and the functions $\mathbf{w} \cdot \mathbf{x}$ and $g(\cdot)$, remain constant throughout the network's operations.

The termination condition of (1-5) is pragmatic. If the objects space X is linearly separable, the procedure is guaranteed to converge to a stable set of weights (this result is know as Rosenblatt theorem). In other cases the process is halted when all the examples have been exhausted. The entire procedure is the computational reality behind the popular claim that neural networks can "train themselves" or "learn from experience." We see that these anthropomorphic phrases should not be taken at face value: neural learning algorithms are based on blind error-minimizing procedures which are far-fetched from what we normally construe as human learning. It is possible that human learning at its lowest level, namely at the neuron's level, is somewhat similar to (1-5), but there is absolutely no biological evidence that this is indeed the case [14].

2.2 First Extension: More Neurons

Suppose now that instead of classifying the customers into *two* categories, the company wishes to discern three categories, as follows: `prime-targets`, `secondary-targets`, and `non-targets`, denoted c_1, c_2 and c_3, respectively.

(This will make sense if, for example, the company has two types of mailing kits which vary in production and mailing cost.)

In order to represent the three categories, we extend the single processing-unit model as follows. First, the input-layer of the network remains the same, with n input-units, one for each customer attribute. This layer however will be connected not to a single processing-unit, as before, but rather to three processing-units, denoted $y_j,\ j = 1, 2, 3$. Each of these units will be connected to the *same* n input lines by a separate weight vector $\mathbf{w_j} =< w_{1j}, \ldots, w_{nj} >$, $j = 1, 2, 3$ (see figure 1-b, and note that for a fixed j the model reduces to the single-unit model depicted in figure 1-a).

When a new customer's profile $\mathbf{x} =< x_1, \ldots, x_n >$ is fed to the network, the three processing-units compute (in parallel) the three inner-products $\mathbf{w}_j\mathbf{x},\ j = 1, 2, 3$. The output-unit with the largest inner-product is then selected as the most promising category of the classified object. The rationale behind this procedure is as follows. As the network sees more and more examples, the learning algorithm continuously adjusts the three weight vectors. If learning is successful, the weight vector $\mathbf{w_j}$ will eventually store the "average" characteristics (features) of the vectors who are known to belong to the class c_j. With that in mind, the inner-products $\mathbf{w}_j\mathbf{x},\ j = 1, 2, 3$ measure the three vectorial similarities (akin to correlations) between the new vector, $\mathbf{x}$, and the up-to-date average characteristics of the three classes $c_j,\ j = 1, 2, 3$. Leaving the question of whether or not this procedure will work to the next section, we note that the extension of a single processing-unit to a layer of multiple units is straightforward. A single unit is a binary classifier. A layer of m units acts as a slab of m competing classifiers, each measuring the vectorial similarity between its own set of weights and the input vector.

2.3 Second Extension: More Layers

The network described in the previous section will not work for two reasons. First, the network lacks a neural solution to the problem of identifying the output-unit which attained the largest value. This, however, is not a major problem: a second neural network, denoted `maxnet`, can be trained to identify which of m input-units contains the greatest value. The two networks can then be merged, connecting the first network's output-units to the input lines of `maxnet`.

However, it may well be that the resulting network will still fail to classify customers correctly. This is because the objects-space (the world of customer characteristics) is probably not linearly separable, whereas the network, in its present form, can carry out only linear classification. In order to visualize this limitation, picture a two-dimensional objects-space in which each object (customer) is represented by a fixed point (x_1, x_2) in the (X_1, X_2) plane (e.g. *income* and *age* values). Furthermore, assume that the customers are partitioned into three classes, i.e. that some points (x_1, x_2) are labeled c_1, others c_2, yet others c_3. Finally, assume that even though the points exist, you still don't get to see them. At that point you are staring at en empty (X_1, X_2) plane on which three lines are randomly drawn.

Suppose now that you are provided with a ruler and a pencil and you are asked to go through the following iterative exercise: in each step, one point (x_1, x_2) will be exposed, along with its label (one of the three c_j's). Your job (in each iteration) is to separate the three sets of already visible points by adjusting the three straight lines. If at some step you'll fail to separate the points correctly, so will the network. To complete the analogy, denote the slopes and offsets of the three lines in step k by the three pairs $(< w_1/w_2, \theta >_j,\ j = 1, 2, 3)_k$. Note that the entire setting corresponds *exactly* to the network's learning procedure, in which the set of weights and threshold values are continuously adjusted to achieve better classification. If the objects-space is not linearly separable, the network will fail to converge to a stable set of weights: the ruler will continue to oscillate, indefinitely.

As it turns out, most interesting classification problems are not linearly separable. For example, consider the customers space spanned by the attributes $X_1, \ldots, X_n$. In general, it will be naive to assume that the factors that determine a purchase likelihood of a new product are simple linear functions of customer attributes. Rather, it is entirely possible that a customer should be classified into, say, c_1, only if, say, $x_1 + x_2$ is greater than, say, $x_2 + x_3$. To make matters worse, it might be that the boundaries of the class c_1 in the object space cannot be expressed in the languages of mathematical functions *at all.* In other words, in the worst case (which is unfortunately quite common) we are facing an opaque classification problem $f : X_1 \times \ldots \times X_n \rightarrow \{c_1, \ldots c_m\}$ in which f may be not only unknown, but also non-descript in mathematical terms.

However, although we may never know the underlying structure of the elusive

f, we still may have access to many examples of its "operation," namely to vectors $\mathbf{x}$ and their correct classifications $f(\mathbf{x})$. Can we use this raw information to build a machine that simulates f without making any assumptions on its underlying structure? This is precisely what *multi-layered* feedforward networks are supposed to do. In a multi-layered network architecture, the input and the output layers are separated by one or more "hidden" layers of intermediate neurons (see figure 1-c). If the network's topology is well-constructed, some hidden neurons will "learn" to recognize subtle (read: non-linear) interaction effects among the input neurons (the object attributes) and the target classes.

To illustrate the role of the hidden units, consider the quintessential example of using a neural network to map fuzzy hand-written letters on the 22 letters of the English alphabet (e.g. [17] and [48]). Although the following description is a simplification of the actual solution, it does capture its main spirit. The basic network consists of three layers of neurons. The neurons of the *input-layer* are linked to the pixels of a bit-mapped retina, on which the hand-written images are laid. The *output-layer* consists of 22 mutually-inhibiting neurons, each representing a different letter in the alphabet. The key player however is an intermediate *hidden layer* whose neurons are designed to capture distinguishing features in the input images. After seeing many examples of images and their correct classifications, the network will learn, for example, that hand-written versions of $\mathcal{A}$, $\mathcal{H}$, and $\mathcal{F}$, have one thing in common: a (more or less) horizontal line positioned (more or less) in the center of the retina. This will cause a certain hidden neuron to fire whenever an image with a center horizontal line is laid on the retina. Through interaction with other hidden neurons (e.g. detectors of vertical and diagonal lines), the network will learn, from experience alone, which are the *distinguishing features* of the target classes.

