Artificial Neural Networks and their Application to Sequence Recognition

Yoshua Bengio

Department of Computer Science,
McGill University, Montréal

June 1991

A thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for
the degree of Doctor of Philosophy.

Abstract

This thesis studies the introduction of a priori structure into the design of learning systems based on artificial neural networks applied to sequence recognition, in particular to phoneme recognition in continuous speech. Because we are interested in sequence analysis, algorithms for training recurrent networks are studied and an original algorithm for constrained recurrent networks is proposed and test results are reported. We also discuss the integration of connectionist models with other analysis tools that have been shown to be useful for sequences, such as dynamic programming and hidden Markov models. We introduce an original algorithm to perform global optimization of a neural network / hidden Markov model hybrid, and show how to perform such a global optimization on all the parameters of the system. Finally, we consider some alternatives to sigmoid networks: Radial Basis Functions, and a method for searching for better learning rules using a priori knowledge and optimization algorithms.

Résumé

Le sujet de cette dissertation est l'introduction de connaissances dans le design de systèmes d'apprentissage de réseaux de neurones appliqués à la reconnaissance de séquences, particulièrement la reconnaissance de phonèmes en parole continue. Pour l'analyse de séquences, il est important de pouvoir estimer les paramètres de réseaux récurrents. Pour ce problème, plusieurs algorithmes sont évalués et un nouvel algorithme est proposé et testé. Afin d'optimiser les performances, on considère l'intégration de ces réseaux avec d'autres méthodes utiles pour la reconnaissance de séquences, telles que la programmation dynamique et les chaînes de Markov. Un algorithme original est introduit et évalué pour faire une optimisation globale d'un système hybride combinant réseau de neurones et chaînes de Markov. Finalement, certaines alternatives aux réseaux à sigmoides et rétropropagation sont étudiées: les fonctions radiales de base, et une méthode pour chercher de meilleures règles d'apprentissage avec des connaissances a priori et des algorithmes d'optimisation.

Contents

List of Figures

List of Tables

Acknowledgements

First, I would like to thank Renato De Mori, my thesis advisor, for his support and guidance. He has provided motivation for many of the themes explored in this thesis, such as the use of a priori knowledge and careful design of networks for speech recognition systems (in particular concerning the use of phonetic features to encode network outputs), and the combination of artificial neural networks with hidden Markov models.

Much of the work presented in this thesis has been done in collaboration with one and sometimes two other researchers. By chronological order, I would like to acknowledge the help of these colleagues and friends. I appreciated the collaboration with Piero Cosi, who implemented the auditory model used in the experiments described in section 3.1.1 and 3.5.1. I would like to thank Ettore Merlo for introducing me to hidden Markov models many years ago, and for his collaboration to the study of generalization of networks to new sounds. I would like to thank Marco Gori, who spent six months in our lab during which we worked together on the BPS algorithm described in section 5.3.3. My old friend Yannick Pouliot provided me with an interesting application of neural networks to sequence recognition: the recognition of particular domains in proteins. He provided the expertise on that task that allowed us to produce a successful neural network/dynamic programming hybrid system for that recognition problem. I have collaborated for many years with Regis Cardin on the application of neural networks to speech recognition. I would like to acknowledge in particular his good work on a hybrid coder in which the outputs of a neural networks are quantized and used as input symbols for discrete hidden Markov models. My own work on the global optimization of a neural network / hidden Markov model hybrid is an attempt to make such a combination optimal. More recently, I have been collaborating with Ralf Kompe and Giovanni Flammia on the project on building a phoneme recognizer based on neural networks and hidden Markov models. Ralf Kompe worked on the implementation of hybrid equations on the HMM side. He also designed the broad phoneme classification network used in these experiments. Giovanni Flammia worked on the design of the plosive recognition network that was also used in the experiments on the neural network / hidden

Markov model hybrid. Finally, the experiments on learning a learning rule for boolean functions were conducted in collaboration with my brother Samy Bengio as well as with Jocelyn Cloutier.

I would like to thank the following people for reading this thesis and their helpful comments: Patrick Haffner, Michael Rossen, Ralf Kompe, Allyn Takahashi and Michael Galler.

I also owe a lot to the many other people with whom I have had stimulating discussions, including: Patrick Haffner, Yann Le Cun, Luc Devroye, Chris Paige, Leon Bottou, Stuart Geman, Cesare Furlanello, John Bridle, Gary Kuhn, Steve Nowlan, Diego Giuliani, Michael Rossen, Charles Snow, Maurizio Omologo, Xavier Driancourt, Yves Normandin, Marcello Federico.

Finally, I must acknowledge the importance of the support of my family and friends. I am particularly grateful to my parents Celia and Carlo, who gave me the desire to go always further, and to Annie who has prevented me from becoming a single-minded researcher, while supporting me when I had to work intensely (that is, often...).

Chapter 1

Introduction

This thesis studies the introduction of a priori structure into the design of learning systems based on artificial neural networks (ANN) applied to sequence recognition, in particular to the problem of phoneme recognition in continuous speech. Instead of considering an ANN as a system learning from *tabula rasa*, we show that significant performance improvement can be achieved if knowledge about a task is used to introduce structure and meaningful representation into the design of such a learning system. Knowledge about a task includes knowledge about existing solutions proposed for this task.

Contributions are made in three interrelated subtopics: how to improve generalization by integrating ANN learning with domain knowledge, how to recognize sequences with ANNs, in particular recurrent ANNs, and how to integrate ANNs into a hybrid system that uses a sequence analysis method such as Hidden Markov Models (HMM). The design and test of an algorithm for training a particular type of recurrent ANNs and the analysis and evaluation of an algorithm for performing the global optimization of an ANN/HMM hybrid are the main theoretical contributions of this thesis. In addition to applying these algorithms, contributions of this thesis to the design of ANNs for sequence recognition concern the use of a priori knowledge to better design ANNs in order to improve their generalization.

There are several motivations for using ANNs. These models are interesting engineering tools that can perform difficult tasks, such as those studied in this thesis. For example,

they allow us to transform large input spaces into compact representations for speaker-independent phoneme recognition. Furthermore, using such computing architectures may be motivated by the analogy between these models and the operation of real nervous systems.

The motivation for associating a priori knowledge and learning from examples is that such a combination may be the optimal way to take advantage both of the available training data and of prior knowledge about the task. This is justified by theoretical arguments that show that good generalization cannot be obtained if the training set size is limited and the ANN has an unbounded number of degrees of freedom. By restricting the transformations that an ANN can perform using a priori knowledge, one reduces the variance of the functions that can be obtained after the limited training data has been used. Experiments described in this thesis show that it is often useful to take advantage of a priori knowledge about the problem to be solved with the ANN. To build efficient recognition systems given the limited amount of training data and training time available in general, it is important to design carefully the preprocessing, input/output coding, architecture and post-processing of the ANNs.

The study of recurrent networks is motivated by the advantages they offer when applying them to sequence recognition problems such as those considered in this dissertation. Algorithms for recurrent ANNs are important for problems of sequence recognition such as speech recognition because recurrence allows us to represent efficiently context. However, we found that even better performance can be obtained if these recurrent ANNs are integrated with other sequence analysis tools, such as dynamic programming or HMMs.

In this introduction, we describe some basic characteristics of connectionist models (or ANNs), introducing in particular the back-propagation and the Boltzmann machine algorithms, since they have been among the most popular connectionist models and they are used in the experiments described in later chapters. We also review results in learning theory that justify the use of ANNs because of their expressive power and set some guidelines as to their design in order get good generalization. Then we briefly introduce some issues in automatic speech recognition.

In the second chapter, we present a formal description of the back-propagation algorithm, which is extended in Chapter 4 to the case of multi-layer ANNs with delays and recurrence. Some problems with back-propagation and possible solutions to these problems, such as acceleration techniques, are also discussed in the second chapter. We present some experiments that compare online and batch learning when the size of the training set is varied.

The integration of speech knowledge and learning from examples is the subject of the third chapter. Experimental examples are used to illustrate several important design issues: preprocessing, input coding, architectural constraints, modularization, and output coding. We describe experiments on an auditory model for ANN preprocessing and present interesting results concerning the generalization of ANNs trained to recognize articulatory features of vowels. The use of a priori knowledge, even if imperfect, and of carefully crafted networks is motivated by the need to reduce appropriately the VC-dimension (a measure of complexity, see section 1.3.3) of those networks in order to improve their generalization when the number of training examples is limited.

Chapter 4 deals specifically with the problem of sequence analysis and the use of ANNs for that problem. An original algorithm for a particular type of recurrent ANN is described and two other algorithms for arbitrary recurrent ANNs are compared. Finally, hybrids of ANNs with dynamic programming and HMMs are considered. An application of an ANN / dynamic programming hybrid to the analysis of amino-acid sequences is described.

The fifth chapter discusses some issues concerning the integration of ANNs with other tools in a modular way. In particular, we are interested in methods of modeling the temporal structure of a signal. We argue that it might be preferable to perform a global optimization of such hybrids, which can be done if learning in each component of the hybrid system depends on a common objective function, and its output is a differentiable function of its inputs.

In Chapter 6, we present an important element of this thesis: a hybrid of ANNs and HMMs with an algorithm for the global optimization of all its parameters. First we explore how a probability density function could be expressed in terms of the output of an ANN.

Then we show for several HMM optimization criteria and model types how to compute the gradient of the optimization criterion with respect to the parameters of the ANN. Finally, an experimental assessment of this method for plosives in continuous speech recognition shows the advantages of using a hybrid and doing global optimization. These experiments are done using some of the techniques described in Chapter 3. In particular, multiple ANN modules are used for different discrimination tasks, with each module using specialized preprocessing adapted to the task.

Finally, in Chapter 7, we consider alternatives to the standard back-propagation algorithm such as Radial Basis Functions, and ideas on how to search for better learning algorithms using a priori knowledge, as well as learning methods.

1.1 Biological Background

The human brain is an organ made of 10^{10} to 10^{12} nerve cells called *neurons*. Neurons are connected to each other through *synapses*. Each neuron has on the order of 10^3 synapses. Most researchers [Byrn87] now accept that animal learning involves changes of synaptic efficacy, i.e., a quantity which measures how much a neuron can affect another one through a synapse. At chemical synapses, the presynaptic terminal of a neuron is very close (about 50 nm) to a postsynaptic *dendrite* of another neuron. When the pre-synaptic neuron fires, neurotransmitters are released on the pre-synaptic side and the reaction of receptors on the post-synaptic side provokes a local change in membrane voltage potential at the post-synaptic site. Through these dendrites, the neuron integrates signals coming from other neurons. When the voltage potential at the surface of a the cell body of a neuron reaches a threshold, the neuron fires: an impulse or a series of impulses is transmitted along its axon to other neurons.

Connectionist models are simplified models inspired from biological neural networks. They involve homogeneous networks of simple processing units. A connectionist model can be described by a graph (see Chapter 2) where each node represents a processing unit that corresponds to a neuron, and links or connections correspond to synapses. Each such unit operates according to a very simple neuron model. Many of the currently used models have

a feedforward architecture (i.e., their graph contains no cycles). However, the architecture of the brain is not homogeneous and it is not feedforward: it contains many feedback paths. Furthermore, many types of neurons, synapses and neurotransmitters have been observed [Gard87, Byrn89, Tam89]. In Section 7.2, we consider a way in which particular neurotransmitters called neuromodulators may propagate information that can be used to adapt synaptic strengths.

Some connectionist learning algorithms are based on "batch" training, that is, observation of a fixed set of training patterns followed by a modification of the system parameters. In contrast, learning in the brain is probably "online", i.e., the brain adapts continuously, depending on its environment.

Although adaptation is a very important feature of brains, learning in biological nervous systems is not from a *tabula rasa*. Instead, each individual brain is constrained in its structure and its potential functions. Genes somehow specify many of the architectural characteristics of the brain, as well as innate behaviors. From a theoretical point of view (see section 1.3), it makes sense to use a priori information about the environment to constrain a learning system, especially if a limited amount of training data, and time to assimilate it, is available. This may be particularly important for newborns and youngsters, who have had only very limited experience and yet must behave in such a way as to survive until they are stronger and more experienced.

1.2 Connectionist Models

Connectionism can be described as the study of certain classes of massively parallel architectures composed of a large number of similar and simple processors, used for learning, and in which most of the learned knowledge is associated to the connections among units [Rume86a]. These architectures are inspired from biological as well as psychological models, but they are being applied to many Artificial Intelligence as well as engineering problems. Although artificial and biological neural networks differ significantly in many aspects, they generally share some interesting properties such as fault-tolerance [Moor88], learning from examples, distributed processing, and associative storage of information (see [Rume86a] for

a discussion about ANN properties).

Connectionist models, or Artificial Neural Networks (ANN), use fairly simple mechanisms for both neuron operation and the modification of synapses; these models can be used to solve some difficult learning problems, including the problem ("hard learning", [Hint87]) of training hidden units[1] of the network when reinforcement or supervision is only available to some neurons (output units). Early artificial neural network models [Rose57, Rose62] were not able to solve that problem [Nils65, Mins69] because no efficient algorithm was known to train a network with hidden units, which are required in order to learn non-linear functions. Minsky and Papert [Mins69, Nils65] have demonstrated the limitations of single-layer perceptron: they cannot separate classes that are non-linearly separable. However, most problems of interest can be solved only with a non-linear transformation.

The more recent, more powerful, abstract models, such as those presented in the book [Rume86a], can learn to perform non-linear mappings. They were used for automatic speech recognition (see [Lipp89] for review), handwritten character recognition [LeC89c], optimization [Hopf84, Pete90], robotics [Jord90, Pome89], financial expert systems (loan applications evaluation, [Coll89]), and other Artificial Intelligence problems. Connectionist models are based on simplifications of biological neural networks, but they often have non-biologically plausible features. As Hinton points out [Hint87], a mathematically derived algorithm such as back-propagation is not plausible as a biological model of learning for many reasons. For example, the transmission of information in neurons and synapses is in both a forward and a backward direction (which is required with the back-propagation algorithm). Furthermore, extensions of the back-propagation algorithm to general recurrent networks are either non-local in time or non-local in space (see section 4.3 for more details on these questions). However, the back-propagation algorithm is one of the most powerful training algorithms for ANNs.

From a statistical point of view, most connectionist models can be considered as techniques to construct — often for classification problems — consistent non-parametric estimators [Gema91], for regressions, as well as probability distributions (see section 1.3.4). Other techniques that have been used to construct non-parametric estimators include Parzen

[1] Hidden units are neither input (sensory) nor output (motor) units.

windows and Nearest Neighbor rules [Duda73], regularization methods [Wahb82, Pogg89], and decision tree methods (ID3 [Quin86], CART [Brei84]).

Some classes of connectionist algorithms will now be briefly described. More details on the back-propagation algorithm can be found in Chapters 2 and 4. This algorithm was used in most of the experiments described in this thesis. The Boltzmann machine algorithm was used in the experiment described in section 3.2.

1.2.1 Back-Propagation

This is one algorithm to train a network with hidden units, such as in Figure 1.1. The neuron output is assumed to be a differentiable function of its inputs. Typically, this function is represented as follows:

$$y_i = f(\sum_j w_{ij} y_j) \tag{1.1}$$

where y_i is the output activation of unit i, that may correspond to the average firing rate of a neuron, w_{ij} is a weight that corresponds to the strength of the synapse that connects neuron j to neuron i. The function $f(\cdot)$ is a non-linear squashing function that is usually taken to be the logistic function, ranging in (0,1):

$$f(x) = \frac{1}{1 + \exp(-x)} \tag{1.2}$$

or the symmetric sigmoid, ranging in (-1,1):

$$f(x) = \tanh(x) \tag{1.3}$$

In the experiments described in this thesis, we generally used a symmetric sigmoid for hidden units and an asymmetric sigmoid for output units. This was motivated by the faster convergence of networks whose inputs have zero mean [LeC90] and because it is easier to interpret network outputs in the range (0,1).

In the back-propagation algorithm, the function $f(\cdot)$ has to be differentiable. This allows us to compute the partial derivative of a neuron's output with respect to the output of neurons which have influenced it. The principle of the algorithm is to use those partial derivatives

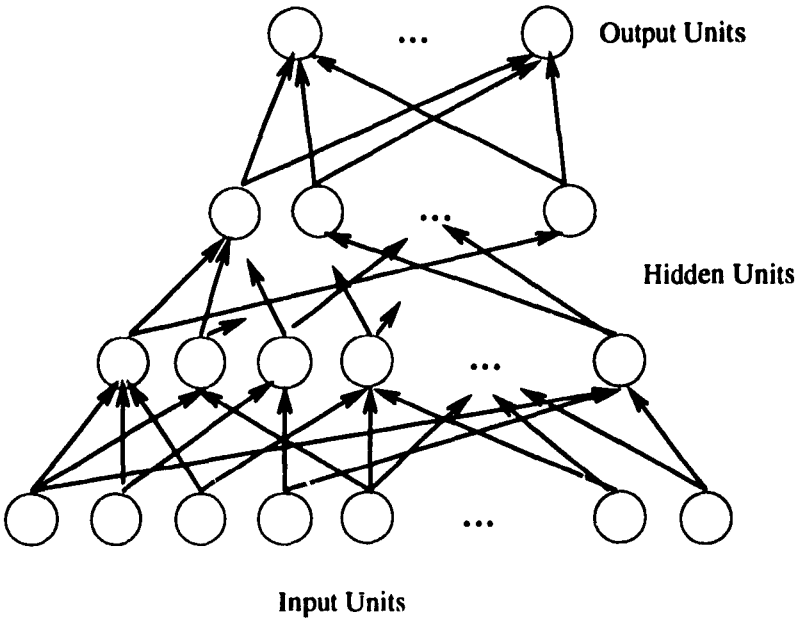


Figure 1.1: Architecture of a multi-layer network trained with back-propagation.

in order to compute the gradient of a cost function with respect to the parameters of the network, and then perform a gradient descent in the space of those parameters in order to minimize that cost function. A commonly used cost function is the Least Mean Square (LMS) criterion, which is the sum of the squares of the differences between activations of output units and target values for those units:

$$C = 0.5 \sum_t \sum_i (y_{it} - target_{it})^2 \tag{1.4}$$

where y_{it} is the activation of output neuron i for pattern t and $target_{it}$ is the corresponding target output or desired value. In general, the target outputs are not chosen to be the saturating values of the sigmoid (e.g., 0 and 1) to avoid resorting to infinite weights[2]. Typical targets used in the experiments described in this thesis are 0.9 and 0.1 to indicate "high" and "low" target values respectively.

The LMS criterion is particularly well suited to the problem of regression estimation, in which one wishes to estimate the parameters of a parametric function $\hat{f}(\theta, x)$, such that the Euclidean distance between that function and a target function $f(x)$ is minimized. The LMS criterion has the nice property that it is differentiable with respect to the function

[2]The sigmoid function reaches saturating values when its argument is $-\infty$ or ∞.

value. Such a criterion is not suitable for every application; in fact, its minimization does not necessarily correspond to the minimization of classification error when an ANN is used for classification (see [Bott91]). However, the true classification error is not a differentiable function of the system parameters, and thus cannot be used to train an ANN with gradient descent.

If the activations of the output units represent a probability distribution over binary vectors[3], then an alternative criterion is the cross-entropy [Hint87] between the desired and actual probability distributions (conditional on the inputs of the networks):

$$C = -\sum_t \sum_i target_{it} \log_2 y_{it} + (1 - target_{it}) \log_2 (1 - y_{it}) \tag{1.5}$$

The back-propagation algorithm was proposed in [Rume86b] but had been independently discovered several times in varying forms and purposes [LeCu85, Park85, Werb74]. It is usually employed in a supervised mode, in which target or desired values are known for a set of training examples, i.e., a set

$$S_T = \{(I_T, D_T)\} \tag{1.6}$$

of pairs of input and output patterns is given. However, the algorithm can also be used for reinforcement learning [Bart81, Bart83, Bart85] as in [Jord90]. With reinforcement learning, instead of a desired output for each pattern and output unit, the network is provided with a scalar reinforcement signal (which may come only once in a while rather than after every input pattern). For example, in the pole balancing problem [Bart83], the ANN receives as input information about the position and the angle of the pole at every time step. Its output controls the force applied at the base of the pole (to avoid falling down). It receives negative reinforcement at the end of a training sequence, which is when the pole falls down (due to the wrong "actions" of the network). The "forward model" of [Jord90] uses an additional network that models the influence of the outputs of the first network upon the reinforcement, thus allowing us to compute the gradient of the desired reinforcement for the outputs of the first network with back-propagation.

[3] Each real-valued output between 0 and 1 represents the probability that the corresponding binary output be 1.

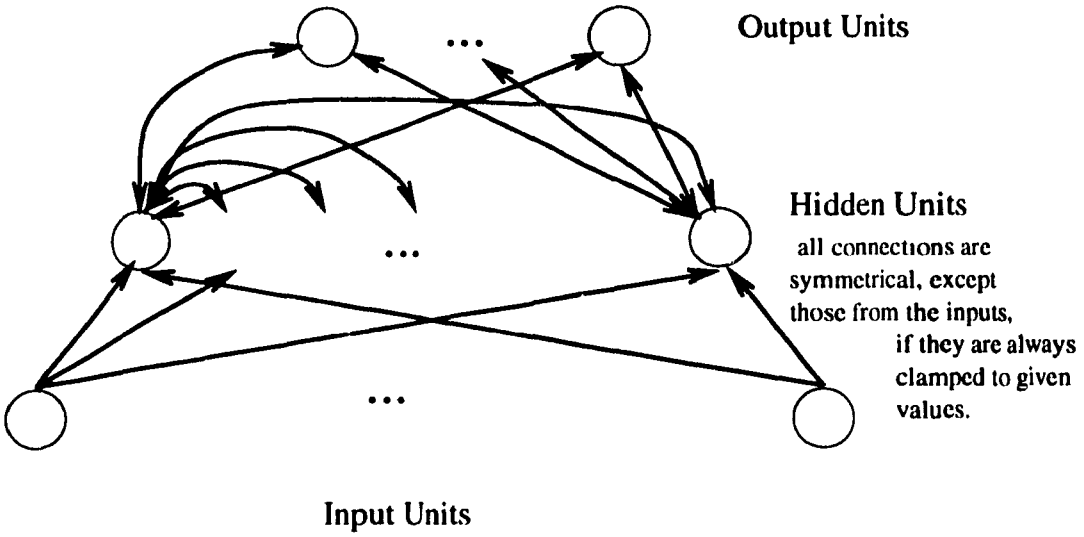


Figure 1.2: Typical architecture of a Boltzmann machine. Connections and corresponding weights are symmetrical.

Several extensions of the basic back-propagation algorithm for recurrent networks have been proposed and some will be discussed in section 4.3. The actual equations for the learning algorithm will be described in section 2.1. In addition, problems of convergence time and local minima will be discussed in section 2.2.

1.2.2 The Boltzmann Machine

This model was introduced in [Fahl83, Hint84]. The Boltzmann machine consists of a network (see Figure 1.2) of stochastic units with boolean outputs that are connected through symmetric connections ($w_{ij} = w_{ji}$). The network is recurrent and it is relaxed by simulated annealing until it reaches a fixed point which should correspond to the global minimum of an energy function. It can thus be considered as a generalization of the Hopfield model [Hopf82], which has no hidden units. In the Hopfield model, all units are connected to all other units, and they are used as inputs, as well as outputs. On the other hand, the Boltzmann machine may have hidden units, allowing it to solve "hard-learning problems" [Hint87] that can't be solved with a linear transformation.

Each unit may assume a value of 0 or 1. The output y_i of the i^{th} unit is stochastically set

to 1 according to a probability

$$p_i = \frac{1}{1 + \exp(-\frac{1}{T}\sum_j w_{ij} y_j)} \tag{1.7}$$

where w_{ij} is the weight of the connection between units i and j, and T is a parameter called "temperature". The Boltzmann machine implements a Monte-Carlo algorithm for reaching a minimum of an energy function which measures the degree of "agreement" among units, i.e., two units i and j maximally "agree" if $y_i y_j w_{ij}$ is large and positive. If the update rule of equation 1.7 is applied iteratively while slowly decreasing the temperature, the network converges to a point of "maximum agreement". The simulated annealing guarantees this state of low "energy" will be reached if the cooling is performed slowly enough [Gema84].

Instead of decreasing a mean squared error, the Boltzmann machine learning algorithm maximizes the likelihood of generating a target input/output discrete distribution. The learning algorithm is based on gradient descent in the space of the weights in order to minimize the Kullback information measure, a measure of the difference between two distributions: the statistical behavior of the network when the activations of the output units depend only on the network state and inputs (phase 1) , and when these activations are fixed from outside with their target or desired values (phase 2). It is interesting to note that this gradient can be expressed as a function of locally measurable quantities: the difference between the probabilities of joint pre- and post-synaptic activation for the two phases, respectively $p_{ij}^{(1)}$ and $p_{ij}^{(2)}$. The weight change that corresponds to descent in this gradient is the following:

$$\Delta w_{ij} = \epsilon(p_{ij}^{(2)} - p_{ij}^{(1)}) \tag{1.8}$$

where ϵ is the learning rate.

The algorithm runs very slowly on sequential machines, because the parallel nature of the algorithm is not taken advantage of, and for each input pattern the simulated annealing process has to be completed – in both phases when training. Furthermore, because of the limited time usually allowed for the measurement of the joint probabilities of activations, their estimation is imprecise, and thus the resulting estimation of the gradient is very noisy, yielding a slow convergence [Hint87].

1.3 Learning Theory for Connectionist Models

Although research on connectionist models has been carried out for only a few years (except for the initial work on artificial neural networks of the late 50's and early 60's, for example [Rose62]), many theoretical results have already been obtained that analyze mathematically artificial neural network capabilities, complexity, generalization power and learning. Many of these results are based on the use of statistical tools. See for example [Lipp87, Barr88, Gall88, Barr89, Haus89, Tish89, Whi89b, Bau+89, Hint90].

1.3.1 Expressive Power

By expressive power, we mean a measure of the "number" of functions that an ANN can approximate precisely. We have already mentioned that some hidden units were required in order for an ANN to perform non-linear transformations. Many researchers have studied the expressive power of several types of connectionist models. For example, [Cybe89, Horn89, Funa89, Stin89] show that a feedforward neural network can perform with arbitrary precision any continuous transformation, given enough hidden units. More precisely, a single hidden layer is theoretically sufficient, except in some pathological cases.

Poggio and Girosi [Pogg89] show that a network of generalized Radial Basis Functions (see section 7.1) can also approximate with arbitrary precision a smooth function. This type of network has the advantage that it produces close to zero outputs in regions of the input space that are very remote from the input examples.

1.3.2 Complexity of the Loading Problem

Although neural networks have great expressive power, it does not mean that it is easy to find those network parameters that allow us to approximate a desired transformation as well as possible. In fact, [Judd88] has shown that the so-called "loading problem" is NP-Complete. The loading problem can be defined as follows. Given a set of examples of input/output pairs and an ANN, is there any choice of network parameters (i.e., weights) for which the examples are satisfied?

However, results by [Baum89] suggest that finding the optimal weights could be done in polynomial time with learning algorithms that are allowed to add units and connections during learning. These types of learning algorithms are called *constructive* algorithms. Examples of constructive algorithms[4] are the cascade-correlation algorithm of [Fahl91] and the algorithm of [Plat91] for automatically allocating resources in a Radial Basis Functions network.

On the other hand, empirical estimates [Hint87] indicate that the learning time for the back-propagation algorithm on a serial machine grows approximately by $O(N_w^3)$, where N_w is the number of weights in a network.

Furthermore, it is important to consider if we must find the global minimum of the cost function, or whether it would be enough to obtain weights that correspond to a quasi-optimal cost. It is usually observed that over-training often yields a loss in generalization. Hence it might even be *better* not to reach the global minimum of the cost function for the training set if our goal is to reduce errors on the underlying distribution sampled with the training set. See [Chau90] and [Mor90b] on stopping short of convergence to improve generalization.

1.3.3 Generalization

Generalization is a crucial question for learning algorithms. Indeed, a learning algorithm would not be very useful if it would learn the training set perfectly but would not be able to compute the expected output for new data in most cases, when the test data is sampled from the same underlying distribution as the training data. One of the attractive features of neural networks is that they seem robust: they tend to generalize well even when the inputs to the network and the training data are noisy. For example, comparisons with symbolic learning methods (e.g., ID3, CART), indicate that, in many cases, ANNs generalize better, particularly in the presence of noise [Tsoi91].

One of the measures of complexity or capacity of a learning system that is becoming

[4] However a polynomial convergence time has not been theoretically demonstrated yet for any of these algorithms.

popular in learning theory for ANNs is the Vapnik-Chervonenkis dimension (VC dimension). It measures the maximum number of dichotomies that can be induced by the learning system. The notion of capacity was introduced by Cover [Cove65] and developed in [Vapn71, Vapn82]. Reviews of this notion and related questions are presented in [Devr88, Blum87, Poll84, Haus89]. Results from learning theory indicate that, when the number of training examples is finite, the generalization of a learning system depends on its VC-dimension [Vapn82, Vapn71], i.e., on the number of functions it can perform, which depends on the number of free parameters of the network, as well as on its architecture. More precisely, theoretical analysis shows that the more training examples are available, the better the generalization will be[5], but the more complex the learning systems (e.g., more weights in a network), the more difference there might be between the training set error and generalization error. This agrees with Occam's razor principle, which suggests choosing the simplest theory that explains our data (training set), in order to maximize its generalization to new data. In this thesis, we sometimes use the term *bias*, as defined mathematically in [Gema91] to describe the constraints and structure imposed on a learning system in order to reduce its VC dimension. This may result in a non-zero error even when an infinite training set is available. However, it may reduce the variance of such systems, i.e., the average difference between networks trained with the same amount of data, thus reducing the difference between training set error and test set error when the number of training patterns is limited.

An interesting result [Bau+89] concerning the generalization of some neural networks gives bounds on the number of training examples necessary in order to obtain an acceptable generalization error. These results are for feedforward networks with hard threshold rather than sigmoid units (because they are simpler to analyze), independently of the type of algorithm used to train the network. If the error on the training set is less than $\frac{\epsilon}{2}$ and the number of examples N_e is bounded as follows:

$$N_e = O(\frac{N_w}{\epsilon}\log(\frac{N_u}{\epsilon})) \tag{1.9}$$

$$N_e = \Omega(\frac{N_w}{\epsilon}) \tag{1.10}$$

[5]because the desired function will be narrowed down more precisely in the space of the functions that correspond to some parameter values for the system, constrained to functions that are consistent with the training set.

then with a confidence approaching certainty, the generalization error will be less than ϵ, where N_w is the number of weights and N_u is the number of units in the network.

1.3.4 Approximation of Posterior Probabilities

In [Whi89b] it is shown that an ANN trained with the LMS criterion approximates the conditional expectation $E(D \mid X)$, i.e., the value of D that will be realized on average, given a particular instance of X, where X and D are random variables representing the input and desired output vectors, respectively. However, this can be obtained only if there are enough hidden units and the training network converges to the asymptotic limit, that is, the conditional expectation of the target given the input.

If the desired outputs D are continuous valued, then the ANN is performing regression estimation and $E(D \mid X)$ is the best predictor of D given X in the mean-squared error sense [Gema91]. If the desired output D is discrete with value 1 for class A and 0 for class B, then the regression becomes $E(D = 1 \mid X) = Pr(\text{Class } A \mid X)$, i.e., the a posteriori class probability conditioned on the input variable X.

If we allow the size of a network to grow with the number of available training examples, then ANN models such as those trained with back-propagation can be considered non-parametric estimators. For example, with the back-propagation algorithm, one can estimate the conditional expectation $E(D \mid X)$ as defined above (see also [Gema91]).

The LMS criterion forces the average of the difference between actual and desired output to be as close to zero as possible, but this may be at the cost of very large errors for values of the inputs X that are unlikely to occur.

1.4 Automatic Speech Recognition

The major sequence recognition task considered in this thesis is that of automatic speech recognition. An automatic speech recognition system is a system that can transform a raw speech signal into a sequence of descriptors that characterize the spoken utterance well

enough so that a computer can use it to take decisions and perform appropriate actions. The lowest level descriptors that such a system can recognize are often *phonemes*, of which there are only a few dozen. Often, the objective is to recognize sequences of *words*. Special application vocabularies may contain as few as several hundred words whereas an unconstrained human talker may attempt to use several tens of thousands of words. A speech *understanding* system may use higher level information such as semantics and syntax in order to recognize *sentences*, of which there may be an unbounded number of instances, but which may be constrained by a particular grammar.

1.4.1 Characteristics of a Speech Recognition System

Let us consider a few characteristics of a speech recognition system that permit evaluating its applicability.

- Vocabulary size. The *vocabulary* is the set of words that the system can recognize. Existing speech recognition systems have a performance that tends to degrade when the number of words in the vocabulary increases. This can be explained by the greater number of pairs of words which can be confused (because the average distance between words in the acoustic space is reduced). A related measure is the *perplexity* of a speech recognition task [Lee 89], which is proportional to the number of possible decisions at each decision point, or the number of bits necessary to specify the next word.

  Many researchers have decided to model sub-word units [Lee 89]. This may permit using a very large vocabulary, since the number of sub-word units that need to be modeled is bounded even when we consider tens of thousands of words. This is because each word can be defined as a combination of these sub-word units. In this thesis, the sub-word units that have been considered are the most commonly used: phonemes. Note that, sharing sub-words units may have the disadvantage that it becomes more difficult to model within-word coarticulation, compared to modeling each word separately.

- Speaker-independent / speaker-dependent recognition. Speaker variability is a very important problem for automatic speech recognition. [Watr89] has shown, for ex-

ample, in the case of formant frequencies for steady-state vowels, that whereas the vowe's were clearly separable for a given speaker, there was a considerable overlap in the formant frequency space when multiple speakers were considered. Although the performance of current speaker-dependent speech recognition system can often be acceptable, it is not the case for speaker-independent systems. Two research alternatives are being considered to deal with this problem. The first one is to perform an automatic *speaker adaptation* [Brid91]. Some speaker-adaptation systems require the talker to pronounce a given reference sentence. The other approach consists of building a truly speaker-independent recognition system, that maps the acoustic signal into a description space that is independent of the speaker. This approach has been pursued in the speech recognition experiments described in this thesis.

