

DEPARTMENT OF ELECTRICAL AND
COMPUTER ENGINEERING
SCHOOL OF ENGINEERING
UNIVERSITY OF NEW MEXICO

AN OVERVIEW OF NEURAL NETWORKS
RESULTS FOR SYSTEMS AND CONTROL

C. Abdallah, G. L. Heileman and D. Hush
Department of Electrical and Computer Engineering
University of New Mexico
Albuquerque, NM 87131

M. Georgiopoulos
Department of Electrical and Computer Engineering
University of Central Florida
Orlando, FL 32816

UNM Technical Report: EECE95-005

Report Date: September 14, 1995

Contents

1	Introduction	1
1.1	Notation	2
2	Neural Net Architectures	3
3	Applications of Neural Networks in Systems and Control	6
3.1	Prediction	7
3.2	Identification	7
3.3	Discrete-Time Case	8
3.3.1	Continuous-Time Case	8
3.4	Control	9
3.4.1	Stability	10
3.5	Monitoring	11
4	Capabilities and Limitations	12
4.1	Approximation Capabilities	12
4.1.1	Approximation of Functions	12
4.1.2	Approximation of Functionals	13
4.1.3	Choosing the Network Size	13
4.2	Information versus Complexity	14
4.2.1	Information	15
4.2.2	Complexity	16
4.2.3	ART Complexity Results	18
5	Conclusions and Future Directions	20
A	Mathematical Background	27
A.1	Definitions	27
A.2	Functional Approximation	28
A.3	Embedding	29

A.4 Stability 30

List of Tables

4.1	Approximation Results	13
-----	---------------------------------	----

List of Figures

2.1	Typical Nonlinearities.	3
4.1	Error Surface Examples.	17
4.2	A block diagram of the ARTMAP architecture.	19
A.1	Block Diagram of General Feedback System.	31

Abstract

In this paper, we survey the applications of neural nets in the areas of systems and control. The paper provides a study of the limitations of neural nets, and stresses “hard” results as opposed to a listing of simulations and applications. Our general approach is to isolate the common properties used by neural networks in control, then to study the difficulties associated with achieving these properties. We avoid as much as possible a listing of neural networks simulations and try to stress their generic properties, independent of a particular architecture. Finally, we address the issues of capabilities versus actual performance and point out results from computational learning theory to show that neural networks can not be used as universal controllers. Although this is a paper on applications of neural networks to systems and control, we focus not on particular experiments but on generic capabilities.

Key Words

Neural networks, Identification, Control, Learning Theory, Complexity

Acknowledgements

The research of C. Abdallah and G. Heileman was supported by a Grant from Boeing Computer Services under Contract W-300445.

Chapter 1

Introduction

Neural networks represent an approach to artificial intelligence (AI) which begins by trying to model physical aspects of the mind. Researchers in conventional AI assume that consciousness is the primary cause of intelligent behavior, and as a result they attempt to model conscious thought processes such as problem solving or reasoning. In contrast, the conventional view among artificial neural networks (or simply neural nets) researchers is that consciousness is merely an emergent phenomenon of the complex interaction of the millions of neurons in our brains [31, 87]. The hope is that someday, neural nets which are sophisticated enough to exhibit the complex behavior known as intelligence, may be built. However, with the current state of knowledge neural nets are best at modeling pre-attentive processes such as low-level pattern recognition and motor control, which are assumed to occur in shallow stages of the nervous system. Unfortunately, it is this connection of neural nets to biology that has created much of the hype and excitement about their capabilities. In this paper our interest in neural nets stems not so much from their philosophical implications in AI, but from the intriguing possibility that models of neurology may also solve real-world problems. This paper discusses the potential, as well as the limitations, of neural nets in systems and control, stressing hard mathematical results. We refer the reader to the following sources for a historical account of the neural net field along with their applications in systems and control [48, 90].

The idea of neural control can be traced back to Norbert Wiener in his book *Cybernetics* [102]. Wiener realized that in order to build “smart” or “adaptive” controllers, some sort of learning mechanism is needed. In fact, he saw control, learning, and communications intermingled in the animal and machine. Control theory has since developed mathematically independent of the work on learning systems and AI. Successful applications of control abound in everyday life and in many technological areas. Why, then one might ask should control theorists (or practitioners) be concerned with neural networks, and is there really a need for such techniques in the study and practice of control? We answer this and many other questions throughout this paper.

Since this paper is mainly concerned with the applications of neural nets to systems and control problems, very little discussion of general neural networks and training algorithms is included. On the other hand, many recent surveys of neural nets applications to control have appeared [48, 66, 90, 104]. Our survey has the following salient features (shared by [90]): We concentrate on theoretical guarantees for such properties as approximation potential, learning capabilities, and stability, while minimizing our discussion of results that have been shown valid on particular examples via simulations. It is in fact surprising that many researchers who would never consider an ad-hoc approach (gradient descent with no proof of stability, gain scheduling, etc.) to controlling or identifying *linear* systems, jump at the occasion of doing so for *nonlinear* systems using poorly understood nonlinear neural net structures. We shall allow for a comparison between the capabilities of neural net systems and more traditional control approaches in an attempt to answer the question posed by Sontag [90]: “What is special about neural

nets?”. In other words, could a different nonlinear structure be substituted for a neural net and achieve similar results?

The test for the usage of neural networks (or any other approach for that matter) evolves around the following questions:

1. Can neural networks solve a previously unsolved problem?
2. Can neural networks better solve a previously solved problem? i.e. faster, more efficiently, etc.

Only an affirmative answer to one of the two questions above can justify the usage of neural networks in control. The biological motivation for using neural control is at best misleading and at worst dishonest. Moreover, simple illustrations via simulation on so-called “difficult” control problems may be misguided as shown in [37].

It turns out that in general, neural nets are capable of computing solutions to a wide variety of control problems by learning to emulate an expert or by learning to minimize a cost objective. They can, however, be very slow learners. This can be attributed to the nonlinear relationship between the network parameters (weights) and the output, and also to the relatively large number of parameters that must be learned. Neural networks typically contain more free parameters than alternative methods. This is in fact what gives them their robustness, namely that information can be distributed across a larger number of parameters. However, this robustness is achieved at the expense of additional storage and computational requirements. It also has an undesirable effect on the generalization and sample size requirements. These limitations will be further discussed later in this paper.

In order to address the issues discussed above, this paper is organized as follows. Section 2 provides a brief review of available neural networks architectures. Their applications in systems and control are overviewed in chapter 3, stressing some common properties and highlighting the need for a theoretical study of their capabilities and limitations. The capabilities and limitations of the different structures are then studied in chapter 4, and our conclusions are provided in chapter 5. The appendices contain a review of some mathematical results used in the paper. The style of the paper is informal and no proofs are included, but the reader is provided with an extensive list of references of all theorems and results.

1.1 Notation

In the following, \mathbb{R}^n equipped with the usual vector addition and (real) scalar multiplication denotes the n -dimensional vector space of real vectors, \mathbb{C}^n equipped with the usual vector addition and (complex) scalar multiplication denotes the complex vector space. The space of continuous functions from the interval $[a, b]$ to \mathbb{R} is denoted $C[a, b]$. The space of functions whose r th derivative is continuous are denoted by $C^r[a, b]$, and the space of smooth functions is $C^\infty[a, b]$. We will use norms of vectors and functions frequently in the paper. Specifically, we use the vector norm $\|x\|_2 = \sqrt{x^T x}$, and the p -function norms described in Appendix A.1. A definition of a generalized sigmoid maybe taken as any continuous function σ which is monotonically non-decreasing from 0 to 1 as x varies from $-\infty$ to ∞ . Unless otherwise stated we will assume this to be the case in this paper.

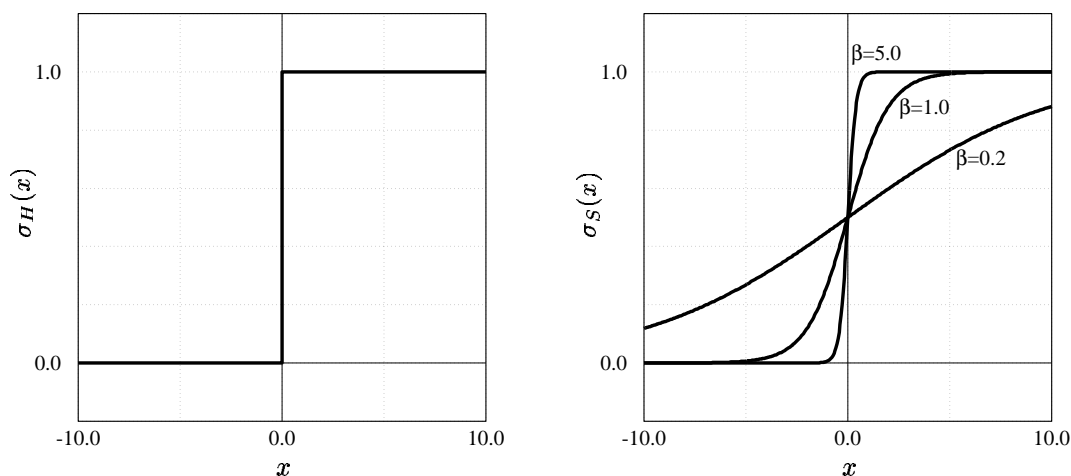
Chapter 2

Neural Net Architectures

When discussing neural nets, we are typically referring to a system built by linearly combining a large collection of simple computing devices (i.e., nodes), each of which performs a nonlinear transformation σ (in general a sigmoid function) on its inputs. These inputs are either external signals supplied to the system, or the outputs of other nodes. We will refer to σ as an *activation function*. Activation functions are in general, sigmoids, but radial basis functions (RBF's) or other nonlinearities may also be used [41]. One of the more popular neural net activation functions is the *hardlimiting threshold* or *Heaviside* function shown in Figure 2.1(a):

$$\sigma_H(x) = \begin{cases} 1 & x > 0 \\ 0 & x \leq 0 \end{cases} \quad (2.1)$$

In order to derive certain learning techniques, a continuous nonlinear activation function is often required. For example, all gradient descent techniques require that the sigmoid be differentiable [2]. Thus the threshold function is commonly approximated using the *sigmoid* function shown in Figure



(a) Hardlimiter Nonlinearity

(b) Sigmoid Nonlinearities

Figure 2.1: Typical Nonlinearities.

2.1(b):

$$\sigma_S(x) = \frac{1}{1 + e^{-\beta x}} \quad (2.2)$$

The gain of the sigmoid, β , determines the steepness of the transition region. Note that as the gain approaches ∞ , the sigmoid approaches a hardlimiting threshold. Often the gain is set equal to one, and β is omitted from the definition in equation (2.2).

Neural net models have two important characteristics. First, since they consist of many nodes, individual nodes carry out only a small amount of the overall computational task. Thus the computational load is *distributed* throughout the network. Second, the large number of *parallel* connections typically found in these systems provide many paths from input to output. These factors combine to make neural nets a very robust model of computing. In theory, damage to a few weights or nodes will not adversely affect the overall performance of the network. In fact, practical implementations indicate that the performance of neural nets tends to degrade gracefully as weights or nodes are destroyed [77, 78].

In order to discuss some results in this paper, we must define what we mean by a neural net more formally.

Definition 2.0.1 (Neural Net) *A k -node neural net \mathcal{G} is a weighted directed graph containing an ordered sequence of m input nodes, and $k - m$ computation nodes. An ordered sequence of p of the computation nodes are designated as output nodes. Associated with each computation node is an activation function $\sigma : \mathbb{R} \rightarrow \mathbb{R}$. The directed edge (v_i, v_j) appears in \mathcal{G} if node v_j receives input from node v_i .*

An important class of neural nets is the feedforward neural net described next.