Thus, when a certain image, say $\mathcal{L}$, is fed to the network, the hidden neuron that detects a left-sided vertical line will cast parallel "votes" in favor of such candidates as `L`, `H`, and `K`, whereas the detector of a bottom horizontal line will vote for `L`, `B`, `E`, and `D`. The votes will be "tallied" (via inner-product computations) by the output neurons that represent the 22 candidates, and the output neuron with the highest vote will emerge as the winning letter. The process is a bit more complicated, since the hidden neurons also cast negative votes against unlikely targets, and in the final tally those negative votes are as important as the positive ones.

In fact, it has been proven mathematically that any classification problem, no matter how complex (barring some exceptions that generally don't inhibit practical applications), can be simulated by a neural network with only *one* hidden layer of neurons [34] [33]. Yet the mathematical proofs are strictly existential: they don't tell us how many neurons the hidden layer should include, what activation functions they should invoke, and what learning algorithm could be used to train them. These questions are at the forefront of today's neural networks research. Although the construction of an effective topology is still more art than science, network construction guidelines are now beginning to emerge [26] [7] [31] [59].

Another open question regards the *meaning* of the network. Given that a certain network topology is indeed a successful classifier, how can we credibly *justify* the network's operations, short of pointing out at an impressive *empirical* track record? If the network is supposed to serve as a *decision support system*, it must offer a certain degree of intuitive face validity and analytic accountability. We'll return to this critical issue later in the paper, when we discuss the potential use of neural networks in the context of strategic decision making.

3 The DSS Framework

Having described the basic building blocks of feedforward networks and learning algorithms, we now turn to explore their potential use and limitations in supporting a variety of *decision making* activities. The discussion will be organized along the DSS framework of Ariav and Ginzberg, a description of which can be found in [6]. The DSS framework was designed with one objective in mind: help researchers and practitioners think *systematically* about the many factors that come to play in the design and use of decision support systems. Inspired by Churchman's approach [12] to analyzing general systems, the framework is organized around three themes: *environment, components*, and *resources.* The basic premise is that under ideal circumstances, the external characteristics of the system's *environment* have to be reflected in its internal *components*, which, in turn, determine the relevant *resources* required to realize the system. In reality, the linkage often flows backwards: the resources that are *actually* used determine the system's architecture, which, in turn, determines the system's ability to effect its designated environment. This view is mirrored in sections 4, 5, and 6, where we explore, respectively, the *environment, components*, and *resources* that either enhance or inhibit the use of neural networks in decision support applications.

4 Environment

Decision processes are seldom conducted in a vacuum; Therefore, the characteristics of the system's *environment* must be taken into consideration by the system designer. By "environment" we refer to a host of external factors that are relevant to the system's operation, but nonetheless are not under the control of the system's designer. Ariav and Ginzberg partitioned the environment into two major categories: *task characteristics*, and *access pattern*. Table 1 gives a more detailed description of these categories.

The task characteristic that affects decision support most critically is *structurability*, i.e., the degree to which the decision maker can apply a single predefined model to bear on the underlying problem (Ginzberg and Stohr, [24]). The second characteristic is the *managerial level* at which the system is supposed to intervene in the decision process (Anthony, [5]). The third characteristic is the cognitive *phase* that the system is intended to support (Simon, [56]). Finally, the fourth characteristic is the *functional area* of the supported application. It goes without saying that different decision problem vary greatly in terms of their underlying structurability, managerial level, phase, and functional area. Each of these variations implies a different set of constraints and design guidelines on the type of decision support which is called for.

4.1 Structurability

As was pointed out earlier in the paper, numerous experiments have indicated that feedforward networks are especially good at classifying fuzzy objects into concrete categories. As it turns out, many important business problems can be cast in terms of classification: assigning people to jobs, placing customers in different mailing lists, determining the risk ratings of commercial loans, and, in general, mapping different objects onto different categories that then merit different treatments from the decision maker.

Dimension	Characteristics	
Task Characteristics	structur-ability	* low ⋮ * high
	managerial level	* operational * control * strategic
	decision process phase	* intelligence * design * choice
	functional area	* finance * marketing * production ⋮
Access Pattern	pace	* intensive, online ⋮ * slow, intermittent
	user community	* size * domain expertise * computer literacy * role
	neighboring systems	* databases * models

Table 1: The DSS Environment (summary)

In general terms, then, the generic structure of problems that lend themselves to feedforward networks is *mapping*, i.e. $f : X \rightarrow C$. Going back to section 2.1, recall that X is a cartesian product $X_1 \times \ldots \times X_n$ of object attributes, e.g. financial ratios or customer characteristics, whereas $C = \{c_1, \ldots, c_m\}$ represents a set of classes, e.g. credit ratings or customer categories, respectively. The decision rule f represents an unknown mapping that assigns vectors in X to classes in C. In general, f can be approximated *deductively*, if there is some explicit knowledge about the classification's rationale, or *inductively*, if there is a good sample $\{< \mathbf{x}, f(\mathbf{x}) > \mid \mathbf{x} \in X\}$ of representative objects and their ex-post classifications.

Complexity – the "unstructurability" of a decision task – can arise in this context from three sources. First, it may be difficult to discern a good set of *attributes*, as the object descriptors may be redundant, correlated, or downright unmeasurable. Second, the set of target *classes* may be partially unknown, as in the case of mapping new products on a set of standard categories that has to be periodically updated. Finally, and perhaps most critically, the *decision rule* f may defy simplistic interpretations. In such cases, traditional models like linear discrimination rules and Bayesian classifiers will fail to provide a good approximation of f.

In general, statistical classifiers as well as rule-based classifiers (e.g. diagnostic expert systems) make strong assumptions on the structure of X, C, and f. For example, linear discrimination models require that (a) the attributes (explaining variables) $x_1, \ldots, x_n$ be independent; (b) the training sets $\{< \mathbf{x}, f(\mathbf{x}) >\}$ be drawn from populations that have multivariate normal distributions and identical covariance matrices; and (c) the objects-space be linearly separable. Models for rule-based inference under uncertainty (e.g. EMYCIN-type systems [60]) seem to be less restrictive, although an analysis of their underlying "belief languages" reveals that they also make strong (although implicit) assumptions on attributes independence [2] [53]. In addition, rule-based systems require at least a partial specification of f – in this case the rule-base elicited from a human domain expert.

In a rather dramatic contrast, neural networks don't have to make any a-priori assumptions on the structure of f. In order to build a neural solution to the classification problem $f : X_1 \times \ldots \times X_n \rightarrow \{c_1, \ldots c_m\}$, one builds a network in which some units represent the n input values, some units represent the m classes, and the remaining units implement the decision rule f. The key point here is that f need not be explicitly specified; Instead, the neural implementation of f will evolve "automatically:" as the network sees more and more examples of $< x_1, \ldots x_n >$ and their correct classifications $f(< x_1, \ldots x_n >)$, it will adjust the definition of f in a direction which minimize its own classification errors.

Hence, compared with traditional classifiers which are widely used in decision support, neural networks are marked by their minimal requirements with respect to problem structure. This makes neural networks particularly suitable to support complex classification problems in which the mapping rationale is either fuzzy, inconsistent, or completely unknown. We conclude that feedforward networks represent a mixed case with respect to structurability; On the one hand, they require that the underlying problem be modeled along the form $f : X \rightarrow C$. On the other hand, once this minimal structural requirement has been satisfied, the problems themselves can be quite *unstructured*, because no apriori knowledge on f is necessary. Needless to say, under-specified or partially-specifiable models were the raison d'etre of decision support systems in the first place [20].