- Continuous speech / isolated word recognition.

  Isolated words are pronounced by leaving a short silence between each word. They are much easier to recognize than continuous speech, for several reasons. First, the recognition problem is broken down into simpler problems: words can be considered independently. Second, when pronouncing isolated words, talkers tend to have a better, slower and clearer pronunciation, thus reducing within-word coarticulation. Third, the problem of coarticulation *between words* is almost eliminated (depending on the pause between words). Unfortunately, pronouncing isolated words is not natural nor comfortable for ordinary users. It is thus desirable to consider continuous speech recognition systems, which can model coarticulation effects, i.e., the influence of left and right phonetic context on the pronunciation of each phoneme.

1.4.2 Some Problems with Current Approaches

Coarticulation problems come from the acoustic variability of phonemes depending on their phonetic context. In continuous speech, this influence may span several phonemes and affect contiguous words, especially short words.

The current approaches either ignore coarticulation, i.e., considering one model per phoneme, or model separately a very large set of *diphones* or *triphones* [Lee 89]. A *diphone* model is

a model of all instantiations of a phoneme in a given right or left context, i.e., considering the left or right phoneme. For example /p-eh/,/p-ih/,/p-ah/ are different diphones for the phoneme /p/ when it is followed by the phonemes /eh/, /ih/, or /ah/, respectively.

Another common characteristic of many speech recognition systems is that they model the speech signal as a concatenation of *independent* segments, as in hidden Markov models (see section 6.1). The psychophysics of speech production seem to indicate a different kind of sequence: moving from one target to another in a space of articulatory features. In that case, there are no clear cut boundaries between speech units, but rather a continuous evolution of articulatory features going from the direction of and toward the direction of targets that may correspond to phonemes or to other subphonemic events (see the temporal decomposition of [Atal83]).

One of the problems with current approaches is their weakness at rejecting or modeling words not in the training vocabulary. In fluent speech, talkers often pronounce non-words such as "ah","hmm", or sounds made with the throat. There are often other noises in the environment that make the observed signal more difficult to recognize. Robustness with respect to noisy signal is thus an important feature of a desirable speech recognition system. In addition, it is important to model sub-word units, in order to be able to characterize words not in the training set (by modeling them as concatenations of sub-word units).

In experiments described in Chapters 3 and 6, we have attempted to address the problem of coarticulation by choosing an output coding that is based on articulatory features. This allowed us representing *articulatory* context in the network outputs, e.g. the horizontal place of articulation of the phoneme that follows a plosive. This results in more compact representations than one output or one model per diphone.

Chapter 2

The Back-Propagation Algorithm

The principle of the back-propagation algorithm was briefly introduced in section 1.2.1. This algorithm was independently discovered by [Werb74][1], [Park85], [LeCu85], and [Rume86b]. It is based on the computation of the first derivatives of a cost function with respect to the parameters of the network. Discrete gradient descent then allows convergence to a minimum of the cost function, but not necessarily to the global minimum. Let us consider a simple generalization of the original algorithm that allows one to use an arbitrary differentiable transfer function F for each unit.

2.1 Formal Description

Consider a network of units forming a directed graph $G = (U, L)$, where $U = \{u_i\}$ is a set of units, and $L = \{l_{ij} : \exists$ link from unit s_{ij} to unit $i\ \}$ where s_{ij} is the unit number of the j^{th} input into node i. That particular formalism was chosen to describe the network graph in order to allow for multiple links with various delays between two units (Chapter 4). Units are either input units, hidden units or output units: the set of input units is U_I, the set of hidden units is U_H, and the set of output units is U_O, so $U = U_I \cup U_H \cup U_O$.

Let us consider for the moment the case of a *static* network, i.e., whose graph contains no

[1] Although in the case of [Werb74] the algorithm is not presented in the context of neural networks.

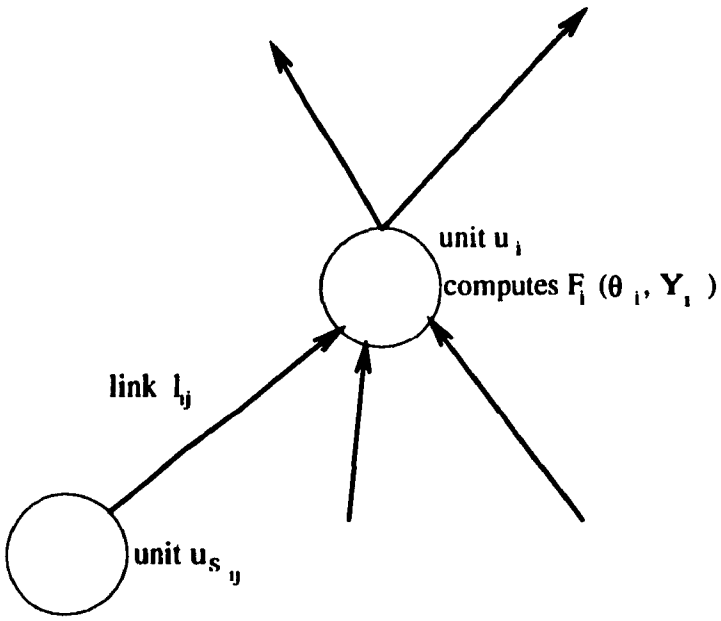


Figure 2.1: Directed graph of an ANN for training with back-propagation. Units are nodes of the graph, connections are links of the graph.

cycles. The cases of a *recurrent* network and of a network with delays will be discussed in section 4.3.

Let us assume without loss of generality that the units of a *static* network are ordered in such a way that if $i < j$ then there is no path in G from u_j to u_i.

2.1.1 Network Operation: Forward Phase

During a forward phase, when the t^{th} input pattern P_{I_t} is presented at the input units, the network computes a transformation that can be read out at the output units. Starting at unit 1, and until unit N_u, each unit computes its activation as follows:

if $u_i \in U_I$,

$$y_i = P_{I_{t,i}}\text{, i.e., the } i^{\text{th}} \text{ element of the } t^{\text{th}} \text{ input pattern.}$$

else,

$$y_i = F_i(\theta_i, Y_i)\text{, i.e., a parametric function of its inputs,}$$

(2.1)

where y_i is the output (also called activation) of unit u_i, θ_i is a set of parameters for the function $F_i(\cdot)$, and Y_i is the set of activations of the units $u_{s_{ij}}$ such that $l_{ij} \in L$, where s_{ij} is the node number of the source of link l_{ij}.

The "standard" model [Rume86b] assumes that $F_i(\cdot) = F(\cdot)$ and is a sigmoid (equations 1.1, 1.2 and 1.3) of a weighted sum of the input activations of the unit u_i:

$$F(w_i, Y_i) = f(\sum_j w_{ij} y_{s_{ij}}) \tag{2.2}$$

2.1.2 Network Learning: Backward Phase

The principle of the backward phase is to obtain the error gradient with respect to the network parameters ($\frac{\partial C}{\partial \theta_{ij}}$) by first computing the derivatives of the local cost C_t with respect to each unit activation ($\frac{\partial C}{\partial y_i(t)}$), using the back-propagation formulae.

A differentiable cost function is defined as a function of the network output units. For example, the Least Mean Square (LMS) criterion is defined as in equation 1.4. In that case, the derivative of the cost function with respect to the activations of the output units is as follows:

$$\frac{\partial C}{\partial y_i(t)} = \sum_t \frac{\partial C_t}{\partial y_i(t)} = \sum_t \sum_{i:u_i \in U_O} (y_i(t) - target_{it}) \tag{2.3}$$

where $target_{it}$ is the target or desired output value for unit i at time t.

In the back-propagation phase, units are considered in the reverse order of the forward phase (from u_N down to u_1). The error gradient of a hidden unit $u_i \in U_H$ is computed as follows, using the chain rule:

$$\frac{\partial C_t}{\partial y_i(t)} = \sum_{s_{jk}=i} \frac{\partial C_t}{\partial y_j(t)} \frac{\partial y_j(t)}{\partial y_i(t)} \tag{2.4}$$

For example, if the function $y_j = F_j(\cdot)$ has the form of equation 2.2 (a squashed weighted sum), then

$$\frac{\partial y_j(t)}{\partial y_i(t)} = f'(\sum_k w_{jk} y_{s_{jk}}) w_{jl} \tag{2.5}$$

where $s_{jl} = i$.

The derivative of $f(\cdot)$ can be easily computed. In the case of an asymmetric sigmoid (equation 1.2),

$$f'(x) = f(x)(1 - f(x)) \tag{2.6}$$

whereas in the case of a symmetric sigmoid (equation 1.3),

$$f'(x) = 0.5(1 + f(x))(1 - f(x)) \tag{2.7}$$

Finally, the gradient with respect to the activations y can be used to compute the gradient with respect to the parameters of $F(\cdot)$:

$$\frac{\partial C_t}{\partial \theta_{ij}} = \frac{\partial C_t}{\partial y_i(t)} \frac{\partial y_i(t)}{\partial \theta_{ij}} \tag{2.8}$$

If the function $y_j = F_j(\cdot)$ has the form of equation 2.2 (a squashed weighted sum), then

$$\frac{\partial y_i(t)}{\partial w_{ij}} = f'(\sum_k w_{ik} y_{s_{ik}}(t)) y_l(t) \tag{2.9}$$

where $s_{ij} = l$.

See section 2.2.1 for a discussion on the application of the error gradient, in particular regarding stochastic *vs* deterministic parameter updating.

2.1.3 Weight Sharing

It is sometimes desirable to constrain a network by forcing some of the parameters used for a unit u_i to be shared by a unit u_j. This is typically the case when u_i and u_j have a different receptive field but should compute the same feature. In that case, weight sharing [Lang88] may reduce the VC dimension of the network while still allowing the network to perform the desired transformation. This usually results in improved generalization (see section 1.3.3).

Weight sharing among parameters in a set $S = \{\theta_{ij}\}$ can be accomplished by simply aver-

aging the gradient contributions from the various instances of the shared parameter:

$$\Delta\theta_{ij} = -\epsilon\left(\sum_{\theta_{ij}\in S}\frac{\partial C}{\partial\theta_{ij}}\right)/\mid S \mid \qquad (2.10)$$

where C is the cost criterion, ϵ is the learning rate.

2.2 Problems

The back-propagation algorithm is a simple algorithm that is easy to implement and it allows us to train networks that have a great expressive power, i.e., that can be used to perform a very large variety of transformations. However, it also has several weaknesses and problems. The most commonly reported problems are the following:

- Slow convergence: some learning tasks may require hundreds, or even thousands of iterations on the training set in order to reach an acceptable cost. However, for perceptual tasks with a very large training set and a lot of redundancy such as speech recognition or handwritten digits recognition, we found that the required number of training iterations may be as low as a couple of dozen and is usually below 100, if appropriate parameters (such as learning rate and architecture) are chosen for the network and stochastic gradient descent is used.

- Local minima: gradient descent does not guarantee reaching a global optimum because it may get stuck in a local minimum. This seems to happen most often in problems such as XOR or parity, but researchers report that it does not appear to be a big problem [Rume86b], particularly in many real world problems. This is borne out in our experience with speech recognition, at least for static networks, in the sense that many trials of initial weight values don't yield significant variance as to the final performance of the network.

- Parameter setting: the user needs to decide on several parameters, such as learning rate and the architecture of the network, including the number of hidden units. New techniques are being explored to design automatically the architecture of the networks, (e.g., with constructive algorithms [Fahl91] or pruning algorithms [Chau90, Moze89, Karm90]) and automatically control some learning parameters [Jaco88].

2.2.1 Acceleration Techniques

Stochastic Weight Update

There two basic ways to perform gradient descent: *deterministic update* and *stochastic update*. In the first case, also called batch update, parameters are modified after the complete gradient of the cost function has been computed over all the training patterns:

$$\Delta\theta_{ij} = -\epsilon \sum_t \frac{\partial C_t}{\partial \theta_{ij}} \tag{2.11}$$

where C_t is the cost for pattern t, ϵ is the learning rate, and θ_{ij} is a parameter of the network.

Stochastic or online update consists in modifying the parameters after each pattern is presented, using a noisy estimate of the gradient based on the local cost C_t:

$$\Delta^t\theta_{ij} = -\epsilon \frac{\partial C_t}{\partial \theta_{ij}} \tag{2.12}$$

The derivative of the cost for one pattern with respect to the parameters θ_{ij} can be considered a noisy estimate of the total gradient. Stochastic gradient descent has been studied for adaptive signal processing (see for example [Ljun83]) and it converges almost surely under certain conditions [Bott91, Whi91]. For ANNs trained with back-propagation, stochastic update was found to be significantly faster than batch update in many instances, especially in pattern recognition problems with a large training set (such as for speech or handwritten digit recognition) [Bott90]. Several reasons may explain the improvements in convergence time obtained with stochastic update. First, when the training set is large, it may contain redundant information: if several gradient components $\frac{\partial C_t}{\partial \theta_{ij}}$ — contributed from different patterns — are all pointing in a similar direction, then batch update wastes a lot of time only to obtain a more precise direction. Yann Le Cun [LeC89a, LeC89b] suggests the following extreme example to illustrate that notion. Suppose the training set is constructed out of two copies of the same training subset. A batch update method would compute the same gradients twice and then add them, thus performing redundant computations. However, for a stochastic update method, a single iteration on this large training set would simply be equivalent to two iterations on the smaller training subset, thus allowing the network to converge twice as fast.

By modifying the weights more often, stochastic update allows us to "try" many more weight values, making the search less deterministic and more exhaustive (in the vicinity of the current weight values). Another advantage may come from the randomness introduced by the noisy evaluation of the gradient used at each update step (based on very little data). This noise may help escape from local minima. This noise is amplified by the learning rate. Thus, starting with a large learning rate and slowly decreasing it, one might conjecture that we can approach the global minimum of the cost function, in a way that is perhaps similar to simulated annealing (L. Bottou 90, personal communication).

In an experiment on vowel recognition using the TIMIT database [Zue90a], the convergence of batch update and online update were compared. The experiments are performed with a recurrent network with a single hidden layer. The hidden and output units are fully connected to each other. There are also delays of 0 and 3 frames from the input to the hidden units and of 0, 2, 4, 6, and 8 frames from the hidden to the output units. These experiments are performed with the 1990 version of TIMIT on the recognition of 11 classes of vowels (as in the experiment described in section 4.3.1). The networks are trained with back-propagation through time. In Figure 2.2, one clearly sees the impressive difference between the two update methods. On the other hand, [Beck89] compared online and batch training, as well as second order methods for a boolean function problem and found little difference between the two methods.

Adaptable / Local Learning Rate

Many schemes have been proposed to accelerate convergence by 1) adapting the learning rate, and 2) using different rates for different parameters. For example, R. Jacobs [Jaco88] proposes such a method, called delta-bar-delta rule. With this method, one increases the rate associated with a parameter when the current gradient for that parameter has the same sign as the decaying average of previous gradients and decreases this rate if the signs are opposite. However, this method, like several acceleration methods, such as the use of a *momentum* term [Rume86b], rely on exact evaluation of the gradient, i.e., batch update. Consequently, the advantages they bring are often offset by the disadvantages of batch update for complex pattern recognition problems with large training sets.

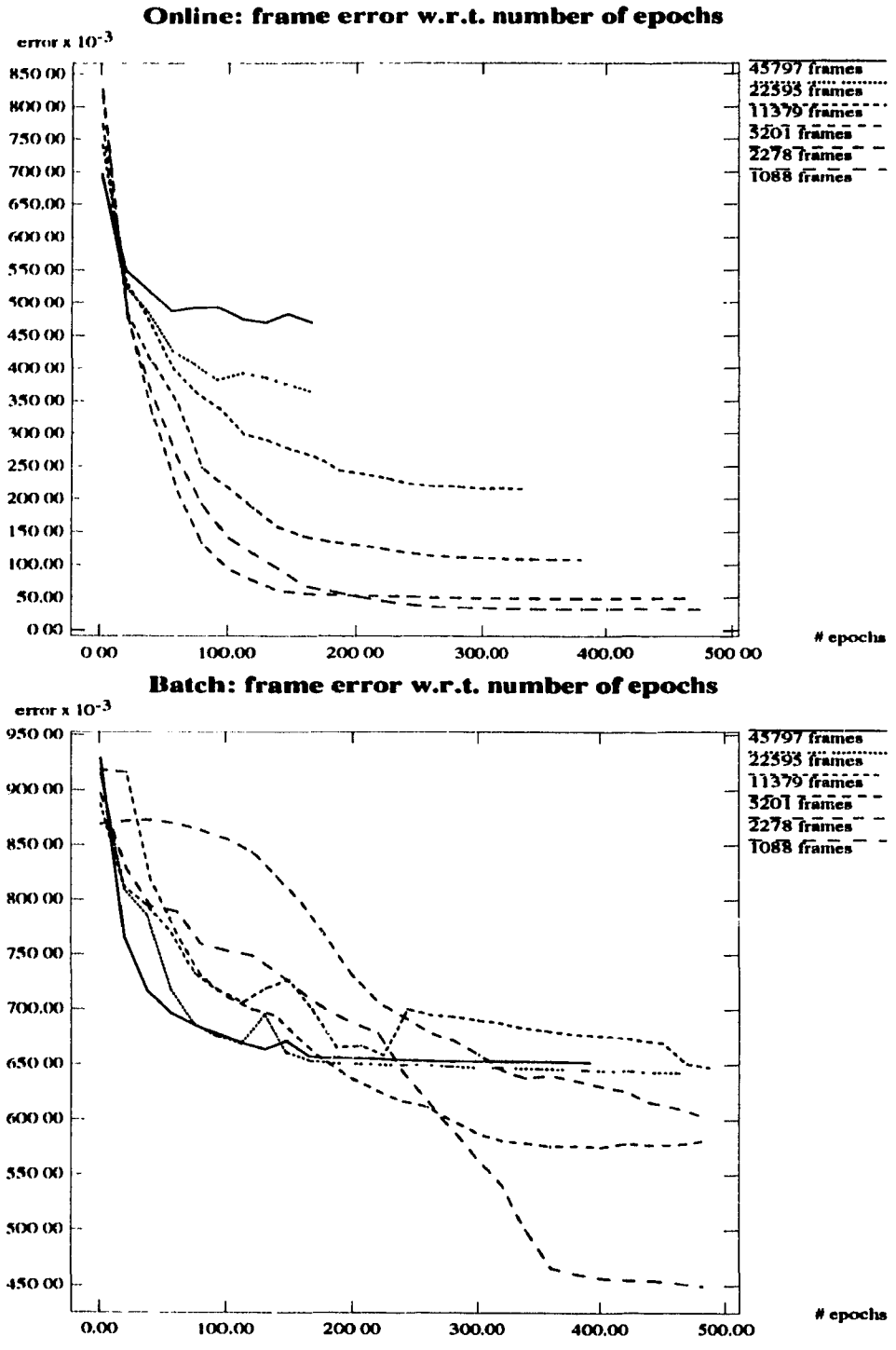


Figure 2.2: Convergence of batch update (a) and online update (b) as the number of training patterns is varied.

An alternative explored in [Ben90a] is to control the learning rate indirectly, by controlling the average weight change in each layer of the ANN. Experiments on small networks showed significantly faster convergence with such an indirect control.

Use of Second Order Information

Several optimization methods using second order information have been explored in order to improve convergence of back-propagation. Already in [Rume86b, Plau86], it was proposed to use a "momentum" term, i.e., add a fixed portion of the previous weight change to the current one. This tends to reduce the zigzagging effect often observed with simple gradient descent, by stabilizing the changes in the direction of *average* steepest descent. Unfortunately, this acceleration due to momentum seems effective mostly with batch rather than online update. The momentum method is actually related to the method of conjugate gradients [Gill81], [Joha90] since in both cases the search direction is obtained through a linear combination of successive gradients. In both cases, second order information is obtained only indirectly and only the first derivatives are explicitly computed. On the other hand, Newton and quasi-Newton methods directly use the second derivatives of the cost with respect to the ANN parameters. Purely Newton methods seem usually impractical in the case of large ANNs because they require computation and storage of the Hessian matrix, which takes $O(N_w^2)$ space and for which inversion requires $O(N_w^3)$ operations (where N_w is the number of weights in the network). The pseudo-Newton method (see [Beck89]) only considers the diagonal of the Hessian, allowing a fast computation and inversion of second derivatives. It also has the advantage that it can be used with stochastic update. However, this method requires careful handling of the problem of near-zero or negative second derivatives. The weight update equation of the Pseudo-Newton method is as follows:

$$\Delta w_{ij} = -\epsilon \frac{\frac{\partial C}{\partial w_{ij}}}{|\frac{\partial^2 C}{\partial w_{ij}^2}| + \mu} \tag{2.13}$$

where C is the cost to be minimized, w_{ij} is a network weight, ϵ is the learning rate, and μ is a parameter used to avoid infinite weight change near zero second derivative values.

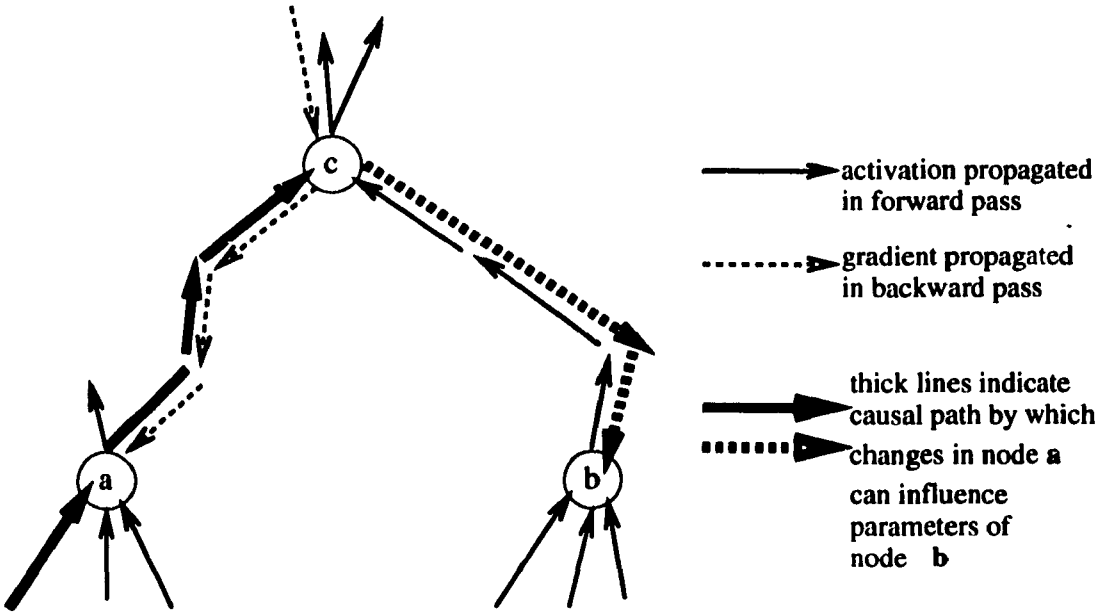


Figure 2.3: Illustration of parameter coupling. Modifying the input weights of node a may influence the gradient to node b and vice-versa, if they are both non-saturating for some pattern.

Parameter Decoupling

One of the reasons why ANN training may be slow is that all the parameters (weights) are strongly coupled. Even nodes that are not directly connected influence each other. Indeed, if a node c is reachable from two nodes a and b in the directed graph of the network, then a change in the function of node a will change the inputs and the behavior of node c. This will influence the gradient ($\frac{\partial C_t}{\partial y_c}$) associated with node c. In turn this may influence the gradient for node b ($\frac{\partial C_t}{\partial y_b}$). If there are some patterns for which neither node a nor node b are saturated (i.e., with a non-zero first derivative of the activation of the node with respect to its input weights), then a change in node a's weights will also influence the input weights of node b (see Figure 2.3).

One of the consequences of this parameter coupling is that it makes the search in the space of those parameters more difficult, i.e., the number of computations grows supralinearly (and maybe exponentially) with the number of parameters (see Section 3.4 on modularization and asymptotic training time). An obvious answer to this problem is modularization. However, this usually implies that the system designer knows enough about the problem

and its solution in order to design modules that can then cooperate together. An imperfect modularization may have the disadvantage that a suboptimal solution is obtained, since parameters are estimated separately, according to different optimization criteria.

What is proposed in the ANN design described later in this thesis is an intermediate solution: first design and train some modules separately (or on top of each other), using a priori knowledge to break up the problem into subtasks. Second, do a *global optimization* of the complete system, by allowing all parameters to be tuned simultaneously in order to minimize a single cost criterion (see chapters 5 and 6).

A different (and maybe complementary) solution is the use of network nodes that have only a *local* response in their input space, such as Radial Basis Functions (RBF) (see [Ben90b] and section 7.1). In that case, the parameter coupling is only between neighboring nodes, i.e., those that have overlapping responses.

Recently, several research groups [Nowl90, Jac91a, Jac91b] proposed an interesting network model, called "Competing Experts" or "Mixtures of Local Experts", which is a kind of hybrid of competitive training and back-propagation, in which a particular type of modularization (over the input space) is performed automatically by the learning algorithm.

Acceleration *vs* Generalization

In some cases, techniques employed to accelerate convergence may have a detrimental effect on generalization of the resulting network. For example Patrick Haffner [Haff89] performed experiments in order to accelerate convergence which often yielded worse generalization error. He compared various forms of the sigmoid which would prevent it saturating and thus slowing down learning. Experiments were also performed with various objective functions, showing there might be a similar trade-off. On the other hand a formula to control learning rates for each unit such that the step size would be reasonably large with no danger of overshooting was used with success. It ensures that the product of the norm of the local gradient and the local learning rate is bounded.

2.2.2 Local Minima

Since gradient descent methods use only locally available information to decide on the next search direction, they may yield *local minima* of the cost function. On the other hand, gradient descent is usually more rapid than methods that are more robust to local minima, such as simulated annealing or genetic algorithms.

Stochastic Weight Update

As was mentioned in section 2.2.1, stochastic weight update may help to escape from some local minima. This may happen because a noisy gradient is used instead of the true error gradient for each parameter update. In fact, a local minimum of the total cost $\sum C_t$ is a stable point of the algorithm only if this point is a local minimum for each C_t [Bott91]. An analogy between stochastic gradient descent (with decreasing learning rate) and simulated annealing is studied in [Bott91].

Simulated Annealing

Instead of allowing only reductions in the cost function at each time step (which is the goal of gradient descent), simulated annealing [Kirk83] allows some upward moves in order to get out of local minima. The probability of making an upward move is controlled by a parameter called "temperature", by analogy with physical systems (cooling metals undergoing an annealing process). Annealing starts with a high temperature that is gradually reduced. If a proper schedule is chosen, the process will converge to the global minimum of the cost function [Gema84].

Although simulated annealing is useful to escape local minima, it does so at a great computational cost when compared with gradient descent.

Genetic Algorithms

An interesting alternative to gradient based optimization methods is the set of optimization methods called "genetic algorithms" (see [Holl75] and [Gold88]). Genetic algorithms are learning algorithms inspired from several features of biological evolution. They consider a population of solutions to a problem, encoded in artificial "chromosomes". Each member of the population is evaluated using an evaluation function. The population undergoes reproduction until a satisfactory performance is attained. During reproduction, one or more "parents" are stochastically chosen to reproduce. This choice favors parents with highest evaluation, i.e., best performance of the evaluation function. Operators are applied to the chromosomes of the parents to produce children that are inserted into the population. Standard operators are mutation, i.e., stochastically modifying one piece of information in a chromosome, and crossover, i.e, combining corresponding pieces of information from two parent chromosomes. Domain knowledge can be exploited to create operators which improve the efficiency of the optimization procedure [Whi89a].

It should be noted that interesting gradient-descent/genetic-algorithm hybrids have been proposed (see for example [Davi89] or [Whi89a]).

One of the advantages of genetic algorithms is that they are very resistant to local minima, because they explore several points in parameter space simultaneously. Unfortunately, this is at the cost of many evaluations of the error function (at all these points, each representing a different individual). Hence in general, if gradient descent can lead to the global minimum, it will probably do so faster than genetic algorithms or simulated annealing.

Chapter 3

Integrating Domain Knowledge and Learning from Examples

Connectionist models are very flexible in the design of architectures based on domain knowledge and their integration with other problem solving tools. It is possible to use domain knowledge in many aspects of network design and thus to take profit of a priori knowledge, as well as of learning from examples. Such knowledge may be used to impose constraints on the network topology, on input and output, or on initial values for some of the system parameters. These constraints, by reducing the VC dimension of the ANN, while still allowing a good solution to be attained, may significantly improve the resulting generalization of the ANN. However, in many cases, it is not easy to use that knowledge, thus the design of ANNs also requires careful crafting and trial and error. The aspects of the design process considered in this chapter are the following: preprocessing, input coding, architecture constraints, modularization, and output coding. Experiments on speech recognition tasks are described to illustrate the importance of each of these aspects of the design of ANNs. To modularize the recognition system and to take advantage of knowledge specific to each type of phoneme (such as plosives, vowels, fricatives), we have generally broken down the phoneme recognition into networks specialized for particular tasks.

3.1 Importance of Preprocessing Input Data

Preprocessing the input data for an ANN has appeared very important in many applications. Considering, for example, automatic speech recognition, our work (see [Ben90a, Ben90c]) as well as the results of others [Ross89] indicate that the choice of signal processing significantly influences the performance of a recognition system. It is important to distinguish between the choice of preprocessing, e.g., acoustic features, from the way these input features are coded before being applied at the input layer of a network (see Section 3.2). We will review in the following subsections some preprocessing techniques for automatic speech recognition.

3.1.1 Auditory Model

In the human auditory system, air pressure variations enter the outer ear and are mechanically transduced through the middle ear to the cochlea (inner ear). These pressure variations from the middle ear reach the cochlea through the oval window near the base of the cochlea, that is filled with a fluid called perilymph. The pressure variations provoke fluid displacements (traveling waves) within the cochlea. These waves appear to have maximum intensity along the length of the basilar membrane, within the cochlea, as a function of the frequency distribution of the acoustic stimuli. On the basilar membrane, about 1500 inner hair cells perform the neural transduction of the acoustic stimuli. When there are vibrations of the basilar membrane near a particular hair cell, its cilia are deflected, provoking an intracellular voltage increase that propagates to the cell body.

In recent years, basilar membrane, inner cell and nerve fiber behavior have been extensively studied by auditory physiologists, and knowledge about the auditory pathway has become more accurate. Through a number of studies, considerable amount of data has been gathered in order to characterize the response of nerve fibers in the eighth nerve of the mammalian auditory system, using simple tones, tone complexes, and synthetic speech stimuli [Delg80, Delg84, Youn79, Sach80, Mill83, Sine83, Kian65]. These studies have been the basis for the development of a mathematical ear model by S. Seneff [Sene84].

The experiments described in this section with an ear model are based on the model pro-

posed by Seneff [Sene84, Sene85, Sene86, Sene88b]. This model is structured into three blocks. The first two blocks are designed using knowledge of the rather well-known responses of the corresponding human auditory stages [Sine83, Kian65]. The third one attempts to extract important speech properties like spectral lines related to formants [Sene84, Sene85].

The speech signal is first sampled at 16 kHz, pre-filtered through a set of four complex zero pairs to eliminate very low and very high frequency components. It is then analyzed, in the first block of the model, by a 40-channel critical-band filter bank. For the experiments described here, frequencies and bandwidths for zeros and poles of each filter were designed almost automatically by P. Cosi [Cosi90] with an interactive technique developed by S. Seneff [Sene85].

The second block of the model implements a non-linear mapping that is intended to capture prominent features of the transformation from basilar membrane vibration, represented by the outputs of the filter bank, to probabilistic response properties of auditory nerve fibers. The outputs of the second block represent the probability of firing for a set of similar fibers acting as a group. This block models four neural mechanisms. The first one corresponds to transduction and is modeled with a half-wave rectifier. The rectified signal is then processed by a module that models neurotransmitter release for the synapse between the inner hair cell and its connected nerve fiber. A membrane model performs short-term adaptation at that synapse [Gold85]. Then a low-pass filter models the loss of synchrony in nerve fiber behavior as stimulus frequency is increased. Finally, an automatic gain control unit models the refractory phenomenon of nerve fibers.

The third block of the auditory model is a synchrony detector, which implements the known "phase locking" property of the nerve fibers responding to low frequency periodic stimuli. It enhances spectral peaks (or *spectral lines*) due to vocal tract resonances. If there is a dominant periodicity in the signal, those channels whose central frequencies are closest to that periodicity will have a more prominent response.

See [Cosi90, Ben89b] for a more detailed description of the auditory model we used and of the experiments performed with this model. Unfortunately, our software implementation of the auditory model was prohibitively slow: several hundred times real time. Thus our

experiments with the auditory model were limited to those described in this section, section 3.3.1 and section 3.5. However, recent work by [Zue90b] indicates that it is possible with a parallel implementation of the auditory model to get performances on the order of 3 times real time with 16 DSP32C processors.

Comparison with the Fast Fourier Transform

Experiments were performed in speaker-independent recognition of 10 English vowels extracted from isolated words [Ben89b, Cosi90]. The Fast Fourier Transform (FFT) and the auditory model were compared as alternative preprocessors for a neural network recognizer. When using 40 coefficients produced by an FFT-based Mel-scale[1] 40-channel filter bank, generalization error [2] (with new speakers) was 13.0%. On the other hand, the auditory model yielded 4.6% generalization error on that task. However, as mentioned above, the preprocessing time for the auditory model required two orders of magnitude more processing time than for the FFT. Errors were on a phoneme per phoneme basis and the recognized phoneme was chosen based on Euclidean distance between the target outputs corresponding to each phoneme and the network outputs.

The speech material consisted of 5 pronunciations of 10 monosyllabic isolated words containing 10 vowels of American English:

{BEEP (/iy/), PIT (/ih/), BED (/eh/), BAT (/ae/), BUT (/ah/), BOOT (/uw/), PUT (/uh/), SAW (/ao/), FAR (/aa/), FUR (/er/)}.

Vowels were automatically singled out with an algorithm proposed in [DeMo85a] and a linear interpolation procedure was used to reduce the number of frames per vowel to ten. The resulting 400 spectral coefficients were the inputs of the ANN (40 spectral coefficients per frame × 10 frames). The voices of 13 speakers (7 male, 6 female) were used for learning, with 5 samples per vowel per speaker. The voices of seven new speakers (3 male, 4 female) with 5 samples per vowel per speaker were used for testing generalization.