Definition 2.0.2 (Feedforward Neural Net) *A feedforward neural net is a special case of a neural net in which \mathcal{G} is acyclic. Furthermore, each of the n input nodes has indegree 0, and each of the r output nodes has outdegree 0.*

The key point to note in this definition is that \mathcal{G} is *acyclic* in feedforward networks.

The simplest feedforward neural net is the *perceptron*, which consists of a single node with a threshold activation function. In any feedforward neural net, it is possible to partition the nodes into layers. In many cases, the input nodes are considered the first layer, the output nodes the last layer, and there are only connections from one layer to the next. Layers which are neither input nor output layers are called *hidden* layers. This type of architecture is often referred to as a *multilayer perceptron* (MLP).

We will use $w_{ji} \in \mathbb{R}$ to denote the weight associated with directed edge (v_i, v_j) in \mathcal{G} . Associated with every feedforward neural net \mathcal{G} is a function $f_{\mathcal{G}} : \mathbb{R}^m \rightarrow \mathbb{R}^p$ that is obtained by composing the node outputs of \mathcal{G} in the obvious way. A neural network with one hidden layer may be compactly described by

$$\begin{aligned} \alpha &= Bu; \text{ where } B \in \mathbb{R}^{L \times m} \\ \Sigma(\alpha) &= \sigma(\alpha^T) \in \mathbb{R}^{m \times L} \\ y &= C\Sigma^T \in \mathbb{R}^{p \times L} \end{aligned} \quad (2.3)$$

so that $y = C\sigma^T[(Bu)^T] = f_{\mathcal{G}}(u) = f_{\mathcal{G}}(B, C, u) = f_{\mathcal{G}}(W, u)$. Note that B and C are matrices of weights to be learned or programmed, and that the notation $\sigma(x)$ for $x \in \mathbb{R}^n$ denotes $\sigma(x) = [\sigma(x_1) \ \cdots \ \sigma(x_n)]^T$.

Definition 2.0.3 (Feedback or Recurrent Neural Net) *Recurrent networks are those neural nets which, even after learning is terminated, are described by dynamical (differential or difference) equations.*

$$\begin{aligned} \dot{x} &= G(x, W, u) \\ y &= h(x, W) \end{aligned} \quad (2.4)$$

where x is the internal variable or the state of the network. As an example, we will later consider the neural network described by

$$\begin{aligned} G(x, W, u) &= \sigma(Ax + Bu) \\ h(x, W) &= Cx \end{aligned} \tag{2.5}$$

where A , B and C are matrices of weights having the appropriate dimensions. Note that since G and h are functions of the internal variable x , the graph \mathcal{G} is cyclic.

Learning in a neural net involves the application of techniques that adjust the weights in \mathcal{G} . This process is referred to as *training* if desired input/output data is made available during learning. Thus by using learning, we have a way of changing $f_{\mathcal{G}}$. Note that learning is difficult even for systems without nonlinearities [3].

Some of the questions we consider in this paper include: What class of functions can \mathcal{G} represent? How much training data must \mathcal{G} be exposed to in order to learn a desired function, and how long will this training take? The training process used most often in neural nets is based on steepest descent minimization of a cost function, namely the backpropagation algorithm (or some variant) as developed in [101] and popularized in [78]. In fact, most supervised learning can be seen as a variant of gradient descent as described in [2].

Static networks are characterized by node equations that are memoryless as given in (2.3). That is, their output $y(t)$ is a function only of the current input $u(t)$, and not of past inputs or outputs. *Dynamic* networks, on the other hand, are systems with memory. Their node equations are typically described by differential or difference equations, see (2.4). The distinction between *programmable* and *learnable* weights is a little more subtle. Programmable weights are designed *a priori*, and placed into the network model so that it will perform the desired operation when the node equations are implemented. A sample of such networks is the basic Hopfield network [45]. In contrast, learnable weights are *sought* by presenting examples to the network, using the node equations to propagate them through the network, and then iteratively adjusting the weights so that the network responds in a desired manner. Another distinction should be made between *supervised* and *unsupervised* learning. When learning the weights of neural net, one could have a set of input/output pairs and supervise the learning procedure by penalizing deviations from the given outputs. Unsupervised learning on the other hand, does not have access to a desired output sequence as the weights are sought. Instead, the objective is usually to form clusters of data that share common characteristics or optimize a performance objective.

A representative static network with learnable weights is the previously mentioned MLP. Recurrent networks are representatives of dynamic networks with programmable and learnable weights. Another recurrent neural net with learnable weights is the adaptive resonance theory (ART) architecture developed by Grossberg [39] and others [16, 18, 36, 42, 43]. ART architectures have been developed using both *unsupervised* and *supervised* learning approaches.

Chapter 3

Applications of Neural Networks in Systems and Control

The idea of modeling a particular behavior using neural networks is attractive for several reasons. Since neural net architectures may be shown to be universal approximators of functions and functionals, the hope is that one can find a neural net learning (weight update) algorithm in order to approximate the data, predict its future, or use it in control. Unfortunately, approximation results are not in general, constructive, i.e. no proof of the approximating capabilities of neural nets can tell the designer the number and types of nodes needed. Moreover, and as stated in the introduction, neural networks will have to hold their own against other approximation schemes. It is however true that the modeling problem can be simplified with neural nets to a parametric identification problem, but much remains to be done.

Another neural network usage in systems relies on the clustering capabilities of such networks as ART, CMAC [65], and others. In such applications, the learning is unsupervised and different controllers may be used depending on the clustering results, as a generalization of a table-lookup control structure.

Because of their properties (clustering and approximation), neural nets have had successful applications in systems and control, and yet very few “hard” results have been derived or used. We follow in this section the general philosophy that neural nets should be motivated by a problem which can not be (efficiently) solved using more traditional approaches. In addition, we separate the issues of what neural nets “can” do and “how” they do it. The capabilities of neural nets may rely on results from approximation theory to show that a certain neural net structure is rich enough to model or control a particular system. The actual operation however has to do with the choice of a training or learning algorithm, and with the availability of a sufficiently rich training set, which may or may not achieve the theoretical performance of the structure. The capabilities of such algorithms has to be studied from a learning theory point of view. This dichotomy is reminiscent of the theory of linear adaptive control [68] where the design is separated into an algebraic step (finding a rich enough structure) and the analytical step (deriving a stable adaptive algorithm).

Independently of neural networks, there can be many reasons for attempting to model a particular phenomenon. The following are four reasons which will be discussed in this paper.

1. Prediction of future behavior.
2. Identification of a real system.
3. Designing a controller for a real system.

4. Monitoring and fault detection.

3.1 Prediction

Neural nets have proved noteworthy in time series prediction. In such a case, the input to an unknown system is also unknown and the neural net is trained on the output sequence, then used to predict the future of the output sequence. A recent collection of different approaches to this problem may be found in [100]. In the following, we discuss one promising approach taken from [84]. As described in Appendix A.3, the theory of Embedding allows the designer to obtain a diffeomorphic image of the attractor of a dynamical system. The theory is applicable to discrete-time systems but the more interesting case of continuous-time will be reviewed next. Consider an undriven dynamical system described by

$$\begin{aligned}\dot{x} &= F(x); \quad x(0) = x_0 \\ y &= h(x)\end{aligned}\tag{3.1}$$

and assume in the following that x is n dimensional, while y is a scalar. The designer has access to y but not to x . Consider the delay vector

$$z(t) = [y(t) \quad y(t - \tau) \quad \cdots \quad y(t - m\tau)]^T\tag{3.2}$$

According to Takens Theorem and its generalizations (see Appendix A.3) [71, 95], $z(t)$ represents a diffeomorphic image of the state $x(t)$ as it settles down to the attractor. In fact, having $z(t)$ available for $t \geq m\tau$, one can attempt to find the recurrent neural network

$$\begin{aligned}\dot{z}(t) &= \sigma(Az) \\ \bar{y}(t) &= Cz\end{aligned}\tag{3.3}$$

by minimizing the error between y and \bar{y} . The resulting system will serve as a predictor of the original time series as discussed in [84].

The key question here is what (if any) are the ingredients of a neural network used in the prediction application? In other words, is the success of neural networks in predicting a time series due to their parallelism, their nonlinear nature, the choice of learning algorithms, the choice of nonlinearity, etc.? In fact, only the approximating capabilities of neural networks are being used, and consequently, many other approximating structures (splines, polynomials, wavelets, etc.) may be used as efficiently in a time series prediction. Moreover, the “goodness” of prediction has more to do with the learning issues associated with training neural networks. These issues will be discussed in Section 4.

3.2 Identification

In many applications, one is interested in identifying or modeling a given system with no ulterior motive such as control. Usually, this makes sense when the open-loop system is stable in the BIBO sense since any bounded input can be used. Identification and modeling may also be seen as a data compression scheme, where a large amount of data is replaced by a structure (neural net) and its parameters (weights). For linear systems, ARMA models and their variants have been extremely adept at this problem [61]. Nonlinear systems however, are so much more general that no one structure has emerged (or will ever emerge) to model every type of nonlinear systems. It is however remarkable that the representation properties of neural nets has placed them at the center of the problem of identifying nonlinear systems. For a recent collection on the usage of neural nets in identification, see [86]. We again try to isolate the useful characteristics of neural networks as they apply to identification. It turns out that the approximating capabilities of neural net are again being used and that the learning issues associated with time series prediction and discussed in Section 4 are important.

3.3 Discrete-Time Case

The simplest conceptual framework for neural nets as identifiers is to consider a discrete-time formulation where the neural net is represented by the functional relationship,

$$\begin{aligned} y(t) &= g(u_{t-1}, y_{t-1}, W) + v(t) \\ u_t &= [u(1), u(2), \dots, u(t)]^T \\ y_t &= [y(1), y(2), \dots, y(t)]^T \end{aligned} \quad (3.4)$$

where $v(t)$ models disturbance or noise. The unknown dynamical system may be represented as

$$\begin{aligned} x_{t+1} &= F(x_t, u_t) \\ y_t &= h(x_t, u_t) \end{aligned} \quad (3.5)$$

The description in (3.4) may then be re-written as a composition of two mappings,

$$\begin{aligned} \phi(t) &= \phi(u_{t-1}, y_{t-1}, \eta) \\ g(u_{t-1}, y_{t-1}, \theta) &= g(\phi(t), \theta) \end{aligned} \quad (3.6)$$

In neural networks terminology, Sontag [89] has shown that

$$\begin{aligned} x_{t+1} &= \sigma(Ax_t + Bu_t) \\ y_t &= Cx_t \end{aligned}$$

can be used to approximate the input output behavior of a large class of discrete-time nonlinear systems described by (3.6). A more detailed discussion follows in the Theorem 3.3.1.

3.3.1 Continuous-Time Case

The continuous-time case is more involved since the availability of the output is no substitute for the state. It can however be shown that the class of recurrent neural networks used in [89] can be used to approximate continuous-time dynamical systems as described in the Theorem 3.3.1. Note however, that the approximation holds over a finite time T and over compact subsets of the state and input spaces.