4.2 Task Level

In his analysis of decision making activities in complex organizations, Anthony described a "pyramid" of managerial tasks consisting of three levels (from bottom to top): operational, control, and strategic [5]. Although it was traditionally argued that decision support systems are mostly needed at the control level, experience has shown that DSS can be used successfully at any task level (e.g. [23] [57] [27]). All the same, the *task level* still plays a major role in determining the nature of the systems that should support various levels of decision making in the organization.

Operational tasks are characterized by routine decisions, automatic transaction processing, and minimal need for human judgment. neural networks are hardly used in this level, with one notable exception: companies whose basic business consists of massive data interpretation, in one way or another. For example, we've already mentioned the possibility of applying neural networks to classify new products into standard categories, based on known classifications of related products. In a similar vein, some organizations use neural networks to index press clips and cross-reference them with existing news databases [3]. Since neural networks are especially effective when there are plenty of examples to work with, such transaction-processing applications lend themselves nicely to neural networks support.

A survey of business applications of neural networks reveals however that the technology is mostly used in the *control* level of management, where it supports decisions that concern resource allocation. Whether they deal with credit rating, scheduling flight crews, or targeting customers, neural networks are typically designed to optimize the use of capital, people, information, and other corporate resources. (Incidentally, resource allocation problems have always attracted decision support systems in general). One obstacle to a widespread use of neural networks at the control level is the need for a technical liaison, namely a network designer. If and when neural network shells will become as easy to use as spreadsheet programs, line managers like credit analysts, factory supervisors, etc., will undoubtedly discover new and creative ways to exploit them in supporting day-to-day decision tasks at the control level.

Strategic decision making, which focuses on such decisions as introducing new products or moving into new markets, can effect the very survival of an enterprise. Technically speaking, much of strategic decision making consists of *evaluating* and then *selecting* alternative courses of action. Neural networks have two inherent limitations that inhibit their use in that level of decision making. First, neural learning algorithms are *inductive*, requiring masses of data and repetitive examples, whereas strategic decision making deals with one-of-a-kind, ad-hoc, type of decisions. Second, neural computing is extremely convoluted, and therefore it is difficult to explain or defend the system's "rationale" (unlike expert systems, where one can trace reasoning chains or invoke some sort of a belief calculus). Therefore, neural networks suffer from low face validity: since their decisions are supported by neither significance tests nor by deductive knowledge, they lack the kind of accountably that is critical in supporting decisions at the strategic level.

4.3 Phase

In his classical research of chess playing strategies and their analogy to human reasoning, Simon [56] identified three generic phases in decision making: *intelligence* (information gathering), *design* (solution construction), and *choice* (solution selection) [56]. DSS researchers used this taxonomy to point out that each phase requires a different set of support services. Traditionally, decision support systems have been employed primarily in the *design* and *choice* phases, with the *intelligence* phase being supported to a much lesser extent. Incidentally, the latter area is precisely where neural networks can prove to be quite useful, thus filling a gap in the range of services provided by other types of decision support technologies.

In business, the intelligence phase consists of an on-going search for problems and opportunities. For example, consider the managerial activities that take place in a mutual funds company. The portfolio manager monitors his positions continuously for non-performing stocks (problems), whereas stock analysts are combing the markets for unknown but promising companies (opportunities). Both search processes are not straightforward, because the factors that make a stock expensive or undervalued are beyond the bounded rationality of most analysts (otherwise the market would have priced the stock correctly). Also, the factors that determine long-term success interact in subtle ways that might go unnoticed by standard analysis. This lack of deductive knowledge, however, can be sometimes compensated by a generous supply of *inductive* experience. Indeed, most mutual fund companies have amassed an abundance of examples of good and bad investments. In some cases, a neural network can be trained to harvest this resource and make investment recommendations based on past experience with related companies, industries, and economic climates.

In contrast, the *design* and *choice* phases of decision making don't lend themselves naturally to neural networks support. These phases focus on *planning* and *selecting*, respectively, alternative courses of action. Since artificial neural networks are not very good in constructing and evaluating solutions, they don't lend themselves to such decision activities. An exception to this statement is the novel use of Hopfield networks in solving hard combinatorial problems (such as integer programming) [32] [36]. Since combinatorial optimization plays a critical role in supporting the *choice* phase of many decision problems, neural networks can help here also, although indirectly.

4.4 Functional Area

For obvious reasons, the business *function* of the underlying task places specific demands and constraints on the type of decision support which is called

Area	Typical Applications
Finance	bankruptcy prediction, customer credit scoring, credit approval, mortgage underwriting, bonds rating, stock and commodity advisory systems, currency trading
marketing	new product analysis, customer characterization, sales forecasting, airline fare management, direct mail optimization
operations	jet engine diagnostic systems, fan motor inspection, assembly and packaged goods inspection, real-time 3D object classification, fabrication plan development, VLSI chip layout, process control, vehicle routing, airline crew scheduling, facility location

Table 2: A sample of business applications of neural networks

for. Table 2 lists representative examples of business applications of neural networks, broken by the functional areas of finance, marketing, and operations management. With the exception of the latter category, neural networks are used primarily in applications that involve forecasting, credit analysis, and customer- or product- classification.

The use of neural networks in economic forecasting is novel, and recent experiments are quite encouraging. Most studies focused on stock market predictions (e.g. [38] and [37]) and on predicting the behavior of individual stocks (e.g. [62]). These studies seem to suggest that when it comes to analyzing time-series, neural networks may have an edge on standard econometric methods because they are capable of picking up and then simulating non-linear relationships in the data set.

Credit rating, a pervasive problem in financial analysis, has attracted numerous neural solutions (e.g. [59] [22] [15] [16] [13]). Since the financial rewards of correct risk analysis are high, financial institutions are constantly on the search for new risk assessment technologies. Indeed, there is a long going tradition in American business of using quantitative methods to analyze risk and forecast defaults, with the practice of bond rating going back to 1919. There is by now a great deal of domain knowledge on the key attributes that determine the financial strength of prospective borrowers, and the historical files of commercial banks contain thousands of examples of good and bad loans. All these factors make credit analysis an attractive application of feedforward neural networks.

In marketing, most business applications of neural networks focus on market segmentation and on targeting customers (e.g. [35] [10]). These, again, are problems that can be cast in terms of fuzzy classification. Given the financial and demographic characteristics of millions of potential adopters, on the one hand, and the attributes of a new product or service, on the other, the problem is to identify the customers who are most likely to make a purchase decision (this application was discussed in section 2.1). If the company has access to many examples of past purchase decisions (in the form transaction files or warranty registration records), a neural network can be trained to classify customers into prime targets, secondary targets, etc. Needless to say, effective targeting holds the key for running efficient direct-mail campaigns, where saving a few cents on each customer can translate to huge savings at the global level.