[1]The Mel-scale compresses high frequencies logarithmically, like the Bark scale described in the next section.

[2]Generalization error is the error on an independent test set. Relative generalization is the difference between the error on the training set and the error on the test set

3.1.2 Bark Scale, Second Order Gradient and Other Features

The FFT is a widely used transformation from which acoustic parameters are computed. This transformation can be computed with high speed but it produces a linear frequency scale (such that frequency resolution does not depend on frequency). In contrast, in the auditory system many more inner hair cells respond to lower frequencies than to higher frequencies. A perceptual measure of frequency resolution, called the Bark scale, was defined [Zwic80] and can be used for speech analysis. It can be approximated by the following relations:

$$\begin{aligned} f < 500 \text{ Hz} &\rightarrow B(f) = 0.01f \\ f < 1220 \text{ Hz} &\rightarrow B(f) = 0.007f \\ f \geq 1220 \text{ Hz} &\rightarrow B(f) = 6 \log f - 32.6 \end{aligned} \tag{3.1}$$

where f represents the actual frequency and $B(f)$ represents the Bark scale transformation approximating human perception. Note that it is basically linear below 1200 Hz and then compresses the high frequencies logarithmically. This transformation was used for the design of the filters used to preprocess the speech signal in the experiments described in sections 3.3.2 and 3.5.2. In these experiments, the signal $S(t)$ was first pre-emphasized in order to counteract the spectral falloff due to the glottal source in voiced speech, according to the following formula [OSha87]:

$$S_{pre}(t) = S(t) - \alpha S(t-1) \tag{3.2}$$

This is a single-zero high-pass filter which may yield formants with similar amplitudes. A value of 0.95 was used for α. Afterwards, a 512-point FFT was computed every 5 ms with 20 ms of the Hamming windowed (and pre-emphasized) signal. The power spectrum computed with the FFT was then smoothed with 32 masking filters. This preprocessing was used on the TIMIT database [Zue90a], for which the speech signal was sampled at 16 kHz. The overlapping triangular filters were equally spaced in the Bark scale described above, with center frequencies ranging from 100 Hz to 7000 Hz [Ben91a].

In addition to the Bark scale spectrum, several other features were considered in order to help recognition. An approximation of a second order partial derivative of the spectral en-

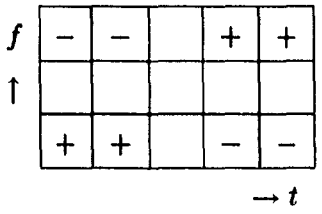


Figure 3.1: Sliding window used for detecting movements of spectral lines. Frequency is on the vertical axis and time on the horizontal axis. Each grid element is 0.5 Bark by 5 ms.

ergy with respect to frequency and time is computed as in [Flam91], which is a modification of the feature proposed by [Stev75]. These feature are obtained by sliding a 20ms by 1.5 Bark window as shown in Figure 3.1 on the peaks of the time/frequency spectrogram. In this way, 30 property detectors are introduced which have strong positive response for spectral lines increasing in frequency and strong negative response for spectral lines decreasing in frequency. Hence these detectors encode information about the *movement* of spectral peaks of high energy, both in time and in frequency. Let us denote by $X(f,t)$ the spectral energy in a frequency band centered at f and in a time interval centered at t. The gradient operator is computed only for the spectral peaks between 300 and 4000 Hz. First, all the local peaks are located in that frequency band. To reduce the number of spurious responses, this window is only applied at *peaks* of the Bark spectra. Second, $G(f,t)$ is computed as in [Flam91]:

$$G(f,t) = \begin{cases} X(f-1,t-2) + X(f-1,t-1) + X(f+1,t+1) + X(f+1,t+2) \\ -X(f+1,t-2) - X(f+1,t-1) - X(f-1,t+1) - X(f-1,t+2) \\ \qquad \text{if } E(t) > \mathit{threshold} \text{ and } X(f,t) \text{ is a spectral peak} \\ 0 \qquad \text{otherwise} \end{cases} \tag{3.3}$$

where $E(t)$ is the total energy of the signal in the window t, and the *threshold* discriminates between speech sounds and silence. In the experiments performed with this approximation of the derivative with respect to time and frequency, the gradient $G(f,t)$ is then smoothed by averaging over the nine neighbors of the point (f,t) in the spectrogram.

Another feature derived from the spectrogram is the spectral slope, which describes the

gross shape of the spectrum with an approximation of the first derivative of spectral energy with respect to frequency. Seven slope features were computed as in [Klat82]. The spectral power in the range 50-600 Hz is a good indicator of degree of voicing. This feature and its time derivative were used to help discriminate among voiced and unvoiced phonemes. The signal energy and its time derivative are very useful features found in many speech recognition systems (e.g., SPHINX, [Lee 89]). Another time-domain feature that was found useful is spectral dissimilarity [Fant73], a normalized dot product of consecutive spectral frames that helps detect abrupt changes in the signal.

All of these features were normalized in order to span the range -0.5 to +0.5. Values outside this range were clipped. This allowed us to use an efficient 1-byte/datum representation of the preprocessed database, as well as making the mean and variance of each input feature of the same order of magnitude. The lower and upper bounds were chosen to clip about 5% of the data.

Although all these features are highly correlated, experiments performed by Giovanni Flammia [Ben91a, Flam91] show that they can improve performance of an ANN designed for the recognition of plosive and nasal sounds in the TIMIT [Zue90a] database (continuous speech, speaker-independent). Many of these features are also motivated by speech knowledge. For example, the evolution of high frequency spectral lines between the burst of a plosive and the consecutive phoneme are important acoustic cues for the recognition of the place of articulation of stop consonants [Stev75, Blum79].

3.2 Input Coding

The experiments described in [Beng88, Ben90a] indicate that the way in which input features are represented in the activations of the input layer influence generalization. In these experiments, a network was trained to discriminate among three classes of vowels according to their horizontal place of articulation (front, center or back). The speech data was preprocessed as follows. Vowels were automatically [DeMo85a] extracted from isolated words pronounced by multiple speakers. Spectral lines were extracted from the spectrogram of the signal with an algorithm described in [DeMo85b] and based on thinning and on skele-

tonization of the spectrogram, which is treated like an image. In the steady-state part of the vowel, average values for frequency and energy are computed for each detected spectral line. Thus the task is one of static classification. The input data for the neural network is a set of energy/frequency pairs for the detected spectral lines.

The best results were obtained with the following coding scheme. The input units were organized into a two-dimensional grid of energy and frequency. Frequency intervals were divided according to a Bark scale and energy intervals were divided according to the observed energy distribution. Frequencies were normalized by subtracting the frequency of the lowest frequency spectral line[3] from the true observed frequencies of the other spectral lines. The number of training examples (72) was particularly small. To improve generalization, a coarse coding scheme was carried out: in addition to exciting one node for each energy/frequency input pair, neighboring nodes also received an input, with intensity decreasing with distance in the grid. The generalization error with a Boltzmann machine was 4.2%. If absolute instead of relative frequencies were used, the error increased to 5.6%. If the energy was coded with a single analog value for each frequency interval[4], the generalization error increased to 8.3%. If instead the coarse coding scheme was not used[5], the generalization error was 9.7%. Slightly worse results were obtained with a feedforward network trained with the back-propagation algorithm [Ben90a].

A surprising result of these experiments is that the ANN generalization was very good, although the number of training examples was very small compared to the number of weights in any of the networks (several tens of thousands). We conjecture this may be due to the input coding, as well as to the fact that training was stopped well before the global minimum of the error function was reached, thus probably limiting the effective VC dimension of the ANN (see [Chau90] and [Mor90b] on stopping short of convergence to improve generalization).

---

[3] This low frequency spectral line should correspond to the fundamental frequency or pitch

[4] For that experiment, the frequency resolution was increased so as to keep a similar number of weights in the network.

[5] In that case each energy/frequency pair corresponded to only one ON input node

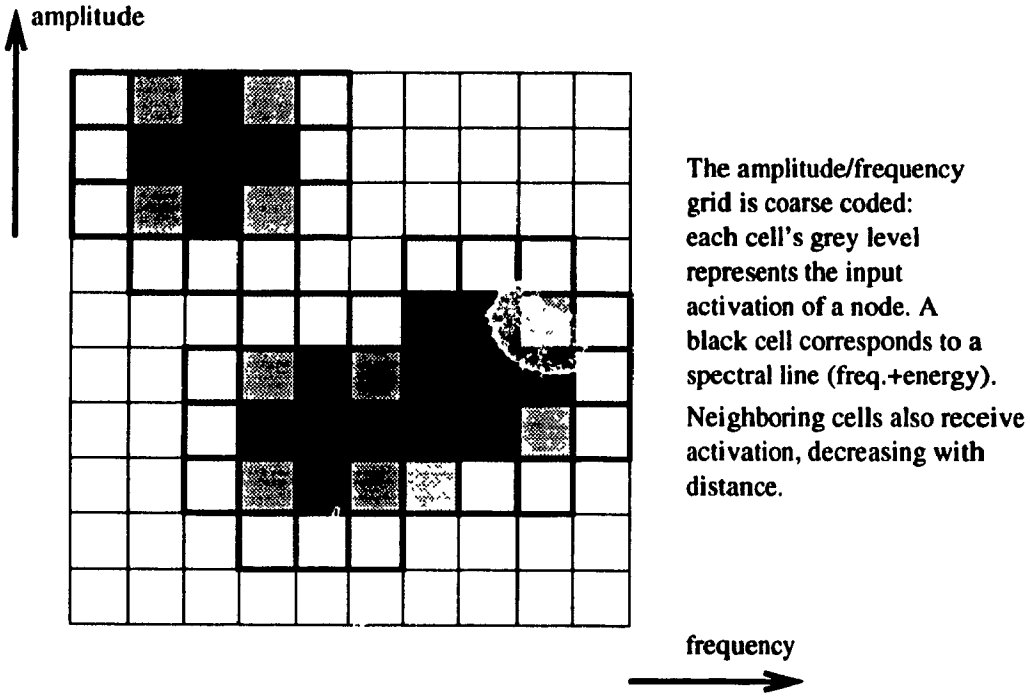


Figure 3.2: Coarse coding scheme with which best results were obtained for the experiments described in section 3.2 (vowel classification). The input grid represents frequency (horizontal axis) and energy (vertical axis).

3.3 Importance of Architecture Constraints on the Network

Many experiments showed that the design of the architecture of an ANN has a significant influence on its performance, regarding both generalization and convergence time. Several experiments in this section illustrate this effect. Other experiments described in this thesis also show the importance of architecture choices: see, for example, the experiments on vowel recognition in TIMIT (section 4.3.1), the experiment on the analysis of amino-acid sequences (section 4.4.2), as well as the experiments on the ANN/HMM and ANN/dynamic programming hybrids (section 6.7).

3.3.1 A Nasal Consonant Discrimination Experiment

In a nasal consonant recognition experiment [Ben90c], a more than 5-fold improvement in generalization was observed by improving the architecture of the ANN. The experiment was performed on the discrimination of nasals /m/ and /n/ in a fixed context, that of letters "m" and "n". The speech material consisted of 294 speech segments from 70 training speakers (male and female with various accents) and 38 speech segments from 10 test speakers. The speech signal was preprocessed with the auditory model and general synchrony detector described in section 3.1.1, yielding 40 input features every 10 ms. Poor results were obtained with early experiments, with a simple output coding with three nodes {vowel, m, n}. A two-layer fully connected feedforward ANN with a window of two consecutive frames at the input and 10 hidden units yielded 15% classification error on the test set.

Better results were obtained by considering observations on speech analysis showing that the most important discriminatory information for the nasal sounds is available during the transition between the vowel and the nasal. This suggested using the following output coding, with four nodes:

{vowel, transition tc m, transition to n, nasal}.

Since the transition was more important than the steady-state, we chose a window of 4

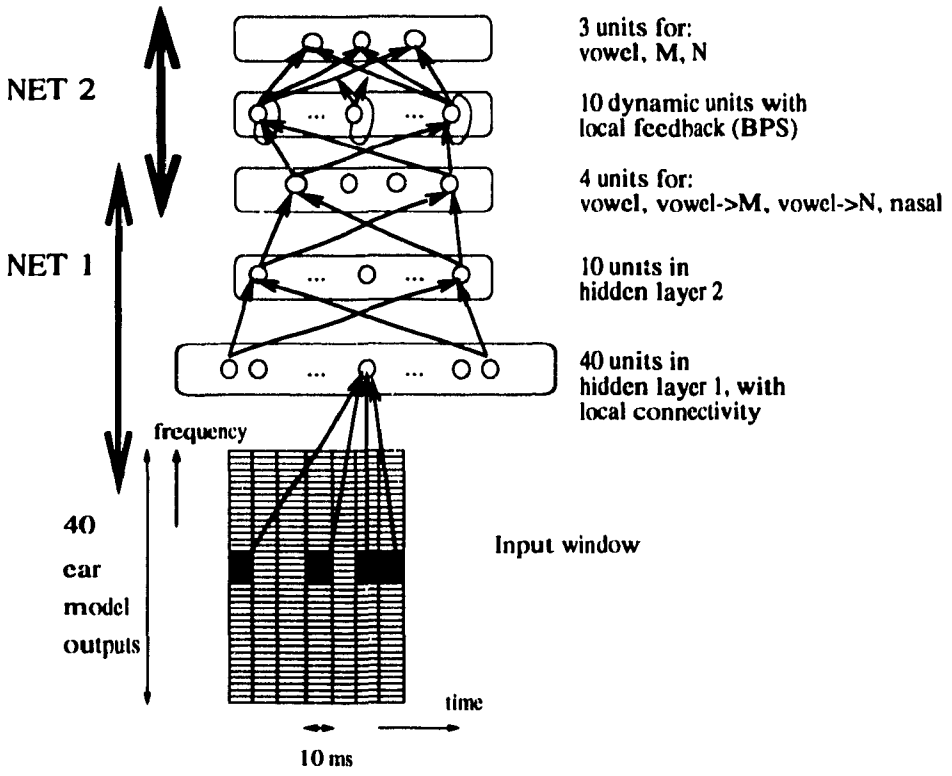


Figure 3.3: Best architecture obtained for the recognition of nasal sounds in a fixed context (see section 3.3.1). The first network is initially trained to recognize transitions from vowel to nasal. The second one models the temporal structure of the output of the first one.

frames (instead of 2 frames) at (t, t - 10 ms, t - 30 ms, t - 70 ms) at the input. To reduce the connectivity in the network, the architecture was modified to include a constrained first hidden layer with 40 units, where each unit was meant to correspond to one of the 40 spectral frequencies of the preprocessing stage. Each such hidden unit, associated to the F^{th} output coefficient of the auditory model synchrony detector, was connected (when possible) to input units corresponding to

auditory model coefficients $(F-2, F-1, F, F+1, F+2)$

and frames $(t,\ t-10\ ms,\ t-30\ ms,\ t-70\ ms)$.

Experiments with the feedforward network (40 inputs - 40 hidden - 10 hidden - 4 outputs)

(network 1 in figure 3.3) showed that, as expected, the strongest clues about the identity of the nasal sound are those associated with the transition from vowel to nasal. Furthermore, this information is available for only a very short segment of time, just before the start of the steady part of the nasal. To extract this critical information, a second network was trained on the outputs of the first one to provide a clear discrimination during the whole duration of the nasal. This higher level network is a recurrent one with local feedback trained with the BPS algorithm (see section 4.3.3), in order to learn about the temporal structure of the task and keep the detected critical information during the length of the nasal. With the 2-network architecture as shown in Figure 3.3, classification performance reached a plateau of 1.1% classification errors on the training set. Generalization was very good for this task, with only a 2.6% error rate on the test set. The dramatic improvement due to the change in architecture may be enhanced by the small size of the training set. In such cases, the structural bias imposed on the network has much more effect than when the training set is very large.

3.3.2 Plosive Recognition

Experiments on plosive recognition on a continuous speech database (TIMIT, [Zue90a]) were performed in collaboration with G. Flammia [Ben91a]. The task was the speaker-independent recognition of the following 7 plosive sounds in continuous speech:

{/p/,/t/,/k/,/b/,/d/,/g/,/dx/}

The best results for this task were obtained with a network with three layers of weights in which the first hidden layer was constrained with a local connectivity both in time and frequency, as shown in Figure 3.4. A recurrent network performed better than a static one [Flam91]. The recurrent network yielded a decrease of generalization error from 35% to 30.7% on a plosive and nasal recognition task with the TIMIT database and its outputs were much less noisy[6]. The input features are those described in section 3.1.2 and the

[6] based on visual inspection of the network outputs

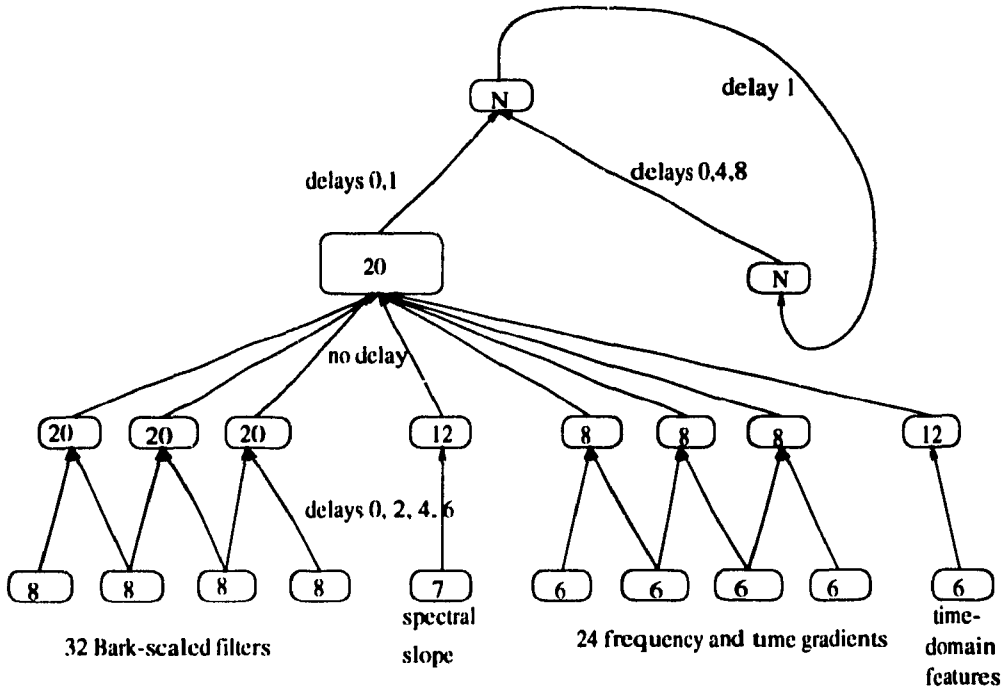


Figure 3.4: Architecture used for the recognition of plosives, nasals and fricatives on the TIMIT database. The first layer has a local connectivity in time and frequency. N is the number of outputs.

output coding, based on articulatory features and representation of context, is described in section 3.5.1.

3.3.3 Position-Invariant Low-Level Features

Several years of research on the application of ANNs trained with back-propagation to handwritten character recognition by Yann Le Cun et al has yielded a highly constrained architecture for this task [LeC89c]. The network has 4 layers of weights. The first hidden layer units have a local receptive field and they are organized in *groups* (he calls *kernels*) of units. All units within a group have the same receptive field and corresponding units in different groups *share weights* (see section 2.1.3). Because of weight sharing the ANN has only 9760 free parameters even though there are 64,660 connections. These constraints are based on a priori assumptions about the nature of the task and its solution: low-level features should be local in space and position-invariant. They yield significantly improved

generalization [LeC89a].

3.4 Modularization

Theoretical analysis indicates that the learning problem for a neural network may be NP-complete [Judd88]. On the other hand, empirical estimations [Hint87] indicate that the learning time on a serial machine is approximately $O(N_w^3)$, where N_w is the number of weights in the network. Thus, significant improvements in learning time can probably be attained by designing a modular system, applying the "divide and conquer" principle.

A modular system can be organized with modules in parallel and/or in series. For example, for the system described in chapter 6, multiple ANNs are specialized to particular types of phonemes or phonetic features. Thus the first level of the system consists of a set of different networks operating in parallel. Each network on the first level computes different types of phonetic features, thus transforming the speech signal into a form that is less speaker-dependent. The operation of the second level is equivalent to a matrix multiplication (a linear layer) that compresses the data from all the outputs of the first level modules in order to feed it to a Hidden Markov Model (HMM), which constitutes the third level.

Waibel et al [Waib89] reported improvements in convergence time for a modular speaker-dependent phoneme recognition system, using Time-Delay Neural Networks (TDNNs, see section 4.2). In this system, phoneme recognition is performed with specialized networks for phoneme subclasses. To merge the individually trained networks, "glue units" are introduced and added to the concatenated subnetworks in order to learn discriminations not learned by any of the individual subnetworks. For example, in the case of plosive recognition, a {/p/,/t/,/k/} network is merged with a {/b/,/d/,/g/} network and the glue units learn about the voiced *vs* unvoiced discrimination that separates {/p/,/t/,/k/} from {/b/,/d/,/g/} while the two subnetworks remain fixed. Finally, with a global tuning of all the parameters, further improvement of the performance of the {/p/,/t/,/k/,/b/,/d/,/g/} network is obtained.

3.4.1 Specialized Networks with Specialized Preprocessing

The modular system used in the ANN/HMM hybrid of Chapter 6 is based on the idea of using specialized networks with specialized preprocessing. Each ANN performs discrimination among some phonemes or phonetic classes while the HMM allows an efficient modeling of the temporal structure of the signal.

We first presented the idea of using specialized networks with specialized preprocessing in [Ben89a]. For the experiments described in [Ben89a], several networks are used depending on the acoustic situation detected by an automatic segmentation program [DeMo85a]. For example, the network MLN1 is executed when a non-sonorant interval is found, followed by a sonorant interval. On the other hand, the network MLN2 is executed in situations characterized by peaks and high energy valleys of the signal energy in which frication noise has not been detected. Each network input is designed for the particular type of spectra that is expected in the corresponding situation. The MLN1 network has 5 input windows, each of which uses a different frequency and time resolution. For example, the first window of MLN1 has two time intervals of 30 ms before the non-sonorant to sonorant interval, and four time intervals of 10 ms after that transition. For that window, the analysis is based on 4 low frequency (100 Hz to 500 Hz) filters. Each filter receives as input a normalized value of the energy in its time-frequency window. For each filter, there is a corresponding input that is activated if a spectral line was detected (using the algorithms described in [Merl86]) inside the associated time-frequency window. The experimental task was the discrimination of the following ten often confused words (nine letters of the alphabet and one digit), related to the often used "E-set":

{"b","c","d","e","g","k","p","t","v","three"}

The experiments were performed with two pronunciations of each word by 80 speakers (40 males, 40 females). The data from 70 speakers were used as a training set while the data from the remaining 10 were used for the test.

An overall error rate of 9.5% was obtained, with a maximum error of 20% for the letter "d". This result was much better than those previously obtained in [DeMo87], and comparable to those obtained by [Bahl88] working only with male speakers on nine letters.

Most of the errors represent cases that are difficult even in human perception. Such cases are confusions "b"→"e" and "d"→"e" representing a low evidence of burst and formant transitions in voiced plosives, confusions "b"→"v", "v"→"b", "d"→"b", "p"→"t", "t"→"p", "t"→"k" indicating wrong estimation of the place of articulation, and confusions "d"→"t", "p"→"b","e"→"b", indicating errors in the characterization of voicing.

3.5 Output Coding

Most ANNs designed for phoneme recognition have a simple output coding scheme consisting of one output unit per phoneme, with a high target for the output unit corresponding to the target phoneme and a low target for the other units (see for example, the previously mentioned work of [Waib89]). In general, one can interpret the output activations of the network as representing degrees of evidence. Using phonetic knowledge, we have explored coding schemes based on phonetic features related to speech production. Examples of such features are horizontal and vertical place of articulation, voicing and nasality. Such a representation is in general more compact than the "one-output-per-phoneme" representation. Furthermore, it describes a more general space of phonetic characteristics, allowing a network trained with some phonemes to generalize to new phonemes.

3.5.1 Articulatory Features for Vowels

In [Ben89b, Cosi90], we describe experiments on the recognition of horizontal and vertical place of articulation for vowels. We found that these features were more difficult to learn than a "one-output-per-phoneme" coding but yielded better generalization, especially for *new sounds* that were not in the training set. Three separate networks were trained to recognize horizontal place of articulation (with 5 outputs corresponding to 5 places), vertical place of articulation (also called manner of articulation, with 5 outputs corresponding to 5 manners), and tenseness (with 2 outputs corresponding to Tense and Lax vowels). Their generalization errors were 4.6%, 5.7% and 5.4%, respectively. These experiments were performed with the speech material and the preprocessing described in section 3.1.1.

To interpret the network outputs, a weighted center of gravity for each of the three features was computed. An interesting generalization experiment was performed as follows. Vowels and diphthongs not in the training set were modeled according to expected evolution of the above mentioned articulatory features when they are pronounced, based on descriptions from theory or past experience and not learned by actual examples. Although HMMs could have been conceived for modeling the time evolution of the articulatory features, a simpler classification method was applied in these experiments. Each new vowel or diphthong model was simply described by a regular expression using symbols corresponding to different quantized values of the weighted centers of gravity (e.g., strong front, weak front, central, weak back, strong back) for the three features. Center of gravity is defined as follows:

$$CG = \frac{\sum_i i\, y_i}{\sum_i y_i} \tag{3.4}$$

where y_i is the i^{th} output corresponding to a certain articulatory feature, such as horizontal place of articulation. The centers of gravity for horizontal and vertical place of articulation were then quantized [Cosi90] using 5 symbols from the following alphabets:

$$\begin{aligned} A_H &= \{F, f, C, b, B\} \\ A_V &= \{H, h, M, l, L\} \end{aligned} \tag{3.5}$$

where F represents strong front, f, weak front, C, central, b, weak back, and B, strong back. Similarly, for vertical place of articulation, H represents strong high, h, weak high, M, medium, l, weak low, and L represents strong low.

The following regular expressions were used to characterize the words containing the new vowels and diphthong:

Letter A of the alphabet :	$(f,h)*(F,H)*$
Letter I of the alphabet :	$(b+C,l)*(f+F,h+H)*$
Letter O of the alphabet :	$(b+B,l)*(b+B,h+H)*$
Diphthong /oy/ :	$(b+B,l)*(f+F,h+H)*$
Diphthong /aw/ :	$(C,l)*(b+B,h+H)*$
Letter U of the alphabet :	$(f+F,h+H)*(b+B,h+H)*$
Letter Y of the alphabet :	$(b+B,h+H)*(C,l+L)*(f+F,h+H)*$

(3.6)

A 2-tuple represents a particular horizontal and vertical place of articulation. The asterisk means "at least 1 repetition", the symbol + here means logical disjunction while a concatenation of terms between parentheses means a sequence in time. A short sequence with intermediate symbols was tolerated in transitions $B \rightarrow F$, $L \rightarrow H$ and vice versa, e.g., for the letter U, a transition of horizontal place of articulation through the value corresponding to the symbol C is tolerated when going from f or F to b or B.

Recognition based on these models yielded 7.5% generalization error, thus providing evidence that the ANN trained to recognize normalized values of horizontal and vertical place of articulation reliably generates feature hypotheses about vowels and diphthong not used for training.

Although very good results were obtained with articulatory features for vowels from isolated words, as described in the previous paragraphs, such good performance was not found in the case of continuous speech. Experiments described in section 4.3.1 with the TIMIT database yielded better performance with the simpler "one-output-per-class" encoding scheme. This may be because the target articulatory values that were used assume well-pronounced vowels in isolated words whereas in continuous speech there is more variation and influence from the context. This is an example of having weak knowledge about the problem. Better performance may be obtained if target articulatory features were used that take the effect of coarticulation into account.

3.5.2 Articulatory Features for Consonants and Representation of Context

Plosive and Nasal Sounds

Experiments on some classes of phonemes (plosives, nasals, fricatives) were performed with the preprocessing described in section 3.1.2. Various output coding schemes were compared in [Ben91a]. It was found that a "one-output-per-phoneme" coding was worse than using horizontal place, vertical place and voicing of the current phoneme. Furthermore, using context-dependent output units improved performance even more. As described in

[Flam91], on the recognition of the 10 classes of plosives and nasals /p,t,k,b,d,g,dx,m,n,ng/, generalization error was 36.4% with one output node per phoneme, but only 28.7% by coding with 7 nodes representing the following phonetic features: labial, alveolar, velar, flap, voiced, stop, nasal.

By adding context-dependent output nodes, generalization error was further reduced to 27%. For each of 4 horizontal places of articulation of plosives, 3 right contexts were considered (front vowel, back vowel, non-vowel). In addition to these 12 nodes for horizontal place of articulation, 2 nodes indicated voiced or unvoiced phoneme, and 6 nodes were used for broad classification of phonemes (liquid, front vocalic, non-front vocalic, nasal, fricative, silence).

The above experiments were performed with a 32-filter Bark-scale spectrum as described in section 3.1.2. By adding the other features described in section 3.1.2. (24 second-derivative detectors and 7 time-domain parameters, including spectral dissimilarity), the error was further reduced to 24.9%.

Fricative Sounds

Experiments on the recognition of 11 fricative classes from the TIMIT database were performed, using the architecture shown in Figure 3.1, preprocessing described in Section 3.1.2, and output coding based on place and manner of articulation of fricatives. The following 11 classes were to be recognized:

/s,z,ch,th,f,sh,zh,jh,dh,v,hh+hv/

The network had 69 inputs: 32 Bark-scale spectrum, 7 spectral slope, 24 spectrum gradient (time and frequency derivative), and 6 time-domain measurements (energies and their derivatives, zero crossing). The 12 output nodes represented the following features: fricative, plosive, nasal, liquid, labial, dental, alveolar, affricate, palatal, glottal, voiced, silence. The plosive, nasal, liquid and silence are used to allow the network to model the rejection class (non-fricatives).

The architecture of the network for fricative recognition is sketched in figure 3.1. It is

a recurrent network with delays and 9224 weights. The network was trained with back-propagation in time (see section 4.3.1). After 34 training iterations, training was stopped. The training was performed on SI and SX sentences from 343 speakers from TIMIT. The test was performed on 77 new speakers of TIMIT [7]. The performance was 24.5% frame errors (on fricatives) on the training set, and 25.1% frame errors on the test set (with different speakers). Insertion of fricatives in non-fricative sounds was 8.2% for the training set and 8.5% for the test set (on a frame-by-frame basis). Note that the relative generalization is excellent (relative difference between the training and test set errors).

The most common errors were the following: voicing (/z/ → /s/, /v/ → /f/), place of articulation (/jh/ → /c/, /th/ → /f/ (less frequent), /zh/ → /sh/ (less frequent)).

3.5.3 Modulating the Targets

As it will be explained in section 4.3.5, supervision can be modulated. In section 4.3.5 we consider a simple boolean operator that controls whether any supervision is provided for a given frame and output unit (see also [Jord88]). Another way to modulate the supervision is suggested in [Kuh90a]. It uses knowledge about the relation between the shape of the signal energy and the onset of vowels to provide accurate supervision to network outputs. In that application (recognition of letters of the alphabet "b", "d", "e" or "v"), the evolution of the target output follows an acoustic feature $r(t)$ which is a shape-dependent energy increase indicator, turning on at times when the vowel turns on.

[7] The training speakers were those with initial between "a" and "r" inclusively and the remaining were used for testing.

Chapter 4

Sequence Analysis

4.1 Problem Definition

Most applications of ANNs to pattern recognition are concerned with classification of static patterns, i.e., the problem is to map input patterns (of fixed size) to an appropriate class. However, there exist several interesting applications in which the input pattern is better described as an ordered *sequence* of fixed-size sub-patterns. The sequences do not necessarily have fixed length, thus methods for static pattern recognition are not easily applicable to those problems. Let us call each of the elementary fixed-size patterns that compose a sequence a *frame*. For some problems, it is required to produce a single classification for the entire sequence. For many other problems, such as continuous speech phoneme recognition, it is instead required to *label* each frame, i.e., to associate it to a particular class or state. Without loss of generality, let us associate the ordering of the frames in the sequence to the flow of *time*, allowing us to talk about past and future times as well as *delays* among frames. This association has also the advantage of relating to the way in which the information is processed for such tasks, i.e., sequentially in time.

A simple approach to this type of problem is to assume that the classification associated to the frame t depends mostly on the surrounding frames, for example in some interval $[t - L, t + R]$ of the sequence. In that case, one can use an input *window* of fixed size $R + L + 1$ to scan the input sequence and produce a classification for each frame using a

static classifier. For example, this is the approach that was employed in NetTalk [Sejn86], in which an ANN maps orthographic descriptions of text into phonetic features used for speech synthesis (in a text-to-speech system). In that case the window is 7 frames wide, with each input frame describing one letter of the alphabet or a punctuation mark. This type of ANN was also used in several experiments described in this thesis. See for example sections 3.3.1 and 4.4.2.

This approach, however, has some drawbacks if a context of variable duration influences the classification of a frame at time t, or if inputs and targets are not well aligned. Consequently, we need less rigid methods to model the temporal structure of the sequence, or the influence of context. A classical method to deal with this problem is to model the sequences with finite-state machines, or more generally, with hidden Markov models (HMM). In that case, it is assumed that the observations were generated by a system which could at any time be in one of N states and that each such state is associated with a distribution on the input. Section 6.1 explains in more detail what an HMM is.

To represent context, one needs a memory of the past. In ANNs, a memory can be provided by introducing delays or feedback links in the network. With the formalism introduced in section 2.1, a delay d_{ij} can be associated to link l_{ij} as follows:

$$y_i(t) = F_i(\theta_i, Y_i(t)) \tag{4.1}$$

where y_i is the output of unit u_i, θ_i is a set of parameters $\{\theta_{ij}\}$ for the function $F_i(\cdot)$, and $Y_i(t)$ is the set of activations $\{y_k(t-d_{ij})\}$ of the units u_k at frames $t-d_{ij}$ such that $l_{ij} \in L$ and $k = s_{ij}$.