Theorem 3.3.1 *Let*

$$\begin{aligned} \dot{x} &= f(x, u) \\ y &= h(x) \end{aligned}$$

with initial conditions $x(t_0) = x_0$. Assume that f and h are in $C^1[\mathbb{R}]$ such that for all inputs, the solution $x(t)$ exists and is unique for all $t \in [0, T]$ and some compact sets $K_1 \subset \mathbb{R}^n$ and $K_2 \subset \mathbb{R}^p$, while $x \in K_1$, $u \in K_2$. Then there exists a recurrent neural network of the form

$$\begin{aligned} \dot{\chi} &= \sigma(A\chi + Bu) \\ y &= C\chi \end{aligned}$$

where the dimension of χ is greater than or equal to that of x , and

$$\begin{aligned} \|x(t) - M(\chi(t))\| &< \epsilon \\ \|h(x(t)) - C\chi(t)\| &< \epsilon \end{aligned}$$

for all $\epsilon > 0$ and all $0 \leq t \leq T$, and some differentiable map M .

Note that the above theorem is not constructive, a problem that plagues other approximation results [35].

3.4 Control

Most applications of neural nets to control problems have been illustrated via simulations and not properly contrasted with more traditional approaches. Neural nets are nonlinear systems (either static or dynamic) and therefore their inclusion in a feedback control setting should be a last resort after linear controller techniques have been exhausted. In particular, neural nets should not be used to control a linear, time-invariant system before having exhausted the following structures: Fixed-structure classical controllers, robust controllers, and adaptive controllers. The reason for this hierarchy is simple: The stability of the feedback structure and its performance is much easier predicted and prescribed using any of the more standard techniques. It is in fact, the nonlinearity of the neural net, even when they are static, that causes many designers to give up any stability guarantees and to resort to “proofs by simulation”. A very common illustration of the validity or the success of a neural net controller is to simulate it in the feedback-loop of a linear system: This illustration proves more about the designer than about the neural net itself!

The following discussion concentrates on the limitations of using neural nets in control applications from two different angles: The fact that neural nets are theoretically incapable of meeting some control requirements, and the very real problem of not being able to achieve theoretically achievable levels of performance.

There are in effect a few cases where it makes sense to use neural nets in control: The controlled system must be uncertain and probably nonlinear. The reason for such a requirement is that for known (linear or nonlinear) systems, control theory is highly developed and successful [55, 98]. In fact, one could argue similarly for uncertain linear systems [4, 12, 28, 29]. Moreover, the neural net usually enters as a “learning” or “adaptive” part of the controller structure [91]. In the case where the usage of neural nets for control is justified, it has been generally been used as follows:

1. **Stabilizing State Feedback:** The starting point for most results on the control of nonlinear systems using neural nets is the assumption that the system is of the form (assumed single-input for simplicity)

$$\dot{x} = f(x) + g(x)u$$

that the state x is available for feedback and that the system is feedback-linearizable [52] but $f(x)$ and $g(x)$ are uncertain. This class of systems include many physical systems including the so-called Lagrange-Euler systems. In this setting, neural networks are basically implementing static state-feedback laws. Therefore, their functional approximating capabilities are called upon in order to implement $u = K(x)$. The main disadvantage of this approach is that the state x is usually unavailable and that noise is always present. Moreover, it is not always guaranteed that the neural net can actually converge to the correct $K(x)$. A typical situation is when a 1-hidden layer is used to implement a controller but a 2-hidden layer is called for because of discontinuities in the desired control action [90, 91]. Of course, feedback-linearizable systems do not suffer from this limitation and thus researchers have almost exclusively concentrated on those types of systems to illustrate the potential of neural nets [60, 96]. The learning issues associated with this control approach will be discussed in Section 4.

2. **Minimizing a Performance Objective:** Suppose that the control objective is to minimize a certain objective $J(x, u)$. The nonlinear controller $u = K(x)$ may again be implemented using a neural net and a gradient-descent learning algorithm [2]. This approach may fail because J may have many local (and global) minima, some of which may lead to unacceptable performance. In this case, the usage of a neural net as an optimizer is required.
3. **Model or Expert Following:** In such cases, the control objective is specified in terms of a model and the neural net attempts to make the plant follow the behavior of the model by minimizing the

error (in a suitable norm) between the plant and the model [8, 69]. Note that the model may be mathematical or an expert. The model following approach suffers from similar limitations to the two approaches above; namely, that the neural net may not be able to converge to an acceptable performance regime.

4. **Open Loop or Inverse Model:** Here, the neural net attempts to implement $u = K(r)$ such that $y = F(K(r)) = y_d$ where y_d is the desired output [8]. If F were invertible and $K = F^{-1}$, then by letting $r = y_d$, the desired output is obtained. The difficulties with this approach are that the inverse may not exist and that open-loop is almost always a bad idea because of instabilities and uncertainties. Moreover, and similarly to the state-feedback case, inverse model applications may require more than the usual 1-hidden layer network as discussed in [90, 91].
5. **Hierarchical Control:** This usage of neural nets is effectively a gain scheduling approach to controlling a plant using different pre-specified controllers [65]. The neural net acts as a classifier which recognizes different operating regions and switches between the different fixed controllers. This is a typical application of networks such as ART, and CMAC [65]. The ability of neural nets to classify patterns is therefore being used. A limitation of such a table-lookup approach, is the lack of stability guarantees as the controller is switched.

When the state is unavailable for feedback, the control problem is further complicated. The standard approach to such problems is to build an observer to reconstruct an estimate of the state which may then be fed back as if it were the true state. The system is then described by

$$\begin{aligned}\dot{x} &= f(x) + g(x)u \\ y &= h(x)\end{aligned}\tag{3.7}$$

where for simplicity, both u and y are scalars. As described in Theorem 3.3.1, one can use a general form of a neural network to model the system, and may also use the same form in order to control it based on input-output measurements. Unfortunately, such control approach requires some structural properties (observability, controllability) which have not always been checked by neural networks researchers.

3.4.1 Stability

Since the control problem arises mostly in the setting of dynamical systems, the question of stability of the feedback system is paramount in control. One can not avoid this question and maintain that he has controlled a given plant. In addition, one should distinguish between guaranteeing the stability of the neural network [23, 103] and guaranteeing the stability of the closed-loop [59]. The stability of the neural network is actually a property of the learning algorithm and the performance objective, while that of the feedback closed-loop structure is more related to the input/output mappings of the system and the neural network. In effect, even the stability of the neural network has not been carefully studied since most learning algorithms use approximations of gradient descent algorithms whose actual behavior is robust, but their stability guarantees non-existent. The stability of the neural network is The problem with the stability concept however is that it has many meanings: It could relate to the stability (or convergence of the weights) of the neural network, and may for example denote Bounded-Input-Bounded-Output (BIBO) stability. That however is not of much use in neural net applications because the output nonlinearity can easily guarantee the boundedness of the output for a particular neural net, while still causing unbounded signals throughout the feedback structure. A more useful stability setting is that of internal stability of the closed-loop systems due to Lyapunov [98], or in the case of driven systems that of Bounded-input-Bounded-State (BIBS) stability described in [92].

As it stands, few researchers have provided stability guarantees for neural nets when used in control applications [60, 73, 82, 83, 96]. Most however have not. This again is reminiscent of the early years of adaptive control of linear system where the MIT rule was blindly applied until an aircraft crash forced

researchers to study the instability mechanisms inherent in an adjustable gradient descent approach [68].

There are two promising approaches to the study the stability of neural nets (or any nonlinear controller) and they rely on either the small-gain theorem or the passivity theorem (see Appendix A.4). In fact, consider Figure A.1 in Appendix A.4, and let the neural network be block H_2 in the figure. Then, the designer may be able to guarantee that at all times, and whether learning is on or off, either the small-gain theorem or the passivity theorem apply. That will require for example that the neural network has a gain $\gamma_2 < 1/\gamma_1$ or that it dissipates enough energy to satisfy the passivity theorem. These theorems provide *sufficient* conditions for stability in a suitable sense. Successful applications of neural networks as feedback controllers have also exploited the particular structure of the controlled plant (block H_1 in Figure A.1), as described for example in [83, 96]. The fact that the weights of the neural networks have converged is of little consequence in guaranteeing closed-loop stability since even a linear, time-invariant, and stable H_2 can lead to closed-loop instabilities.

3.5 Monitoring

It is also conceivable that a neural net can be used as a monitoring device, in order to detect major changes in the operation of the system. Specifically, the neural net is trained on a well-behaving system, and then operated with no more training in parallel to the actual system. The neural net output will then be compared to that of the physical system, and any anomalies in the output of our system will be detected. In some cases, the neural net may also be used to accommodate the change of behavior in a manner similar to the hierarchical control approach [74]. In others, it is used simply as a fault detection device where the clustering capabilities of networks such as ART or CMAC are called upon [42].

Chapter 4

Capabilities and Limitations

As described in the previous chapter, some generic capabilities of neural nets are usually used in applications to systems and control. These capabilities are broadly divided into approximation capabilities and clustering capabilities. The discussion in this chapter deals with these two capabilities along with a more practical aspect: How close to the theoretical capabilities can we get when learning the weights of a neural net? This will give rise to the issues of information and complexity [1]. Information deals with the idea that in order to learn, a sufficient amount of information (in a suitable form) must be available to the learner. On the other hand, the learner is always limited in his time, space and other resources which requires him to keep a low complexity in his learning approach. Another related issue which is consistently overlooked by users of neural nets in control is the issue of memorizing versus learning. Memorizing refers to the act of recalling as closely as possible, the data used in teaching the network a particular behavior. A neural network can memorize given long enough time and enough samples. Learning, on the other hand, relates to the idea of generalization, or to the extrapolating the knowledge encoded from seeing the data used in memorization, to explain data that was not seen before. Therefore, learning requires that the network is provided with the right type of training (large information content) and enough of them, along with a sufficient amount of resources. This issue is related to the concepts of “sufficiently rich” or “persistently exciting inputs” studied in the context of identification and adaptive control of linear systems [68].

4.1 Approximation Capabilities

One of the common uses of neural nets in control is in approximating a functional relation $y = F(u)$ given a sample of input/output pairs $(u_i, y_i); i = 1, \dots, N$. These results usually rely on the theorems reviewed in Appendix A.2. First, let us note that there is a difference between *what a neural net can approximate*, and *how it does it*. In other words, if a neural net structure is rich enough to approximate a particular behavior, there is no guarantee that the particular learning algorithm used will actually converge to the correct approximating weights. Second, *there is a difference between approximating functions and approximating functionals*. In other words, a neural net which does well on a finite set of input/output data pairs, may not do the same if the input and output are given as functions.

4.1.1 Approximation of Functions

It is now known that a 1-hidden layer network static network is capable of approximating an arbitrary (continuous) function. Many proofs of this result have appeared of which we recall the ones in [24, 46].

Reference	Activation Function	Approximation In	Proof
[24]	Continuous Sigmoid	$C[K]$	Existential
[24]	Bounded Sigmoid	$L_p[K]$	Existential
[46]	Monotone Sigmoid	$C[K]$	Constructive
[22]	Bounded Sigmoid	$C[\mathbb{R}^n]$	Constructive

Table 4.1: Approximation Results

Until recently, these proofs have used either Kolmogorov’s results or the Stone-Weierstrass theorem (See Appendix A.2) and required the continuity or even differentiability of the sigmoid (or nonlinearities) in the neural net. Chen et al. [22], building on the research of Sandberg [79, 80, 81] have recently shown however that all is needed is the boundedness of the sigmoid building block. Table 4.1.1 is taken from [22] and summarizes some available results for the approximation of functions. The set K denotes a compact subset of \mathbb{R}^n . Note that even those results labeled “constructive” still ignore the facts associated with the training algorithm and the available data.