4.5 Access Pattern

Continuing to follow the DSS framework (table 1), we now turn to discuss another environmental aspect of decision support systems: *access pattern.* Access pattern encompasses three key characteristics of system's usage: (a) the *pace*, or the intensity, of the supported decision process; (b) the given features of the system's *users community*; and (c) the relationship between the system and its *neighboring systems.*

Pace The intensity of decision processes varies from intensive and online (as in crisis management, for example), to slow, evolving, and intermittent (as in strategic planning). Apart from the "natural pace" of the underlying decision process, some DSS resources *impose* long response times, as in the case of lengthy database searches or step-wise optimizations. In the case of neural networks, the *training* phase is a prolonged process that might require *days* of uninterrupted CPU-time. However, once training is over and the network is used in the field, response-time is excellent. Hence, neural networks are adequate in situations where a relatively long setup time is available, and a fast response during normal execution is considered an advantage. Many business applications seem to fit this description. For example, the ability to quickly confirm a loan application is an important competitive advantage in the banking industry. This is at least one reason why several banks are presently studying the potential use of neural networks in supporting the work of loan officers.

User community As was pointed out elsewhere in the paper, neural networks are unique in their inability to "explain" their own decisions to their users. This limitation, coupled with lack of intuitive face validity, places certain constraints on the users community of neural networks. In particular, the network's behavior should be scrutinized by a domain expert, especially in the network's design and learning phases. Furthermore, a network that exhibits excellent *average* performance during its operational phase can still generate freak individual decisions that can go unnoticed without human monitoring. Therefore, users of neural networks should be trained to identify exceptional outputs and decisions which seem to be off-target, in which case a domain expert must be consulted. In that sense neural networks is essentially a decision *support* technology, as final judgment should be reserved to human operators (at least in critical applications).

Neighboring information systems: Since decision processes are never conducted in a vacuum, the presence of *neighboring information systems* must be recognized by the DSS designer. Indeed, we observe that stand-alone DSS's with indirect or no access to other information systems are gradually being phased out in favor of systems which tap directly into corporate databases and information resources [42]. On the other end of this link, most DSS's are now expected to be able to pipe their modeling outputs into neighboring systems. In order to facilitate such an interaction, most neural network shells now provide effective means for incorporating data from standard database and spreadsheet files. This allows users to build and edit training data sets using familiar environments like dBASE and Lotus, and then use the *staging function* of the shell to import the data into the learning schedule of the network.

5 Components

According to the DSS design literature (e.g., [9] and [57]), a decision support system consists of three functional *components*: model management, data management, and dialog management. The three components interact with each other in a pattern that makes up the system's *arrangement*. In the DSS framework, the components are treated at a conceptual level which is kept separate from their actual implementations, the argument being that the DSS design process should not be biased by resource availability.

For example, the designer of a trading DSS need not be constrained from the outset by the presumption that the DSS will be eventually implemented on a spreadsheet program. Starting from the user's view of the trader's job, the designer must think in broad terms about the model-, dialog-, and data management, functions which are called for *by the application*. Only then, the designer must seek the best *resources* to realize his conceptual design. We follow this notion by separating the treatment of components and resources into two independent sections.

Dialog management: During the last decade neural networks have proven to be quite useful in several applications related to user interface design: handwriting recognition, speech recognition, and speech synthesis. For example, Sejnowski and Rosenberg have built a network, called NetTalk, which is capable of correctly pronouncing written text [55]. In a similar vein, Lippmann and his colleagues have shown that neural networks are generally better than other traditional classifiers in the reverse task of speech recognition [44]. These encouraging results should be qualified by the fact that, unlike human speech recognition and synthesis, artificial networks understand nothing about the *meaning* of the underlying text. Nonetheless, they clearly hold promise for enhancing the dialog management function of decision support systems.

Component	Characteristics	
Dialog Management	dialog control	* user-led ⋮ * system-led
	user interface	* menu-driven ⋮ * command-driven
	request constructor	interface to data + model management
Data Management	data depository	* files * databases * spreadsheets
	data directory	data descriptions and definitions
	query facility	interface to dialog and model management
	staging	* manual ⋮ * automatic
Model Management	MBMS	* scattered models * model-base
	model execution	* invoking ⋮ * linking
	modeling command processor	interface to dialog management
	data interface	interface to data management

Table 3: DSS components and their parts (summary)

Data management: The potential use of neural networks in data management for decision support is significant. A great deal of business practice and education focuses on the benefits that managers can reap from searching information about *similar* companies, *related* products, and *relevant* strategies (see for example the PIMS project, case-based reasoning, and hypertext). The fantastic amounts of available data, the absence of a unifying data structure, and the lack of indexing mechanisms, all make "strategic search" a prime target for computer-based decision support.

Indeed, several experiments have indicated that neural networks are well-suited to support a free-form quest for information, using a distributed memory organization and associative recall algorithms [50] [58] [8]. In a neural database (for lack of a better term) such as the Hamming network, the data is not organized in the conventional linear format of files and databases. In fact, the concept of *functional dependency*, which is central in ordinary data models, does not exist in the neural representation of data. Instead, the data is spread across the network in a distributed format that does not lend itself to supporting any one particular query, and allows retrieval of information through inexact or incomplete keys. As a result, users can venture freely in a web of facts and rules, using pattern recognition, rather than rigid indexing schemes, to retrieve data. In the context of a business application, this flexibility will enable managers and analysts to trace chains of associations and recognize patterns in surprising and unpredictable ways. With that in mind, we anticipate that neural networks will play an important role in the data management function of future decision support systems.

Model management: The explicit management of models and the support of modeling activity are the most distinguishing aspects of DSS among all other information processing systems. Correspondingly, the ability to specify, invoke, run, change, combine, and inspect, models is a key capability in decision support platforms. Ideally, the model management functionality should be achieved through a *modelbase* to store models and a *modelbase management system* (MBMS) to handle them [40].

If the DSS platform is intended to support classification tasks, the MBMS should offer access to a variety of alternative models like cluster analysis, discriminant analysis, and induction, using the same terminology and user-interface across the board. Since the feedforward network paradigm can be viewed as yet another data analysis model, its inclusion is a generalized DSS environment should pose no difficulties from the standpoint of software engineering. Once incorporated in the environment, the MBMS should allow users of neural networks to accept inputs from, and pipe outputs to, other models in the model-base.

6 Resources

It is only after the DSS has been *designed* – after the desired architecture has been logically developed – that resources should be taken into consideration. At this point the key questions are: How can the proposed system best be realized by the available technology? How close to the ideal can a feasible system come? Which specific resources should be used to build the system? In this section we are primarily interested in the latter question. Following the DSS framework, we discern four categories of DSS resources, as follows: hardware, software, people, and data (see Table 4).

Hardware resources were critical in the 70's, when DSS's required specialized input/output devices. With the advent of desktop computing in the last decade, the hardware resource are no longer a binding constraint in DSS design. Software resources, on the other hand, still play a prominent role in the design process. Even though any system can be written in any general-purpose *programming language*, most DSS designers augment this basic resource with a variety of *tools* like screen generators, data dictionaries, specialized editors, etc. These resources support at least one of the three major DSS functions – dialog, data, or model management – providing building blocks that speed up the DSS design process.