A recurrent network differs from a static network in the property that its graph $G = (U, L)$ has at least one cycle, where $U = \{u_i\}$ is a set of units, $L = \{l_{ij} : \exists$ link in G from unit s_{ij} to unit i, with delay $d_{ij}\}$. Note that a recurrent ANN must have some delays as well, otherwise there will be a cycle of zero delay in the network, which can't be simulated because it corresponds to a physically impossible situation.

With delays, we can use information from a fixed relative past (e.g., from frame $t - D$, when looking at current frame t). On the other hand, with recurrent networks, an input frame from an arbitrarily long past may influence the computations for the current frame.

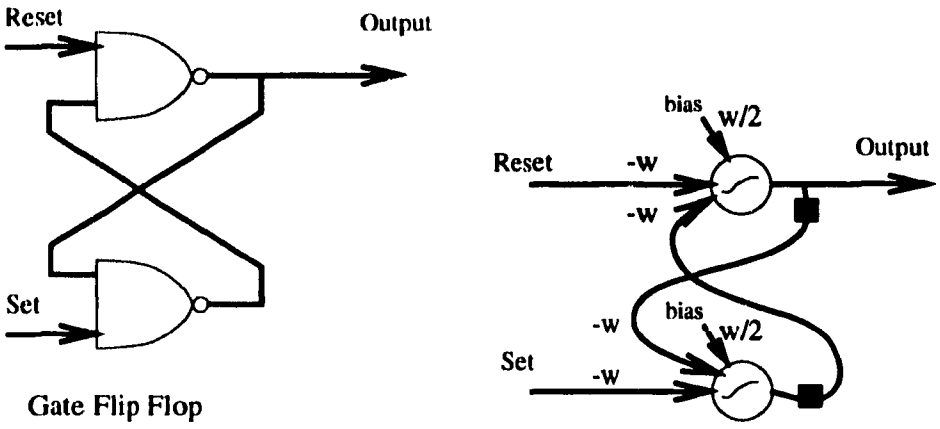


Figure 4.1: A recurrent network that retains a bit of information for an arbitrarily long time, provided weights are large enough. It is just two soft NOR gates with delay lines.

In practice, the memory of a **trained** recurrent network is limited, but not by a hard limit. To realize that this is true, consider the network in Figure 4.1 that implements a flip-flop. Since a flip-flop can be realized with 2 NOR gates and some delay lines, one can store a bit for an arbitrarily long time in this recurrent network, provided the magnitudes of the weights of the network are large enough [1]. This large weight condition is required only for networks of sigmoid units. In the case of threshold units, a regular flip-flop is realized. However, no efficient method is known to train networks of threshold units when they have one or more hidden layer[2].

For the purpose of characterizing the various algorithms for recurrent network described later in this chapter, let us define the following two notions:

- **Local in Time**: a learning algorithm is local in time if it can be executed as input frames arrive, using only temporally local information (such as values of some variables

[1] However, training a network with large weights with gradient descent may be very difficult, because saturated units, with output close to 1 or 0, send an almost zero gradient to the units that feed them, thus considerably slowing down the training. This may help explain why recurrent networks tend to concentrate on short-term context

[2] However, see [Gros89] for an approach based on learning of internal representations that works with threshold units

in a fixed and small time window). Such an algorithm could be used for example with an unbounded training sequence and could adapt the parameters of the network as the sequence is presented, for a cost per frame that does not increase with the length of the sequence.

- **Local in Space**: a learning algorithm is local in space if computations involving each unit require only information concerning its immediate neighbors in the network. Such an algorithm could be naturally executed with a parallel implementation of the network, with each node corresponding to a physical processing unit, and the communication cost per node depending only on the connectivity (fan-in, fan-out) of the network, and not on the total number of nodes in the network.

4.2 Time Delay Neural Networks

For the recognition of isolated words, it is possible to use the following simple approach to deal with variable length input sequences: transform the input sequence to a sequence of fixed length, either by compressing or dilating the sequence with a simple linear interpolation scheme. This is the approach chosen for example in the experiments described in sections 3.1.1 and 3.5.1, with vowels extracted from short isolated words. This scheme may allow a static ANN classifier to use all the input frames simultaneously, which in general yields networks with a great number of inputs and thus a great number of free parameters. Let us call that type of network a Type 1 network. Unfortunately, this approach, by imposing very little structure on the static recognizer, may yield poor generalization when the training set is of limited size (this is usually the case, see section 1.3). This may yield networks that are for example sensitive to the alignment of the input sequence. Furthermore, it can only be applied when each input sequence corresponds to a single (or a fixed number of) class(es), as in the case of isolated word recognition.

Time Delay Neural Networks (TDNN) were proposed in [Lang88, Waib87]. They are based on ANNs with time delays, and outputs averaged averaged over time, as illustrated in Figure 4.2. The basic idea of TDNNs is to impose some constraints on the Type 1 network described in the previous paragraph by forcing the ANN to use the same set of learned

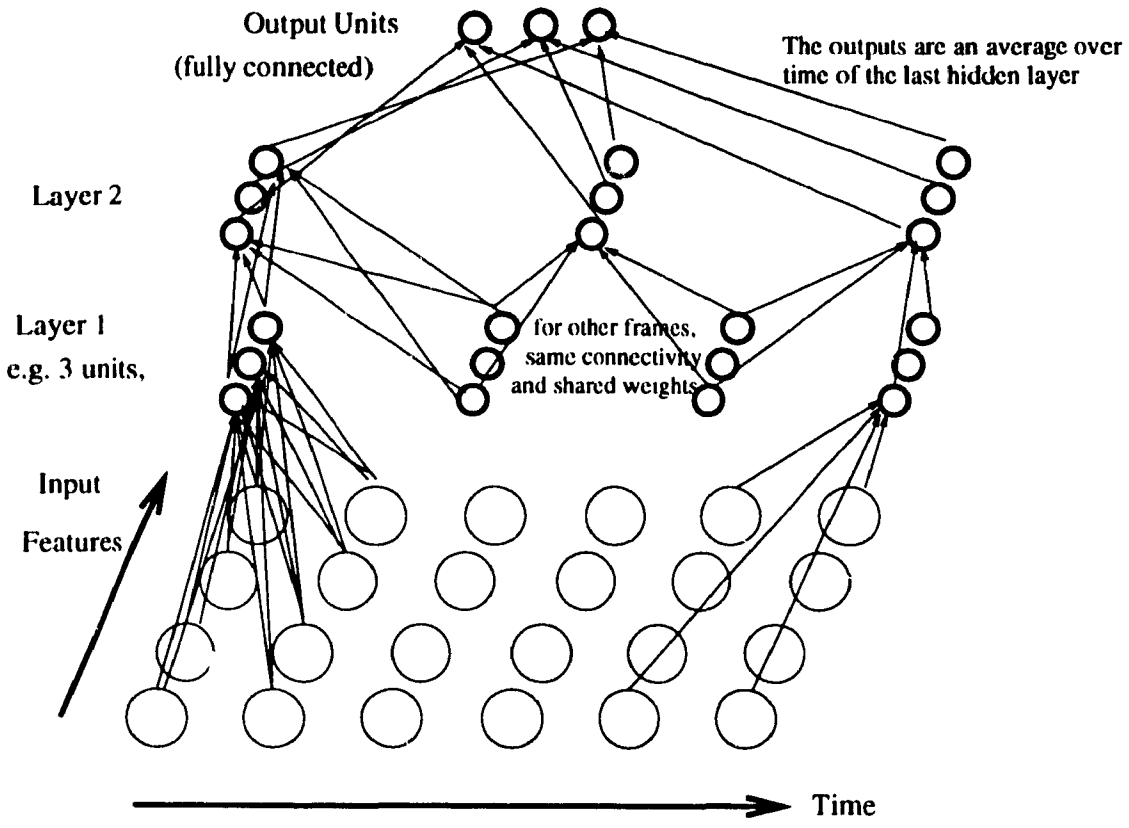


Figure 4.2: Typical architecture of a TDNN used for phoneme recognition.

internal features for all frames of the sequence. The learned internal features are the functions performed by the hidden units on each hidden layer. Starting from a Type 1 network, one can use weight sharing (see section 2.1.3) and local connectivity to constrain the internal features to be *time-shift invariant*. In addition, in order to account for the possibly imprecise alignment between inputs and targets, the outputs are computed by performing an average over time of the activations of the last hidden layer. This is already a significant improvement over the Type 1 network but is still quite rigid in its representation of context: each output unit "sees" information from a *fixed window* (or interval) of the input sequence.

4.3 Recurrent Networks

To avoid the above-mentioned problem of fixed windows, one can introduce cycles in the graph of the network. A recurrent network has this property: cycles in its graph allow it to keep information about past inputs for an amount of time that is not fixed a priori, but rather depends on its weights and on the input data. In Section 1.1 and Figure 1.1, we illustrated how a recurrent network may hold information for an arbitrary amount of time by building a flip-flop with a recurrent network.

4.3.1 Back-propagation in Time

The time-unfolding algorithm was proposed in [Rume86b] and another version of it (from discretized differential equations) can be found in [Pear89]. This algorithm computes the error gradient of an unconstrained discrete recurrent ANN. It requires keeping a memory of the activations of the units evaluated during the forward phase (going forward in time) in order to compute the first derivative of the cost with respect to the activations during the backward phase (backward in time).

To clarify our use of partial derivatives, let us define two types of partial derivatives, as in [Werb88]:

- The conventional partial derivative $\frac{\partial u}{\partial v}$ is calculated by differentiating the function u as it would normally be written as a function of its direct arguments, without any substitutions.
- The *ordered derivative* $\frac{\partial^+ u}{\partial v}$ refers to the total causal impact, including direct and indirect effects of v upon u.

Let us now consider a simple generalization of the time-unfolding algorithm based on the generalized unit equation (4.1). To perform gradient descent or another gradient based optimization technique, such as conjugate-gradient, one has to compute the partial derivatives of the cost C_p associated to each sequence p with respect to the parameters of the network (here, the θ_{ij} associated to each unit u_i). Gradient descent itself can be done following equation 2.12 for stochastic update, for example. The total cost for the training set can be decomposed as follows:

$$C = \sum_p C_p = \sum_p \sum_t C_{pt} \tag{4.2}$$

where C_{pt} refers to the t^{th} frame of the p^{th} training sequence. Since parameters θ_{ij} only affect $y_i(t)$ directly (equation 4.1),

$$\frac{\partial^+ C_p}{\partial \theta_{ij}} = \sum_t \frac{\partial^+ C_p}{\partial y_i(t)} \frac{\partial y_i(t)}{\partial \theta_{ij}} \tag{4.3}$$

The next important equation of the algorithm describes the actual *back-propagation in time*, i.e., the **recursive** computation (in the reverse time direction) of the ordered partial derivatives of the cost function with respect to the activations of each unit:

$$\frac{\partial^+ C_p}{\partial y_i(t)} = \sum_{s_{jk}=i} \frac{\partial^+ C_p}{\partial y_j(t+d_{jk})} \frac{\partial y_j(t+d_{jk})}{\partial y_i(t)} + I_{u_i \in U_O} \frac{\partial C_p}{\partial y_i(t)} \tag{4.4}$$

where the symbol I_c takes value 1 when c is true and 0 otherwise. A proof of the validity of the application of the chain rule as above can be found in [Werb74]. The last term of the above equation takes the following value for output units when the optimization criterion is the minimization of the squares of the differences between outputs and targets (LMS):

$$\frac{\partial C_p}{\partial y_i(t)} = (y_i(t) - target_i(t)) \tag{4.5}$$

Let us now consider the special case of units that compute a sigmoid $f(\cdot)$ of the weighted sum of their delayed inputs. Such a node operation was used in all of the experiments

described in this thesis unless otherwise indicated.

$$y_i(t) = f(\sum_l w_{il} y_{s_{il}}(t - d_{il})) \tag{4.6}$$

The required partial derivatives for equations 4.3 and 4.4 above are then the following:

$$\frac{\partial y_i(t)}{\partial w_{ij}} = f'(\sum_l w_{il} y_{s_{il}}(t - d_{il})) y_{s_{ij}}(t - d_{ij}) \tag{4.7}$$

and

$$\frac{\partial y_j(t + d_{jk})}{\partial y_i(t)} = f'(\sum_l w_{il} y_{s_{il}}(t - d_{il})) w_{jk} \tag{4.8}$$

where $s_{jk} = i$.

The first derivative $f'(\cdot)$ can be efficiently computed from $f(\cdot)$ as in equations 2.6 and 2.7, for the asymmetric and the symmetric sigmoid respectively.

The algorithm for back-propagation in time requires $O(N_w)$ operations per frame, and $O(L \times N_u + N_w)$ total space, where L is the maximum length of a sequence, N_u is the number of units in the network and N_w is the number of weights (parameters) of the network. The algorithm is local in space but not local in time, since we have to store all past activations of the network units and run the algorithm backward in time.

Static versus Recurrent Networks Experiments

Speech recognition experiments were performed with recurrent ANNs trained with the above described algorithm. Several architectures were compared, including architectures with no recurrence, to evaluate the improvements brought by recurrence in the network.

In experiments on the recognition of plosives and nasals in the TIMIT database described in [Ben91a], better generalization was obtained on that task with a recurrent network than with a static network, with the preprocessing and output coding described in sections 3.1.2 and 3.5, respectively. The recurrent network had links from the output layer to one of the groups of hidden units.

We will now describe in more detail further experiments on the recognition of vowels in the TIMIT database, also performed with both recurrent and static networks. The task was

Table 4.1: Comparison of various architectures and output codings for a vowel recognition problem in continuous speech (TIMIT 90). Output coding scheme: *1–1* means one phoneme per class, *form.* means relative formant frequencies, *place* means horizontal and vertical place of articulation.

exp. #	# hidden layers	# hidden units	recurrent network	# weights	output coding scheme	% frame error (train.)	% frame error (test)
1	0	0	no	3883	1-1	61.1%	63.8%
2	1	200	no	8811	1-1	60.3%	61.4%
3	1	60	no	8531	1-1	60.2%	61.8%
4	1	20	yes	3031	1-1	48.5%	52.7%
5	2	40	yes	7811	1-1	47.0%	51.0%
6	3	60	yes	16753	1-1	46.4%	51.7%
7	3	60	yes	16739	form.	62.9%	64.6%
8	2	40	yes	7811	place	52.9%	55.6%
9	2	40	yes	10979	place	50.7%	52.3%

the discrimination of the following 11 vowel classes (based on the grouping of phonemes proposed by [LeeH 89]), in every context:

{eh, ao, aa, uw + ux, er + ix + axr + ax-h, ax, ih, ae, ah, uh, iy }

The training set was obtained by extracting vowels (based on the database segmentation) from 1024 training SI and SX sentences of TIMIT (1990 version) from the training speakers. The 192 SI and SX sentences from the core test were used for evaluating generalization. Training was stopped when no more significant improvement could be obtained on the training set from one training epoch to the next. The input to the neural network was simply a 32-filter Bark-scale spectrogram (as described in section 3.1.2). Unless otherwise indicated there were 11 outputs, one per vowel. The results are summarized in table 4.1.

1. No hidden units. Static network. Delays 0,2,4, ... 20 between the input and the output units. 3883 weights. 61.1% frame error on the training set and 63.8% frame error on the test set.

2. One hidden layer with 200 units. Static network. no delays. 8811 weights. 60.3% on training and 61.4% on test set.

3. One hidden layer with 60 units. Static network. Delays 0 and 3 between the input and the hidden units. Delays 0,2,... 12 between the hidden and the output units. 8531 weights. 60.2% frame error on the training set and 61.8% frame error on the test set.

4. One hidden layer with 20 units. Recurrent network. Hidden and output units are connect to each other and to themselves recurrently with 1-frame delays, except for the links from hidden to output, which have delays 0, 2, 4, 6, 8. There are also delay 0 and 3 from the input to the hidden units. 3031 weights. 48.5% frame error on the training set and 52.7% on the test set.

5. Two hidden layers with 20 units each. Recurrent network. Delays 0 and 3 from the input to both hidden layer. Delays 0,2,4,6,8 from the input and both hidden layers to the output. Single delay of 1 frame from the output to both hidden layers and from each hidden layer to the other one. 7811 weights. 47% frame error on the training set and 51% frame error on the test set.

6. Three hidden layers with 20 units each. Recurrent network. Delays 0,1,2,3 from input to hidden units; 0,1,2,...8 from input and hidden units to output; from hidden1 to hidden2, from hidden2 to hidden3, from hidden3 to hidden1, and from output to hidden units, a single 1-frame delay. 16739 weights. 46.4% error on the training set and 51.7% error on the test set.

7. As the previous experiment, but using expected relative formant frequencies (F2-F1,F1-F0) as targets (which are related to place and and manner of articulation). 10 outputs. 62.9% frame error on the training set and 64.6% on the test set.

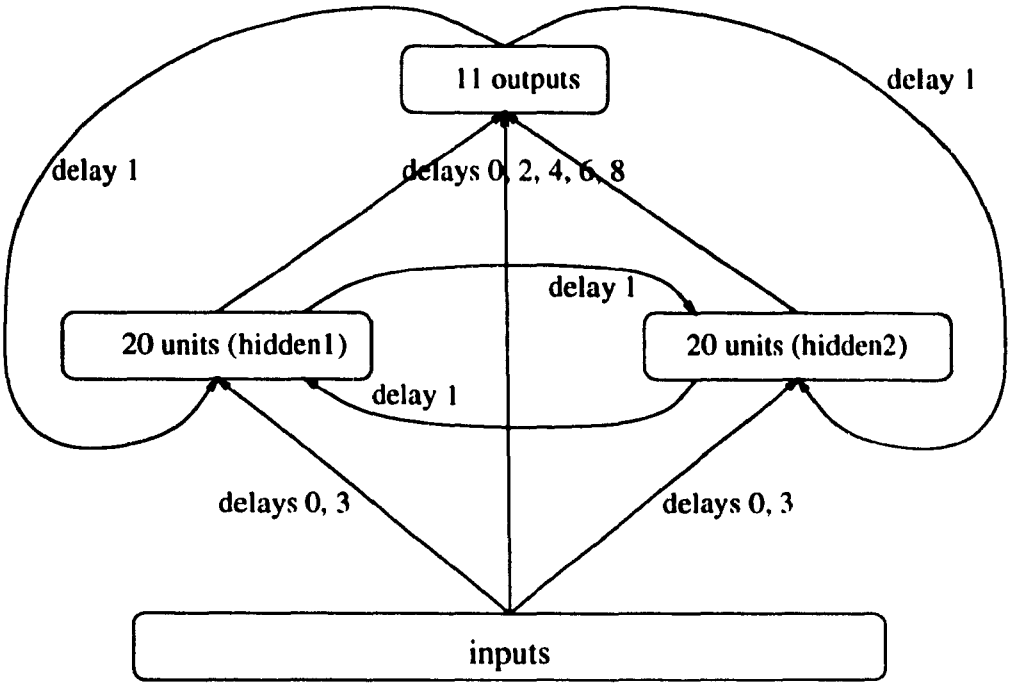
8. Same architecture as experiment 5, but using an output coding based on place and manner of articulation of vowels. The 11 outputs are as follows: 5 nodes for horizontal place of articulation, 5 nodes for vertical place of articulation, 1 node for indicating the phoneme is a schwa [3]. 52.9% frame error on the training set and 55.6% on the test set.

9. As the previous experiment but using more delays to the output units (from the hidden units: delays 0,1,2,.. 8; from the inputs: 0,2,4,... 16). 10979 weights. 50.7% frame error on the training set and 52.3% on the test set.

Note the drastic improvement in using recurrence: from more than 61% frame error with static networks to around 51% error with recurrent networks. One should also notice that the use of articulatory features in the vowel targets does not seem to be as successful in that case, when compared to the simpler "one-output-per-class" scheme. This may be explained as follows. Ideal values of place and manner of articulation (or of formant frequencies) were derived in simple settings such as short, well-pronounced isolated words. In contrast, the speech signal of TIMIT is continuous and one may observe that the target formant frequencies or place of articulation of vowels does not seem to be reached, most of the time, because the vowel is too short and is very much influenced by left and right contexts.

Another static versus dynamic comparison was done, for the BPS algorithm described in Section 1.3.1. There again, the recurrent network performed significantly better than the

[3] Previous experiments without the schwa node indicated many confusions involved one of the schwa phonemes /er/, /ax/, /ix/, /ux/, /axr/ and /ax-h/.



Best architecture obtained for vowel recognition in continuous speech on TIMIT (experiment #5 in Table 4.1).

static network.

However, in some cases in which relevant information in the input sequence is mostly local, recurrence doesn't help performance. This is the case of the broad classification network used in the experiments of section 6.7.1.

4.3.2 Forward Propagation

Another algorithm for training recurrent networks with no restriction on the architecture was independently proposed in various forms in [Kuhn87, Kuh90b] and [Will88]. This algorithm avoids the back-propagation in time, which requires storing the complete sequence of network activations. It achieves this by computing recursively and keeping in memory during the regular forward pass partial derivatives which indicate how each weight of the network influences each unit activation. A generalized derivation of this algorithm will now be presented, using the already introduced formalism for describing recurrent networks.

To perform "frame-by-frame" online adaptation, one needs to compute the gradient of the local cost C_t with respect to the system parameters $\{\theta_{ij}\}$. The parameter gradient can be first decomposed as follows, using the same conventions as in the previous section:

$$\frac{\partial^+ C_t}{\partial \theta_{ij}} = \sum_k \frac{\partial C_t}{\partial y_k(t)} \frac{\partial^+ y_k(t)}{\partial \theta_{ij}} \tag{4.9}$$

The first factor in the right hand side of the above equation is zero except for units for which a target is provided at frame t. In that case it is simply $\frac{\partial C_t}{\partial y_k(t)} = (y_k(t) - target_{kt})$ with the LMS criterion.

The partial derivatives of unit activation ($y_k(t)$) with respect to every parameter (θ_{ij}) can be computed recursively as follows:

$$\frac{\partial^+ y_k(t)}{\partial \theta_{ij}} = \sum_l \frac{\partial y_k(t)}{\partial y_{s_{kl}}(t - d_{kl})} \frac{\partial^+ y_{s_{kl}}(t - d_{kl})}{\partial \theta_{ij}} + \delta_{ik} \frac{\partial y_k(t)}{\partial \theta_{ij}} \tag{4.10}$$

The first factor depends only on the definition of $F_k(\cdot)$ (see for example equations 4.5 and 4.7). The last term is non-zero only if $k = i$.

This algorithm requires in the worst case $O(N_u N_w D)$ storage, where N_u is the number of units, N_w is the number of network parameters θ, and D is the maximum delay between two units. In the simple case in which $D = 1$ and a small fixed set of output units is used, the storage requirement is $O(N_o N_w)$, where N_o is the number of output units. The required computing time is also prohibitive: it can be as bad as $O(N_u^4)$ in the worse case. This is always worse than the $O(N_w)$ of the algorithm for back-propagation in time, unless it allows — when sequences are very long — to perform much more frequent parameter updates with stochastic gradient descent, thus allowing a faster convergence (see section 2.2). Hence in comparison with the back-propagation in time algorithm, this algorithm trades-off locality in space and efficiency for locality in time, because the computation of the derivatives $\frac{\partial y_k}{\partial w_{ij}}$ is not local in space. However, the complexity of the forward propagation algorithm can be limited as in the experiments described in [Kuh90a], for example by using only self-loops. This is similar to the architectural constraint of the BPS algorithm, which relies on combined forward and backward propagation of partial derivatives.

4.3.3 BPS

The BPS algorithm was developed in collaboration with Marco Gori [Ben90c, Gori89]. We proposed the BPS algorithm as an efficient algorithm for training a particular kind of constrained recurrent ANN. The algorithm has been called BPS for Back Propagation for Sequences, and operates on a simple class of recurrent ANNs in which *dynamic neurons* — with a single feedback to themselves — have only incoming connections from the input layer. This model is similar to the model of M. Mozer [Moze88] which was discovered independently and is called "focused" back-propagation. It is also somewhat related to the forward propagation model when constrained to the same architecture (local feedbacks and direct input connections to the dynamic units), as proposed in [Kuh90a].

In this model, two kinds of units are distinguished: static unit and dynamic unit. The first kind is as described in section 2.1:

$$y_i(t) = f(\sum_j w_{ij} y_{s_{ij}}(t)) \tag{4.11}$$

where $f(\cdot)$ is a non-linear function such as the sigmoid (equations 1.2 or 1.3).

Dynamic units compute their activation as follows:

$$y_i(t) = f(x_i(t)) \tag{4.12}$$

where the intermediate variable $x_i(t)$ is defined as follows:

$$x_i(t) = \sum_{\tau}^{D} a_{i,\tau} x_i(t-\tau) + \sum_j w_{ij} y_{s_{ij}}(t - d_{ij}) \tag{4.13}$$

where s_{ij} is the node number of the *source* of link l_{ij}, as defined in Section 2.1. The summation over τ is taken over some positive values, e.g., $\tau = 1, 2, ...D$. The above equation can be seen as the equation of an Infinite Impulse Response filter. The parameters $a_{i,\tau}$ are called decays because they can be interpreted as exponential decay factors for the intermediate variable x_i. This is a generalization of the algorithm we presented in [Ben90c, Gori89], in which only a single time delay was considered.

Gradient Computation

The error derivative can be expressed by

$$\frac{\partial^+ C_t}{\partial w_{ij}} = \frac{\partial^+ C_t}{\partial x_i(t)} \frac{\partial^+ x_i(t)}{\partial w_{ij}} \tag{4.14}$$

This simple chain rule equation is also used to derive the back-propagation algorithm for static networks (equation 2.8). It is not valid for an arbitrary recurrent network but it is valid for the BPS architecture. Indeed, with an arbitrary recurrent network, the correct gradient of the local cost C_t would be written

$$\frac{\partial^+ C_t}{\partial w_{ij}} = \sum_{\tau \leq t} \frac{\partial^+ C_t}{\partial x_i(t-\tau)} \frac{\partial^+ x_i(t-\tau)}{\partial w_{ij}} \tag{4.15}$$

On the other hand, with the BPS architecture, $x_i(t-\tau)$ ($\tau > 0$ and u_i a dynamic unit) may influence the local cost C_t, but only through the influence it has on $x_i(t)$. This influence is taken into account in equation 4.16. On the other hand, if the inputs of the dynamic units were not input units, it would be necessary to take into account all the influences of those units on the error through all the dynamic units. The resulting equations would then be closer to the forward propagation model described in the previous section, requiring many more partial derivatives be stored and computed than with the BPS algorithm.

With the BPS architecture, the first factor of equation 4.14 can be computed as for static networks (see Section 2.1), as we proceed along the sequence, instead of waiting for the end of the sequence to back-propagate gradients (the forward propagation algorithm of section 4.3.2 has the same characteristic: locality in time).

$$\frac{\partial^+ C_t}{\partial x_i(t)} = \begin{cases} (y_i(t) - target_{it}) f'(x_i(t)) & \text{for } u_i \in U_O \\ \sum_{j:s_{jk}=i} \frac{\partial^+ C_t}{\partial x_j(t)} \frac{\partial x_j(t)}{\partial x_i(t)} = \sum_{j:s_{jk}=i} \frac{\partial^+ C_t}{\partial x_j(t)} w_{jk} f'(x_i(t)) & \text{for } u_i \in U_H \end{cases} \tag{4.16}$$

An interesting aspect of the BPS algorithm is that the second factor of equation 4.14 can be computed recursively during the *forward* phase of the algorithm:

$$\frac{\partial^+ x_i(t)}{\partial w_{ij}} = \frac{\partial x_i(t)}{\partial w_{ij}} + \sum_{\tau}^{D} \frac{\partial x_i(t)}{\partial x_i(t-\tau)} \frac{\partial^+ x_i(t-\tau)}{\partial w_{ij}}$$

$$= y_{s_{ij}}(t - d_{ij}) + \sum_{\tau}^{D} \alpha_{i,\tau} \frac{\partial^+ x_i(t-\tau)}{\partial w_{ij}} \tag{4.17}$$

The parameters $\alpha_{i,\tau}$ describe the temporal response of the dynamic unit u_i. For example, with a single delay for the feedback, the impulse response of the unit is a discrete time decaying exponential (for the variable x). In general, the intermediate variable x corresponds to the output of an arbitrary Infinite Impulse Response filter for the weighted input vector. The $\alpha_{i,\tau}$ can be fixed a priori or they can be learned, again using back-propagation. The gradient of the error with respect to these parameters can be computed as follows:

$$\frac{\partial^+ C_t}{\partial \alpha_{i,\tau}} = \frac{\partial^+ C_t}{\partial x_i(t)} \frac{\partial^+ x_i(t)}{\partial \alpha_{i,\tau}} \tag{4.18}$$

where the first factor is computed as in equation 4.15 and the second factor can be computed recursively as follows:

$$\frac{\partial^+ x_i(t)}{\partial \alpha_{i,\tau}} = \alpha_{i,\tau} \frac{\partial^+ x_i(t-\tau)}{\partial \alpha_{i,\tau}} + x_i(t-\tau) \tag{4.19}$$

In the case of $D = 1$ (single delay), local constraints can be easily satisfied in order to guarantee *stability* of the recurrent network. Each dynamic weight $\alpha_{i,1}$ is assumed to depend on another parameter λ_i:

$$\alpha_{i,1}(\lambda_i) = B \tanh(\lambda_i) \tag{4.20}$$

where $B \leq 1$ is a decay bound constant. It is interesting to note that by controlling B, one can modulate the dynamics of the network: as $B \to 0$, the network becomes static.

The gradient with respect to λ_i can be computed with the chain rule:

$$\frac{\partial^+ C_t}{\partial \lambda_i} = \frac{\partial^+ C_t}{\partial \alpha_{i,1}} \frac{\partial \alpha_{i,1}}{\partial \lambda_i} \tag{4.21}$$

where the second factor can be computed as in equation 2.7.

Hence the BPS algorithm requires $O((N_w + N_u)D)$ storage and computation, where N_w is the number of weights, N_u is the number of units, and D is the maximum delay in the network. For $D = 1$, the space and time costs are the same as for the back-propagation

algorithm for static networks. On the other hand its space requirement is less than that necessary with backpropagation through time ($O(N_w + LN_u)$). Furthermore, it has the advantage of being both local in time and in space, since all the computations at each unit can be performed with a fixed amount of storage and time per frame, independently of the length of the sequence and of the rest of the network, except for the immediate neighbors of the unit. For $D > 1$, additional storage and computations are required, growing linearly with D. Since the algorithm is local in time, it can be executed in an online mode, unlike the backpropagation through time algorithm. In an online mode, the network weights are update after each *frame*. In contrast, with the back-propagation through time algorithm, the weights may be updated only after each sequence. If the sequences are long, the advantage of online training may be significant (see discussion and experiments in Section 2.2.1 concerning stochastic vs deterministic gradient descent).

4.3.4 Experimental Assessment of BPS

Experiments were performed [Gori89, Ben90c] in order to compare the BPS algorithm with back-propagation for static networks on a limited but difficult speech recognition task: the discrimination between the two stop consonants /b/ and /d/ independent of the speaker.

At every time frame t the ANN was fed with the output of a preprocessor that computed 91 parameters for ev[illegible] 10 ms of speech. Seventy-eight of these parameters are the outputs of 2 frames of an [illegible] 1 Mel-scale 39-frequency filter bank, one produced at t-10 ms and one at t. The remaining 16 parameters consisted of

- the zero-crossing rate,
- the energy of the signal (within 20 ms window),
- the energy of the derivative of the signal,
- a high to low frequency energy ratio (6 kHz - 9 kHz over 1 kHz - 4 kHz), for 4 intervals of 10, 5, 5 and 10 ms respectively around t.

Hence there was an overlap of information presented at successive frames. The experiments

were performed on pronunciations of the letters "b" and "d" of the alphabet, by 60 speakers (30 males, 30 females). Each speaker pronounced the letters twice. The first 50 speakers were used for training and the last 10 for testing.

Both the static network and the BPS network had 10 hidden units and 3 output units {/silence/,/b/,/d/} and both were initialized with identical initial (static) weights. The BPS network used a single delay of 1 frame for the dynamic units. After convergence, the static network performed with 6.9% error on the test set whereas the BPS network reached 3.45% error on the test set.

Hence, the recurrence provided by the dynamic units has helped the BPS network capture important information about the temporal structure of the input signal, allowing it to perform better than a static network with delays.

4.3.5 Supervision of a Recurrent Network Does Not Need To Be Everywhere

The environment for training a recurrent ANN is composed of a sequence of frames. The supervision, consisting of imposing target values to network outputs, is not necessarily associated with every frame, but can be arbitrarily provided at one or many points along the sequence. Because of recurrence, future errors can be influenced by past inputs and activations. Hence, the learning, as well as the recognition processes operate on sequences and the ordering of these sequences is very important. The target values can be represented by a temporal sequence. To supervise the network at particular points in the sequence, the LMS cost function can be modified as follows:

$$C = \sum_t \sum_{i \in U_0} (target_{it} - y_i(t))^2 c_{it} \tag{4.22}$$

where c_{it} is a binary switch which specify that a target is provided at time points t and for output units i. Note that, this is a particular case of the weighted LMS criterion, in which the covariance matrix Σ for the Gaussian error between the targets and the outputs is known (see also [Jord88]):

$$C = \sum_t (\mathbf{target}_t - \mathbf{y}_t)^t \Sigma_t^{-1} (\mathbf{target}_t - \mathbf{y}_t) \tag{4.23}$$

where $\mathbf{target}_t$ and $\mathbf{y}_t$ are target and output vectors respectively.

The computation of the error derivatives are the same as usual when $c_{it} = 1$ but $\frac{\partial C_t}{\partial y_i} = 0$ when $c_{it} = 0$, i.e., there is no cost associated to the output of unit i at time t.

4.3.6 Problems with Recurrent Networks trained with Back-Propagation

Let us consider some problems about recurrent networks trained with back-propagation:

- **dynamic instability**: Our experiments with unconstrained recurrent ANNs indicate that during training, units saturate more often than in the case of static networks. The recurrent ANN is not really unstable if unit outputs are bounded (e.g., with a squashing function), but in the case of saturation, learning ceases to occur because the first derivative of a unit output (with respect to its weights or its inputs) is almost zero.

- **local minima**: Although in theory a recurrent ANN could learn about the influences of very remote past upon current error, this seems to require very large weights, which slows down convergence. Hence in practice, it is often observed that the network parameters fall in a local minimum which corresponds to learning about the short term influences of the inputs upon the error.