4.1.2 Approximation of Functionals

The approximation of functional by a neural net structure is more involved, since the neural net is now approximating a mapping from a space of functions $u(t)$ to another space of functions $y(t)$. Here again, the work of Sandberg [79, 80, 81] on approximately-finite memory systems was instrumental both in discrete and continuous time. Recently, Newcomb and de Figueiredo [70] have used ideas from Volterra functionals to design a network which can identify dynamical systems. Similarly, the results of Sontag and Albertini previously discussed in Theorem 3.3.1 concentrate on the problem of approximating functionals. Finally, Chen and Chen [21] have recently discussed the problem of approximating functionals and how to then use it in identifying nonlinear dynamical systems. Note however that in all of these results, The approximation is over a finite time and when used in identification, over compact sets of inputs and states. More importantly, nowhere is a learning algorithm provided to achieve the capabilities of the proposed networks. In other words, none of the proposed structures is guaranteed to provide a valid model by training on a given data set.

We have discussed the fact that neural nets are capable of approximating arbitrary nonlinear (static or dynamic) mappings. For example, given a set of input/output data, the Backpropagation algorithm can be called upon to learn the mapping described by an MLP at the example points. However, there are a number of practical concerns which should be addressed. The first is the matter of choosing the network size. The second is the time complexity of learning. That is, we may ask if it is possible to learn the desired mapping in a reasonable amount of time. Finally, we are concerned with the ability of our network to generalize, that is its ability to produce accurate results on new samples outside the training set. These concerns are addressed in the sections that follow.

4.1.3 Choosing the Network Size

First, theoretical results indicate that the MLP is capable of forming arbitrarily close approximations to any continuous nonlinear mapping; but this is true only as the size of the network grows arbitrarily large [24, 34]. In general it is not known what (finite) size network works best for a given problem, although the results of Barron [5] may be called upon in a particular problem. Further, it is not likely that this issue will be resolved in the general case since each problem will demand different capabilities from the network. Choosing the proper network size is important for the following reasons. If the network is too small it will not be capable of forming a good model of the problem. On the other hand, if the network

is too big then it may be *too capable*. That is it may be able to implement *numerous solutions* that are consistent with the training data, but most of these are likely to be poor approximations to the actual problem.

Ultimately we would like to find a network whose size best matches its capability to the structure of underlying problem, or since the data is sometimes not sufficient to describe all of the intricacies of the underlying problem, we would like a network whose size best captures the structure of the data. With some specific knowledge about the structure of the problem (or data), and a fundamental understanding of how the MLP might go about implementing this structure, one can sometimes form a good estimate of the proper network size.

With little or no prior knowledge of the problem however, one must determine the network size by trial and error. A methodical procedure is recommended. One approach is to start with the smallest possible network and gradually increase the size until the performance begins to level off (or worsen). In this approach each size network is trained independently. A closely related approach is called “growing a network”. The idea here is to start with one node and create additional nodes as they are needed. Approaches that use such a technique include Cascade Correlation [30], the Group Method of Data Handling (GMDH) [6, 53], Projection Pursuit [32, 33], the Algorithm for Synthesis of Polynomial Networks (ASPEN) [6], and others [7]. These approaches differ from the previous approach in that additional nodes are created *during the training process*.

Another possibility is to start with a large network and then apply a pruning technique that destroys weights and/or nodes which end up contributing little or nothing to the solution [58]. With this approach one must have some idea of what size network constitutes a “large” network, that is “what size network is probably too big for the problem?”. The following guidelines are useful in placing an upper bound on the network size. For a fully connected MLP network, no more than two hidden layers are typically used, and in most cases only one. Numerous bounds exist on the number of hidden layer nodes needed in one-hidden layer networks. For example it has been shown that an upper bound on the number of hidden layer nodes needed for the MLP to implement the training data exactly is on the order of M , the number of training samples [47]. This suggests that one should never use more hidden layer nodes than training samples. Actually, the number of hidden layer nodes should almost always be much less than the number of training samples, otherwise the network simply “memorizes” the training samples, and as a result yields poor generalization. It is more useful to consider bounds on the number of hidden layer nodes that are expressed as a function of n , the dimension of the input pattern. It is possible to find problems that require the number of hidden layers nodes to be exponential in m , but it is generally recommended that MLPs be used only for problems that require a polynomial number of hidden layer nodes [26, 40, 85]. For example, if the problem is one that requires a pattern class to be completely enclosed by a spherical decision boundary then the number of hidden layer nodes should be approximately $3m$ [49]. These issues are considered in more detail in the following section.

4.2 Information versus Complexity

In Section 2 we posed the question: How much training data must \mathcal{G} be exposed to in order to learn a desired function? In order to answer this question, we must first define what it means for a neural net to *learn* a function. The most widely studied model of learning is the distribution-free probably approximately correct (PAC) model, described by Valiant [97]. A discussion of the PAC learning model is presented in this section. Using this model, we will be able to rigorously define *generalization*, and therefore classify the *learnability* of various problems using neural nets. Generalization is concerned with the ability of a neural net to correctly classify data it has *not* been exposed to previously. The generalization issue becomes critical when using neural nets in modeling and controlling nonlinear systems.

An interesting trade-off between *information* and *complexity* arises when considering generalization.

The amount of information contained in the training data sets an upper bound on the number of examples that a neural net must be exposed to in order to achieve some level of generalization. While the issue of complexity considers how long it takes to perform the learning. Larger neural nets tend to have better generalization capabilities than smaller ones; however, the training time tends to increase rapidly with the network size, as does the number of training examples needed for achieving generalization.

4.2.1 Information

The PAC Learning Model. Within the PAC framework, a learning function is supplied with examples (classified objects) from the *instance domain* X . Members of X (i.e., instances) are classified according to their membership or non-membership in the unknown *target concept* $c \subseteq X$. Thus we may represent the target concept as the function $c : X \rightarrow \{0, 1\}$ which partitions the instance domain according to membership (1), or non-membership (0) in the target concept. Furthermore, the target concept is assumed to be a member of a known collection \mathcal{C} of subsets of X referred to as the *concept class*.

Examples are assumed to be generated from a fixed but unknown probability distribution P on X , and a *training sample* s consists of m such examples drawn from c :

$$s = \{x_1, c(x_1), (x_2, c(x_2)), \dots, (x_m, c(x_m))\}$$

where m is the *sample size*, and $x_i \in X$ for $i = 1, 2, \dots, m$.

The goal of the learning function is to construct a hypothesis $h : X \rightarrow \{0, 1\}$ that is a good approximation of c . Specifically, h is said to ϵ -approximate c if the probability that h and c disagree on a random instance drawn from X according to distribution \mathcal{D} is at most ϵ . A hypothesis h is said to be *consistent* with a training sample s if $h(x_i) = c(x_i)$ for $i = 1, 2, \dots, m$ (i.e., h correctly classifies all examples in the training sample).

Once again, we assume that any hypothesis is a member of a known collection \mathcal{H} of subsets of X . We may also define a *graded hypothesis space* $\mathcal{H} = \bigcup \mathcal{H}_n$, whenever \mathcal{H} is made up of a disjoint union of hypotheses parameterized by n . In the case of a graded real hypothesis space, each \mathcal{H}_n is a real hypothesis space defined on (possibly some subset of) \mathbb{R}^n .

Given the previous definitions, we may now define an important learning model in which our problem will be framed.

Definition 4.2.1 (PAC Model) Let $\mathcal{C} = \bigcup \mathcal{C}_n$ be a concept class over $X = \bigcup X_n$. A learning algorithm L is said to be a *probably approximately correct (PAC)* for \mathcal{C} if it has the following property: for every $c \in \mathcal{C}$, for every distribution \mathcal{D} on X , and for all $0 < \epsilon < 1/2$ and $0 < \delta < 1/2$, if L is given access to labeled training examples of some c drawn according to \mathcal{D} , and inputs ϵ and δ , then with probability $1 - \delta$, L outputs a hypothesis $h \in \mathcal{H} = \bigcup \mathcal{H}_n$ that ϵ -approximates c .

If a learning algorithm L with the property described in this definition can be shown to exist, we say that \mathcal{C} is PAC-learnable. If \mathcal{C} is PAC-learnable, then the sample size m_0 for which L can PAC learn any $c \in \mathcal{C}$ is referred to as the *sample complexity* of L .

VC Dimension and Learnability. The Vapnik-Chervonenkis dimension (VC dimension) is an important combinatorial tool introduced into learnability theory by Blumer et al. [14] and has been extensively used in the analysis of learning problems specified within the PAC framework. Below we formally define the VC dimension of a concept class, and then state some important results related to this measure. After which, we will apply these results to neural nets.

Definition 4.2.2 (VC Dimension) Let X be a set and \mathcal{C} a collection of subsets of X (i.e., $C \subseteq 2^X$). The subset $S \subseteq X$ is said to be shattered by \mathcal{C} if for every partition of S into disjoint subsets S_1 and S_2 , there exists a $C \in \mathcal{C}$ such that $S_1 \subseteq C$ and $S_2 \cap C = \emptyset$. The VC dimension of \mathcal{C} is the maximum cardinality of any $S \subseteq X$ shattered by \mathcal{C} , or ∞ if arbitrarily large subsets may be shattered.

We will use $\text{VCdim}(\mathcal{C})$ to denote the VC dimension of the concept class \mathcal{C} .

The importance of the VC dimension in learning is considered next. The following theorem demonstrates that if $\text{VCdim}(\mathcal{C}) = \infty$, then the concept class \mathcal{C} is not PAC learnable; conversely, if $\text{VCdim}(\mathcal{C})$ is finite, then \mathcal{C} is PAC learnable (but possibly computationally unbounded). Here and throughout the paper, \log denotes the logarithm base 2, and \ln denotes the natural logarithm.

Theorem 4.2.1 (Blumer et al. [15]) Let \mathcal{C} be any concept class, \mathcal{H} be any hypothesis class of VC dimension d , and L be any algorithm that takes as input a random set S of labeled examples of a concept in \mathcal{C} , and produces a hypothesis $h \in \mathcal{H}$ that is consistent with S . Then L is a PAC learning algorithm for \mathcal{C} using \mathcal{H} , if

$$|S| \geq \max \left(\frac{4}{\epsilon} \log \frac{2}{\delta}, \frac{8d}{\epsilon} \log \frac{13}{\epsilon} \right)$$

Hence, if the VC dimension of a neural net can be determined, then Theorem 4.2.1 gives us one way of determining the *information complexity* of a learning problem—i.e., the minimal amount of training data that must be collected.

It is not difficult to show that the VC dimension of an n -input perceptron net is $n + 1$. However, Minsky and Papert [67] showed these networks to be fairly limited in terms of the functions they are able to represent. Let us consider more general feedforward neural nets which, as described in Section 2, are able to approximate arbitrary functions. Baum and Haussler [10] showed the VC dimension of a feedforward neural net with threshold activation functions is $O(W \log(e \cdot r))$, where e is the base of the natural logarithm, W is the number of adjustable weights, and r is the number of computation nodes in the network. Determining the VC dimension of feedforward neural nets with sigmoid activation functions was a much more difficult problem. Macintyre and Sontag [64] used deep results in logic to show that the VC dimension of these networks is finite, but no explicit bounds were derived. Goldberg and Jerrum [38] were, however, able to derive a polynomial bound using a continuous piecewise polynomial approximation for the sigmoid activation function.