In addition to hardware and software, the two other resources that come into play in the DSS design process are *people* and *data*. The four resources interact with each other in a number of different ways. Whereas the tradeoff between hardware and software is obvious, there are other tradeoffs as well. For example, the *people* resource (in the form of inexpensive personnel) can sometime substitute the need for an expensive *data* resource (e.g. a computerized news clipping service). In a similar vein, the availability of a *software* resource (e.g. an auditing expert system) can compensate for a scarce *people* resource (e.g. experienced auditors).

Resource	Characteristics
Hardware	* personal computers * workstations * mainframes * storage media * data communication
Software	* languages * tools * generators * environments
People	* designers * operators * "chauffeurs"
Data	* internal * operational * external

Table 4: DSS Resources and some of their characteristics (summary)

Of course, the final architecture of a DSS is always a compromise between ideal design and available resources. If a DSS is implemented on a DBMS platform, it will typically provide good data management support, but little if any support in the way of model management. Conversely, a spreadsheet-based DSS will exhibit opposite strengths and limitations. Although several "integrated" software resources claim to support data, dialog, and model, management equally well, they typically excel in at most one function. Hence, once we commit ourselves to a certain resource, we typically compromise one DSS functionality in lieu of another.

6.1 Hardware

In general there is a sequence of hardware configurations which offer increasingly powerful capabilities for neural computing. The simplest configuration is an ordinary personal computer loaded with a neural networks software simulator (to be discussed shortly). The next step is an ordinary PC or a workstation equipped with an accelerator board which offers parallel processing capabilities (e.g. SAIC's Delta II board, ANZA's Plus board). Finally there are "true" parallel processors like connectionist machines which lend themselves nicely to neural applications. Of particular interest in the future will be optical computers with programmable memory buses. Data transfer in these machines is carried out by laser beams, which, unlike physical circuits (and human synapse) can criss-cross without losing information. As of this writing, optical architectures are beginning to enable the implementation of complex network topologies which are unfeasible on conventional, hard-wired machines.

With respect to processing speed, it's important to reiterate that neural networks are CPU-intensive only in their training phase, when thousands of weights have to be continuously adjusted as the system learns how to classify historical objects, or "cases." Once training is over, however, the network's operation (in the way of classifying new objects) is reduced to computing many inner-products – a straightforward calculation that can be efficiently done by any computer. Therefore, the use of parallel processing and accelerator boards is critical only during the network's design phase. Once the network has been deployed in the field, it can run on practically any machine. This is especially true in business applications of neural networks, where network sizes rarely exceeds several dozen neurons and several hundred connections.

6.2 Software

The DSS framework distinguishes among four types of software resources: programming languages, DSS tools, DSS generators, and DSS environments. Whereas a DSS *tool* is similar to a general-purpose subroutine, a DSS *generator* is essentially a streamlined collection of DSS tools. DSS generators attempt to address all three functions of DSS (at least to some extent). Therefore, they can be used to construct – quite easily – a wide variety of specialized systems in diverse and unrelated areas of application. The fourth software resource – *a DSS environment*, differs from a DSS generator in that it supports a *specific* class of problems. For example, a *scheduling* DSS environment can be used to assign professors to classes, operators to shifts, or specialists to projects. These scheduling decisions share a similar structure, and, furthermore, a common model-base. Therefore, the same generic DSS environment can be used to support them.

When it comes to constructing neural networks, there are essentially three alternatives design approaches. At one extreme, a specific network can be "hard-wired" in software using a conventional language like Pascal or C. Indeed, since the backpropagation learning algorithm was published in 1986, [51] feedforward networks were implemented by numerous researchers, especially those who were interested in studying only one aspect of neural computing. At the other extreme of "canned support" one finds neural network shells like Nestor's "NDS" and Neuralwork's "Professional" – stand-alone software packages that enable the definition, training, and execution, of a wide spectrum of neural network models. In terms of table 4, neural network shells correspond to DSS generators: software environments which can be used to custom-tailor neural solutions to specific problems with minimal programming.

The design process of a new network begins by choosing a particular neural architecture from a library of several dozen candidate architectures (perceptrons, feedforward, autoassociative, etc.). Once a specific architecture has been chosen, the shell invokes a template topology which is placed in a graphical editor. This enables the designer to create arrays of neurons and connections with minimal effort, moving the mouse around and manipulating graphical objects and pull-down menus. Once the network has been constructed, the designer can select a learning algorithm and a training schedule that will best fit the problem at hand. Although training does not require human supervision, some shells allow the designer to intervene in the process and exercise partial control (as in *clamping* weights) during the network's training phase.

The designer-interface of the shell is important because the construction and training of a neural networks can benefit greatly from graphical insights. For example, some shells feature *Hinton Diagrams* [30] – a chart that packs, in a simple to read format, a great deal of real-time information about the network's behavior during learning. Such graphical tools enable the designer to debug the network structure in real-time, as well as pinpoint subtle relationships between the objects-space and the target categories – relationships that might go undetected in a non-graphical interface.

Hence, the shells provide an integrated solution to the tasks of constructing, training, and executing, a wide family of neural network models. The networks that the shells produce can run either on dedicated hardware, or on general-purpose PC's through software simulation. In both cases the shells do all the necessary mappings from user-defined definitions of neurons and connections to the corresponding hardware processing elements (or to their counterpart elements in software simulations).

Aside of the two design extremes – building a network from scratch through programming versus customizing a template network with a shell – there is also the interim alternative of using a library of neural utilities that can be compiled with conventional languages. For example, the Axon language (by Hecht-Nielsen Neurocomputers) allows users to build networks via an object-oriented definition language that can be compiled with regular C code. In a similar vein, the Rochester Connectionist Simulator [25] [19] offers an elaborate library of procedures that can be compiled with C source code to produce custom-made neural architectures. The library contains procedures for setting up arrays of neurons and connections, selecting (and even defining) activation functions, and controlling learning parameters. In addition, the simulator features a programmable user-interface in which networks can be displayed and tested in real-time. Using programming, the designer can determine which neurons and connections will be displayed, what shapes and sizes they should attain, and what frequency should govern the display process (continuous, only when the neuron fires, only when the neuron's output changes, etc.) Whereas neural network *shells* conform to the definition of *DSS generators*, neural *libraries* are essentially *DSS tools*. As tools, they don't provide a complete design solution, but rather a set of building blocks that speeds up and streamlines the network design process.

We see that contrary to common belief, the task of building a neural network is technically straightforward; The hard part is the logical design of the network, namely selecting the appropriate architecture at the outset and constructing an effective network topology (effective in terms of classification). Once these questions are resolved, the actual implementation of the network is a straightforward technical exercise that can be assisted by a variety of software design aids.

6.3 Data

Unlike the deductive nature of expert systems, neural networks learn to perform their designated tasks by example, using *inductive* learning algorithms. In the absence of a fixed inferenecing mechanism, neural networks require massive data sets in order to achieve meaningful learning. In the case of classification, the data sets consist of examples of historical objects (e.g. companies) whose class-memberships (e.g. bond-ratings) are known *ex post facto*. The goal of the learning algorithm is to use these data to teach the network how to correctly classify new objects (e.g. assign credible bond-ratings to new prospective borrowers).

Due to the centrality of data in the learning process, special attention must be paid to the data's structure, quality, and quantity. As with many other data analysis models, the main challenge is to reduce wholistic objects (like companies) to structured tuples of attributes, or "descriptors." The chosen attributes must be complete, relevant, measurable, and independent [54]. Once a data structure has been built, a mechanism has to be designed to filter data into it from historical files. In business applications, external data sources (like industry and trade databases) are typically used to supplement internal data sources.