- **biological implausibility**: As mentioned in Section 5.1, one of the attractive features of ANN models is their resemblance to the workings of the brain. However, powerful learning algorithms such as back-propagation are not biologically plausible. First, the algorithm requires the transmission of information in neurons and synapses in both a forward direction and a backward direction (see section 7.2 about the search of more plausible learning algorithms). Furthermore, algorithms for recurrent networks with arbitrary connectivity, such as, those presented in sections 4.3.1 and 4.3.2 are either non-local in time or non-local in space, which makes them biologically implausible. The BPS algorithm, is both local in time and in space but has a constrained architecture. Note also that algorithms related to the Boltzmann machine algorithm - including the more efficient mean field version of the Boltzmann

machine [Hint89] - are both local in time and in space.

4.4 Hybrids

In sections 3.4 (modularization in the design of ANNs), 2.2.1 (parameter decoupling), and 5.2 (integrating ANNs with other tools), we have argued about the advantages of modularizing a learning system[4], using modules based on ANNs or other algorithms. Particular cases of integration of ANNs with other algorithms for the recognition of sequences will be considered here. Two major types of hybrids will be discussed: ANNs with dynamic programming and ANNs with hidden Markov models (HMMs)[5] A learning algorithm for the latter type of hybrid is proposed and evaluated in Chapter 6.

4.4.1 ANNs and Dynamic Programming

The principle of dynamic programming [Bell57] is to use recursively previously computed information to reduce the complexity of a computation, typically a search among a large number of paths, or taking a sequence of interrelated decisions. If the problem satisfies some locality constraints, it can be broken down into subproblems and each subproblem is solved by using previously computed results for smaller subtasks. This may allow reduction of the complexity of a problem which could require an exponential number of computations (if each possible subproblem was solved separately) to a polynomial time complexity. Typically, this requires storing the results of all subtasks of the previous level (e.g., path length) in a table and using them to compute the results of the next level of subproblems.

In the case of the application of dynamic programming (DP) to the analysis of sequences, the problem can be expressed in one of the following forms.

$$C = \sum_{p} \prod_{t}^{L} C_{pt} \tag{4.24}$$

[4]when the training set and the computing resources are bounded and practically insufficient to use a completely unbiased ANN.

[5]Note that these two tools are actually related since recognition with HMM algorithms uses dynamic programming.

or

$$C = \min_{p} \sum_{t}^{L} C_{pt} \tag{4.25}$$

where p is a sequence $\{i_1, i_2, ..i_L\}$. The task in both cases is to compute C (and in the second case maybe also provide the best path p that was found). However, in general the total number of paths N_p grows exponentially with L, for example $N_p = N_s^L$, where N_s would indicate the number of possible *states* of the system, i.e., the number of possible values for each i_t. The complexity of the problem can be drastically reduced when many of the terms C_{pt} are identical, for example, if they depend only on the state i_t at time t in path p and on the previous state i_{t-1} at time $t-1$:

$$C_{pt} = I_{t>0}\, T_{i_t,i_{t-1},t} + U_{i_t,t} \tag{4.26}$$

or

$$C_{pt} = (T_{i_t,i_{t-1},t})^{I_{t>0}} U_{i_t,t} \tag{4.27}$$

I_c is a boolean indicator that takes value 1 when c is true and 0 otherwise. $T_{i,j,t}$ and $U_{i,t}$ are problem dependent quantities that respectively depend on the transition from the previous state to the current state, and on the local state at the current frame t.

One can then express the above equations as follows:

$$C = \sum_{i} S_{i,L} \tag{4.28}$$

or

$$C = \min_{i} S_{i,L} \tag{4.29}$$

where $S_{i,t}$ is a sum (or min) over sub-paths q, considering only the first t frames:

$$S_{i,t} = \sum_{q} \prod_{\tau<=t} C_{q\tau} \tag{4.30}$$

or

$$S_{i,t} = \min_{q} \sum_{\tau<=t} C_{q\tau} \tag{4.31}$$

Because of the locality of C_{pt}, one can thus compute recursively (and efficiently)

$$S_{i,t} = (\sum_{j}^{N_s} S_{j,t-1} T_{i,j,t})^{I_{t>0}}\, U_{i,t} \tag{4.32}$$

or

$$S_{i,t} = I_{t>0}(\min_j\{S_{j,t-1} + T_{i,j,t}\}) + U_{i,t} \tag{4.33}$$

When one needs to keep track of the required path (for the min case), the arg min of the above equation is kept:

$$P_{i,t} = \arg\min_j(S_{j,t-1} + T_{i,j,t}) \tag{4.34}$$

This gives a pointer to the best previous state. Hence the best path can be retrieved as follows in a backward pass:

$$\begin{aligned} i_L &= \arg\min_j S_{j,L} \\ i_t &= P_{i_{t+1},t+1} \end{aligned} \tag{4.35}$$

In the Baum-Welsh algorithm for training HMMs, the sum of products is used, with $T_{i,j}$ representing state transition probabilities and $U_{i,t}$ representing observation probabilities. These variables have the same meaning in the Viterbi algorithm (for training HMMs) except that a max of sums of log probabilities is used instead of the sum of products of probabilities. The backward pass also provides the best path for the Viterbi training algorithm and for recognition in HMMs.

In the dynamic programming approach used in sections 5.1.2 and 5.1.3, $T_{i,j,t}$ can be used for costing transitions, including a duration cost[6], associated to the number of consecutive frames during which the state kept the same value. Hence $T_{i,j,t}$ may represent temporal constraints, whereas $U_{i,t}$ is used to measure a local cost associated to being in state i at frame t (e.g., computed by or derived from the output of an ANN).

This algorithm can also be used to perform template matching with dynamic time warping, by computing a cost C for each prototype that is a measure of distance between the input sequence and each prototype sequence. The selected prototype is the one "closest" to the input sequence according to that warped distance measure. In that case $T_{i,j}$ may constrain the warping to skip or repeat only a limited number of frames from the prototype and $U_{i,t}$

[6] However, as explained in [Jouv88], using explicit duration probabilities when changing state may yield suboptimal paths in some cases, because the cost function is no longer local: it depends on decisions to be taken in the future (when exiting a state). On the other hand, duration probabilities as used in Section 6.7 greatly improved recognition performance.

is a measure of distance between the t^{th} frame of the input sequence and the i^{th} frame of the prototype.

4.4.2 Analysis of Amino-Acid Sequences

As described in [Ben90f], we have used an ANN integrated with dynamic programming to perform the recognition of immunoglobulin domains from amino acid sequences. The ANN/DP hybrid was designed to identify proteins exhibiting such domains with minimal rates of false positives and false negatives. The National Biomedical Research Foundation (NEW) protein sequences were scanned to evaluate the performance of the system in recognizing mouse immunoglobulin sequences. For such sequences, the recognition efficiency was 98.2% with an overall false positive rate of 7.3%. These results showed that ANN-based search programs are well suited to search for sequences characterized by only a few well conserved subsequences. By using the recognition system on previously unseen data it was discovered that the Epstein-Barr virus envelope protein bears an Ig-like domain [Cash90], thus demonstrating the usefulness of such a recognition system.

Problem Background

Domains are characteristic amino-acid subsequences found in many different proteins. Immunoglobulin (Ig) domains are sets of β-sheets bound by disulfide bonds which exhibit a characteristic tertiary structure. Because proteins are assembled from different domains, it is useful to be able to scan amino acid sequence databases to identify proteins that might comprise a domain of interest. This capability would also be useful in the analysis of newly sequenced proteins. However, current search programs incorporating algorithms such as the Wilbur-Lipman algorithm [Wilb83] or the Needleman-Wunsch algorithm [Need70] and its modification by [Smit81] are ill-designed for detecting domains because they implicitly consider each amino acid to be equally important. This is not the case for residues within domains such as the Ig domain, since only some amino acids are well conserved, while most are variable. Search programs incorporating algorithms that do not reflect this will therefore not detect efficiently proteins bearing the domain of interest. A solution is to

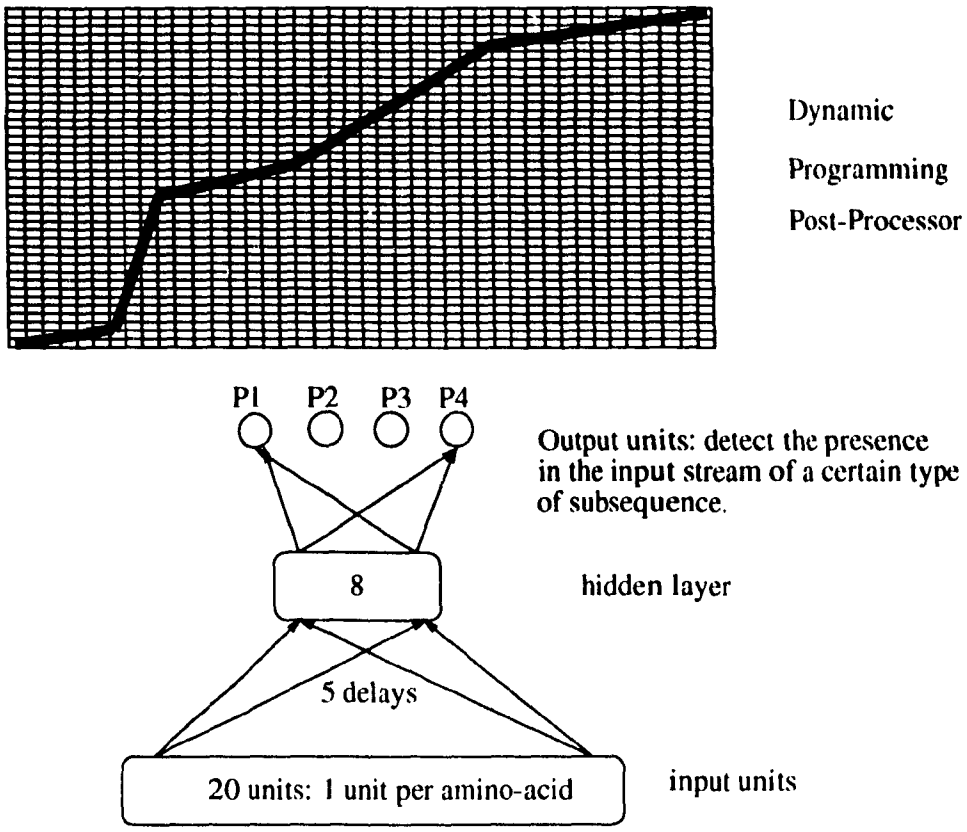


Figure 4.3: Architecture of the system for the recognition of Ig domains in amino-acid sequences. The input window is 5 amino-acid long.

use statistical occurrence of a residue at a particular position [Grib87, Wang89, Deve84]. Although programs based on this idea (**Profile Analysis**, [Deve84]) can be applied, they often suffer from a high rate of false negatives and positives, especially when there are considerable variations in domain length to be accounted for, as in the case of Ig domains. Perceptrons and other types of ANNs have been used previously in research on protein and DNA analysis [Stor82, Bohr88, Qian88, Holl89]. Our results indicate that they are well suited for detecting complex pattern sequences such as those that characterize Ig domains.

System Design

The design of the ANN we described in [Ben90f] capitalizes on the following knowledge about the particular task at hand: the Ig domain is characterized by several highly conserved groups of amino acids whose locations have been identified for certain well studied

Ig proteins. In particular, the β-strands B, C, D, E and F of the Ig domain are very well conserved segments of the domain [Wil88b]. Using this information, the design of the system has focused on the recognition of sequences of four conserved sub-regions corresponding to β-strands B, C, E and F, here designated P1 to P4 respectively. Amino acids in these segments are not necessarily all conserved, but in each sub-region (P1-P4), they show a distribution very different from the distribution observed elsewhere in these proteins. Hence the system was trained to recognize these distributions, with appropriate temporal constraints. The recognition problem is thus divided into two subproblems:

1. Detect subsequences similar to P1, P2, P3 or P4.
2. Detect occurrences of P1 followed by P2, P3, and P4, in the right order and with some duration constraints.

The first task is performed by an ANN trained with back-propagation, with 4 outputs for each of the 4 regions of interest. The network is a simple static network with delays, with 8 hidden units that scan proteins with a window of 5 residues. Since each residue is represented with 20 input units (for 20 amino acids), there are 100 inputs to the network. Better results were obtained with a multi-layer architecture than with no hidden layer. In comparative experiments on the network itself, performance of the multi-layer network and of the network with no hidden layer were 2.1% and 4.3% frame error, respectively [Ben90e].

The second stage of the system, based on the stream of outputs from the P1-P4 detector, evaluates whether a region similar to the Ig domain has been detected. Dynamic programming was used, with constraints on the order and the distance (duration) between detected subsequences. A weak duration cost was defined for each interesting transition, with zero cost for values within the bounds found within the training set proteins, and linearly increasing cost for shorter or longer gaps. A cut-off value of the total cost computed for each potential solution subsequence was used to accept or reject a subsequence. This value was selected in order to optimize the trade-off between false positives and false negatives.

As a training set, a group of 30 proteins comprising *bona fide* Ig domains was used [Wil88b]. Because of the small number of these sequences, the training set was artificially augmented using knowledge about residue distributions in general, thus generating a very large number

of pseudo-sequences. These sequences were generated by stochastically substituting residues known to be in variable positions of the domain with variable residues found elsewhere in the sequence. This was necessary in order to obtain good generalization. This is a good illustration of the concepts described in section 1.3 and in Chapter 3 about the use of domain knowledge to reduce the effective VC-dimension of the ANN. Knowledge about the problem was also used to modularize the system and combine the ANN with an appropriate sequence analysis tool.

Results

The system was evaluated by scanning the National Biomedical Research Foundation (NBRF) protein database (NEW, version 19). By scanning 1718 proteins from this database, 191 proteins were identified as possessing at least one Ig domain. All proteins in the training set were detected. Even though only human major histo-compatibility complex (MHC) class I and II proteins were included in the training set, both mouse H-2 class I and II proteins were detected. Proteins from unseen species, such as bovine and rat transplantation antigens were also detected. Recognition of human and mouse immunoglobulin sequences was used to measure recognition efficiency. The rate of false positive was 7.3%. Most of them possess features somewhat similar to those of true Ig domains and they score low, i.e., near the threshold (see [Ben90f] for a more detailed analysis). The recognition efficiencies for mouse and human immunoglobulins were 98.2% and 93.8% respectively. By carefully analyzing the results, an interesting discovery was made. The usefulness of this system was demonstrated when the Epstein-Barr virus (EBV) envelope protein p110 (NBRF filename QQBE1) was discovered to bear an Ig-like domain by the ANN/DP hybrid [Cash90].

4.4.3 Phoneme Recognition

A dynamic programming algorithm can also be used to impose some temporal constraints on the output of an ANN used for phoneme recognition. Robinson and Fallside [Robi90a] obtained excellent results with a phoneme duration constraint in a dynamic postprocessor for a recurrent ANN. We propose here a similar scheme in which we also incorporate statis-

tical information about bigram probabilities (conditional probability that phoneme i follows phoneme j), a priori frame-based class probabilities (probability of phoneme i at frame t), as well as observation probabilities (conditional probability of network outputs given a class). Several researchers have interpreted network outputs as probabilities [Bour88, Fran90] or combined them directly with duration probabilities [Robi90a]. We have instead chosen to compute observation probabilities (based on a normal distribution or a mixture of normal distributions), as described in equation 4.40.

The resulting cost function associated to each sequence is the following:

$$C = -\sum_{t=1}^{L} \log P(s(t)) + \log P(Y(t) \mid s(t)) + I_{s(t)\neq s(t-1)}(\log P(s(t) \mid s(t-1)) + \log P(\text{duration of } s(t-1))) \quad (4.36)$$

where $s(t)$ represents the *state* of the system, here the class of interest, e.g., a phoneme. $Y(t)$ is the vector output of the ANN at time t. The dynamic programming algorithm finds the sequence $s(1), s(2), ...s(L)$ that minimizes[7] the cost C.

The training segmentation provided correct sequences $s(1), s(2), ...s(L)$ with which the required statistics were evaluated as follows (considering all the training sequences).

The bigram probabilities are estimated as follows:

$$P(s(t) = p \mid s(t-1) = q) = \frac{\sum_t I_{s(t)=p \wedge s(t-1)=q}}{\sum_t I_{s(t-1)=q}} \quad (4.37)$$

The duration probabilities can be modeled with a gamma distribution for each phoneme (or class):

$$P(\text{duration of phoneme } s(t) = \tau) = \frac{\tau^{-(\kappa+1)} e^{-\frac{\tau}{\theta}}}{\theta^{-\kappa}\Gamma(\kappa)} \quad (4.38)$$

where κ and θ are estimated as in [Gree60]. The advantage of a gamma distribution over the more common normal distribution is that it is more appropriate to model *positive* random variables, such as duration.

[7] As already mentioned, the resulting path may be suboptimal because of the non-locality of explicit duration constraints (see [Jouv88]). It is however very useful in a dynamic programming postprocessor (see experiments in section 6.7, table 6.1).

The frame-based phoneme probability can be estimated as follows:

$$P(\mathrm{s(t)} = p) = \frac{\sum_t I_{s(t)=p}}{L} \tag{4.39}$$

The observation probabilities were estimated with likelihood maximization, using a normal distribution with diagonal covariance matrix for each phoneme.

$$P(\mathbf{Y}(t) \mid s(t) = p) = \frac{\exp(-0.5(\mathbf{Y}(t) - \mu_p)^t \Sigma_p^{-1} (\mathbf{Y}(t) - \mu_p))}{((2\Pi)^n \mid \Sigma_p \mid)^{1/2}} \tag{4.40}$$

where n is the number of network outputs; μ_p and Σ_p are evaluated as usual [Duda73].

It is possible to perform a global optimization of the ANN/dynamic programming hybrid. There are several ways to do that, depending on how the two algorithms are combined. For example, Patrick Haffner [Haff91] uses dynamic programming to obtain the sequence of output nodes that minimizes a cost computed by summing network outputs on the selected path while respecting some temporal constraints. It is then possible to back-propagate from that global cost, through those selected outputs, to all network parameters. On the other hand, [Dria91] uses the resulting best path to provide actual *targets* to the ANN. In [Tebe91], a related scheme is presented. Dynamic programming is used to minimize a cost which depends on how well networks associated to states predict the next input frame.

4.4.4 ANNs and Hidden Markov Models

Interesting papers have been published recently, describing attempts at combining ANNs with HMMs. In some of the proposed approaches (e.g., [Fran90, Brid90]) the activation value of each output node of the network are used as observation probabilities. The ANN is trained to compute these observation probabilities for the best sequence of states produced by the alignment. In [Fran90] the input data are aligned with the model of the spoken utterance using the Viterbi algorithm. This is used to provide a "high" target to nodes corresponding to states being on the selected path at a given time, and a "low" target to the other network outputs. The approach proposed by Bridle [Brid90] consists of viewing the HMM as a recurrent ANN with linear nodes and product nodes. Another ANN provides observation probabilities and gradient descent is used to estimate network parameters (including the parameters of the HMM). Other hybrid systems combining ANNs with HMMs

[Bour88, Mor90a] theoretically require that the ANN parameter estimation has converged to the global minimum in order to express the a posteriori probability $P(state \mid observation)$. Our previous work on hybrid models [Ben90d] used ANNs merely to compute an additional set of symbols considered as observations for discrete HMMs. A vector-quantized codebook was generated for these parameters and added to codebooks obtained for other popular parameter sets. This did not require any assumption on the network outputs but had the disadvantage that the ANN and the HMM were optimized separately.

Chapter 5

Integrating ANNs with Other Systems

In many applications, ANNs are integrated with other systems, either for preprocessing, for performing alternative processing, or postprocessing. We consider the possibility of an integration of ANNs with other systems to be an important quality.

For example, in section 3.1, we discussed the importance of preprocessing for ANNs applied to automatic speech recognition. ANNs can also transform signals in parallel with other tools: [Ben90d] describes the speaker-independent recognition of connected digits from the TI/NIST database using an ANN as one of the modules providing symbols to a discrete HMM. In addition to the ANN, standard preprocessing techniques (cepstrum, cepstrum derivatives: often employed with HMMs) are used with vector-quantization (see [Gray84] for a review) to provide other input symbols to the discrete HMM. The outputs of the ANN are also discretized in order to provide discrete codes to the HMM. The role of the ANN in this system is to discriminate among types of sounds with which a simpler recognition system (cepstrum + HMM) has difficulties (see also Section 4.4.4 about ANN/HMM hybrids in general, and Chapter 6 about the global optimization of an ANN/HMM hybrid).

Another example of successful interaction of ANNs with other algorithms has been implemented in an application to handwritten digit recognition [LeC89c] (see also section 3.3.3).

In this system, the back-propagation network is followed by a statistical decision process that acts as a postprocessor.

To introduce temporal constraints, Bourlard et al. [Bour88, Mor90a] integrate a multi-layer perceptron (trained with back-propagation) with dynamic programming (more specifically, the Viterbi algorithm). Several other researchers have combined ANNs with dynamic programming. We have used such a combination for the analysis of amino-acid sequences and for the recognition of phonemes (see sections 4.4 and 6.7).

5.1 Advantages and Disadvantages of Current Algorithms for ANNs

An analysis of advantages and disadvantages of ANNs will help to motivate the use of ANNs with other systems. Such an analysis should help us to design hybrids that succeed in profiting as much as possible from ANNs advantages, while attempting to circumvent their disadvantages.

5.1.1 Advantages

- **Flexibility**: when comparing ANNs with other non-parametric inference tools, such as k-Nearest Neighbor or Parzen windows, we found that ANNs are particularly well suited to incorporate a priori knowledge with learning from examples. This is important in order to find the right equilibrium of bias and variance to maximize generalization given the size of the training set (see [Gema91] for a discussion of the bias/variance dilemma). In Chapter 3, we studied various ways to introduce such bias or constraints into ANNs, while using the training data to tune the network parameters. In addition to allowing designers a ready integration of domain knowledge, ANNs are also flexible regarding their integration with other tools. In Section 5.2, we propose a method for integrating ANNs with other tools in a modular way and performing global optimization. In Section 4.4 we discuss various hybrids for sequence recognition and in Chapter 6 we describe and evaluate an algorithm for performing

global optimization of an ANN/HMM hybrid.

- **Robustness**: As demonstrated in experiments described in this dissertation (see chapters 3 and 6 in particular), ANNs seem to be able to generalize well in difficult problems such as speaker-independent continuous speech recognition, which present a lot of variability and noise. In addition to robustness to variations in the inputs, ANNs can be robust to variations in the operation performed by each node. Because of the distributed representations usually found in the hidden layers of ANNs, physical implementations of ANNs are naturally fault-tolerant: if a synapse or a neuron fails, it does not disrupt the whole network operation. Instead, performance is seen to degrade gracefully as the number of faulty elements increases [Dzwo91, Stev90]. This is a very attractive feature for VLSI parallel implementations of ANNs.

- **Expressive power**: As shown by several authors, multi-layer networks can in theory approximate arbitrary continuous transformations if the number of hidden units is chosen in function of the precision of the approximation [Cybe89, Horn89, Funa89, Stin89]. Furthermore, because ANNs are non-parametric inference systems, they do not make any explicit assumptions on the input data (on the other hand, for example, HMMs assume a certain statistical model for the input distribution corresponding to each speech unit). This may be an important property for complex problems for which proper statistical models are hard to identify. On the other hand, it is still possible to introduce a priori bias in the design of ANNs.

- **Discriminant learning**: With the LMS criterion (used for example with the back-propagation trained networks), the ANN models *discrimination surfaces* between classes. On the other hand, with maximum-likelihood models (e.g., HMMs trained with maximum likelihood estimation, or Boltzmann machines[1]), one models the complete input/output distribution. An advantage of discriminant learning is that it concentrates its efforts (and the use of parameters) for the ultimate goal of learning, that is, reducing classification error, which are usually due to confusions of one class for another. The input probability density function for each class contains more

[1] Boltzmann machines in which the inputs are not clamped during the free running phase learn to generate the joint input/output distribution. On the other hand, if the inputs are clamped in both phases, the network learns about the conditional distribution of the output given the input.

information than the corresponding discrimination surfaces between classes, thus it requires less parameters to model discriminant surfaces than a complete input/output distribution. For this reason, discriminant methods might perform better given the same (too small) training set (see Section 1.3.3 on the relation between the number of parameters, generalization and training set size). However, there may be other advantages with maximum-likelihood methods. For example, in the case of HMMs, a fast estimation algorithm exists for maximum-likelihood estimation (the Baum-Welsh algorithm [Bahl83]) whereas discriminant learning (Maximum Mutual Information Estimation) is performed with a much slower gradient descent. This quality of ANNs (building precise discrimination surfaces) may explain their success at performing fine spectral discriminations for phoneme recognition [Lipp89][2].

- **Large input space:** Several experiments in speech recognition indicate that ANNs trained with back-propagation can use a large input space with many correlated features to their advantage (see Section 3.1 and [Ross89]). This can be very useful when many such alternative input features are available and can be combined to help recognition (see for example the experiments described in section 3.1.2 and [Ben91a]). In contrast, other methods used for speech recognition, such as HMMs, which are model-based methods, require a small number of observation (input) features in order to generalize correctly given the relatively small[3] training sets usually available.

- **Parallel Implementation:** ANNs are attractive as computing machines because they can be naturally implemented in parallel hardware. Hence several research teams are now working on such implementations, some in digital VLSI [LeC89d], some in analog VLSI [Mead89], and others with optical or electro-optical implementations [Psal90].

- **A Priori Knowledge about Learning Architectures:** In the same way that we found that a priori knowledge about a task was very helpful in designing ANNs for this task, we can conjecture that using information about how the *brain* works to constrain the design of learning algorithms may help us create systems that are successful at solving the kind of problems that the brain can handle. Although current

[2]However, this does not guarantee improved *word* recognition, as shown by the experiments of [Bour89]

[3]given the number of input features and the complexity of the task

ANN models already incorporate some basic principles of brain operation, they have many biologically implausible features. In section 7.4, we argue about using even more biological constraints in the design of ANNs, in particular for their learning algorithms.

- **Sharing of Internal Features and Distributed Representation**: In an ANN with several outputs receiving their inputs from the same hidden units, these hidden units will tend to learn functions that are useful for several of the output units. This in general will lead to compact and distributed internal representations. The mapping to these representations will require fewer degrees of freedom than if each class is modeled separately (as for HMMs, for example). In the frequent case in which the outputs of an ANN represent related concepts, the sharing of internal features may improve generalization because fewer parameters are required and the function of hidden units is more constrained, since each of them is trained with a greater number of examples.

5.1.2 Disadvantages

- **Training Time**: Most current ANN models are very CPU-intensive, especially if simulated on sequential machines. However, a lot of research effort is invested in improving convergence time of current algorithms and creating faster learning algorithms. See for example an alternative to standard back-propagation in Chapter 7, as well as some acceleration techniques mentioned in section 2.2.1. Although training time is important for developing a recognition system, it is also important to consider recognition time, which is generally short with ANNs, especially if they are implemented in parallel hardware.

- **Temporal Modeling**: Although ANNs with delays and recurrence can in theory be used to represent any temporal structure, the experiments described in Chapter 6 indicate that the architectures we are currently using are inefficient at capturing some important aspects of temporal structure, such as the duration of a phoneme. Similar observations have prompted many researchers, including ourselves, to look at ways to combine ANNs with other tools that had been previously seen to be useful

at analyzing the temporal structure of signals such as speech. Such combinations are described in Chapters 4 and 6, in particular with dynamic programming methods and with HMMs.

- **Theoretical Underpinnings**: More theoretical work is needed to analyze the capabilities of neural networks, and in particular of their learning algorithms (e.g., convergence proofs for ANNs with hidden units, and on the optimal choices of architectures).

5.2 Modularization and Global Optimization

As discussed in sections 3.4 and 2.2.1, modularization is a promising way to build complex systems for learning to perform a task when not enough training data or training time are available. It may significantly help the convergence, as well as generalization of ANNs. Modularization allows incorporation of some a priori knowledge about the problem and its solution, in particular about the task decomposition. In such a system, each module may correspond to a particular subtask.

A practical question concerning modular systems is how to integrate all the modules together in the "best" possible way. An answer to this question comes from a *common* definition of what is *best*: a global error criterion or cost function, i.e., a common goal of learning for all the modules. If such a global cost function is defined, then in order to allow the parameters of the modular system to be optimal for this cost function, one can perform a *global optimization* of all these parameters.

5.2.1 Why Global Optimization is Necessary

It is easy to imagine cases in which each module in a modular system is trained separately, which is suboptimal for the given complete architecture, even though each module may have reached a local optimum of its cost function.

Suppose a modular system contains two modules: network 1 and network 2, with outputs

$y_1(x)$ and $y_2(y1(x))$, respectively, and parameter sets $\theta_1 = \{\theta_{1i}\}$ and $\theta_2 = \{\theta_{2i}\}$. The input of network 1 is the input x of the system, and the input of network 2 is $y_1(x)$, i.e., network 1 feeds network 2, as depicted in Figure 5.1. Let us suppose this architecture design corresponds to an ideal function decomposition:

$$F(x) = f(g(x)) \tag{5.1}$$

Consequently the target outputs for network 1 and network 2 when x is given as input are $g(x)$ and $F(x)$, respectively. With the Least Mean Square criterion, for example, the costs to be minimized by network 1 and network 2 are respectively

$$C_1 = E_T(\mid y_1(x) - g(x) \mid^2) \tag{5.2}$$

$$C_2 = E_T(\mid y_2(y_1(x)) - F(x) \mid^2) \tag{5.3}$$

where E_T is expected value over the training set T. At the end of training, assuming a *local* optimum was reached but a zero mean square was not achieved[4]:

$$\frac{\partial C_1}{\partial \theta_{1i}} = 0 \text{ and } C_1 \neq 0 \tag{5.4}$$

and

$$\frac{\partial C_2}{\partial \theta_{2i}} = 0 \text{ and } C_2 \neq 0 \tag{5.5}$$

Since $C_2 \neq 0$, there is in general a non-zero gradient $\frac{\partial C_2}{\partial y_1}$, i.e., there exists a way to keep θ_2 fixed and change y_1 that would further reduce C_2, except in the trivially uninteresting case in which the input y_1 to network 2 does not influence its output y_2.

Now since

$$\frac{\partial C_2}{\partial \theta_{1i}} = \frac{\partial C_2}{\partial y_1}\frac{\partial y_1}{\partial \theta_{1i}} \tag{5.6}$$

We have

$$\frac{\partial C_2}{\partial \theta_{1i}} \neq \frac{\partial C_1}{\partial \theta_{1i}} = 0, \tag{5.7}$$

except in the uninteresting case in which θ_1 does not influence y_1. Because of the previous inequality, one can reduce the *global* cost C_2 by *further modifying* θ_1 along the direction of the gradient $\frac{\partial C_2}{\partial \theta_{1i}}$. Hence separate training is suboptimal, because in general each module cannot perform *perfectly* the desired transformations from the preconceived task decomposition. On the other hand, a final tuning based on the global optimization of the all the parameters of network 1 and network 2 can find an optimum of the global cost C_2.

[4]This is generally the case for sufficiently interesting, thus complex, problems

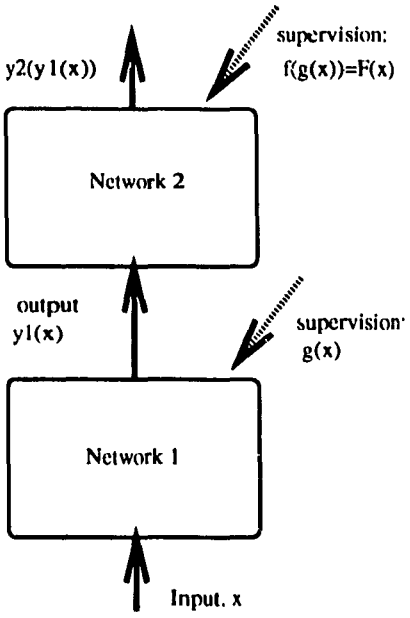


Figure 5.1: Typical example in which global optimization is necessary: network 1 feeds network 2 and the decomposition $F(x) = f(g(x))$ is not learned perfectly by each module.

5.2.2 Global Optimization of a Modular System with Stochastic Gradient Descent

The global error measure or cost function may depend directly on the outputs of one or more modules, but all the modules directly or indirectly influence this cost. If we want to use stochastic gradient descent as a learning method for one or more modules (as we have done for training the ANNs described in this thesis), then the following characteristic of the modules allows a global optimization. If we define a module as a subsystem having some inputs and some outputs, then we require that it be possible to compute the first derivative of each of the module's outputs with respect to each of its inputs. Furthermore, if the module is not static (e.g., it is a recurrent network), then one must be able to compute the first derivative of each of the module's outputs for frame t w.r.t each of its inputs for previous frames $\tau \leq t$. If this derivative does not exist[5] it is not possible to perform gradient descent, but other optimization methods may be applicable (e.g., simulated annealing,

[5] For example, if the outputs are boolean functions of the inputs, this derivative is zero everywhere except at certain points at which it is infinite

genetic algorithms).

Let us briefly formalize these remarks. Consider a set of modules that are interconnected in a graph, following the formalism introduced in section 2.1 for a static system and generalized in section 4.3 for a dynamic system. We will consider here the general case of an arbitrary connectivity among and within modules, which thus could be recurrent, and of input patterns which may be sequences (a static pattern is just a special case of a sequence of length 1).

Without loss of generality, let us further decompose the modular structure in such a way that each remaining module is memory-less, i.e., has no internal delay or recurrence: these are solely represented with the connectivity of the graph $G = (M, L)$, where M is a set of modules $\{m_i\}$ and L is a set of links $\{l_{ij}$: link from $m_{s_{ij}}$ to m_i with delay $d_{ij}\}$. Such a decomposition is illustrated in figure 5.2. Let us denote each module's operation as follows:

$$\mathbf{y}_i(t) = \mathbf{F}_i(\theta_i, Y_i(t)) \tag{5.8}$$

where $\mathbf{y}_i = (y_{i1}, y_{i2}, ..., y_{iN_i})$ is the output vector of module i, $\mathbf{F}_i(\cdot)$ is a vector function for module m_i that depends on two sets: θ_i is a set of parameters $\{\theta_{ij}\}$ (e.g., weights of an ANN) for module m_i, whereas $Y_i(t)$ is the set of vector outputs $\mathbf{y}_k(t - d_{ij})$ of modules m_k at frames $t - d_{ij}$ such that $l_{ij} \in L$ and $k = s_{ij}$.