Let us now apply the VC dimension results to the learning problem. Baum and Haussler [10] used Blumer et al.'s results to show that feedforward neural net with threshold activation functions will have good generalization if they are consistent with a sufficiently large training set. Specifically,

Theorem 4.2.2 (Baum and Haussler [10]) A feedforward neural net \mathcal{G} of threshold activation nodes having d input nodes, 1 output node, and W adjustable weights has $\text{VCdim}(\mathcal{G}) = O(W \log W)$ for any weight-space $W \subseteq \mathbb{R}$ and any concept space $X \subseteq \mathbb{R}$.

Thus the VC dimension of a single-output feedforward network grows polynomially with the size of the network. Furthermore, if we had access to a consistent algorithm for training such a network, then using Theorems 4.2.1 and 4.2.2 it easy to see that the sample complexity of the learning algorithm would also be polynomial in the size of the network. This then suggests that neural networks can be efficiently used in control applications where one has enough training samples.

4.2.2 Complexity

Even if one is able to determine the optimal network size, and if one has enough samples, it turns out that finding the correct weights for a network is an inherently difficult problem. The problem of finding

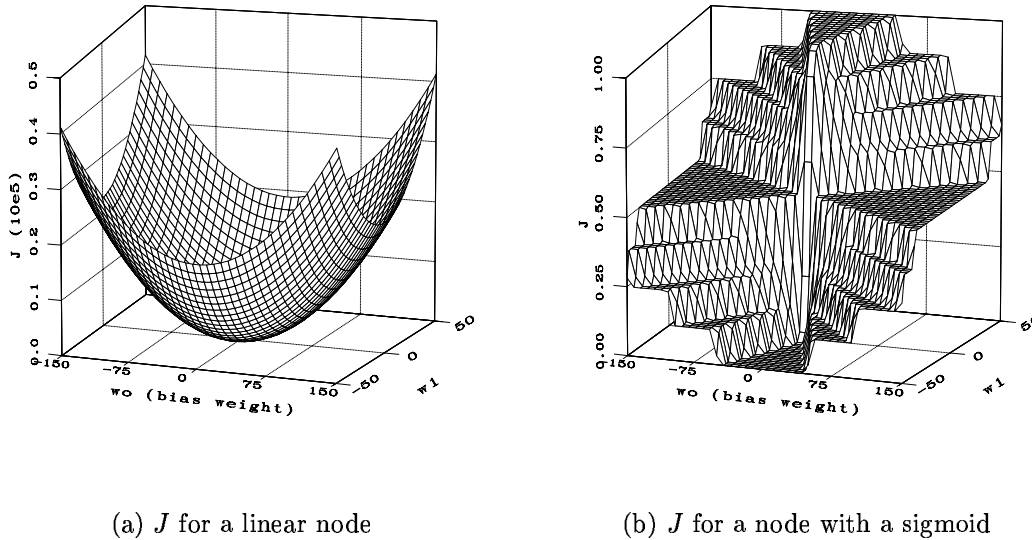


Figure 4.1: Error Surface Examples.

a set of weights for a fixed size network which performs the desired mapping exactly for some training set is known as the *loading problem*. Recently it has been shown that the loading problem is NP-complete [13, 54]. This suggests that if we have a very large problem (e.g. if the dimension of the input space is very large), then it is unlikely that we will be able to determine if a weight solution for the exact mapping exists in a reasonable amount of time. Therefore, control applications where a quick response is critical are definitely out, and the claim that neural networks are “fast” misses the point that their training time is usually too long for real-time applications [83].

On the other hand, learning algorithms like Backpropagation are based on a gradient search [2], which is a greedy algorithm that seeks out a *local minimum* and thus may not yield the exact mapping. Gradient search algorithms usually do not take exponential time to run (if a suitable stopping criterion is employed). However, it is well known that finding weight solutions using Backpropagation is extraordinarily slow. One way to explain this sluggishness is to characterize the error surface which is being searched.

In the case of a single perceptron with linear activation function the error surface is a quadratic bowl (with a single (global) minimum), and thus it is a relatively agreeable surface to search. For MLPs however, it turns out that the surface is quite harsh [50]. This is illustrated in Figure 4.1 which shows the performance index to be minimized J for two different cases. In both cases there is a single node with two weights and the training samples are $\{-4, -3, -2, -1\}$ for class 1 and $\{1, 2, 3, 4\}$ for class 2. The only difference between the two cases is that in Figure 4.1(a) a linear activation function is used, and in Figure 4.1(b) a sigmoid is used. This surface tends to have a large amount of flatness as well as extreme steepness, but not much in between. It is difficult to determine if the search has even terminated with this type of surface since the transient flat spots “look” much the same as minima, i.e. the gradient is very small. Furthermore, with this type of surface a gradient search moves very slowly along these flat parts. It is dangerous to increase the learning rate to compensate for the sluggishness in these areas because the algorithm may then exhibit instabilities when it reaches the steep parts of the surface. The surface in Figure 4.1(b) is for a single node only, but it has been shown that many of the characteristics of this surface are typical of surfaces for multilayer perceptrons as well [50]. This then suggests that nonlinear neural networks should not be used to control known, linear time-invariant systems.

Attempts to speed learning include variations on simple gradient search [72, 76, 94], line search methods [51], and second order methods [57, 99]. Although most of these have been somewhat successful, they usually introduce additional parameters which are difficult to determine, must be varied from one problem to the next, and if not chosen properly can actually slow the rate of convergence.

The choice of initial weight values is also important since the criterion function for the MLP may have many minima (which correspond to different solutions) and the final convergence point depends on the initial conditions. Typically, a problem is solved by running Backpropagation several times with different initial weight vectors until an adequate solution is found. The problem of finding a consistent neural net (i.e., finding network weights that correctly classify the training data) is often referred to as the *credit-assignment problem* for the weights of a network. For perceptron networks with real-valued weights, the credit-assignment problem is exactly the linear programming problem, which can be solved in polynomial time using Karmarker's algorithm.

Unfortunately, for the more general class of feedforward neural nets with threshold activation functions, the credit-assignment problem is believed to be intractable. Blum and Rivest [13] demonstrated this by showing that finding network weights for a simple 3-node feedforward neural net that correctly classify the training data is NP-hard.

Baum [9] subsequently came up with a positive result which showed that it is possible to efficiently (i.e., polynomial time) PAC learn the simple networks considered by Blum and Rivest if queries are allowed. Baum and Lang [11] empirically showed that this approach can also be used to learn far more complicated functions. E.g., using queries to construct hidden units, they were able to provide a very good approximation to Wieland's two spirals problems using only 30 minutes of CPU time.

The previous results have dealt only with feedforward with threshold activation functions. An open question posed by Blum and Rivest [13] is whether or not the credit-assignment problem remains NP-hard if sigmoid activation are used. Interestingly, Sontag in [88] provides a mathematical construction which shows that there do exist continuous activation functions for which the credit-assignment problem is not NP-hard; however, the ones he found were not "reasonable". He poses the following open question: "Do reasonable continuous activation functions lead to NP-completeness?" and conjectures "yes." DasGupta, Siegelmann, and Sontag [25] do have a result which demonstrates NP-completeness for one class of continuous activation function, but they have not been able to prove NP-completeness for the most important case, the sigmoid function of equation (2.2).

Finally, Wolfgang Maass [63] has some results extending the PAC model to the case of continuous inputs and outputs, and therefore allowing the analysis of MLP networks with continuous activations.

These complexity results suggest that for control systems applications, a neural network can never be used as a universal controller. The complexity of the control task along with the training data are actually more important than the fact that a so-called neural network is being used.

4.2.3 ART Complexity Results

In chapter 3 we mentioned the use of ART networks in applications of monitoring, fault detection, gain scheduling and clustering. Because such applications are different from those usually associated with neural controllers, we now present some results that should be used as guidelines when considering the use of ART networks in these applications. The proofs of the results presented here can be found in [36]. We consider two unsupervised varieties of ART networks: ART1 [17] and Fuzzy ART [20]. ART1 requires binary valued input patterns, while Fuzzy ART can process real-valued inputs. We also discuss two supervised varieties that make use of ART1 and Fuzzy ART, these are ARTMAP [19] and Fuzzy ARTMAP [18], respectively. A block diagram of the ARTMAP network is shown in Figure 4.2. This network consists of two unsupervised ART1 modules, denoted ART_a and ART_b in the figure. The Fuzzy ARTMAP architecture is constructed in essentially the same fashion, using Fuzzy ART modules

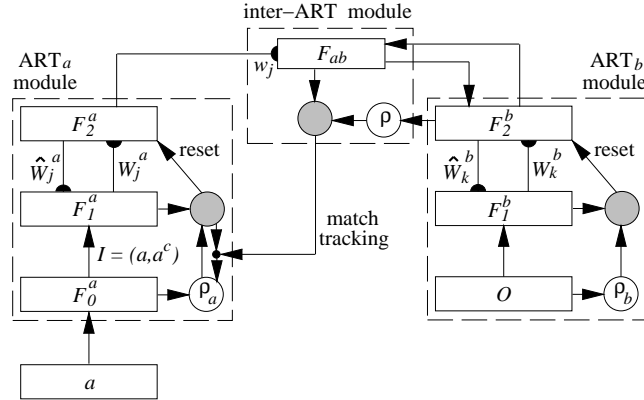


Figure 4.2: A block diagram of the ARTMAP architecture.

instead of ART1 modules.

All of the results presented here assume training in the “fast learning” mode of operation. This implies that all inputs to ART1 or Fuzzy ART modules are held at the network inputs long enough for the network weights to converge to their steady-state values.

One of the principle advantages of using ART-based networks in applications is that they can be trained very quickly. For example, it can be shown that if an ART1, Fuzzy ART, or ARTMAP network is repeatedly presented a list of input patterns, and the network parameter β_a is chosen small, then learning will be complete in at most m list presentations, where m is the size of the input patterns. In the case of unsupervised ART networks, completion of learning essentially means that a stable pattern cluttering has occurred. With supervised ART networks, learning is complete when the network is able to produce a consistent hypothesis.

For example, consider an ARTMAP network with $m = 20$ input nodes. Even though there could be as many as $2^{20} \approx 1,000,000$ input patterns, the previous result guarantees that the hypothesis produced by the network will be consistent after at most 20 list presentations. In order to produce this consistent hypothesis, however, as many as 2^{20} nodes may be required in both the F_2^a and F_2^b layers. An important open question is: how small can the F_2^a and F_2^b layers be made before the network is no longer able to produce a consistent hypothesis? The answer to this question depends on the function we are trying to learn. Answering this question is also important for another reason, the VC dimension of the ARTMAP network is given by $|F_2^a|$. Thus, in order to make use of Theorem 4.2.1, we also want $|F_2^a|$ to be as small as possible.

A related results states that if an ART1 or Fuzzy ART network is repeatedly presented a list of input patterns, and the network parameter β_a is chosen small, then learning will be complete in p list presentations, where p represents the number of distinct size patterns in the input list. Because the preprocessing in Fuzzy ART create input patters that are all the same size, this result actually shows that Fuzzy ART needs only one list presentation to learn the list of input patterns presented to it. In short, ART structures may then be used as fast classifiers so that different pre-designed controllers may be switched on and off.