The critical role that *data* plays in neural applications implies that DSS that include, or are based on, neural models, must be equipped with powerful data *staging* mechanisms. In the DSS framework, "staging" refers to *importing* data from operational and historical databases, *preprocessing* data in a variety of different ways, and *piping* data into specific DSS modules. The designer of neural networks can use these services to eliminate irrelevant attributes, aggregate other attributes in order to counter multi-colinearity, and translate attributes from one data-type to another, as dictated by the network architecture. In addition, staging services might include access to a variety of *sampling retrieval* techniques, in which an algorithm methodically selects "good" training objects from a relatively small sample of data. (Lack of data is a notorious problem in neural networks training, and modern sampling techniques like those described in [18] are likely to play a major role in alleviating this shortcoming.)

6.4 People

Since the operation of a neural network is not based on explicit reasoning and textbook knowledge, there is a no need for knowledge engineering in the conventional sense of the word. Rather, the network design process draws on the expertise of three individuals: a network designer, a domain expert, and a data specialist. Instead of eliciting rules, the designer and the domain expert focus on defining attributes, discerning classes, and formulating a network topology. To a large extent, the topology is based on assumptions about attributes interaction, on the one hand, and on the *separability* of the objects-space (the geometry of the decision boundaries), on the other. These assumptions are articulated and then tested through a dialog between the network designer and the domain expert.

Once the network has been set up, the learning process must be fueled with massive amounts of data. Since the data must conform to the topology and typology of the input neurons, preprocessing and staging are inevitable. This is where the data specialist, e.g. a database administrator, enters the picture. In many cases, one needs to pool data from multiple sources like transaction files, historical databases, and spreadsheets. This, in turn, opens the door to the perils of data redundancy and inconsistency. Hence, access to raw data is a necessary, but insufficient, condition for proper learning. A corporate data dictionary and a cooperative database administrator are essential ingredients as well.

7 Decision Support Themes

We conclude the paper with a brief discussion of three critical subjects for prospective adopters and educators of neural networks: *face validity* considerations, the interplay of *neural networks and expert systems*, and the *teaching* of a neural networks course or module in academic institutions and in industry training programs.

7.1 Face Validity

Neural networks are constructed in a step-wise fashion. First, a general architecture is chosen, and a specific network topology is laid out. The number of neurons and layers are determined according to "generally-accepted" construction rules, but the actual topology of the network evolves more or less through what may be termed an educated trial and error process. More often than not, a certain topology will prove to be an effective classifier, but the designer will not be able to *explain* why this network is better than others. Also, the weights that emerge from the learning process are not easily labeled, and their meaning with respect to the features of the classified objects is not directly discernable.

Recalling the complex non-linear computations that neural networks perform, It is not surprising that their structure and operations defeat simplistic explanations. When it comes to *real* brian circuitry, there is a similar phenomenon: most cognitive processes are hard to explain, and yet we tend to rely on them almost blindly, without ever stopping to question their validity. For example, although it is hard to justify formally why we prefer a certain person to a certain job, we don't hesitate to use our judgement and make a hiring decision. The reason for our confidence is twofold. First, we are dealing with a familiar classifier – our own brain. Knowing that we've already used this classifier (whichever form it might have) to make many good hiring decisions in the past, we are willing to give it another try. Second, we typically entertain the belief that even though we might err, we'll never err big. For example, although we might end up hiring a lazy secretary, it's unlikely that he or she will shred all the company's files or set the office on fire.

Unfortunately, the same sense of confidence does not translate well to artificial neural networks. Even though we can confirm empirically that a particular network performs well on a large training set, we often can't explain this success analytically. Therefore, we can't guarantee that a network will not make freak decisions at some point in the future, and we can't justify its analytic rationale. This problem places neural networks is an disadvantage, compared to other, more traditional classifiers. For example, statistical models offer significance tests that enable the user to assess with precision the statistical power of the model's predictions; Diagnostic expert systems offer another form of accountability: they allow the user to trace reasoning chains and understand the rationale that led to a certain recommendation.

Unlike statistical models and rule-based classifiers, neural networks offer no simple and convincing means to assess the credibility of their outputs. Furthermore, it is difficult, and often impossible, to make sense of intermediate network constructs and "revealed" features. Hence, it is not clear to what extent the application of a neural network can contribute reliably to a deeper *understanding* of the underlying problem. This might present problems in the context of decision support systems, as the facilitation of learning is cited by some researchers as *the* main source of DSS value [47]).

At present, the only way to "sell" a neural solution to a decision maker is to point at an impressive *empirical* track record, which is typically better than those of linear regression, discriminant analysis, induction, and diagnostic rule-based algorithms. This poses an interesting dilemma to prospective clients of neural networks: should they use a classifier with, say, 90% hit-rate but vague accountability, or, rather, one that offers 70% hit-rate and excellent accountability[2]? We believe that this dilemma has caused many DSS practitioners to shun away from neural networks for business applications, where one has to be accountable to one's decisions. Clearly, "teaching" a network to explain its own operations is going to be a major research challenge in the next few years.

7.2 Neural Networks and Expert Systems

neural networks are related to expert systems in a number of ways. First, neural algorithms can be used to extract a set of rules that specify how an expert has reached past decisions in a certain domain. [21] [52]. This application of neural

[2]By "accountability" we refer to some form of analytic, normative, or logical, justification.

networks is somewhat similar to the use of the ID3 algorithm in knowledge elicitation. Both methods rely on massive amounts of data (examples), and both involve an optimization process: minimizing classification error in the case of neural networks, maximizing entropy in the case of ID3.

Second, a set of rules can be used to override or modify the training schedule of a neural network. Instead of invoking an unattended learning algorithm that walks the network through thousands of examples in a blind fashion, a domain expert (who is also trained in neural computing) can be assigned to oversee the learning process. Monitoring the classification behavior of the network as it "sees" new examples, the expert can intervene in the learning process by forcing certain neurons to certain outputs, clamping weights to fixed values, and fine-tuning the network's topology.

Finally, it is feasible that future systems will be based on hybrid architectures that incorporate elements of both inductive and deductive reasoning [4] [1]. When one extracts rules from a domain expert, one often hears statements like: *at this point I would make this or that decision, but I can't explain exactly why.* If the expert will be able to provide many *examples* of such intuitive judgements, a neural networks could be trained to simulate his local decisions, without attempting to explain them. The inputs and outputs of the network can then be linked to the heads and tails, respectively, of standard `IF ... THEN ...` rules. This, again, is a promising and open area of future research.

7.3 Teaching Neural Networks

The introduction of a new topic to an education program is always constrained by the availability of three resources: established course curricula, textbooks, and software. This is particularly true in the case of neural networks, a subject which is relatively unknown to most instructors, and yet can be covered effectively in many different ways and styles. We conclude this section with a few ideas on how neural networks can be taught in academic courses and in corporate training programs.