This formalism is a straightforward generalization to vectors of the one used to define recurrent networks in Chapter 4. Such a modular system can thus also be trained with similar learning algorithms. For example, let us consider back-propagation in time and stochastic gradient descent:

$$\Delta^p \theta_{ij} = -\epsilon \frac{\partial^+ C_p}{\partial \theta_{ij}} \tag{5.9}$$

where $\frac{\partial^+ u}{\partial v}$ denotes an ordered derivative, as defined in section 4.3.1. When a particular sequence p is presented, we wish to compute how the corresponding error C_p is influenced by all the parameters of the system:

$$\frac{\partial^+ C_p}{\partial \theta_{ij}} = \sum_t \sum_k \frac{\partial^+ C_p}{\partial y_{ik}(t)} \frac{\partial y_{ik}(t)}{\partial \theta_{ij}} \tag{5.10}$$

The second factor in the above equation depends on the definition of module m_i's operation

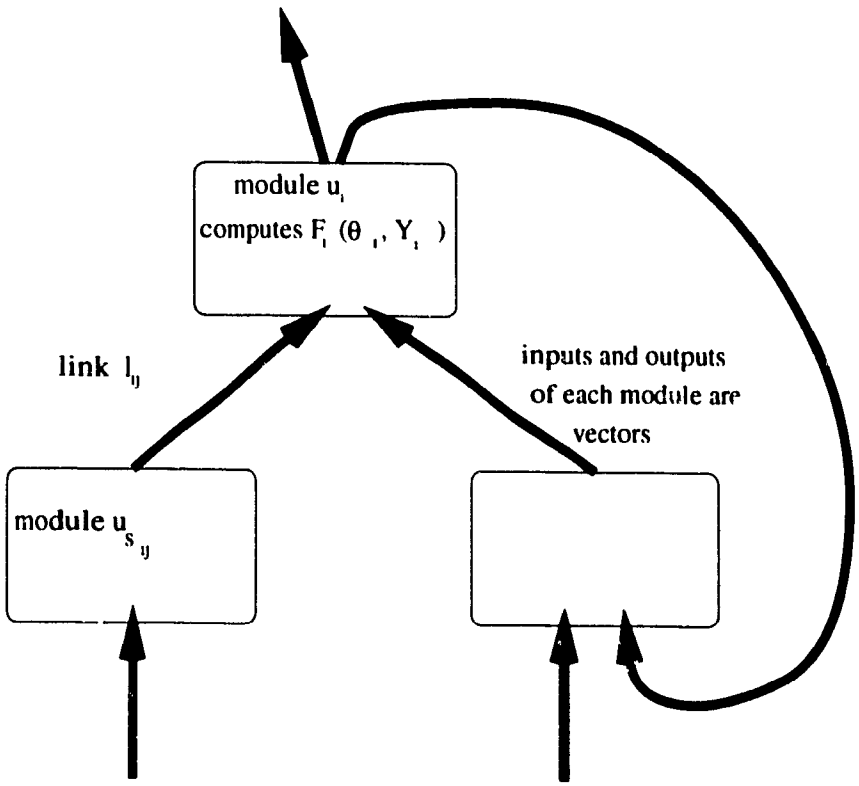


Figure 5.2: General modular system that can be trained with stochastic gradient descent and back-propagation through time.

($\mathbf{F}_i(\cdot)$). The first factor could be computed recursively by back-propagation in time:

$$\frac{\partial^+ C_p}{\partial y_{ik}(t)} = \sum_{m,q=i} \sum_n \frac{\partial^+ C_p}{\partial y_{mn}(t+d_{mq})} \frac{\partial y_{mn}(t+d_{mq})}{\partial y_{ik}(t)} + \frac{\partial C_p}{\partial y_{ik}(t)} \tag{5.11}$$

The first factor is similar to the left hand side of the equation and can be computed recursively. The second factor depends on the definition of $\mathbf{F}_m(\cdot)$. The last term is zero unless supervision is provided to the k^{th} output of module i at time t. In that case its exact value depends on the choice of objective function. For the LMS criterion, it is as in equation 4.5.

5.3 Suggestions for ANN Design

Tools other than ANNs were required in the hybrid systems of sections 4.4.2 and 6.7, even though in theory they could learn any continuous mapping. It could be because of inherent weaknesses in current ANN algorithms and architectures, or it could be because we have not yet found ways to appropriately design such architectures and algorithms for ANNs. A way to do that could be to learn how these other tools efficiently perform their task in order to help us design ANNs that can do it as well or better.

In the experiment of section 6.7, an interesting result was obtained: by adding a dynamic programming postprocessor, which requires as few as 22 free parameters (in its simplest form, with the duration probabilities only), the total error of the system is almost halved. The ANN is recurrent, with delays, and has over 20000 parameters. This suggests that such simple temporal constraints as durations of phonemes are not efficiently captured by multi-layer ANNs with commonly used architectures, even those with delays and recurrence. This should encourage us either to search for better architectures for ANNs in order to model temporal structure, or simply to construct hybrids, as we have done in chapters 4 and 6. An extreme case of the first option is to view those non-connectionist tools such as HMMs as very particular types of ANNs (with products, sums and recurrence), as did J. Bridle in [Bridle90].

Chapter 6

A Hybrid of ANNs and HMMs with Global Optimization

In this chapter, we describe an ANN/HMM hybrid of which all the parameters can be simultaneously estimated in relation to a single cost function. As argued in Chapter 5, hybrids and modular systems may be advantageous for both convergence time and generalization performance. They may allow taking advantage of the strengths of the various modules and algorithms. However, performing a global optimization of the combined system allows the learning algorithms to yield a better value of the cost function. In the first section of this chapter we consider how the outputs of an ANN might be used to express a probability density function. We choose to make some assumptions about the form of the input distribution in order to obtain a computationally simple algorithm for performing the global optimization of the ANN/HMM hybrid. Then we derive equations for performing global optimization of an ANN/HMM hybrid in which the ANN supplies observations to the HMM, as shown in Figure 6.2. We show how to compute the gradient of the HMM optimization criterion with respect to the ANN outputs. At the end of this chapter, we evaluate this new algorithm by comparing it with other systems, such as an ANN/DP hybrid, an ANN alone or an HMM alone, with speaker-independent continuous speech recognition. Stochastic gradient descent is used for the ANN parameters and two alternatives for performing parameter update in the HMM within the hybrid are evaluated.

It is found that the HMM parameters can also be advantageously updated with an online algorithm.

6.1 Expressing a Density in Terms of a Continuous Transformation of some Observations

Let X and Y be random variables that take values in $\Omega_x \subset \Re^{n_x}$ and $\Omega_y \subset \Re^{n_y}$, respectively. Suppose that Y is a deterministic parametric function of X:

$$Y = y(X;\omega) \tag{6.1}$$

where Y and X are vectors of dimension n_y and n_x, respectively, and ω is a set of parameters. Let us suppose that Y has been observed and a parametric model has been assumed for its distribution, and its probability density function (p.d.f.) is $f_y(y;\theta)$, where θ is a set of parameters.

6.1.1 Gradient Computation

We would like to express a probability density function for X in terms of Y, for example,

$$f_x(x;\omega,\theta) = f_y(y(x;\omega),\theta)\, g(x) \tag{6.2}$$

If a cost function C is to be minimized and this cost function depends on f_x, we need in general to compute $\frac{\partial C}{\partial \theta}$ and $\frac{\partial C}{\partial \omega}$, which depend on $\frac{\partial C}{\partial f_x}$, $\frac{\partial f_x}{\partial \theta}$, and $\frac{\partial f_x}{\partial \omega}$.

$\frac{\partial f_x}{\partial \theta}$ is simply

$$\frac{\partial f_x}{\partial \theta} = \frac{\partial f_y}{\partial \theta} g(x) \tag{6.3}$$

since $g(x)$ is independent of θ.

In the case of maximum likelihood estimation (MLE), to estimate θ we can solve the following equation, as if only Y were considered,

$$\frac{\partial f_y}{\partial \theta} = 0 \tag{6.4}$$

For example, when f_y is a normal density or a Gaussian mixture, we obtain the usual maximum likelihood estimation or re-estimation formulae [Duda73, Rabi89], in function of Y.

Since $g(x)$ may be influenced by ω we may have to compute the gradient of f_x with respect to ω as follows:

$$\frac{\partial f_x}{\partial \omega} = \sum_{i=1}^{n_y} \frac{\partial f_y}{\partial y_i} \frac{\partial y_i}{\partial \omega} g(x) + f_y(y(x;\omega),\theta) \frac{\partial g}{\partial \omega} \tag{6.5}$$

where y_i denotes the i^{th} element of the vector y. When $y(x;\omega)$ is non-linear, e.g., the output of an ANN, ω should be obtained by descending the gradient defined in equation 6.5.

6.1.2 Integral Equation

For f_x to be a proper p.d.f. for X, we have to find a non-negative function $g(x)$ such that

$$\int_{\Omega_x} f_y(y(x;\omega);\theta)\, g(x)\, dx = 1 \tag{6.6}$$

Unfortunately, this is an underdetermined integral equation in g. In order to find the form of a solution for g we will start from the equivalent integral for Y:

$$\int_{\Omega_y} f_y(y;\theta)\, dy = 1 \tag{6.7}$$

Let us first consider the simple case in which $n_x = n_y = n$. Then we can do a change of variable $y = y(x;\omega)$ in the above integral and obtain the following:

$$\int_{\Omega_x} f_y(y(x;\omega),\theta) \mid \frac{\partial(y_1, y_2, \ldots y_n)}{\partial(x_1, x_2, \ldots x_n)} \mid dx = 1 \tag{6.8}$$

Hence $g(x)$ could be chosen to be the absolute value of the determinant of the Jacobian of y with respect to x, evaluated at x.

Unfortunately, we are generally interested in transformations from $\Omega_x \subset \Re^{n_x}$ to $\Omega_y \subset \Re^{n_y}$ that reduce the dimensionality of the data, i.e., $n_y < n_x$.

6.1.3 Case of a Linear Transformation

To better understand the problem when $n_y < n_x$, let us consider the case in which $y(x)$ is a linear transformation from $\Omega_x \subset \Re^{n_x}$ to $\Omega_y \subset \Re^{n_y}$.

$$y = Tx \tag{6.9}$$

where T is a $n_y \times n_x$ matrix.

Let us decompose T with a singular value decomposition:

$$T = UDV^t \tag{6.10}$$

where U and V^t are square unitary[1] matrices of dimension $n_y \times n_y$ and $n_x \times n_x$, respectively. D is a rectangular matrix of dimension $n_y \times n_x$, with zeros everywhere except in the "diagonal" from $(1,1)$ to (n_y, n_y). Let us call s the product of the singular values:

$$s = \prod_{i}^{n_y} D_{ii} \tag{6.11}$$

If $n_x = n_y$, then $s = det(D) = det(T)$ and $\mid s \mid = \mid det(\frac{(y_1,\ldots,y_n)}{(x_1,\ldots,x_n)}) \mid$, since $det(U) = det(V) = 1$.

As shown in Figure 6.1 for the case $n_y = 1$ and $n_x = 2$, lines in $\Re^2$ are mapped by V^t into vertical lines in $\Re^2$, then by D into points in $\Re$, then by U into points in $\Re$ that represent values of y:

$$y = TX = U(D(V^t x)) \tag{6.12}$$

Let us define an intermediate random variable Z, with values $z \in \Omega_z, \Omega_z \subset \Re^{n_x}$, as follows:

$$z = V^t x \tag{6.13}$$

Thus y can be defined in terms of z:

$$y = UDz \tag{6.14}$$

Because D has only zeros after the column n_y, y depends only on the first n_y elements of z. Let us decompose the variable Z into two variables Z' and Z'', such that $Z = (Z', Z'')$,

[1] Unitary matrices have orthonormal columns, hence $U^t U = I$ and $det(U) = 1$.

i.e.,

$$z' = (z_1, \ldots, z_{n_y})$$
$$z'' = (z_{n_y+1}, \ldots, z_{n_x}) \tag{6.15}$$

Let us denote by $\Omega_{z'}$ and $\Omega_{z''}$ the corresponding subspaces of Ω_z. In this way, we can express the total derivative of y in function of the total derivative of z',

$$dy_1 dy_2 \ldots dy_{n_y} = \left| \frac{\partial(y_1, \ldots, y_{n_y})}{\partial(z_1, \ldots, z_{n_y})} \right| dz_1 dz_2 \ldots dz_{n_y} = |\, s \,|\, dz_1 dz_2 \ldots dz_{n_y} \tag{6.16}$$

or in shorter notation,

$$dy = |\, s \,|\, dz' \tag{6.17}$$

since the determinant of U is 1, and s is the determinant of the truncated matrix D' obtained by keeping only the first n_y columns of D. We can make a change of variable $y = UD'z'$ in equation 6.7:

$$1 = \int_{\Omega_y} f_y(y;\theta)\, dy = \int_{\Omega_{z'}} f_y(UD'z';\theta)\, s\, dz' \tag{6.18}$$

To account for Z'', the remaining dimensions of Z, we have to assume some distribution along each hyperplane[2] of dimension $n_x - n_y$ in $\Re^{n_x}$ that corresponds to a point y. In terms of probabilities, this distribution is the p.d.f. for Z'' when Z' is given, and corresponds to a conditional p.d.f. for X given that $Y = y$. Let us call that fonction $h'_{z'}(z'')$:

$$\int_{\Omega_{z''}} h'_{z'}(z'')\, dz'' = 1 \tag{6.19}$$

By specifying this p.d.f. we constrain sufficiently $g(x)$ to allow us to find a solution similar to equation 6.8. Let us multiply the integrals in equations 6.18 and 6.19 together:

$$\begin{aligned} 1 &= \int_{\Omega_{z''}} h'_{z'}(z'')\, dz'' \int_{\Omega_{z'}} f_y(UD'z';\theta)\, s\, dz' \\ &= \int_{\Omega_z} f_y(UD'z';\theta)\, h'_{z'}(z'')\, s\, dz \end{aligned} \tag{6.20}$$

Now we can make a change of variable $z \to x$, $z = V^t x$, hence $UD'z' = UDV^t x = Tx = y$:

$$1 = \int_{\Omega_x} f_y(y(x);\theta)\, h_{y(x)}(x)\, s\, dx \tag{6.21}$$

[2]e.g. a straight line when $n_x = 2$, as in Figure 6.1.

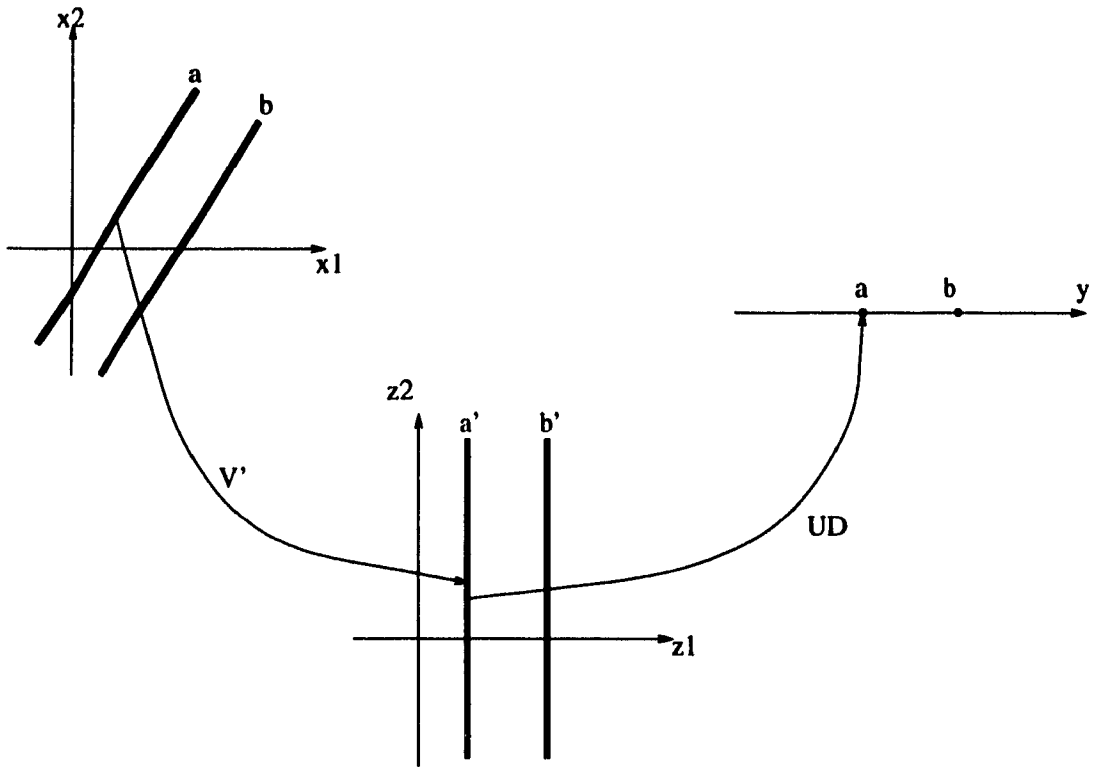


Figure 6.1: Mapping from Ω_x to Ω_y, going through the intermediate space Ω_z obtained by singular value decomposition of the Jacobian of Y w.r.t. X. Here we consider a mapping from $\Re^2$ to $\Re$, and either the mapping is linear or a is very close to b.

where $h_{y(x)}(x) = h'_{z'}(z'')$. Note that the Jacobian of the mapping $x \rightarrow z$ is 1.

Hence we have found how to express a solution in the case where $y = Tx$ is a linear function of x:

$$g(x) = h_y(x)s \tag{6.22}$$

where s is the product of the singular values of T and $h_y(x)$ is a conditional p.d.f. for X given $Y = y$, to be specified, for hypersurfaces of dimension $n_x - n_y$ in Ω_x.

6.1.4 Case of a Non-Linear Transformation

We will now attempt to generalize the solution previously obtained with the change of variable $y = y(x)$ to the case $n_y < n_x$ and $y(x)$ a non-linear transformation.

Let T be the Jacobian of the transformation $Y \rightarrow X$. Let us suppose a singular value decomposition for T, as before:

$$T = UDV^t \tag{6.23}$$

where U and V^t are square unitary matrices of dimension $n_y \times n_y$ and $n_x \times n_x$, respectively. D is a rectangular matrix of dimension $n_y \times n_x$, with zeros everywhere except in the "diagonal" from $(1,1)$ to (n_y, n_y). Let us call s the product of the singular values:

$$s = \prod_{i}^{n_y} D_{ii} \tag{6.24}$$

Let us again define an intermediate random variable Z of dimension n_x that has the following property:

$$\frac{\partial(z_1, z_2, ..., z_{n_x})}{\partial(x_1, x_2, ..., x_{n_x})} = V^t \tag{6.25}$$

where V^t varies in function of x.

As for the linear case, let us decompose Z into Z' and Z'':

$$\begin{aligned} z' &= (z_1, ..., z_{n_y}) \\ z'' &= (z_{n_y+1}, ..., z_{n_x}) \end{aligned} \tag{6.26}$$

Then we can perform the same changes of variables as in the linear case, except now s, U and V^t depend on x:

$$1 = \int_{\Omega_y} f_y(y;\theta)\, dy \;=\; \int_{\Omega_{z'}} f_y(UD'z';\theta)\; s\; dz' \tag{6.27}$$

We also have to specify a p.d.f. $h'_{z'}(z'')$ for Z'' when Z' is given, i.e., a conditional p.d.f. $h_y(x)$ for X when $Y = y$ is given.

$$\int_{\Omega_{z''}} h'_{z'}(z'')\, dz'' \;= 1 \tag{6.28}$$

We obtain again the form of the solution by multiplication of the two integrals in equations 6.27 and 6.28:

$$1 = \int_{\Omega_x} f_y(y(x);\theta)\, h_{y(x)}(x) s\; dx, \tag{6.29}$$

i.e., we can choose $g(x)$ as follows:

$$g(x) = h_y(x) s(x) \tag{6.30}$$

6.1.5 Choice of Distribution

Let us consider the following uniform distribution for z'', when z' is given:

$$h_{z'}(z'') = \begin{cases} K(z') & \text{if } (z', z'') \in A'(z') \\ 0 & \text{otherwise} \end{cases} \tag{6.31}$$

where $A'(z')$ is a hypersurface in Ω_z that is spanned by $(z_{n_y+1}, ..., z_{n_x})$, i.e., that is orthogonal to the axes of $\Omega_{z'}$.

The constant K is defined as follows:

$$K(z') = \frac{1}{\int_{A'(z')} dz''} \tag{6.32}$$

Interestingly, the area of $A' \subset \Omega_z$ is the same as the area of the corresponding surface A in Ω_x, because the Jacobian of the mapping $x \to z$ is V^t, whose determinant is 1:

$$\text{area}(A') = \int_{(z',z'')\in A'(z')} dz'' = \int_{A'} dz = \int_A dx = \text{area}(A) \tag{6.33}$$

where A is the hypersurface in Ω_x, corresponding to a given z' and to all $z'' \in A'$.

Choosing a uniform distribution and a certain $K(z')$ is thus equivalent to assuming that the domain of the random variable X covers a certain bounded volume, and that for each value of Y, X is uniformly distributed along the hypersurface that corresponds to $Y = y(x)$.

Unfortunately, when $y(x)$ is a non-linear transformation, there does not appear to be any easy solution to the problem of finding the appropriate constants K for each value of Y, when the domain of X is given. One way to do so would be following:

1. Compute the function $z(x)$: this requires to solve the simultaneous partial differential equations given by $\frac{\partial(z_1, \ldots, z_{n_x})}{\partial(x_1, \ldots, x_{n_x})} = V'$. This matrix equation can be transformed into n_x systems of functional equations for each z_i, each with n_x equations and n_x variables.
2. Map the known domain Ω_x to $\Omega_z = z(\Omega_x)$.
3. Compute the area of $A'(z')$ (for all z') that lie within Ω_z.

Although choosing a proper $h_y(x)$ remains an open problem, the following considerations may justify a simple choice for this distribution. If $Y = y(x)$ is the output of an ANN which reduces the dimensionality of X, it means that some "components" of X have been discarded. These are precisely represented by Z'', which is obtained by a non-linear mapping from X as already described. Y depends only on the first n_y components of Z, that is Z'. If we wish to use the output of the ANN for some *classification* task, then we are not really interested in *modeling*[3] those aspects of the input data that correspond to variations along $\Omega_{Z''}$. This may help justify the choice of an arbitrary distribution for Z'' which requires no parameters and simplifies the estimation of ω. A choice of distribution which would considerably reduce the computational complexity of the estimation of ω is the following:

$$h_y(x) = \frac{K}{s(x)} \tag{6.34}$$

where K is some constant that ensures that the integral of $h_y(x)$ over the bounded hypersurface A is equal to 1. In that case we can write the p.d.f. for X as follows:

$$f_x(x; \omega, \theta) = f_y(y(x; \omega); \theta) K \tag{6.35}$$

[3] i.e., using some degrees of freedom.

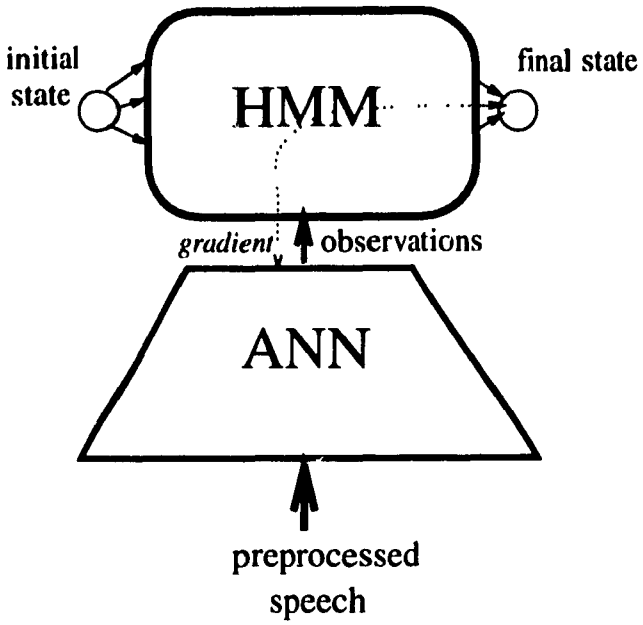


Figure 6.2: Basic architecture of the ANN/HMM hybrid: the ANN supplies observations to the HMM; the HMM returns an error gradient to the ANN.

When trying to minimize or maximize f_x, one can drop the constant K, whose precise value is unknown in general, since its value does not affect the estimation of ω or θ.

In this chapter we consider a hybrid of ANNs and HMMs. In analogy with the concepts introduced in this section, X is the input of the ANN, i.e. the preprocessed speech signal, and Y represents the output of the ANN and the observations of the HMM. To optimize the ANN/HMM hybrid we must compute the gradient of a common cost function with respect to all the parameters of the hybrid. These parameters are ω and θ for the ANN and the HMM, respectively. The cost functions that we consider for the hybrid can all be expressed in terms of the likelihood of the observed data given some model. Hence to estimate the parameters of the system we need to compute the gradient of this likelihood with respect to these parameters. In sections 6.3 and 6.4 we show how to compute the gradient of the likelihood of the HMM observations, Y. By the previous arguments, we can consider this gradient to be proportional to the gradient of the likelihood of the input data X, when making some assumptions about the distribution of X along hypersurfaces that correspond to fixed values of Y.

6.2 Hidden Markov Models

Hidden Markov Models (HMMs) are parametric stochastic models of non-stationary processes. Although they have been known for many decades, efficient algorithms for estimating their parameters have been known only since the 60's [Baum63, Baum70]. An HMM models a stochastic process generated by an underlying Markov chain with a finite number of states, and a set of distributions associated to each state in the model [Levi83]. For computational reasons, applications of HMMs to automatic speech recognition are generally limited to Markov chains of order 1, i.e., transition probabilities depend only on the current and the previous state.

An HMM can have N states $q_1, q_2, q_3, ...q_N$, and it can be specified in terms of

- initial state probabilities $\pi_1, \pi_2, ...\pi_N$, where π_i is the probability of being in state i at time 0.
- a state transition matrix $A = [a_{ij}]$, where a_{ij} is the probability $P(q_j \text{ at } t+1 \mid q_i \text{ at } t)$.
- a random process associated to each state, described either by a discrete or a continuous distribution: $b_{i,t} = P(\mathbf{Y}_t \mid q_i \text{ at } t)$, where $\mathbf{Y}_t$ is the observation vector at time t.

Algorithms exist to estimate iteratively these probabilities (or the parameters of probability distributions). These algorithms are guaranteed to converge to a local maximum of a criterion function. For example, the Baum-Welsh algorithm [Rabi89, Baum70, Levi83] performs maximization of the likelihood:

$$L = P(\mathbf{Y}_1^T \mid M) \tag{6.36}$$

where M represents the model, with all its parameters.

In the application of HMMs to speech recognition, two types of problems are distinguished: isolated words (or units) recognition, and continuous speech recognition. In the first case, a single model per word is used and the word corresponding to the model with greater likelihood is selected during recognition. In the second case, several units are concatenated.

During recognition, the most likely sequence of units can be obtained with an algorithm based on dynamic programming. In the following sections, we derive equations for computing the gradient of two HMM criterion functions (maximum likelihood and maximum mutual information) for the problem of isolated and continuous speech recognition.

6.3 Definitions

Let Y_t be the vector of ANN outputs at time t. These outputs are considered as observations of a continuous density HMM used in the scheme shown in Figure 6.2. Here, we assume HMMs with a single final state per model. Let Y_1^T be the whole observation sequence for the HMM, T is the length of the observation sequence, and Y_t a particular observation, made when the HMM is in the state S_t at time t. Let a_{ij} be the transition probability from state q_i to state q_j. The probability that the HMM generates Y_t in state S_t at time t is denoted as $b_{i,t} = P(Y_t \mid S_t = q_i)$. Parameter estimation algorithms [Rabi89] allow one to compute recursively with a dynamic programming procedure the following probabilities for partial sequences (up to time t, from time $t+1$ on):

$$\begin{aligned} \alpha_{i,t} &= P(Y_1^t \text{ and } S_t = q_i \mid model) &= b_{i,t}\sum_j a_{ji}\alpha_{j,t-1} \\ \beta_{i,t} &= P(Y_{t+1}^T \mid S_t = q_i \text{ and } model) &= \sum_j a_{ij}b_{j,t+1}\beta_{j,t+1} \end{aligned} \tag{6.37}$$

with appropriate boundary conditions [Lee 89]. Y_a^b is the subsequence of observations from frame a to frame b. If the task is to model isolated units (e.g., isolated words), there will be multiple models ω, one for each unit. For continuous speech recognition, unit models (e.g., phonemes) are concatenated to make word and sentence models.

The likelihood that an HMM has generated the observation corresponding to the pronunciation of the unit ω is $L_\omega = \alpha_{F_\omega,T}$, where F_ω is the final state for model ω. HMM parameters can be estimated with different criteria. Two popular criteria are Maximum Likelihood (ML) and Maximum Mutual Information (MMI). Modeling with these two criteria is discussed in [Nada88]. Maximum Likelihood Estimation (MLE) is based on the maximization of the criterion C expressed as $C_{MLE} = L_c$. For isolated unit modeling, c represents the pronounced unit. For continuous speech, c represents the training model, a constrained

model built from the concatenatation of the units corresponding to the training sentence.

6.3.1 Maximum Mutual Information Criterion

Training an HMM with the Maximum Likelihood criterion is based on the assumption that the true input distribution is a member of the family of distributions covered by the choice of model, i.e., the choice of model topology and the choice of the form of the observation distributions. However, this assumption is not exact in general. With the Maximum Mutual Information (MMI) criterion, one searches for parameters of the HMM that minimize the recognizer's uncertainty of what the correct word sequence is, given the observations [Bahl87]. The resulting criterion has the advantage in comparison with MLE of being discriminant, i.e., it attempts to model each class as it differs from the others, rather than independently from the others.

Assuming we have a set of alternative models $\{model_w\}$ that could have generated the observation $\mathbf{Y}_1^T$, the mutual information between the correct model c and the observation Y_1^T is

$$I = \log(\frac{P(Y_1^T, model_c)}{P(Y_1^T)P(model_c)}) = \log(\frac{P(Y_1^T \mid model_c)}{\sum_\omega P(Y_1^T \mid model_\omega)P(model_\omega)}) \tag{6.38}$$

In the case of Maximum Mutual Information Estimation (MMIE) for isolated unit modeling, the following criterion can be used:

$$C_{MMIE} = \log(\Pi_{isolated}) = \log(\frac{L_c}{\sum_\omega L_\omega}) \tag{6.39}$$

where

$$\Pi_{isolated} = \frac{L_c}{\sum_\omega L_\omega} \tag{6.40}$$

Assuming equal a priori probabilities for each model, maximizing C_{MMIE} as in equation 6.39 also maximizes the mutual information I.

For continuous speech, we assume that an HMM is built by concatenating unit models. During *training*, we consider a constrained model τ that is made of the concatenation of the units that form the training sentence. On the other hand, during *recognition*, no such

knowledge is available and we use an unconstrained[4] model ρ, for example a loop model [LeeH 89]. Hence, for continuous speech, C_{MMIE} can be expressed as

$$C_{MMIE} = \log(H_{continuous}) = \log(\frac{L_\tau}{L_\rho}), \tag{6.41}$$

where

$$H_{continuous} = \frac{L_\tau}{L_\rho} \tag{6.42}$$

$L_\tau = \alpha_{F_\tau,T}$ denotes the likelihood of the training model and $L_\rho = \alpha_{F_\rho,T}$ denotes the likelihood of the recognition model. By optimizing one of the criteria described above for the hybrid system, we can replace the usual Least Mean Square criterion and direct supervision of the ANN by a supervision which is derived from the temporal modeling in the HMM.

6.3.2 Observation Probability

For the ANN/HMM hybrid, any continuous distribution can be used as long as it is possible to compute the derivative of $b_{i,t}$ with respect to the observations Y_t. For the experiments described in this chapter, we assume $b_{i,t}$ can be represented by a Gaussian mixture as follows:

$$b_{i,t} = \sum_k \frac{Z_k}{((2\pi)^n \mid \Sigma_k \mid)^{1/2}} \exp(-\frac{1}{2}(Y_t - \mu_k)^t \Sigma_k^{-1}(Y_t - \mu_k)) \tag{6.43}$$

where n is the number of observation features (i.e., the number of ANN outputs that are sent to the HMM). The transition probabilities a_{ij}, normal distribution mean vectors μ_k, covariance matrices Σ_k, and gains Z_k can be estimated as in [Rabi89]. A derivative of the cost function C with respect to $b_{i,t}$ can be computed and used for estimating the parameters of the ANN as will be shown in the next section.

6.4 Estimation of ANN parameters

As the optimization criterion C depends on the parameters Y_1^T computed by the ANN, it is possible to express C as a function of them and derive the following equation, using the

[4] Except for those constraints that could be introduced by a grammar on some transitions (particularly from unit to unit)

chain rule:

$$\frac{\partial^+ C}{\partial Y_{jt}} = \sum_i \frac{\partial^+ C}{\partial b_{i,t}} \frac{\partial b_{i,t}}{\partial Y_{jt}} \tag{6.44}$$

where Y_{jt} is the j^{th} element of the network output vector Y_t. The negative of this gradient can be used with back-propagation[5] to estimate the ANN parameters.

In the case of MLE, the derivative of C_{MLE} with respect to $b_{i,t}$ is simply

$$\frac{\partial^+ C_{MLE}}{\partial b_{i,t}} = \frac{\partial^+ L_{model}}{\partial b_{i,t}} = \frac{\partial^+ \alpha_{F_{model},T}}{\partial b_{i,t}} \tag{6.45}$$

where *model* is the training model (the correct word model, in the case of isolated units modeling).

In the case of MMIE, the gradient of the optimization criterion C_{MMIE} with respect to the observation probabilities $b_{i,t}$ can be expressed as

$$\frac{\partial^+ C}{\partial b_{i,t}} = \frac{1}{H} \frac{\partial^+ H}{\partial b_{i,t}} \tag{6.46}$$

where H is defined as in equations 6.40 and 6.42 for isolated and continuous speech modeling, respectively.