Chapter 5

Conclusions and Future Directions

The current status of neural nets in control and systems applications is reminiscent of the early days of adaptive control. Prior to the stability proofs of adaptive control, many results promised the control of uncertain systems using the simple MIT gradient descent rule [68]. In controlling a nonlinear system, neural networks are but one tool to be used cautiously. In fact, no special properties of neural networks have been used in the control of uncertain nonlinear systems. The results reviewed in this paper suggest that what neural networks are good at (approximating nonlinear mappings) is but a small part of their usage in control. We still have to worry about having a rich enough training set, an efficient learning algorithm, and most of all, a stable operation in a feedback loop containing the neural network. As we learn of the limitations of neural networks, we also gain a greater appreciation of their capabilities. The future of neural networks in control will be written by researchers who not only use them to control a real system, but by those who can prove their need and guarantee a certain level of performance in neural controllers.

Bibliography

- [1] Y. Abu-Mostafa. The Vapnik–Chervonenkis dimension: Information versus complexity in learning. *Neural Computation*, 1(3):312–317, 1989.
- [2] P. Baldi. Gradient descent learning algorithm overview: A general dynamical systems perspective. *IEEE Trans. Neural Nets*, 6(1):182–195, 1995.
- [3] P. Baldi and K. Hornik. Learning in linear neural networks: A survey. *IEEE Trans. Neural Nets*, 6(4):837–858, 1995.
- [4] B. Barmish. *New Tools for Robustness of Linear Systems*. MacMillan, New York, NY, 1st edition, 1993.
- [5] A. Barron. Universal approximation of bounds for superpositions of a sigmoidal function. *IEEE Trans. Info. Th*, 39(3):930–945, 1993.
- [6] A. Barron and R. Barron. Statistical learning networks: A unifying view. In E. Wegman, D. Gantz, and J. Miller, editors, *Computing Science and Statistics: Proc. of the 20th Symposium on the Interface*, pages 192–202, 1989.
- [7] A. Barron, F. van Straten, and R. Barron. Adaptive learning network approach to weather forecasting: a summary. In *Proceedings of the IEEE Int. Conf. on Cybernetics and Society*, pages 724–727, 1977.
- [8] A. Barto. Connectionist learning for control: An overview. In W. Miller, R. Sutton, and P. Werbos, editors, *Neural Networks For Control*, pages 5–58. MIT Press, Cambridge, MA, 1990.
- [9] E. B. Baum. Neural net algorithms that learn in polynomial time from examples and queries. *IEEE Trans. Neural Networks*, 2(1), 1991.
- [10] E. B. Baum and D. Haussler. What size net gives valid generalization? *Neural Computation*, 1(1):151–160, 1989.
- [11] E. B. Baum and K. J. Lang. Constructing hidden units using examples and queries. In R. Lippmann, J. Moody, and D. Touretzky, editors, *Advances in Neural Information Processing Systems 3*. Morgan Kaufmann, San Mateo, CA, 1991.
- [12] S. Bhattacharyya, H. Chapellat, and L. Keel. *Robust Control: The Parametric Approach*. Prentice-Hall PTR, Upper Saddle River, NJ, 1st edition, 1995.
- [13] A. Blum and R. L. Rivest. Training a 3-node neural net is NP-complete. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 1*. Morgan Kaufmann, San Mateo, CA, 1989.

- [14] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. Warmuth. Learnability and the Vapnik–Chervonenkis dimension. *Journal of the Association for Computing Machinery*, 36(4):929–965, 1989.
- [15] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth. Learnability and the Vapnik–Chervonenkis dimension. *Journal of the ACM*, 36(4):929–965, 1989.
- [16] G. A. Carpenter. Neural Networks: From Foundations to Applications Short Course, May 1991. Wang Institute of Boston University, Tyngsboro, MA.
- [17] G. A. 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.
- [18] G. A. Carpenter, S. Grossberg, N. Markuzon, J. H. Reynolds, and D. B. Rosen. Fuzzy ARTMAP: A neural network architecture for incremental supervised learning of analog multidimensional maps. *IEEE Transactions on Neural Networks*, 3(5):698–713, 1992.
- [19] G. A. Carpenter, S. Grossberg, and J. H. Reynolds. ARTMAP: Supervised real-time learning and classification of nonstationary data by a self-organizing neural network. *Neural Networks*, 4(5):565–588, 1991.
- [20] G. A. Carpenter, S. Grossberg, and D. B. Rosen. Fuzzy ART: Fast stable learning and categorization of analog patterns by an adaptive resonance system. *Neural Networks*, 4(6):759–771, 1991.
- [21] T. Chen and H. Chen. Universal approximation of nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems. *IEEE Trans. Neural Nets*, 6(4):911–917, 1995.
- [22] T. Chen, H. Chen, and R.-W. Liu. Approximation capability in $C(\bar{R}^n)$ by multilayer feedforward networks and related problems. *IEEE Trans. Neural Nets*, 6(1):25–30, 1995.
- [23] M. Cohen and S. Grossberg. Absolute stability of global pattern formation and parallel memory storage by competitive neural networks. *IEEE Transactions on Systems, Man, and Cybernetics*, 13:815–825, September/October 1983.
- [24] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, 2(4):303–314, 1989.
- [25] B. DasGupta, H. T. Siegelmann, and E. Sontag. On a learnability question associated to neural networks with continuous activations. In *Proceedings of Computational Learning Theory (COLT)*, pages 47–56. 1995.
- [26] J. Denker, D. Schwartz, B. Wittner, S. Solla, R. Howard, L. Jackel, and J. Hopfield. Large automatic learning, rule extraction, and generalization. *Complex Systems*, 1:877–922, 1987.
- [27] C. Desoer and M. Vidyasagar. *Feedback Systems: Input-Output Properties*. Academic Press, New York, N.Y., 1st edition, 1975.
- [28] P. Dorato, L. Fortuna, and G. Muscato. *Robust Control for Unstructured Perturbations - An Introduction*. Springer-Verlag, Berlin, Germany, 1st edition, 1992.
- [29] J. Doyle, B. Francis, and A. Tannenbaum. *Feedback Control Theory*. Macmillan Publishing Co., New York, NY, 1st edition, 1992.
- [30] S. Fahlman and C. Lebiere. The cascade–correlation learning architecture. In D. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, pages 524–532. Morgan Kaufmann, 1990.

- [31] J. Fodor and Z. Pylyshyn. Connectionism and cognitive architecture: A critical analysis. *Cognition*, 28:3–72, 1988.
- [32] J. Friedman and W. Stuetzle. Projection pursuit regression. *J. Amer. Stat. Assoc.*, 76:817–823, 1981.
- [33] J. Friedman and J. Tukey. A projection pursuit algorithm for exploratory data analysis. *IEEE Transactions on Computers*, C-23(9):881–890, 1974.
- [34] K. Funahashi. On the approximate realization of continuous mappings by neural networks. *Neural Networks*, 2(3):183–192, 1989.
- [35] K. Funahashi and Y. Nakamura. Approximation of dynamical systems by continuous time neural networks. *Neural Networks*, 6:801–806, 1993.
- [36] M. Georgiopoulos, G. Heileman, and J. Huang. Convergence properties of learning in ART1. *Neural Computation*, 2(4):502–509, 1990.
- [37] S. Geva, J. Sitte, and G. Willshire. A one neuron truck backer-upper. In *Proc. Int. Joint Conf. Neural Nets*, pages 1–1, Baltimore, MD, 1992.
- [38] P. W. Goldberg and M. R. Jerrum. Bounding the Vapnik-Chervonenkis dimension of concept classes parameterized by real numbers. *Machine Learning*, 18(2/3):131–148, 1995.
- [39] S. Grossberg. Adaptive pattern recognition and universal recoding II: Feedback, expectation, olfaction, and illusions. *Biological Cybernetics*, 23:187–202, 1976.
- [40] A. Hajnal, W. Maass, P. Pudlák, M. Szegedy, and G. Turán. Threshold circuits of bounded depth. In *Proceedings of the 1987 IEEE Symposium on the Foundations of Computer Science*, pages 99–110, 1987.
- [41] S. Haykin. *Neural Networks: A Comprehensive Foundation*. Macmillan, New York, N.Y., 1994.
- [42] M. J. Healy, T. P. Caudell, and S. D. G. Smith. A neural architecture for pattern sequence verification through inferencing. *IEEE Transactions on Neural Networks*, 4(1):9–20, 1993.
- [43] G. L. Heileman, M. Georgiopoulos, and C. Abdallah. A dynamical adaptive resonance architecture. to appear in *IEEE Transactions on Neural Networks*, 1994.
- [44] D. Hilbert. Mathematical problems. In *Mathematical Developments Arising From Hilbert Problems*, volume 28, pages 1–34. American Mathematical Society, Providence, RI, 1976.
- [45] J. Hopfield and D. Tank. Neural computation of decisions in optimization problems. *Biological Cybernetics*, 52:141–152, 1985.
- [46] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.
- [47] S. Huang and Y. Huang. Bounds on the number of hidden neurons in multilayer perceptrons. *IEEE Transactions on Neural Networks*, 2(1):47–55, 1991.
- [48] K. Hunt, D. Sbarbaro, R. Żbikowski, and P. Gawthrop. Neural networks for control systems—A survey. *Automatica*, pages 1083–1112, 1995.
- [49] D. Hush. Classification with neural networks: A performance analysis. In *Proceedings of the IEEE International Conference on Systems Engineering*, pages 277–280, 1989.
- [50] D. Hush, B. Horne, and J. Salas. Error surfaces for multi-layer perceptrons. *IEEE Transactions on Systems, Man and Cybernetics*, 22(4), 1992.

- [51] D. Hush and J. Salas. Improving the learning rate of back-propagation with the gradient reuse algorithm. In *Proceedings of the IEEE International Conference on Neural Networks*, volume 1, pages 441–448, 1988.
- [52] A. Isidori. *Nonlinear Control Systems*. Springer-Verlag, Berlin, 2nd edition, 1989.
- [53] A. Ivakhnenko. Polynomial theory of complex systems. *IEEE Transactions on Systems, Man, and Cybernetics*, 1:364–378, 1971.
- [54] J. Judd. *Neural Network Design and the Complexity of Learning*. MIT Press, Cambridge, MA, 1990.
- [55] T. Kailath. *Linear Systems*. Prentice-Hall, Inc., Englewood Cliffs, N.J., 1st edition, 1980.
- [56] A. Kolmogorov. On the representation of continuous functions of several variables by superposition of continuous functions of smaller number of variables. *Dokl. Akad. Nauk SSSR, English Version*, 114:179–182, 1957.
- [57] Y. le Cun. *Models connexionnistes de l'apprentissage*. Doctoral Dissertation, University of Pierre and Marie Curie, Paris, France, 1987.
- [58] Y. le Cun, J. Denker, and S. Solla. Optimal brain damage. In D. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, pages 598–605. Morgan Kaufmann, 1990.
- [59] F. Lewis, S. Jagannathan, and A. Yesildirek. Neural network control of robot arms and nonlinear systems. In D. Elliot, editor, *Neural Networks for Control*, page To appear. Ablex Publishing Corp., 1995.
- [60] F. Lewis, K. Liu, and A. Yesildirek. Neural net robot controller with guaranteed tracking performance. *IEEE Trans. Neural Nets*, 6(3):703–715, 1995.
- [61] L. Ljung. *System Identification: theory for the User*. Prentice-Hall, Inc., Englewood Cliffs, N.J., 1st edition, 1987.
- [62] G. Lorentz. The 13th problem of Hilbert. In *Mathematical Developments Arising From Hilbert Problems*, volume 28, pages 419–430. American Mathematical Society, Providence, RI, 1976.
- [63] W. Maass. On the computational complexity of networks of spiking neurons. In G. Tesauro, D. Touretzky, and T. Leen, editors, *Advances in Neural Information Processing Systems 3*, pages 183–190. Morgan Kaufmann, San Mateo, CA, 1995.
- [64] M. Macintyre and E. Sontag. Finiteness results for sigmoidal ‘neural’ networks. In *Proc. 25th Annual Symp. Theory Computing*. San Diego, 1993.
- [65] W. Miller, F. Glanz, and L. Kraft. CMAC: An associative neural network alternative to back-propagation. *Proceedings of the IEEE*, 78:1561–1567, October 1990.
- [66] W. Miller, R. Sutton, and P. Werbos, editors. *Neural Networks for Control*. MIT Press, Cambridge, MA, 1990.
- [67] M. Minsky and S. Papert. *Perceptrons: An Introduction to Computational Geometry*. MIT Press, Cambridge, MA, expanded edition, 1988.
- [68] K. Narendra and A. Annaswamy. *Stable Adaptive Systems*. Prentice-Hall, Inc., Englewood Cliffs, N.J., 1st edition, 1989.
- [69] K. S. Narendra and K. Parthasarathy. Identification and control of dynamical systems using neural networks. *IEEE Trans. Neural Nets*, 1(1):4–27, 1990.