In a computer science department, the study of neural networks can easily fill a one-semester or even a two-semester graduate-level elective course. The theoretical part of the course will focus on the mathematical and statistical backdrops of neural architectures and algorithms, whereas the applied part might consists of student presentations of key applications and applied term projects. In business information systems departments, the subject does not seem to merit an entire course. Instead, a more balanced treatment would be a neural networks module (of, say, 2 to 6 class meetings) within an elective course on expert systems, AI applications, or decision support systems. In order to minimize confusion and avoid technical clutter, it is recommended that this module will expose the students to no more than a single neural paradigm, feedforward networks being a natural candidate. In industry, the need for neural networks education arises when companies wish to either see "what this technology is all about" or train systems analysts and knowledge engineers to apply the technology to business applications. For this purpose, several companies now offer a 3-days or a one week workshop of intense training. All vendors of neural networks products also offer such workshops, but for obvious reasons their own products often take the center stage in these programs.

As regards literature, there are by now several excellent neural networks survey/tutorial books. In this rapidly developing field, *"Parallel Distributed Processing"* by Rumelhart et al. [51], a 1986 publication, is now considered a classic. This two-volume book gives a comprehensive review of all the key ideas that led to the development of feedforward networks and backpropagation algorithms, along with a series of articles on theory and applications written by some of the top researchers in the field. As such, it can be used to support a full-semester graduate-level course on neural networks in a computer science department. On the lighter side of the literature, there are several introductory books which vary in quality and focus. Some authors have managed to give an accurate and compact view of the field without getting into too much technical clutter; Wasserman's *"Neural Computing: Theory and Practice"* [61] and Alexander and Morton's *"Introduction to Neural Computing"* [4] are good examples. These books are very readable, and they can be best utilized in supporting elective courses in information systems programs.

In addition to literature, there is by now a good selection of reasonably priced and quite powerful neural network software shells. These shells can be easily installed on personal computers and thus in academic PC labs. Most software vendors, like Nestor and KnowledgeWare, offer educational versions of their shells for significant discounts. Shells that originated from academic institutions, like the Rochester Connectionist Simulator, can be also obtained (by other universities) for nominal fees. In short, instructors who wish to teach a course or a module on neural networks will find a good selection of books and software to choose from.

As a rule, it is our belief that instructors should avoid the trap of building their curricula around specific commercial tools. Instead, the tools should be used chiefly in the way of *demonstrating* fundamental ideas and making the students educated consumers of neural network products. This can be done by arranging the curriculum around the sequence of (a) analyzing a *real* problem that calls for decision support; (b) developing an "idealized" system design; and (c) assessing and critically examining which neural (as well as conventional) resources will best realize the ideal design.

8 Conclusion

During the last decade, neural networks have proliferated to so many directions that it is no longer clear what the term "neural networks" stand for. In order to keep track of the field, one has to monitor several journals in neuroscience, computer science, psychology, mathematics, physics, and statistics. Worse yet, it is entirely possible that the solution to a certain marketing problem is buried in a physics journal, where is applied to analyzing spin glass. Hence, potential adopters of the technology are quite confused by the overwhelming gamut and esoteric terminology of neural computing. The situation is not helped by the multitude of hyperbolic articles on neural networks which appear in an alarming rate in popular journals and in conference proceedings.

For this reason, DSS designers and practitioners often find it difficult to map the essential features of the technology on their *actual* decision support needs. With that in mind, we've taken in this paper a different stance, one that assumes that the acceptability (or the lack of it) of any new DSS resource must be examined in the *functional* context of its potential use. Hence, rather than focusing on the technology itself, we gave a systemic description of the various components and features that characterize the development of *any* decision support system. Only then we proceeded to map the capabilities and limitations of neural networks on DSS design and application. We believe that this view might will help researchers and practitioners realize the exciting possibilities that neural networks entail, without losing sight of the limitations that still inhibit their use in decision support applications.

References

[1] *DARPA Neural Network Study.* Fairfax, VA: AFCEA International Press, 1988.

[2] J.B. Adams. Probabilistic reasoning and certainty factors. In B.G. Buchanan and E.H. Shortliffe, editors, *Rule-Based Expert Systems*, pages 263–271, Addison-Wesley, 1984.

[3] E.R. Addison. Using news understanding and neural networks in foreign currency options trading. In *Proceedings of the 1st Intl. Conf. on Artificial Intelligence Applications on Wall Street*, pages 319–323, 1991.

[4] I. Alexander and M. Morton. *An Introduction to Neural Computing.* London, UK: Chapman and Hall, 1990.

[5] R.N. Anthony. *Planning and Control Systems: A Framework for Analysis.* Technical Report, Cambridge, MA: Harvard University GSBA, 1965. Studies in Management Control.

[6] G. Ariav and M.J. Ginzberg. Dss design—a systemic view of decision support. *Communications of the ACM*, 28(10):1045–1052, 1985.

[7] E.B. Baum and D. Haussler. What size net gives valid generalizations? *Neural Computation*, 1:151–160, 1988.

[8] E.B. Baum, J. Moody, and F. Wilczek. Internal representations for associative memory. *Biological Cybernetics*, 1988.

[9] R.H. Bonczek, C.W. Holsapple, and A.B. Whinston. *Foundations of Decision Support Systems.* New York: Academic Press, 1981.

[10] J.E. Bowen. Marketing and artificial intelligence: a neural network segmentation example. In *Proceedings of the 1st Intl. Conf. on Artificial Intelligence Applications on Wall Street*, pages 251–256, 1991.

[11] G. Carpenter and S. Grossberg. A massively parallel architecture for a self-organizing neural pattern recognition machine. *Computer Vision, Graphics, and Image Processing*, 37:54–115, 1987.

[12] C.W. Churchman. *The System Approach.* New York: Dell Publishing, 1968.

[13] E. Collins, S. Gosh, and C.L. Scofield. An application of multiple neural network learning system to emulation of mortgage underwriting judge-

ments. In *Proc. of the Int. Joint Conf. on Neural Networks, San Diego: SOS Printing*, pages II–459–466, 1988.

[14] F. Crick. *What Mad Pursuit.* New York: Basic Books, 1988.

[15] Cadden. D.T. Neural networks and the mathematics of chaos – an investigation of accurate predictions of corporate bankruptcy. In *Proceedings of the 1st Intl. Conf. on Artificial Intelligence Applications on Wall Street*, pages 52–57, 1991.

[16] S. Dutta and S Shekhar. Bond rating: a non-conservative application of neural networks. In *Proc. of the Int. Joint Conf. on Neural Networks, San Diego: SOS Printing*, pages II–443–450, 1988.

[17] G.M. Edelman. *Neural Darwinism.* New York: Basic Books, 1987.

[18] B. Efron and G. Gong. A leisurely look at the bootstrap, the jackknife, and cross-validation techniques. *The American Statistician*, 37(1):36–48, February 1983.

[19] J.A. Feldman, M.A. Fanty, N.H. Goddard, and K.J. Lynne. Computing with structured connectionist networks. *Communications of the ACM*, 31(2):170–187, 1988.

[20] Gorry G.A. and M.S. Scott-Morton. A framework for management information systems. *Sloan Management Review*, 13(1):55–70, 1971.

[21] S.I. Gallant. Connectionist expert systems. *Communications of the ACM*, 31(2):152–169, 1988.