In the case of isolated units modeling (MMIE), for states i that are in a unit model ω, the partial derivative on the right hand side can be expressed as follows:

$$\frac{\partial^+ H_{isolated}}{\partial b_{i,t}} = \frac{(\delta_{c\omega} - H_c)}{\sum_\omega L_\omega} \frac{\partial^+ \alpha_{F_\omega,T}}{\partial b_{i,t}} \tag{6.47}$$

For continuous speech (MMIE), we have the following derivative:

$$\frac{\partial H_{continuous}}{\partial b_{i,t}} = \begin{cases} \frac{1}{\alpha_{F_\rho,T}} \frac{\partial \alpha_{F_\tau,T}}{\partial b_{i,t}} & \text{for state } i \text{ in training model } \tau \\ -\frac{\alpha_{F_\tau,T}}{\alpha^2_{F_\rho,T}} \frac{\partial \alpha_{F_\rho,T}}{\partial b_{i,t}} & \text{for state } i \text{ in recognition model } \rho \end{cases} \tag{6.48}$$

In general, for every optimization criterion C that can be expressed as a differentiable function of the likelihood L, it is possible to compute $\frac{\partial C}{\partial L}$.

By differentiating equation 6.43, $\frac{\partial b_{i,t}}{\partial Y_{jt}}$ can be expressed as follows:

$$\frac{\partial b_{i,t}}{\partial Y_{jt}} = \sum_k \frac{Z_k}{((2\pi)^n \mid \Sigma_k \mid)^{1/2}} \left(\sum_l d_{k,l,j}(\mu_{kl} - Y_{lt})\right) \exp\left(-\frac{1}{2}(Y_t - \mu_k)^t \Sigma_k^{-1}(Y_t - \mu_k)\right) \tag{6.49}$$

[5] It replaces the usual $\partial E/\partial Y_{jt} = (Y_{jt} - target_{jt})$ for output units, as used in equation 4.5, where $target_{jt}$ would be the desired output at time t for unit j

where d_{k,l_j} is the element (l,j) of the inverse of the covariance matrix (Σ^{-1}) for the k^{th} Gaussian distribution and μ_{kl} is the l^{th} element of the k^{th} Gaussian mean vector μ_k.

Then, similarly to [Brid90], it is possible to compute the following derivative using equation 6.37, for any hidden Markov *model*. Here the symbol *model* can stand for any of the above mentioned models, e.g., c (correct word), ω (any word) for isolated units modeling, or ρ (recognition model), or τ (training model) for continuous speech modeling.

$$\begin{aligned}
\frac{\partial \alpha_{F_{model},T}}{\partial b_{i,t}} &= \frac{\partial \alpha_{F_{model},T}}{\partial \alpha_{i,t}} \frac{\partial \alpha_{i,t}}{\partial b_{i,t}} \\
&= \left(\sum_j \frac{\partial \alpha_{j,t+1}}{\partial \alpha_{i,t}} \frac{\partial \alpha_{F_{model},T}}{\partial \alpha_{j,t+1}}\right)\left(\sum_j a_{ji}\alpha_{j,t-1}\right) \\
&= \left(\sum_j b_{j,t+1} a_{ji} \frac{\partial \alpha_{F_{model},T}}{\partial \alpha_{j,t+1}}\right)\left(\sum_j a_{ji}\alpha_{j,t-1}\right) \\
&= \beta_{i,t} \frac{\alpha_{i,t}}{b_{i,t}}
\end{aligned} \tag{6.50}$$

The equality on the previous line can be justified because the recursive definition of $\beta_{i,t}$ (see equation 6.37) is the same as the recursive computation of $\frac{\partial \alpha_{F_{model},T}}{\partial \alpha_{i,t}}$:

$$\frac{\partial \alpha_{F_{model},T}}{\partial \alpha_{i,t}} = \sum_j a_{ij} b_{j,t+1} \frac{\partial \alpha_{F_{model},T}}{\partial \alpha_{j,t+1}} \tag{6.51}$$

with

$$\frac{\partial \alpha_{F_{model},T}}{\partial \alpha_{F_{model},T}} = \beta_{F_{model},T} = 1 \tag{6.52}$$

so we have:

$$\frac{\partial \alpha_{F_{model},T}}{\partial \alpha_{i,t}} = \beta_{i,t} \tag{6.53}$$

6.5 Initialization of the ANN Weights

There are several ways for initializing the weights of the ANN. If initialization is with random weights, training might be very slow because ANN training is in general slower than HMM training, and the HMM parameters depend on the transformation performed by the ANN.

6.5.1 Data Compression and Deterministic Initialization

Another possibility consists of a deterministic initialization that would decrease the chance of getting stuck in a local minimum (see [Irin90])[6]. Another advantage of a deterministic initialization is that parameter estimation does not depend on random initial conditions. Different initial conditions may give rise to different networks, with very different performances [Kole90]. A useful preprocessing step that the ANN can perform may consist of the computation of the principal components of the input data. An interesting alternative is the computation of linear discriminants [Brow87]. In both cases, it is reasonable to model the distribution of network outputs with a mixture of Gaussians with diagonal covariance matrices. This considerably reduces the number of parameters of the HMM.

The computation of the principal components can be split into two parts by decomposing the principal components or the linear discriminants matrix A into the product of two matrices, e.g., with an LU decomposition [Stew73]:

$$A = LU \tag{6.54}$$

This decomposition leads to a 2-layer network with inputs connected to the hidden layer and the hidden layer connected to the output layer. The hidden layer weight matrix is initialized with ϵL where ϵ is a small positive number, and the output layer weight matrix is initialized with U. By multiplying L by a small scalar we make sure that initially the weights of the connections to hidden nodes are very small so that the input-output transformation of each hidden node is almost linear. If the hidden units compute symmetric sigmoids, with output ranging in the interval [-1,1] rather than [0,1], then the ANN output Y is approximately as follows:

$$Y \approx \epsilon L U X = \epsilon A X \tag{6.55}$$

where X is the network input, which could be fed by a small number of spectral frames of the input signal.

[6] A similar situation occurs for Radial Basis Function networks, which can be trained very fast if the hidden layer is initialized using a cluster analysis of the input data (see Section 7.1)

Initially the network settled as described performs a decorrelation of the input data over the spectra and a few frames of speech. Indeed, we know that there is a degree of redundancy in each spectrum, as well as in a set of adjacent spectra (the spectra normally change smoothly over time). The network reduces the dimensionality of the input data so that the process described by the network output can be better modeled by an HMM trained with limited sample data.

The advantage of the proposed scheme over a simple fixed linear transformation that computes the principal components or the linear discriminants in a preprocessing phase is that the function computed by the network can evolve with further training in order to optimize the learning criterion, based on the reduction of the rate of recognition errors.

6.5.2 Initialization with A Priori Targets

To initialize the ANN, we can use prior knowledge about the task to modularize it and bootstrap the modules such that their outputs approximate features of interest (see arguments in Chapters 3 and 5). Rather than having a single ANN that computes the vector $\mathbf{Y_t}$ of parameters, we can have a hierarchy of networks, as shown in Figure 2. Such an architecture is built on three levels. Level 3 contains the HMMs. Level 2 is made of a single ANN that acts as an integrator of parameters generated by more specialized ANNs or computed by other algorithms, like the ones of an ear model (see [Cohe89] or [Cosi90]). Networks at level 1 are specialized to compute parameters that are particularly useful for characterizing certain acoustic situations. As an example, a set of parameters may represent the acoustic cues useful for distinguishing between sonorant and non-sonorant sounds. A network that computes these parameters may have some special acoustic features at the input that are different from those feeding other networks. Networks at level 1 can be added when systematic recognition errors have been noticed (for example a frequent confusion between voiced and unvoiced plosives) and suitable time and frequency representations can be conceived for the input of these networks (see for example [Ben89b]). The outlined hierarchical system can be trained as described in sections 6.3 and 6.4. The gradient of the optimization criterion can be back-propagated from level 2 to level 1.

Initially, networks at level 1 can be trained separately as classifiers of certain phoneme sets. An alternative would be to train networks explicitly as detectors of phonetic properties such as place and manner of articulation. After level 1 networks are trained, the level 2 network can be initialized so that it computes the principal components of the output of the level 1 modules. This approach has been chosen for the comparative experiments performed in Section 6.8.

6.6 Some Extensions

The algorithms introduced above can be used even if the ANNs are recurrent. Networks of this type can capture short-time temporal regularities with less parameters than a feedforward network with time delays (see Section 4.3). In general, the output of a recurrent network is smoother than the output of a feedforward network scanning the input data. In that case, the HMM temporal resolution could be reduced, thus reducing the number of necessary parameters. In the experiments described in Section 6.8, the temporal resolution of the ANN was 5 ms and that of the HMM was 10 ms.

ANNs trained as proposed can perform speaker adaptation. For such a purpose, the network has to map speaker-dependent observations into a speaker-independent representation. This objective could be obtained by first training a hybrid system as previously described for multiple speakers, and in a second step, adapting *only some or all ANN parameters* with known or unknown sentences from the new speaker. If the sentences are known then a constrained HMM can be used to adapt the ANN parameters. In such a system, the ANN adaptation represents a tuning of the feature space to the new speaker, whereas the temporal model remains unchanged. The gradient of the optimization criteria is still passed from the HMM to the ANN, but HMM parameters remain unchanged. The ANN is trained in such a way that different speakers tend to produce similar output parameters for the same speech unit (see [Brid91] for a related speaker adaptation mechanism).

6.7 Methodology

In summary, we have seen how to compute the gradient of a training criterion for HMMs with respect to the parameters of the ANN. In particular we have considered the MLE and the MMIE criterion for both isolated and continuous speech models. To implement such a hybrid system, the following methodology can be applied.

1. First, ANNs are trained to recognize, for example, phonetically relevant features, such as, place and manner of articulation.
2. Second, the output vector of these networks is compressed by principal component analysis, in order to provide a smaller input vector for the HMM.
3. Third, a first estimation iteration computes initial values for the HMM parameters, keeping the ANNs parameters fixed.
4. Finally, the global optimization procedure can be applied in order to tune the HMM and ANN parameters.

In the next section, an application of this algorithm is described in more details.

6.8 Experimental Evaluation

A preliminary experiment has been performed using a prototype system based on the integration of ANNs with HMMs. The purpose of the experiment is to show the benefits of an ANN/HMM hybrid and of its global optimization. The task is the recognition of plosive sounds in every context and pronounced by a large speaker population. The TIMIT continuous speech database (Zue, Seneff & Glass 90) has been used for this purpose. SI and SX sentences from regions 2, 3 and 6 were used, with 1080 training sentences and 224 test sentences, 135 speakers in the training set and 28 speakers in the test set[7]. The following 8 classes were considered: /p/,/t/,/k/,/b/,/d/,/g/,/dx/[8], /all other phonemes/.

[7] The training speakers were those whose initial was between "a" and "r" inclusively. The remaining speakers were used for testing

[8] The flapped alveolar plosive */dx/* is considered as a distinct phoneme in the TIMIT database.

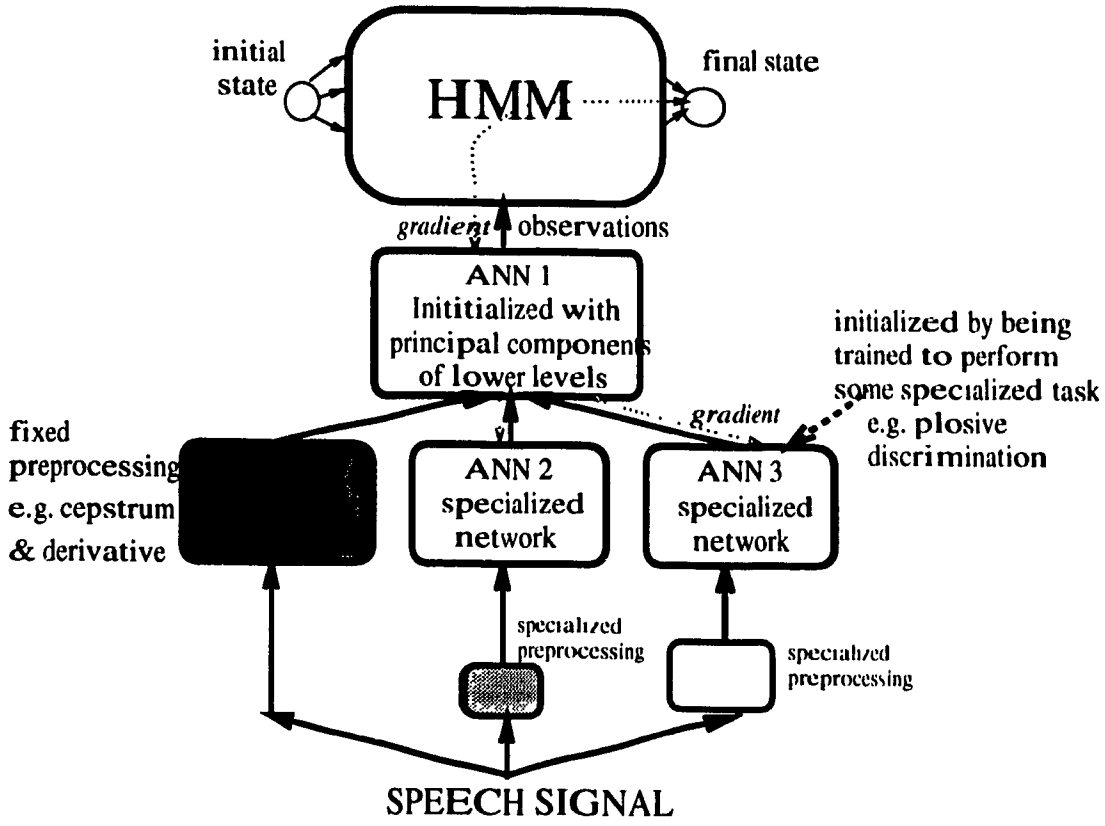


Figure 6.3: Extension of the ANN/HMM hybrid to a hierarchy of modules, with three levels.

Speaker-independent recognition of plosive sounds in continuous speech is a particularly difficult task because these sounds are made of short and non-stationary events that are often confused with other acoustically similar consonants or may be included into other unit segments by a recognition system.

6.8.1 Architecture

As discussed in [Ben89b, Ben90a, Ben91b] and Chapter 3, speech knowledge can be used to design the input, output, and architecture of the system and of each one of the networks. The approach that we have taken is to select different input parameters and different ANN architectures depending on the phonetic features to be recognized.

The experimental system is based on the scheme shown in Figure 6.3. Rather than having a single ANN that computes the vector Y of parameters, we have a hierarchy of networks. For the HMM hybrid, the architecture is built on three levels. Level 3 contains the HMMs. Level 2 is made of a single ANN that acts as an integrator of parameters generated by more specialized ANNs: ANN1 is a linear network that initially computes the principal components of the concatenated output vectors of the lower level networks (ANN2 and ANN3). At level 1, two ANNs are initially trained to perform plosive recognition (ANN3) and broad classification (ANN2) respectively. The preprocessing and topology of these networks are similar to the networks described in [Ben91b] and their outputs describe articulatory features such as the place and manner of articulation and a degree of voicing. Input parameters are fed to the networks every 5 msec.

Broad Classification Network

This network was designed by Ralf Kompe and is described in [Ben91a]. The broad classification net (ANN2) has five outputs corresponding to five broad categories:

{non-nasal sonorant, nasal, plosive, fricative, silence }.

The twelve input nodes to ANN2 are the energies of five band-pass filters in the time domain covering the range up to 7 kHz, the signal total energy, and their six time derivatives. The

filters were IIR (infinite impulse response) Butterworth pass-band filters with the following -3 dB bandwidth specifics: 150-350 Hz, 60-500 Hz, 500-2500 Hz, 2500-3500 Hz, and 4000-7000 Hz. The non-linear phase response of the filters was not corrected. For the total energy and for the filters in the 150-350 and 60-500 Hz bands, an input window of 20 ms was used. A window of 5 ms was used for every other filter. The derivatives were computed by linear regression over 9 consecutive frames (45 msec). The filter bandwidths were chosen based on acoustic-phonetics knowledge (see for example [Stev81, OSha87]). This input feature representation was found to perform better than other spectral representations based on the computation of energies from the Fast Fourier Transform of a fixed analysis window (see below).

ANN2 has four fully connected layers (12 30 15 5) but only time-delay links (from the input at frame t and frame $t - 20$ ms to the first hidden layer, and from the second hidden layer at frame t and $t - 20$ ms to the output layer), as it was found that recurrence did not help its performance. Also there were direct links without any delay from the input layer to the second hidden layer and the output layer, and from the first hidden layer to the output layer. This architecture was optimized after some trials. The total frame error was 17.7% on the test set and 17.6% on the training set. With a reduced set of classes obtained by merging *fricative* with *plosive* and and *nasal* with *non-nasal sonorants*, a frame error rate of 11.1% on the test set and 11.0% on the training set were obtained. This performance compares favorably with other published works [Chig88, Cole88].

Plosive Network

The plosive recognition net (ANN3) is as described in sections 3.1.2 (inputs), 3.3.2 (architecture, see Figure 3.4), and 3.5.2 (output coding). ANN3 has time-delay links between the input nodes and the hidden layer, and recurrent links between some of the hidden nodes and the output nodes. It has two hidden layers, and the first hidden layer nodes have input connectivity that is local in frequency. It has sixteen outputs with different instantiations of each place node depending on the right context[9]. The 74 inputs to ANN3 are the out-

[9] Each of the four different places of articulation (labial, alveolar, velar, and flapped alveolar) corresponds to two different nodes, depending on whether the following phoneme has a forward or backward place of

puts of 32 Bark scaled (logarithmic) triangular filters computed from the short-time Fast Fourier Transform of the windowed signal, 30 property detectors approximating a second order derivative over short intervals of frequency and time (as described in Section 3.1.2). 7 slope coefficients describing the frequency derivative of the spectrum, the total energy and the voicing energy (in the 60-500 Hz band) and their time derivatives, and a measure of distance (normalized dot product) between neighboring spectral frames. This particular selection of input parameters is the fruit of some preliminary experiments [Ben91a]. In general, we have found that using many correlated input parameters and using specialized ANN topologies with such a distributed output encoding improves both the phonetic classification performance and the convergence rate of the learning algorithm, when compared to using the spectrogram only as input and the simpler "one output node per phoneme" encoding.

Principal Components Network

Computing the principal components of a set of vector patterns results in a projection of these vectors upon the eigenvectors of their covariance matrix. This is a rotation of the axes so that they coincide with these eigenvectors. This is done to reduce dimensionality, by keeping only the first few eigenvectors, when they are ordered by decreasing eigenvalue. The first few eigenvectors account for most of the variance of the patterns.

After the covariance matrix has been computed and its eigenvectors evaluated, the input to the principal components network (ANN1) are projected onto this lower dimension space and the mean and variance in this space are computed and used to normalize the projected data to have zero mean and unit variance. The linear network is then initialized so that it computes this translated and scaled projection onto the principal component space.

ANN1 initially computes the principal components of the concatenated output vectors of the lower level networks, ANN2 (16 outputs) and ANN3 (5 outputs). Several choices of number of principal components were tried and the best one that was found was 8. Experiments with a simpler HMM topology yielded 78% accuracy with 21 inputs to the HMM, 82%

articulation: The remaining eight nodes are labeled: unvoiced plosive, voiced plosive, vocalic front,vocalic non-front, liquid, fricative, nasal, silence.

Table 6.1: Comparative Results: Neural networks alone, with Dynamic Programming, with Hidden Markov Models, and with global optimization.

	% rec	% ins	% del	% subs	% acc
ANNs alone	85	32	0.04	15	53
HMMs alone	76	6.3	2.2	22.3	69
ANNs+DP (no bigrams)	88	16	0.01	11	72
ANNs+DP (bigrams)	88	14	0.01	11	74
ANNs+HMM	87	6.8	0.9	12	81
ANNs+HMM+global opt.	90	3.8	1.4	9.0	86

accuracy with 10 inputs and 84% with 8 inputs.

Hence the linear network initialized to compute the translation scaling and rotation has 21 inputs, 8 outputs and 176 weights ($8\times(5 + 16 + 1$ (bias))).

6.8.2 Comparison of Post-Processors

For the dynamic programming (DP) hybrid, the outputs of the networks at level 1 are indirectly used in the cost function of the dynamic programming optimization. That cost function is based on the product of several probabilities (as described in Section 4.4.3): a priori class probabilities, duration probabilities for each phoneme, bigram probabilities (all estimated from the TIMIT labeling of the training set), and observation probabilities conditional on each class (modeled with a maximum likelihood Gaussian mixture with diagonal covariance matrix).

In the case of the HMM postprocessor, ANN1 computes 8 features for the continuous densities HMM. The combined network (ANN1+ANN2+ANN3) has 23578 free parameters. Each of the 11 HMM unit models [10] had 14 states, 28 transitions, 3 self loops, without

[10] To improve its modeling, the rejection class was composed of four models: nasals, fricatives, non-nasal sonorants, and silence. The recognition results are obtained by merging these four subclasses, such that the total number of classes to recognize is eight.

explicitly modeling the state duration. Each HMM has tied distributions with 3 basic different distributions characterizing the beginning, middle and final part of a segment modeled by the unit. Each of these distributions is modeled by a Gaussian mixture with 5 densities. The covariance matrix is assumed to be diagonal since the parameters are initially principal components and this assumption reduces significantly the number of parameters to be estimated.

To assess the value of the proposed approach as well as the improvement brought by the HMM or DP as postprocessors for time alignment, the performance of the hybrid systems were compared with that of a simple postprocessor applied to the outputs of the ANNs. The simple postprocessor assigns a symbol to each output frame of the ANNs by comparing the actual output vectors with target output vectors and choosing the one with shortest Euclidean distance. This simple postprocessor then smooths the resulting string to remove very short segments and merges consecutive segments that have the same symbol. This system is denoted in Table 6.1 by *ANNs alone*.

To evaluate the advantages of using an ANN as a sophisticated preprocessor for the HMM, the same HMM models were used to perform recognition (denoted in Table 6.1 by *HMMs alone*), and using only a standard set of acoustic features as input to the HMM: 8 cepstral coefficients, 8 cepstral derivatives, the signal energy and its derivative. Results using only the cepstrum and the energy were slightly worse.

The comparative results on the test set for the various systems are summarized in Table 6.1. The overall recognition rate (100% - %deletions - %substitutions) for the 8 classes with the hybrid system after two training iterations is 90% on a total of 7214 phonemes, and its accuracy (100% - %deletions - %substitutions - %insertions) is 86%. Note that this is an important improvement over the performance obtained with an HMM trained without global optimization (86% recognition and 81% accuracy), as well as in comparison with the two ANN/DP systems (88% recognition and 72% accuracy without bigrams and 88% recognition and 74% accuracy with bigrams). Note that the biggest improvement in comparison to the ANNs alone comes from modeling the durations rather than the bigrams. The ANNs alone yielded 85% recognition but only 53% accuracy, because of the high number of insertions (32%), mostly due to short plosive segments. The ANNs classify

Table 6.2: Generalization as a function of the number of iterations and the HMM parameter update method.

	% rec	% ins	% del	% subs	% acc
Iteration 0	87.6	6.8	0.9	11.5	80.7
Iteration 1 (batch)	87.1	3.6	2.2	10.7	83.5
Iteration 2 (batch)	87.1	3.8	1.9	11	83.4
Iteration 1 (online)	89.5	4.0	1.3	9.2	85.5
Iteration 2 (online)	89.6	3.8	1.4	9.0	85.8
Iteration 3 (online)	87.6	3.6	2.4	10	84.0

well but have a noisy output with many insertions. The HMM or DP duration modeling eliminates most of these insertions because of their better duration and temporal structure modeling. However, for the experiments with HMMs alone with cepstral input features (+ energy and their derivative), the performance was slightly worse than with the ANN/DP hybrid, and remarkably worse than with the ANN/HMM hybrid Within this globally optimized hybrid, in addition to providing a good temporal model, the HMM provides more appropriate gradient for the outputs of the ANN. With these "moving targets" for the ANN, the hybrid system further improves its performance. It is interesting to note that the effect of equation 6.49 is to generate a gradient that tends to bring the output of the ANN closer to the means of the normal densities which are close to the ANN output, as well as consistent with the the training string.

6.8.3 Batch *vs* Online Parameter Update for the HMM

Our previous experience, as well as other results [Bott90, LeC89a] (see discussion and experiments in Section 2.2.1) indicate that online or stochastic update yields faster convergence than batch update of ANN parameters, especially for pattern recognition problems such as those in speech recognition. Comparative experiments performed with the hybrid system indicate that better results can be obtained with online update for the HMM as well as for the ANN.

In Table 6.2, the two update methods for the HMM parameters within the hybrid system are compared. Traditionally, the HMM parameters are updated after having compiled statistics over the whole training set. The alternative update method used in the experiments is a smoothed online parameter update:

$$\theta_{i,p} = (1 - \alpha)\theta_{i,p-1} + \alpha\hat{\theta}_{i,p} \tag{6.56}$$

where $\theta_{i,p}$ is the new value of the i^{th} parameter after sentence p, α is a small constant[11], and $\hat{\theta}_{i,p}$ is the estimation of the parameter θ_i given the observations in sentence p, using usual HMM parameter estimation algorithms [Rabi89]. Table 6.2 also shows the evolution of generalization errors after one and two training iterations of the hybrid system with global optimization. In the experiments, an error minimum was reached after only two iterations. Further training only reduced generalization. This fast training behavior is typical of Continuous Densities HMMs.

6.9 Summary

A system has been proposed to combine the advantages of ANNs and HMMs for speech recognition. The parameters of the ANN and HMM subsystems can influence each other. We showed how to perform a global optimization of such a system by driving the network gradient descent with quantities computed by the HMM parameter estimation algorithm. Although the algorithm is based on making some assumptions about the distribution of the input data for fixed values of the output of the ANN, the results of the above-described experiments were encouraging. They indicate that global optimization of a hybrid ANN / HMM system brings some important performance benefits and suggest that the possibilities of such a hybrid system should be further explored. We have seen how such a hybrid system could integrate multiple ANN modules, which may be recurrent.

[11] We used $\alpha = 0.005$, except for the variances of the observation distributions which were updated with a semi-batch algorithm, because the estimation of the second moment of the distributions requires more observations:

$\sigma_{i,p} = (1 - \frac{p}{N})\sigma_{i,p-1} + \frac{p}{N}\hat{\sigma}_{i,1,p}$,

where N is the number of sentences, and $\hat{\sigma}_{i,1,p}$ is the estimation of the parameter σ_i given *all the observations from sentence 1 to sentence p*, using standard HMM parameter estimation algorithms [Rabi89]. This method forces a slow initial adaptation of the variances but computes their final value using all the training data

An interesting extension would be to perform speaker adaptation with the hybrid system. This could be obtained by first training the system as previously described for multiple speakers, and in a second step, adapting *only the ANN parameters* with known or unknown sentences from the new speaker, in order to maximize the likelihood of the data given the global (ANN / HMM) model. In such a system, the ANN adaptation represents a tuning of the feature space to the new speaker, whereas the temporal model remains unchanged.

Although the ANNs used in the experiments were recurrent, they did not capture the temporal structure of the speech signal as well as the ANN/HMM or the ANN/DP hybrid systems. Note that very few parameters were used in the HMM or the DP postprocessors to describe the temporal structure of the observations (transition probabilities or duration probabilities, respectively). This may indicate that current ANN topologies and related algorithms are inefficient in modeling temporal structures. It should be observed that HMMs generally used for speech recognition have a left-to-right structure rather than a full connectivity from state to state. It may be possible to improve the way in which temporal structures are modeled in ANNs by imposing appropriate constraints on their architecture for the particular problem of learning to recognize sequences.

Chapter 7

Radial Basis Functions and Local Representation

In Chapters 3, 5 and 6 we have considered modularization as a way to deal with the problem of parameter coupling. In this chapter we consider another approach based on a different type of node operation that yields a local rather than a distributed representation: Radial Basis Functions (RBF). Several phoneme recognition experiments are described with such networks, showing that they can be trained more rapidly but may require more memory resources than the "standard" sigmoid networks. Networks of RBFs with recurrent connections and a hybrid of RBFs and sigmoids are also considered.

7.1 Radial Basis Functions Networks

Sigmoid units such as used in the previous chapters can be interpreted as computing the sigmoid of a signed distance between an input point in the space of inputs of the unit and a hyperplane in that space, defined by the input weights of that unit. Hence, input points near the hyperplane produce an intermediate output of 0.5 or 0 (for asymmetric and symmetric units respectively), while points far from the hyperplane yield values close to the saturation values of the sigmoid (e.g., 0 and 1, depending on which side of the hyperplane they lie), as illustrated in figure 7.1.

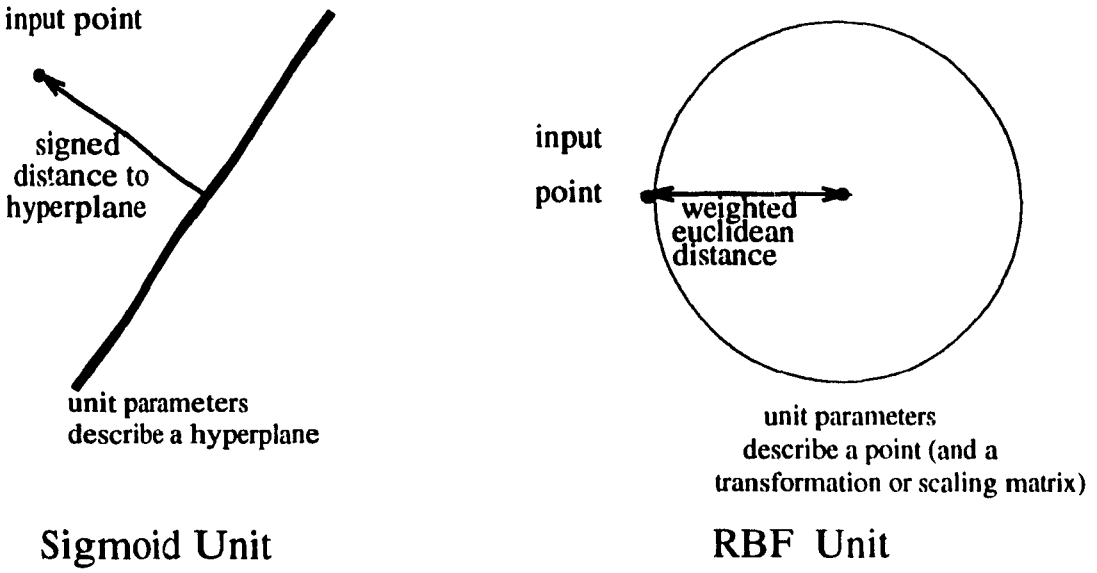


Figure 7.1: Geometrical interpretation of RBF *vs* sigmoid units.

On the other hand, Radial Basis Functions (RBF) units produce an output which depends on the distance between the input point and a "prototype" point in the input space. The output of an RBF network can be written as the weighted sum of the outputs of those RBF units:

$$\hat{F}_i(\mathbf{X}) = \sum_j w_{ij} h(|\ \mathbf{X} - \mathbf{P}_j\ |) \tag{7.1}$$

where $\mathbf{X}$ is the input pattern vector, $\mathbf{P}_j$ is the prototype point (vector) associated to the RBF unit u_j and the basis function $h(\cdot)$ may be, for example, a Gaussian:

$$h(r) = e^{-r^2} \tag{7.2}$$

The norm in equation 7.1 can be weighted by a matrix, equivalent to a linear transformation on the input.

[Pogg89] showed that RBFs perform smooth function approximation arbitrarily well, i.e., approximate $\mathbf{F(X)}$ with $\hat{\mathbf{F}}(\mathbf{X},\theta)$ by hypersurface reconstruction, when given a training set of examples $\{\mathbf{F(X)},\mathbf{X}\}$ which may be noisy. This reconstruction is based on the use of regularization [Tikh77]. More specifically, it imposes a smoothness constraint on the resulting mapping. Poggio [Pogg89] suggests that the parameters of an RBF network as defined in equation 7.1 can be initialized efficiently (e.g., using K-means) and then tuned

using gradient descent. Such "generalized RBFs" [Pogg89] are mathematically related to the well known RBFs used for strict interpolation tasks. However, less units than training examples are required in the "generalized RBFs". This approximation scheme is also related to methods such as Parzen windows, generalized splines and vector quantization.

7.2 Neurobiological Plausibility

A multidimensional Gaussian can be represented as the product of lower dimensional Gaussians. This property suggests a way for neurons to compute RBFs. Gaussian radial basis functions in one or two dimensions can be implemented as coarse coded receptive fields: a dimension is represented as an array of neurons, each reacting only to values of the variables in a certain range. What is required in addition to coarse coded receptive fields is the multiplication of signals. Some special type of synapse has two (or more) incoming inputs and performs an operation similar to a product of the two incoming signals [Mel90]. Neurons with such synapses are called sigma-pi neurons. Hence a RBF network could be implemented with Gaussian receptive fields and sigma-pi neurons without explicitly computing the exponential of the norm in equation 7.1.

7.3 Relation to Vector Quantization, Clustering, and Semi-Continuous HMMs

RBFs are related to vector quantization (VQ) as follows (see [Gray84] for a review).system. VQ partitions the input space into mutually disjoint regions (e.g., Voronoi polygons, separated by line segments at equal distance of neighboring cluster centers). VQ algorithms cluster the input points into these disjoint regions and transform each input point to a symbol associated to the corresponding cluster. This is similar to RBFs that would have a boolean output, with a single RBF responding to a given input pattern. Instead, RBFs represent the input pattern by a vector of proximity measures (between 0 and 1) over the set of hidden units of the RBF network.

Kohonen's neural network models for vector quantization [Koho88] also compute Euclidean distances of cluster centers to the input patterns. These algorithms can be seen as online forms of the k-means algorithm [MacQ67], often used for VQ. Kohonen's algorithms are competitive algorithms in which only the unit closest to the input pattern responds. In particular, in the feature map algorithm, the units in competition are laid out in a low-dimensional spatial structure (e.g., a 2-dimensional grid) in which a neighborhood is defined, such that adjacent units in that "map" respond to similar vectors.