- [70] R. Newcomb and R. de Figueirido. A functional artificial neural network. In *Proceedings of the 3rd Int. Conf. on Aut. Rob. and Comp. Vision*, pages 566–570, 1994.
- [71] E. Ott, T. Sauer, and J. Yorke. The theory of Embedding. In E. Ott, T. Sauer, and J. Yorke, editors, *Coping With Chaos: Analysis of Chaotic Data and the Exploitation of Chaotic Systems*, Wiley Series in Nonlinear Science, pages 41–64. Wiley-Interscience, New York, NY, 1994.
- [72] D. Plaut, S. Nowlan, and G. Hinton. Experiments on learning back propagation. Technical Report CMU-CS-86-126, Carnegie-Mellon University, Pittsburgh, PA, 1986.
- [73] M. Polycarpou and P. Ioannou. Identification and control of nonlinear systems using neural network models: Design and stability analysis. Technical Report 91-09-01, Dept. of Electrical Engineering-Systems, University of Southern California, Los Angeles, CA 90089, 1991.
- [74] M. Polycarpou and A. Vemuri. “Learning methodology for failure detection and accomodation. *IEEE Contr. Sys. Mag.*, 15(3):16–24, 1995.
- [75] W. Rudin. *Principle of Mathematical Analysis*. McGraw-Hill, Auckland, Australia, 3 edition, 1976.
- [76] D. Rumelhart, G. Hinton, and R. Williams. Learning internal representations by error propagation. In D. Rumelhart and J. McClelland, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, pages 318–362. MIT Press, Cambridge, MA, 1986.
- [77] D. Rumelhart and J. McClelland. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Foundations*, volume 1. MIT Press, Cambridge, MA, 1986.
- [78] D. Rumelhart and J. McClelland. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Psychological and Biological Models*, volume 2. MIT Press, Cambridge, MA, 1986.
- [79] I. Sandberg. Structure theorems for nonlinear systems. *Multidim. Syst. and Sign. Proc.*, 2:267–286, 1991.
- [80] I. Sandberg. Uniform approximation and the circle criterion. *IEEE Trans. Automat. Control*, 38(10):1450–1458, 1993.
- [81] I. Sandberg. General structures for classification. *IEEE Trans. Circ. and Syst.-1*, 41(5):372–376, 1994.
- [82] R. Sanner and J. Slotine. Gaussian networks for direct adaptive control. Technical Report Technical Report NSL-910503, Nonlinear Systems Laboratory, MIT, Cambridge, MA 02139, May 1991.
- [83] R. Sanner and J.-J. Slotine. Stable adaptive control of robot manipulators using “neural” networks. *Neural Comp.*, 7(4):753–790, 1995.
- [84] T. Sauer. Time series prediction by using delay embedding. In A. Weigend and N. Gershenfeld, editors, *Time Series prediction: Forecasting the Future and Understanding the Past*, volume 15 of *Santa Fe Institute Studies in the Sciences of Complexity*, pages 175–193. Addison Wesley, Reading, MA, 1994.
- [85] K.-Y. Siu, V. Roychowdhury, and T. Kailath. Depth-size tradeoffs for neural computation. Technical report, Information Systems Laboratory, Stanford University, 1990.
- [86] J. Sjöberg. *Nonlinear System Identification with Neural Networks*. PhD thesis, Linköping University, Dept. of Electrical Engineering, S-581 83 Linköping, Sweden, 1995.

- [87] P. Smolensky. On the proper treatment of connectionism. *Behavioral and Brain Sciences*, 11:1–74, 1988.
- [88] E. Sontag. Feedforward nets for interpolation and classification. *J. Comp. Syst. Sci.*, 45:20–48, 1992.
- [89] E. Sontag. Systems combining linearity and saturations and relations to neural networks. Technical Report SYCON–92–01, Rutgers Center for Systems and Control, 1992.
- [90] E. Sontag. Neural networks for control. In H. Trentelman and J. Willems, editors, *Essays on Control: Perspectives in the Theory and its Applications*, pages 339–380. Birkhauser, Boston, MA, 1993.
- [91] E. Sontag. Some topics in neural networks and control. Technical Report LS93-02, Rutgers University, Department of Mathematics, New Brunswick, NJ 08903, 1993.
- [92] E. Sontag. State-space and i/o stability for nonlinear systems. *Lecture Notes in Control and Information Sciences*, pages 215–235. Springer-Verlag, Berlin, 1995.
- [93] D. Sprecher. On the structure of continuous functions of several variables. *Trans. Amer. Math. Soc.*, 115:340–355, 1964.
- [94] W. Stornetta and B. Huberman. An improved three-layer back propagation algorithm. In *Proceedings of the IEEE 1st International Conference on Neural Networks*, volume 2, pages 637–644, 1987.
- [95] F. Takens. Detecting strange attractors in turbulence. In D. Rand and L.-S. Young, editors, *Dynamical Systems and Turbulence*, volume 898 of *Lecture Notes in Mathematics*, pages 336–381. Springer-Verlag, Berlin, 1981.
- [96] E. Tzirkel-Hancock and F. Fallside. Stable control of nonlinear systems using neural networks. Technical Report CUED/F-INFENG/TR.81, Cambridge University Engineering Department, 1991.
- [97] L. G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.
- [98] M. Vidyasagar. *Nonlinear Systems Analysis*. Prentice-Hall, Inc., Englewood Cliffs, N.J., 2nd edition, 1993.
- [99] R. Watrous. Learning algorithms for connectionist networks: applied gradient methods of nonlinear optimization. In *Proceedings IEEE 1st International Conference on Neural Networks*, volume 2, pages 619–628, 1987.
- [100] A. Weigend and N. Gershenfeld, editors. *Time Series prediction: Forecasting the Future and Understanding the Past*, volume 15 of *Santa Fe Institute Studies in the Sciences of Complexity*. Addison Wesley, Reading, MA, 1994.
- [101] P. Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. Doctoral Dissertation, Applied Mathematics, Harvard University, Boston, MA, November 1974.
- [102] N. Wiener. *Cybernetics*. The MIT Press, Cambridge, MA, 2nd edition, 1961.
- [103] M. Zak. Non-Lipschitzian dynamics for neural modeling. *Applied Mathematics Letters*, 2(1):69–74, 1989.
- [104] R. Żbikowski, K. Hunt, A. Dzielinński, R. Murray-Smith, and P. Gawthrop. A review of advances in neural adaptive control systems. Technical Report ESPRIT III Project 8039: NACT, Dept. of Mathematics, Royal Institute of Technology, Daimler-Benz AG and University of Glasgow, 1994.

Appendix A

Mathematical Background

A.1 Definitions

A set A of functions from $K \subset \mathbb{R}^n$ to \mathbb{R} is an Algebra if $\forall f, g \in A$ and $\forall \alpha \in \mathbb{R}$:

1. $f + g \in A$,
2. $fg \in A$, and $\alpha f \in A$.

A norm $\| \cdot \|$ of a vector x is a real-valued function defined on the vector space X such that

1. $\| x \| > 0$ for all $x \in X$ with $\| x \| = 0$ if and only if $x = 0$.
2. $\| \alpha x \| = | \alpha | \| x \|$ for all $x \in X$ and any scalar α .
3. $\| x + y \| \leq \| x \| + \| y \|$ for all $x, y \in X$.

A function $f(\cdot) : [0, \infty) \rightarrow \mathbb{R}$ is uniformly continuous if for any $\epsilon > 0$, there is a $\delta(\epsilon)$ such that

$$|t - t_0| < \delta(\epsilon) \implies |f(t) - f(t_0)| < \epsilon$$

Then, f is said to belong to L_p if for $p \in [1, \infty)$,

$$\int_0^\infty |f(t)|^p dt < \infty$$

f is said to belong to L_∞ if it is bounded i.e. if $\sup_{t \in [0, \infty)} |f(t)| \leq B < \infty$, where $\sup \{ \cdot \}$ denotes the supremum of $\{ \cdot \}$.

In some cases, we would like to deal with signals that are bounded for finite times but may become unbounded as time goes to infinity. This leads us to define the extended L_p spaces. Thus consider the function

$$f_T(t) = \begin{cases} f(t) & \text{if } t \leq T \\ 0 & \text{if } t > T \end{cases} \quad (\text{A.1})$$

then, the extended L_p space is defined by

$$L_{pe} = \{ f(t) : f_T(t) \in L_p \}$$

where $T < \infty$. We also define the norm on L_{pe} as

$$\|f(\cdot)_T\|_p = \|f(\cdot)\|_{T_p}$$

A subset S of a vector space $(X, \|\cdot\|)$ is *open* if for each $x \in S$, there is an $\epsilon > 0$ such that the ball of radius ϵ and centered at x is completely in S . A *closed set* is one whose complement in X is open. The *closure* of an open set S is the smallest closed set which contains S . A *bounded set* $S \subset X$ is one in which for each $\epsilon > 0$, there is a finite set of points $y_i \in S$; $i = 1, \dots, n$ such that for $x \in X$, $\|x - y_j\| < \epsilon$ for some $y_j \in \{y_1, \dots, y_n\}$. A *compact set* is a set which is closed and bounded. A vector space is said to be *complete* if every Cauchy sequence is convergent. If the vector space is equipped with an inner product (which induces a norm) is complete, then it is a *Hilbert space*. If the vector space is equipped with a norm (but not with an inner product) and is complete, then it is a *Banach space*. A set $C \subset \mathbb{R}^n$ is *convex* if given any a_1 and a_2 tow points in C , then for all $\lambda \in [0, 1]$, $\lambda a_1 + (1 - \lambda)a_2 \in C$. The *convex hull* of a not necessarily convex set $B \subset \mathbb{R}^n$, is the smallest convex set containing B . A function $f : \mathbb{R} \rightarrow \mathbb{R}$ is *odd* if $f(-x) = -f(x)$ and *even* if $f(-x) = f(x)$.

A.2 Functional Approximation

In 1900, Hilbert delivered before the International Congress of Mathematics in Paris a lecture which has since become the Mathematicians call to arms. In that lecture [44], Hilbert listed 23 open problems, the 13th of which was: *Prove that the equation of the seventh degree $x^7 + ax^3 + bx^2 + cx + 1 = 0$ is not solvable with the help of any continuous functions of only two variables.* The study of this problem has had a large impact on the field of Neural Networks as discussed below. Noting that the solvability of that equation is an algebraic question, while the representation of the solution by a continuous functions of only two variables is an analytical one, we concentrate our review (and our interest) on the analytical part. The first step is then to re-formulate the problem as follows *Prove that there are continuous functions of 3 variables which can not be represented by continuous functions of 2 variables.* For a discussion of the problem and its implications see [62]. Kolmogorov [56] disproved Hilbert's conjecture as stated in the following theorem.