[22] S. Garavaglia. An application of a counterpropagation neural network: simulating the s&p corporate bond rating systems. In *Proceedings of the 1st Intl. Conf. on Artificial Intelligence Applications on Wall Street*, pages 278–287, 1991.

[23] M.J Ginzberg. Dss success: measurement and facilitation. In C.W. Holsapple and A.B. Whinston, editors, *Data-Base Management: Theory and Applications*, pages 367–387, Dordrecht, Netherlands: D. Reidel, 1983.

[24] M.J. Ginzberg and E.H. Stohr. Decision support systems: issues and perspectives. In M.J. Ginzberg, W.R. Reitman, and E.A. Stohr, editors, *Decision Support Systems*, pages 9–32, North-Holland, Amsterdam, 1982.

[25] N.H. Goddard, K.J. Lynne, and T. Mintz. *Rochester Connectionist Simulator.* Technical Report, Technical Report 233, Computer Science Department, University of Rochester, 1988.

[26] R.P. Gorman and T.J. Sejnowski. Analysis of hidden units in a layered network trained to classify sonar targets. *Neural Networks*, 1(1):75–90, 1988.

[27] G.A. Gorry and R.B. Krumland. Artificial intelligence research and decision support systems. In J.L. Bennett, editor, *Building Decision Support Systems*, pages 205–219, Reading, MA: Addison-Wesley, 1983.

[28] D.O. Hebb. *The Organization of Behavior.* New York: Wiley and Sons, 1849.

[29] R. Hecht-Nielsen. Counterpropagation networks. *Applied Optics*, 26(23):4979–4984, 1987.

[30] G.E. Hinton. Learning distributed representations of concepts. In *Proceedings of the 8th Annual Conference of the Cognitive Science Society*, pages 1–12, 1986.

[31] C. Ho. On multi-layered connectionist models: adding layers vs increasing width. In *Proceedings of the 11th Intl. Joint Conf. on Artificial Intelligence (Volume 1)*, pages 176–179, 1989.

[32] J.J. Hopfield and D.W. Tank. Neural computations of decisions in optimization problems. *Biological Cybernetics*, 52:141–152, 1985.

[33] K. Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4:251–257, 1991.

[34] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366, 1989.

[35] W.R Hutchinson and K.R. Stephens. The airline marketing tactician: a commercial application of adaptive networking. In IEEE *First Int. Conf. on Neural Networks (pp. IV:753-757) San Diego: SOS Printing*, 1987.

[36] D. Johnson. More approaches to the travelling salesman guide. *Nature*, 330, December 1987.

[37] K. Kamijo and T. Tanigawa. Stock price recognition – a recurrent neural net approach. In *Proc. of the Int. Joint Conf. on Neural Networks, San Diego: SOS Printing*, page I:589, 1990.

[38] T. Kimoto and K. Asakawa. Stock market prediction with modular neural networks. In *Proc. of the Int. Joint Conf. on Neural Networks, San Diego: SOS Printing*, pages I:1–6, 1990.

[39] T. Kohonen. *Self-Organization and Associative Memory.* New York: Springer-Verlag, 1988.

[40] B. Konsynski. On the structure of a generalized model management systems. In *Proc. of the 14th Annual Hawaii International Conference on System Sciences*, pages 19–31, 1980.

[41] B. Kosko. Bi-directional associative memories. *IEEE Transactions on Systems, Man, and Cybernetics*, 18(1):49–60, 1987.

[42] L.J. Laning, G.O. Walla, and L.S. Airaghi. A dss oversight—historical databases. In G.W. Dickson, editor, *DSS-82 Transactions*, pages 87–95, 1982.

[43] R.P. Lippmann. An introduction to computing with neural nets. *IEEE ASSP Magazine*, 4–21, 1987. April 1987.

[44] R.P. Lippmann. Review of neural networks for speech recognition. *Neural Computation*, 1:1, 1989.

[45] W.W. McCulloch and W. Pitts. A logical calculus of the ideas imminent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–33, 1943.

[46] M. Minsky and S. Pappert. *Perceptrons: an Introduction ro Computational Geometry.* Cambridge: MIT Press, 1969.

[47] Keen. P.G.W. and T.J. Gambino. Building a decision support system: the mythical man-month revisited. In J.L. Bennett, editor, *Building Decision Support Systems*, pages 133–172, Reading, MA: Addison-Wesley, 1983.

[48] A. Rajavelu, M.T. Musavi, and M.V. Shirvaikar. A neural network approach to character recognition. *Neural Networks*, 2(5):387–394, 1989.

[49] F. Rosenblatt. *Principles of Neurodynamics.* New York: Spartan Books, 1962.

[50] D.E. Rumelhart, G.E. Hinton, and R.J. Williams. Learning internal representations by error propagation. In D.E. Rumelhart and J.L. McClelland, editors, *Parallel Distributed Processing: Exploring the Microstructure of Cognition*, Cambridge, MA: MIT Press, 1986.

[51] D.E. Rumelhart and J.L. McClelland. *Parallel Distributed Processing: Exploring the Microstructure of Cognition.* Cambridge, MA: MIT Press, 1986.

[52] T. Samad. Towards connectionist rule-based systems. In *Proc. of the Int. Joint Conf. on Neural Networks, San Diego: SOS Printing*, pages II–525–532, 1988.

[53] S. Schocken and P. R. Kleindorfer. Artificial intelligence dialects of the bayesian belief revision language. *IEEE Transactions on Systems, Man, and Cybernetics*, 19:1106–1121, 1989.

[54] S. Schocken and R. Weitz. *A Survey of Classification Models in Management Science.* Technical Report, Information Systems Dept., NYU's Stern School of Business, 1991.

[55] T.J. Sejnowski and C.R. Rosenberg. Parallel networks that learn to pronounce english text. *Complex Systems*, 1:145–168, 1987.

[56] H.A. Simon. *The New Science of Management Decision.* New York: Harper and Row, 1960.

[57] R.H. Sprague and E.D. Carlson. *Building Effective Decision Support Systems.* Englewood Cliffs, N.J.: Prentice-Hall, 1982.

[58] C. Stanfill and D. Waltz. Toward memory-based reasoning. *Communications of the ACM*, 29(12):1213–1228, 1986.

[59] J. Utans and J. Moody. Selecting neural network architectures via the prediction risk: application to corporate bond rating prediction. In *Proceedings of the 1st Intl. Conf. on Artificial Intelligence Applications on Wall Street*, pages 35–41, 1991.

[60] W. van Melle, E.H. Shortliffe, and B.G. Buchanan. Emycin: a knowledge-engineer's tool for constructing rule-based expert systems. In E.H. Shortliffe and B.G. Buchanan, editors, *Rule-Based Expert Systems*, pages 302–313, Addison-Wesley, 1984.

[61] P.D. Wasserman. *Neural Computing: Theory and Practice.* New York: Van Nostrand Reinhold, 1989.

[62] H. White. Economic prediction using neural networks: the case of the ibm daily stock returns. In *Proc. of the Int. Joint Conf. on Neural Networks, San Diego: SOS Printing*, pages II:443–50, 1988.

[63] B. Widrow and Hoff M.E. Adaptive switching circuits. In *IRE WESCON Convention Record, Part 4*, pages 96–104, New York: Institute of Radio Engineers, 1960.