Theorem A.2.1 *Any continuous function $f(x_1, x_2, \dots, x_n)$ on the n -dimensional cube E^n can be represented exactly as a composition of real and continuous functions χ_i and ϕ_{ij} of one variable as follows:*

$$f(x_1, x_2, \dots, x_n) = \sum_{i=1}^{2n+1} \chi_i \left(\sum_{j=1}^n \phi_{ij}(x_j) \right) \quad (\text{A.2})$$

The result was improved by Lorentz, who showed that a single χ is needed [62], and by Sprecher who showed that ϕ_{ij} can be generated as linear combinations $\alpha_{ij}\phi_j$ [93]. Hecht-Nielsen stated Sprecher's result in neural net terminology as follows.

Theorem A.2.2 *Any continuous function $f(x_1, x_2, \dots, x_n)$ on the n -dimensional cube E^n can be represented exactly as the output of a three-layered (one-hidden layer) network having $2n + 1$ nodes in the hidden layer. The inputs x_j to the network are processed through the functions $\alpha_{ij}\phi_j$, which are the nodes of the hidden layer. The output of the hidden layer is processed through χ to give the output of the neural net.*

Yet another result called upon to show the approximation capabilities of neural networks is the Stone-Weirestrass theorem [75].

Theorem A.2.3 *Let A be an Algebra of some continuous functions from a compact set $K \subset \mathbb{R}^{p+m+qn}$ to \mathbb{R}^q , such that A separates points on K and vanishes at no point of K . Then the uniform closure B of A consists of all continuous functions from K to \mathbb{R} .*

An important and sometimes misunderstood issue of approximation is the complexity of an approximating algorithm. This is best illustrated through the results of Barron [5]. Barron relied on the following result which he attributed to B. Maurey and can also be found in [5].

Lemma A.2.1 *If \bar{f} is in the closure of the convex hull of a subset G of a Hilbert space H , where for any $g \in G$, $\|g\| \leq b < \infty$, then for every $n \geq 1$ and every $c > b^2 - \|\bar{f}\|^2$, there is an f_n in the convex hull of n points in G such that*

$$\|\bar{f} - f_n\|^2 \leq \frac{c}{n} \quad (\text{A.3})$$

In other words, one can approximate \bar{f} arbitrarily close with f_n . Using this powerful result, Barron was able to show that

Theorem A.2.4 *For any function f such that*

$$\int_{\mathbb{R}^d} |w| \cdot |F(w)| dw = C_f < \infty \quad (\text{A.4})$$

where $F(w)$ is the Fourier transform of $f(x)$, and for any $n \geq 1$, there exists a linear combination of sigmoidal functions

$$f_n = \sum_{k=1}^n c_k \sigma(a_k \cdot x + b_k) + c_0 \quad (\text{A.5})$$

such that

$$\int_{B_r} [f(x) - f_n(x)]^2 \mu(dx) \leq \frac{(2rC_f)^2}{n} \quad (\text{A.6})$$

where $B_r = \{x; |x| \leq r\}$ is the ball of radius r , and μ is a suitable measure. If in addition, f is such that $C_f \leq C$ for some $C > 0$, then

$$\sum_{k=1}^n |c_k| \leq 2rC; \quad c_0 = f(0) \quad (\text{A.7})$$

Note that this approximation result holds in a Hilbert space G but not in general spaces.

A.3 Embedding

Suppose we are given a dynamical system

$$\begin{aligned} \dot{x} &= F(x) \\ y &= h(x) \end{aligned} \quad (\text{A.8})$$

where x is n -dimensional and y is a scalar. The only accessible information about the system is obtained through measuring $y(t)$. One can then collect $y(t)$ and its delayed values in a vector

$$z(t) = [y(t) \ y(t - \tau) \ \cdots \ y(t - m\tau)]^T \quad (\text{A.9})$$

Since $y = h(x)$, we can then obtain $z = H(x)$. The question then becomes: What conditions on the F and h will guarantee that the information contained in z can be used to study the behavior of x ? Note that we should assume that x has settled down to an attractor, a bounded set in the space of all states.

In 1981, Takens showed that generically, the output measurements of continuous-time systems may be used to reconstruct an image of the attractor [95]. A more recent result by Sauer [71] states the following.

Theorem A.3.1 *Assume a continuous-time dynamical system has a compact invariant set A of box-counting dimension D . Let $m > 2D$ and τ be a time delay. Assume that A contains only a finite number of equilibria, a finite number of periodic orbits of period $p\tau$ for $3 \leq m$ and that there are no periodic orbits of period τ or 2τ . Then, with probability 1, any choice of the measurement function h yields a delay coordinate function H which is one-to-one from A to $H(A)$.*

A.4 Stability

Let x_e be an equilibrium (or fixed) state at t_0 of the free continuous-time, possibly time-varying nonlinear system

$$\dot{x}(t) = f(x, t),$$

i.e. $f(x_e, t) = 0; \forall t \geq t_0$, where x, f are n vectors.

1. *Stability:* x_e is stable in the sense of Lyapunov (SL) at t_0 , if starting close enough to x_e at t_0 , the state will always stay close to x_e at later times. More precisely, x_e is SL at t_0 , if for any given $\epsilon > 0$, there exists a positive $\delta(\epsilon, t_0)$ such that if

$$\|x_0 - x_e\| < \delta(\epsilon, t_0)$$

then

$$\|x(t) - x_e\| < \epsilon; \text{ for all } t \geq t_0$$

x_e is stable in the sense of Lyapunov if it is stable for any given t_0 .

2. *Instability:* x_e is unstable in the sense of Lyapunov (UL), if no matter how close to x_e the state starts, it will not be confined to the vicinity of x_e at some later time. In other words, x_e is unstable if it is not stable at t_0 .
3. *Convergence:* x_e is convergent (C) at t_0 , if states starting close to x_e will eventually converge to x_e . In other words, x_e is convergent at t_0 if for any positive ϵ_1 , there exists a positive $\delta_1(t_0)$ and a positive $T(\epsilon_1, x_0, t_0)$ such that if

$$\|x_0 - x_e\| < \delta_1(t_0)$$

then

$$\|x(t) - x_e\| < \epsilon_1; \text{ for all } t \geq t_0 + T(\epsilon_1, x_0, t_0)$$

x_e is convergent, if it is convergent for any t_0 .

4. *Asymptotic Stability:* x_e is asymptotically stable (AS) at t_0 if states starting sufficiently close to x_e will stay close and will eventually converge to it. More precisely, x_e is AS at t_0 if it is both convergent and stable at t_0 . x_e is AS if it is AS for any t_0 .

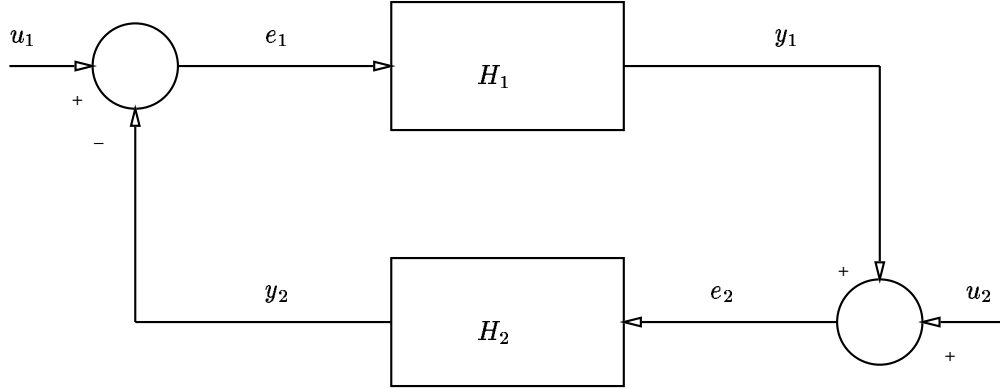


Figure A.1: Block Diagram of General Feedback System.

5. *Global Asymptotic Stability*: x_e is globally asymptotically stable (GAS) at t_0 if any initial state will stay close to x_e and will eventually converge to it. In other words, x_e is GAS if it is stable at t_0 , and if every $x(t)$ converges to x_e as time goes to infinity. x_e is GAS if it is GAS for any t_0 and the system is said to be GAS in this case, since it can only have one equilibrium point x_e .

All these definitions become uniform (uniform stability, uniform asymptotic stability, etc.) if the constants δ_i may be made independent of t_0 .

We also define the concept of bounded-input-bounded-output (BIBO) stability as follows: A system H with input u and output y is BIBO stable if $y \in L_\infty$ whenever $u \in L_\infty$.

Consider the block diagram shown in Figure A.1. The blocks labeled H_1 and H_2 represent two systems (linear or nonlinear) which operate on the inputs e_1 and e_2 as follows

$$y_1 = H_1 e_1 = H_1(u_1 - y_2)$$

$$y_2 = H_2 e_2 = H_2(u_2 + y_1)$$

Let H_1 be an $m \times p$ matrix function, and H_2 be a $p \times m$ matrix function. Therefore, u_1 and y_2 are p vectors, while u_2 and y_1 are m vectors. The first theorem gives sufficient conditions to guarantee that the closed-loop system shown in Figure A.1 is BIBO stable and is given in [27].

Theorem A.4.1 Small gain Theorem Let $H_1 : L_{ep}^p \rightarrow L_{ep}^m$ and $H_2 : L_{ep}^m \rightarrow L_{ep}^p$. Therefore H_1 and H_2 satisfy the inequalities

$$\|(H_1 e_1)_T\| \leq \gamma_1 \|e_{1T}\| + \beta_1 ; \gamma_1 > 0, \beta_1 \in \mathbf{R}$$

$$\|(H_2 e_2)_T\| \leq \gamma_2 \|e_{2T}\| + \beta_2 ; \gamma_2 > 0, \beta_2 \in \mathbf{R}$$

for all $T \in [0, \infty)$ and suppose that $u_1 \in L_\infty^p$, $u_2 \in L_\infty^m$. If

$$\gamma_1 \gamma_2 < 1$$

Then $e_1, y_2, \in L_\infty^p$ and $y_1, e_2 \in L_\infty^m$.

Basically, the small-gain theorem states that a feedback interconnection of two systems is BIBO stable if the loop-gain is less than unity. In other words, if a signal traverses the feedback-loop and decreases in magnitude then the closed-loop system can not go unstable.

Next, we discuss passive systems which were initially defined in terms of electrical networks but have since become important in the study of nonlinear systems. A system H is said to be passive if for any input $u \in L_e$ and an output $y = Hu$, there exist $\beta \in \mathbb{R}$ such that

$$\int_0^T y^T(t)u(t)dt \geq \beta; \quad \forall T \in \mathbb{R}^+ \quad (\text{A.10})$$

H is said to be strictly passive if there exist $\delta \in \mathbb{R}^+$ and $\beta \in \mathbb{R}$ such that

$$\int_0^T y^T(t)u(t)dt \geq \delta \int_0^T y^T(t)u(t)dt + \beta; \quad \forall T \in \mathbb{R}^+ \quad (\text{A.11})$$

Theorem A.4.2 Passivity Theorem *If in Figure A.1, $u_2 = 0$, H_1 is passive, H_2 is strictly passive, and $u_1 \in L^2$, then $y_1 \in L_2$.*