Semi-continuous HMMs [Huan89] are more and more used for automatic speech recognition. In these models, observation distributions are generated by a mixture of a set of Gaussians *common* to all states. In the more classical discrete HMMs, observation distributions are approximated by a non-parametric distribution over a set of *common* symbols generated with VQ. On the other hand, with continuous densities HMMs, each state in the model is associated to its private set of Gaussian densities. In semi-continuous HMMs, each state is characterized by the coefficients of the mixture, i.e., the contributions of each Gaussian. The probability distributions modeled by such mixtures are functionally equivalent to the outputs of an RBF network, however the interpretation of outputs and the training methods are different.

7.4 Methodology

The basic implementation advantages of RBFs derive from their representation: parameters have a simple meaning with regard to the pattern examples. The following fast training method was chosen to take advantage of this property in the experiments described in Section 7.5 (see also [Pogg89]):

1. Initialize the parameters of the RBF units. That can be done by choosing a random subset of the examples, or with the result of a cluster analysis, such as the cluster centers produced by k-means or Kohonen's LVQ2 algorithm [Koho89]. These two algorithms are simple to implement and yield acceptable results rapidly. In the experiments, the k-means algorithm was used. This initialization step can be interpreted as an unsupervised, competitive learning step which encodes the input pattern in a

local representation.

2. Initialize the output weights of the RBF network. This can be done with the Penrose [Penr55] pseudo-inverse or with stochastic gradient descent. The latter seemed more efficient as the number of training patterns increased. This step is a supervised learning step and can also be accomplished very fast, since it is a linear problem with no local minimum.

3. Perform a global optimization of all the parameters of the system. This can be achieved efficiently with stochastic gradient descent since the output of an RBF unit can be differentiated with respect to the adjustable parameters of the unit (cluster mean and rotation matrix). Experiments described in the next section show that this global tuning step indeed improves performance, confirming our previous theoretical (Section 5.2) and experimental (Section 6.7) results indicating that global optimization of a learning system improves its performance (see also [Broo88]).

7.5 Experiments on Phoneme Recognition with RBFs

Several experiments were performed to evaluate the performance of RBFs in a difficult speech recognition problem. The task was to recognize 39 phoneme classes from the TIMIT continuous speech multi-speaker database. Complete SI and SX sentences from regions 2, 3 and 6 of TIMIT were used, with 135 speakers in the training sets (292623 frames) and 28 speakers in the test set (61428 frames). The preprocessing for all the experiments described in this section produced 24 spectral coefficients on the Bark scale (see Section 3.1.2), in addition to the signal energy computed over a 20 ms window. These parameters were computed every 10 ms and sent to the networks.

Unless specified the input window of the network was 4 consecutive frames, i.e., a 100-dimensional input vector representing 40 ms of speech. Output units of the networks were asymmetric sigmoids rather than linear units. This is equivalent since the sigmoid is invertible but is it has the advantage that the outputs are limited to the range (0,1). For the RBF units, only scaling was used in the Gaussian, because of the higher cost of a complete rotation matrix.

Table 7.1: Comparison of global k-means and k-means per class to initialize an RBF network for 39 phoneme classes recognition on TIMIT.

global k-means	78 clusters, no class info	58.1% frame error (test set)
k-means per class	39 classes × 2 cluster/class	52.2% frame error (test set)

7.5.1 Supervised *vs* Unsupervised Initialization

The k-means algorithm generates a set of clusters with input patterns associated to each cluster. The cluster centers are used directly to initialize the centers of the RBF units. The variances of patterns within each cluster was used to initialize the scaling parameters of the corresponding RBF unit (different for each dimension).

Should we use a *completely unsupervised* algorithm to find those parameters? Some points from two classes may form two clusters that mostly overlap, such that the k-means algorithm represents them with a single cluster. A simple but sub-optimal solution to that problem is to *apply k-means separately for each of the classes.* For simplicity, a fixed number of clusters per class was used. An empirical comparison for the experimental task of global k-means (no class information used) *vs* k-means per class showed a significant improvement with k-means per class (see table 7.1). The experiments were performed with 78 hidden RBF units, and output weights were obtained with the pseudo-inverse procedure [Penr55]. All other experiments with RBFs described in this section were therefore performed with k-means per class to initialize the network.

7.5.2 RBFs *vs* Sigmoid Units

The next set of experiments are comparative experiments performed in order to verify that the gain in training time obtained with RBFs is not lost in generalization. RBFs may be trained faster for at least two reasons:

- A powerful initialization procedure exists for RBFs, whereas sigmoid networks are initialized with random weights.

Table 7.2: Comparison of RBF network and sigmoid network on the task of recognizing 39 phoneme classes in TIMIT.

	Sigmoid net	RBF net (only init.)	RBF net (+ glob. opt.)
Training epochs	22	0	10
Generalization error	51.2%	52.2%	47.8%

- There is much less parameter coupling with RBF networks than sigmoid networks, because RBF unit have only a local response. Hence modifying the parameters of one hidden unit in an RBF network only influences the small subset of hidden units that have a nearby cluster center, i.e., which response overlap.

Comparative experiments were performed on the same task already described at the beginning of this section, with the same target outputs and the same architecture for both an RBF network and a sigmoid network. Both networks had 78 hidden units.

As shown in Table 7.2, the generalization error of the RBF net after initialization is almost as good as that obtained with the sigmoid network. After 10 epochs of global optimization (stochastic gradient descent on all the RBF network parameters: output weights, cluster centers and spreads), the generalization error was reduced well below that of the sigmoid network. Hence, with less training time an RBF network with the same number of hidden units performed better than a sigmoid network.

7.5.3 Effect of Context and Architecture

Various architectures were explored to optimize performance on the phoneme recognition problem, using multiple delays between the hidden units and the output units. In all of these experiments there were four delays (0,1,2,3) between the input and the hidden layer.

Results are shown in Table 7.3. Note that the last experiment is with 3 clusters/class, i.e., 117 hidden units. Using only the outputs of the RBF units at frame t did not produce as good results as using several links with various delays (e.g., 0, 2, and 4 frames). This

Table 7.3: Comparison of various sets of delays between the hidden layer and the output layer in RBF nets for 39 phoneme classes recognition on TIMIT.

#clusters/class	hidden→output delays	generalization error
2	0	52.2%
2	0,4	48.6%
2	0,4,8	47.7%
2	0,2,4	46.5%
3	0,2,4	45.6%

result agrees with speech knowledge: information about the changes in speech spectra is very useful for classification of phonemes. Considering only one frame at a time makes several classes overlap in the input space. However, using too many frames may result in generalization problems.

7.5.4 Adding a Recurrent Hidden Layer and Combining RBFs with Sigmoids

The improvement obtained as shown in Table 7.3 by providing more context with delay links show the importance of context for the recognition of phonemes. However, as argued in section 4.3, recurrent networks offer an interesting alternative to the representation of context. This motivated the next experiment in which a recurrent hidden layer was added to the best previously obtained network (5th experiment in Table 7.3). This experiment was also an attempt at combining RBF units with sigmoid units, in order to evaluate if such a hybrid network could perform better than the RBF or the sigmoid network.

The extra layer had 40 sigmoid units feeding the outputs with 3 delays (0, 6 and 12 frames), and receiving its inputs from the 117 RBF units with 3 delays (1, 3 and 5 frames). It also received recurrent inputs from the 39 output units, with 2 delays (0 and 4 frames). In addition, each of these sigmoid hidden units had a self-loop with a fixed-weight decay of 0.93, which corresponds to a time constant (time to halve activation) of 9.5 frames. The

initial weights from these units to the outputs were set uniformly between -0.01 and 0.01, so as not to disturb the network too much by the introduction of this layer. After a few cycles of training, significant improvement was observed. Before the introduction of the recurrent hidden layer the generalization error was 45.6%. After 11 epochs of stochastic gradient descent (update after each sentence), the generalization error was reduced to 41.8%. Relative generalization was good since the error on the training set was 40.9%.

However, the resulting network was very large, with about 54000 weights, and each epoch required more than a day of CPU time on a UNIX workstation.

In spite of this problem, these experiments suggested interesting conclusions. First, alternative networks based on local representations, such as RBF networks, may offer faster training than the standard networks of sigmoid units. Second, their initialization could be improved if information about the class of each input pattern is used to compute initial clusters. Third, performing a global optimization of the RBF network after the RBF units and the output weights have been initialized lowers the error rate. Third, improving the representation of context with delays and recurrence significantly improves performance. Fourth, combining the local representation units (RBFs) and the distributed representation units (sigmoid units) may yield improved performance.

Chapter 8

Biological Constraints for the Automated Design of a Learning Rule

Up to this point in this thesis we have considered only basic variations of the back-propagation algorithms to train ANNs. However, as discussed in section 2.2, this *training* algorithm[1] has several weaknesses, including lack of biological plausibility. In this chapter we consider an alternative to research based on the back-propagation algorithm. The proposed approach is based on the desire to help the search for better learning algorithms for ANNs using *automatic* methods. One of the conclusions of this thesis is that the design of ANNs for a complex task such as phoneme recognition is significantly improved if one uses knowledge about the problem and its solution for many aspects of this design. On the other hand ANNs themselves were originally designed by taking inspiration from knowledge of neurobiology, psychology, mathematics and sometimes physics. When searching for learning rules for ANNs, we propose to use even more knowledge of biology to *constrain* that search, since the space of learning algorithms is so large. An automatized search for such biologically constrained learning rules can be achieved by describing such rules with *parametric functions* and estimating those parameters that minimize a cost. Some toy ex-

[1] Even though the network and unit operation are biologically plausible, the training algorithm is not.

periments with gradient descent show that it is indeed possible to perform a "second-order optimization", i.e., learning a learning rule. The basic hypothesis behind the ideas and experiments presented in this section is that it is possible to search for a synaptic learning rule with learning algorithms. Because the space of learning algorithms is very large it is proposed to use biological knowledge about synaptic mechanisms, in order to design the form of this rule. The proposed method of finding the learning rule automatically relies on the idea of considering the synaptic modification rule as a parametric function, which has *local* inputs, and is the same in many neurons. The parameters that define this function can be optimized with known learning methods. For the experiments described here, gradient descent was used to optimize the learning rules. Estimation of parameters of synaptic modification rules consists of a joint global optimization of the rules themselves, as well as of multiple networks that learn to perform some tasks with these rules.

Initial experiments are described in order to assess the feasibility of the proposed method for very simple tasks. Experiments of classical conditioning for *Aplysia* (an invertebrate commonly used in neurological studies) yielded a rule that allowed a network to reproduce five basic conditioning phenomena: habituation, conditioning, blocking, second-order conditioning, and extinction. Experiments with boolean functions yielded a rule for a network with a hidden layer that could be used to learn some non-linearly separable transformations. The rule was trained with a set of boolean functions and tested successfully on a different set.

8.1 Optimizing a learning rule

In order to search for a learning rule, the following assumptions were made:

1. The same rule is used in many neurons [2].
2. There exists an input/output mapping (that may be stochastic) that corresponds to the learning rule.
3. This mapping can be approximated with a parametric function.

[2]This constraint is relaxed to several rules for several types of neurons or synapses.

Consider a network (that may be recurrent) of neurons and synapses (with strength w_i), and an optimization criteria C, which is a function of the behavior of the network, and that is to be minimized. Let us assume that $\frac{\partial C}{\partial w_i}$ can be computed (for example, with back-propagation, as described in Section 4.3). Let synaptic weight update at time t be defined as follows:

$$w_i(t+1) = w_i(t) + \Delta w_i(t) \tag{8.1}$$

and let $\Delta w_i(t)$ be a function of local observable quantities, as well as of a *set of parameters* θ shared by all (or a lot of) synapses:

$$\Delta w_i(t) = \Delta w(\textit{local variables}, \theta) \tag{8.2}$$

For example, the synaptic change function used in experiments described in this section has as local arguments a measure of presynaptic activity (y_{pre}), postsynaptic potential (x_{post}), synaptic strength (w_i) and of activity of a facilitatory neuron (or of concentration of a diffusely acting neuromodulator) (y_{modul}) that modulates synaptic plasticity[3]:

$$\Delta w_i(t) = \Delta w(x_{post(i)}(t), y_{pre(i)}(t-1), w_i(t), y_{modul(i)}(t), \theta) \tag{8.3}$$

To perform gradient descent on θ one computes the following derivatives:

$$\frac{\partial^+ C}{\partial \theta_j} = \sum_i \frac{\partial^+ C}{\partial w_i(t)} \frac{\partial^+ w_i(t)}{\partial \theta_j} \tag{8.4}$$

where the ordered derivative $\frac{\partial^+ w_i(t)}{\partial \theta_j}$ (as defined in Section 4.3) can be computed recursively (with $\frac{\partial^+ w_i(0)}{\partial \theta_j} = 0$):

$$\frac{\partial^+ w_i(t)}{\partial \theta_j} = \frac{\partial^+ w_i(t-1)}{\partial \theta_j} + \frac{\partial \Delta w_i(t)}{\partial \theta_j} \tag{8.5}$$

Hence $\Delta w_i(\cdot)$ must be a differentiable function, both of parameters θ and of its inputs (e.g., x_{post}, y_{pre} and y_{modul})[4].

To avoid that θ be estimated as a function of a particular synapse, neuron or network performing a particular task[5], it is important that the function $\Delta w(\theta)$ and the parameter

[3] Facilitatory neurons have been modeled in [Hawk89b]. They emit chemical substances, either very locally or in large areas, that influence synaptic plasticity.

[4] The inputs of $\Delta w_i(\cdot)$ may be influenced by some other weight w_j, e.g., through y_{pre}, hence it may be necessary to compute $\frac{\partial \Delta w_i(t)}{\partial y_{pre}}$.

[5] David Chalmers [Chal90] performed experiments on learning of a learning rule with genetic algorithms for a single neuron learning boolean linearly separable mappings. He observed that for a rule to generalize to new tasks, at least 10 "training tasks" were necessary.

set θ that defines it be the same for all (or a large number of) synapses, and that θ be estimated simultaneously with a population of networks learning to perform different tasks.

8.2 Conditioning Experiments

Preliminary experiments were performed in order to assess the feasibility of using gradient descent to learn the parameters of a learning rule. The first task considered was the simulation of some behavioral phenomena observed in a simple organism, *Aplysia*. The architecture of the network used for this experiment (see Figure 7.3) was inspired from a hypothesized circuit [Hawk89b, Hawk83] for the *Aplysia* gill- and syphon-withdrawal reflex and its modification by tail stimulation.

The network model employed a very simple discrete-time neuron equation:

$$y_i(t) = \text{sigmoid}(x_i(t)) = \frac{1}{1 + e^{-(\sum_j w_{ij} y_j(t-1))}} \tag{8.6}$$

The synapse model was also restricted to a simple transformation (see Figure 7.2). The weight change was a linear function of the three external inputs of equation 3, as well as of the following three products:

- $y_{pre} \times y_{modul}$: this type of term is hypothesized in the synapse model of [Hawk89b, Hawk83] or [Done89].
- $y_{pre} \times x_{post}$: this is simply a Hebbian mechanism.
- $y_{pre} \times w$: this term corresponds to a context-free decremental process, that was useful to model habituation [Gluc87].

Biological knowledge or theories are thus used to bootstrap the function $\Delta w(\theta)$ so that it initially has access to a set of a priori subfunctions equivalent to known or hypothesized synaptic modification mechanisms (see Figure 7.4a). This approach can be seen as a continuation of our previous work that demonstrated the advantages of using prior knowledge in the design of ANNs architectures, inputs and outputs. Here we wish to use prior knowledge

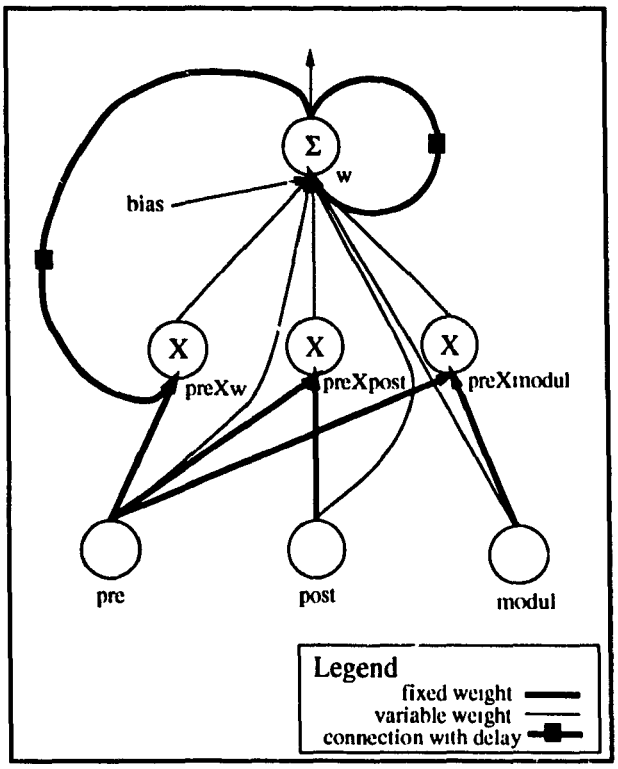


Figure 8.1: Structure of the simple learning rule used in experiments, with 7 free parameters.

for the design of ANN learning rules. Such knowledge comes mainly from neurobiological data, but mathematical and information sciences can also guide us in such as design.

With the architecture in Figure 7.3 and the form for the learning rule of Figure 7.2, we succeeded in estimating parameters of the learning rule that allowed the network to display the following five behaviors:

- Habituation: Initially, CS1 and CS2 elicit a weak response. Presenting repetitively CS1 or CS2 alone reduces that response even more.
- Conditioning: CS1 is followed by US. The response to CS1 gradually increases until it saturates at a level slightly lower than US response.
- Blocking: After CS1 has been conditioned, CS1 and CS2 are paired and followed by US. CS2 does not become conditioned.[6]

[6] However, in the experiments, there was a slight increase in the response to CS2, even though the target value was a constant response. A similar behavior was obtained with Hawkins's model [Hawk89b].

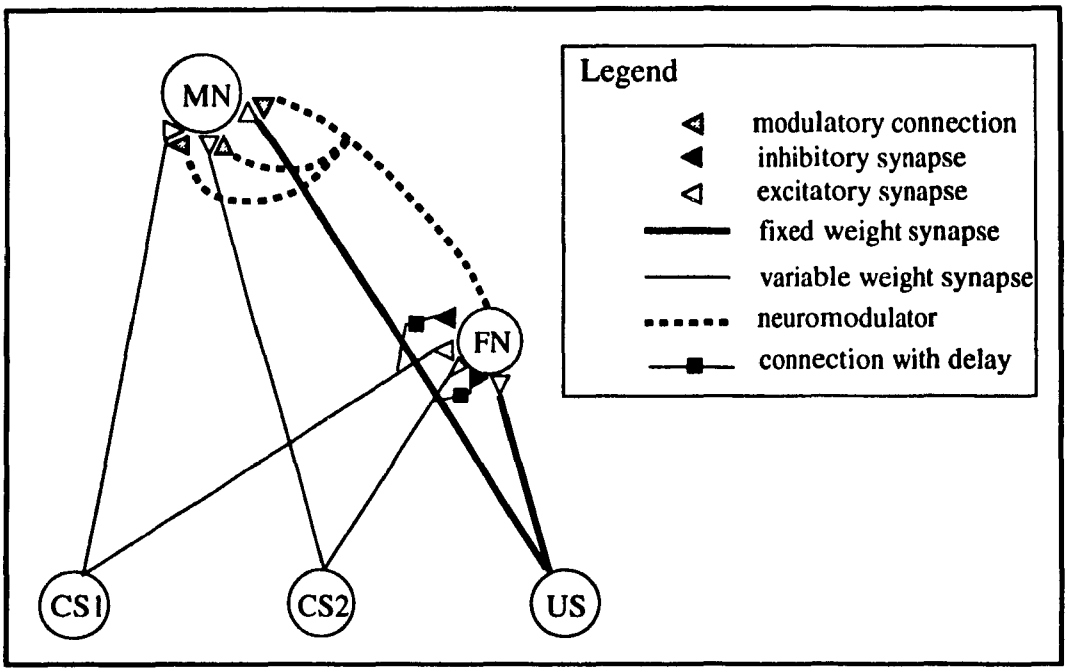


Figure 8.2: Architecture of the network used in conditioning experiments. Two input synapses from CS1 or CS2 to the facilitatory neuron are inhibitory and excitatory, with delay 0 and delay 1, respectively

- Second-order conditioning: After CS1 has been conditioned, it can be used to condition CS2 by presenting CS2 followed by CS1.
- Extinction: After CS1 or CS2 have been conditioned, a repetitive presentation of CS1 or CS2 alone reduces their response to their original levels.

The learning rule of input synapses of the motor neuron was allowed to be different from the learning rule of input synapses of the facilitator neuron. In general one may allow multiple $\Delta w(\theta)$ functions for diverse types of synapses or neurons that are observed. Various types of synapses, neurotransmitter, pre-, epi- and postsynaptic mechanisms were, for example, modeled in [Gard87].

The learning rule parameters were estimated in 1000 training epochs. A simulator that allows product, summation as well as sigmoid nodes was used to perform experiments, allowing to embed synaptic modification networks into the larger "task" network and forcing parameters of the synaptic rule to be the same in various places with weight sharing.

Table 8.1: Summary of results for the boolean functions experiments. L stands for Linear transformations, NL for Non-Linear transformations.

#training tasks	type of training task	#epochs (for finding the rule)	generalization (new tasks) L	NL
1	L	3	yes	no
1	NL	15	yes	no
4	L	5	yes	no
5	4L, 1NL	100	yes	yes

8.3 Boolean Function Experiments

The goal of these experiments was to explore in a very simple setting the possibility of searching for a learning rule that could be used to train a network with hidden units. These experiments are not meant to be biologically plausible. Instead they allowed us to evaluate the applicability of our method to a computationally motivated problem. The same form for the learning rule was used as in previous conditioning experiments (Figure 7.2). Fully connected networks with two inputs, a single output and 1 or 2 hidden units were trained to perform linear and non-linear mappings. The solution chosen here for providing information to hidden units about their contribution to errors is based on use of backward paths, with neurons that may modulate synaptic change on corresponding forward paths (Figure 7.4b). In simulations, a symmetric sigmoid was used for backward paths neurons. However, using asymmetric neurons with an appropriate adaptable threshold can be shown to be equivalent (see [Carl87] for activity-dependent threshold). Results are summarized in Table 7.4. These experiments were performed in collaboration with Samy Bengio and Jocelyn Cloutier.

In a more difficult additional non-linear mapping experiment with a 2-unit hidden layer and no connection from input to output, the resulting rule was successful with 76% of initial network weight values. Rule parameters were updated after each network training epoch, which consisted of 800 input patterns. At the beginning of each epoch, the network weights

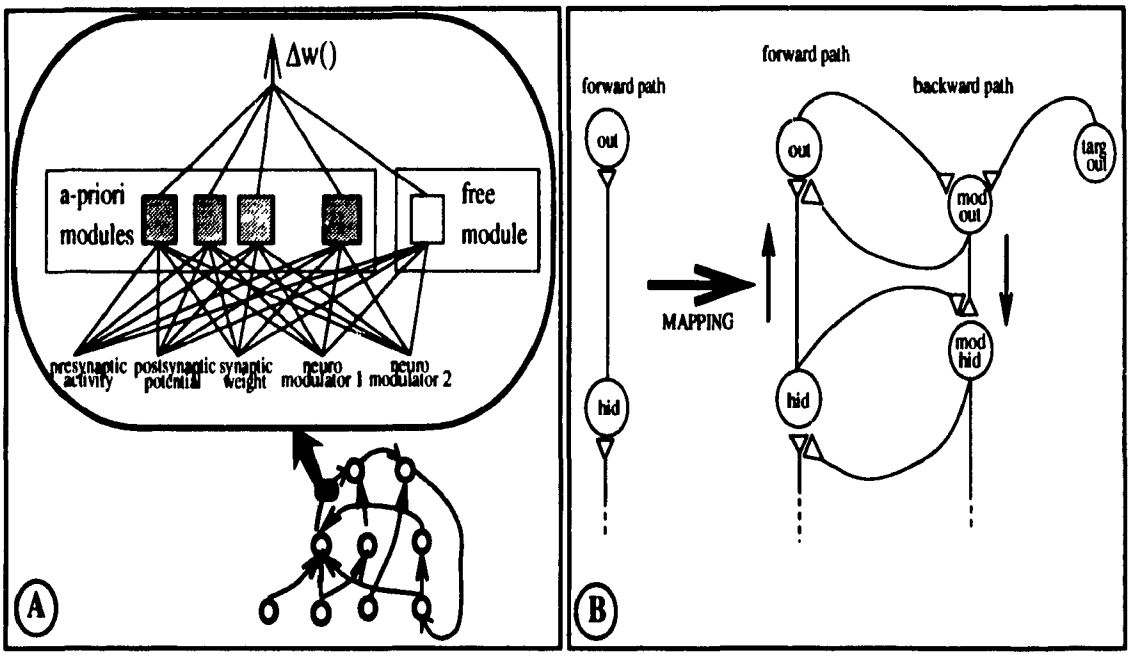


Figure 8.3: (a) Learning rule instantiated in every variable synapse of the network, designed using prior knowledge. (b) Boolean functions experiments: forward connections are mapped into a network that also contains a corresponding backward path. Neurons on this path modulate synapses of forward connections.

were initialized randomly.

It is interesting to note a few things about these experiments. Convergence of rule parameters was very sensitive to their initial values. The best set of initial values we found was equivalent to the delta rule for output units (i.e., 1 for the presynaptic $\times$ modulator factor and 0 for others). However, it was necessary to further optimize all rule parameters. Another interesting observation is that, as expected with results of [Chal90], generalization to new tasks is improved if more tasks are used for training the learning rule. Finally, as for the conditioning experiments, we were not able to make the rule converge unless the parameter set θ associated to forward paths neurons was allowed to be different from the one associated to neuromodulating neurons. This indicates that multiple learning rules for various types of neurons in a network may allow more powerful learning, as is suggested by numerous neuron types and synaptic mechanisms in the brain.

8.4 Is It Possible to Learn a Learning Rule?

In this section, an original approach to neural modeling was presented, based on the idea of searching *with learning methods* for a parametric synaptic learning rule that is biologically plausible, as well as yielding networks that can learn to perform computationally motivated tasks requiring hidden units[7]. The networks architecture, as well as learning function have been designed with constraints derived from biological considerations, thus using prior information to help solve the problem. The experiments presented here deal only with very simple cases, but results indicate that it is possible to learn a learning rule, and this should encourage researchers to apply these ideas to more difficult tasks and more complex forms for the learning rule.

[7] An initial version of this proposal can be found in [Ben90g].

Chapter 9

Conclusion

This thesis addressed the question of the integration of a priori knowledge with learning from examples, for systems based on artificial neural networks and applied to the recognition of sequences. In particular, several problems in automatic speech recognition were considered, such as the speaker-independent recognition of phonemes in continuous speech. The nature of these tasks brought us to the analysis and the design of algorithms for recurrent networks and the integration of artificial neural networks with other systems, such as hidden Markov models, which are well suited for modeling sequences.

We found artificial neural networks to be flexible on at least two levels: the integration of algorithms for learning from examples with prior information about a task, and the integration of these networks with other systems[1]. For the first point, we studied the following aspects of network design in particular. The preprocessing and input coding were found to be important and results indicated that artificial neural networks could take advantage of a *large input space*, even when the input features were highly correlated or redundant. Differently from most others in the field of speech recognition with multi-layer networks, we used output coding schemes that were more compact than the usual "one-output-per-phoneme" scheme. Our approach is based on the use of phonetic or *articulatory features* of speech signals. These representations inspired from phonetics yielded better performance, except

[1] Using ANNs in conjunction with other algorithms that have been shown to be successful for the task is also a way to use prior knowledge about this task

in the case of vowels in continuous speech[2]. Concerning the architecture of networks, *modular systems* based on prior knowledge of problem decomposition were found most effective. This is in fact a particular case of useful local connectivity, which may improve both generalization and convergence, by adding useful *bias* and by reducing *parameter coupling*. In general we preferred specialized networks (for particular types of phonetic discriminations), which may have *specialized inputs*. Theoretical and experimental evidence were presented that argue in favor of performing a *global optimization* of modular systems. We preferred to perform such a step after the individual modules (neural or not) have been bootstrapped, taking advantage of prior knowledge about problem decomposition.

To train artificial neural networks, in particular recurrent ones, we studied learning algorithms. It was found that stochastic update converges much faster than batch learning. An original algorithm for training a particularly interesting kind of constrained recurrent networks was proposed and evaluated. However, we found that even with the more general training algorithms, such as back-propagation in time, recurrent networks -- at least with the common types of architectures we studied -- are inefficient at capturing many aspects of the temporal structure of training data. This was supported by experiments in which the addition of a dynamic programming postprocessor with a few dozens free parameters to a recurrent network with more than 20000 weights almost halved the total error. Hence one should either search for more appropriate architectures or, as we did in this dissertation, consider hybrids of artificial neural networks with other algorithms that have been shown to model sequences well.

Given the objective of performing a global optimization of modular systems, we derived an algorithm for jointly estimating the parameters of a hybrid of artificial neural networks and hidden Markov models. This system was evaluated and compared favorably with either neural networks alone, hidden Markov models alone, or other (simpler) postprocessors for artificial neural networks. We also found that the global optimization step improved performance (reduction of the phoneme error from 20% to 14% in the experiments), even though the computation of the gradient in this algorithm makes some assumptions which

[2]That may be because classical descriptions of the articulatory features of vowels are based on the pronunciation of short isolated words, for which vowels are "well-pronounced". In our experiments, target values derived from these data did not appear to be adequate.

may be wrong about the distribution of the inputs for fixed values of the ANN outputs. We have found the form of a correct gradient for the parameters of the ANN in such a hybrid. It depends on the product of the singular values of the Jacobian of the transformation performed by the ANN, as well as on the distribution of the inputs when given the activations of the output units. Unfortunately, expressing such a distribution consistently with the input data remains an open problem.

Finally, some alternatives to the algorithms used in this thesis for training neural networks were considered. First we considered Radial Basis Functions networks, which are based on local representations and were found to require less training time, mostly because of a useful bootstrapping of the network parameters and because of a partial decoupling of those parameters.

Second, we considered a drastic question concerning learning algorithms for neural networks: is it possible to *search* for such algorithms, using both prior knowledge (about learning) and learning from examples (here, examples are tasks to be learned with a certain network architecture). This is a second-order learning problem: learning about learning rules. Although it may seem infeasible, preliminary experiments indicated that it can be done. They also showed the importance of using prior knowledge in the design of the *form* of a learning rule and its initialization. Here, that knowledge is either biological, mathematical or empirical.

To conclude, all these apparently varied contributions are related to the central theme of this thesis: the integration of prior information with automated optimization, in particular, for artificial neural networks and their application to sequence recognition problems.

Bibliography

[Atal83] Atal B.S. (1983). Efficient coding of LPC parameters by temporal decomposition. *Proceedings of the International Conference on Acoustics, Speech and Signal Processing*, Boston, pp. 81-84.

[Bahl83] Bahl L.R., Jelinek F., and Mercer R.L. (1983). A Maximum Likelihood Approach to Continuous Speech Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. ASSP-5, no. 2, pp. 179-190.

[Bahl87] Bahl L.R., Brown P., de Souza P.V. and Mercer R.L. (1987). Speech recognition with continuous-parameter hidden Markov models. *Computer, Speech and Language*, **2**, pp.219-234.

[Bahl88] Bahl L.R., Brown P., de Souza P.V. and Mercer R.L. (1987). Speech recognition with continuous-parameter hidden Markov models. *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing* (ICASSP-88), New-York, pp. 40-43.

[Bair90] Baird B. (1990). Learning with synaptic nonlinearities in a coupled oscillator model of olfactory cortex. To appear in *Analysis and Modeling of Neural Systems*, F.H. Eeckerman ed.

[Barr88] Barron A.R. and Barron R.L. (1988). Statistical learning networks: A unifying view. Computing Science and Statistics, Proc. 20th Symp. Interface, E. Wegman ed., Amer. Statist. Assoc. Washington DC., pp. 192-203.

[Barr89] Barron A.R. (1989). Statistical properties of artificial neural networks. *Proc. of the 28th conf. on Decision and Control*, Tampa, Florida, pp.280-285.

[Bart91] Bartha G.T., Thompson R.F., and Gluck M.A. (1991). Sensorimotor learning and the cerebellum. To appear in *Visual Structures and Integrating Functions*, M. Arbib & J. Ewert eds, Springer-Verlag.

[Bart81] Barto A.G., Sutton R.S. and Brouwer P.S. (1981). Associative search network: a reinforcement learning associative memory. *Biological Cybernetics*, 40, pp. 201-211.

[Bart83] Barto A.G., Sutton R.S. and Anderson C.W. (1983). Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Trans. on Systems, Man, and Cybernetics*, vol. SMC-13, no. 5, pp. 834-846.

[Bart85] Barto A.G. (1985). Learning by statistical cooperation of self-interested neuron-like computing elements. *Human Neurobiology*, 4:229-256.

[Baum63] Baum L.E., and Eagon, J. (1963). An inequality with applications to statistical prediction for functions of Markov processes and to a model of ecology. *Bull. Amer. Math. Soc.*, **73**, pp.360-363.

[Baum70] Baum L.E., Petrie T., Soules G., and Weiss N. (1970). A maximization technique occuring in the statistical analysis of probabilistic functions of Markov chains. *Ann. Math. Statistic.*,41, pp.164-171.

[Baum89] Baum E. B. (1989). A proposal for more powerful learning algorithms. *Neural Computation*, vol.1, no.2, pp. 201-207.

[Bau+89] Baum E. B. and Haussler D. (1989). What size net gives valid generalization? *Neural Computation*, vol.1, no.1, pp. 151-160.

[Beck89] Becker, S. and Le Cun, Y. (1989). Improving the convergence of back-propagation learning with second-order methods. In Touretzky, Hinton and Sejnowski eds., *Proc. of the 1988 Connectionist Summer School*, pp. 29-37, San Mateo. Morgan Kaufmann.

[Bell57] Bellman R.E. (1957). *Dynamic Programming*, Princeton, NJ. Princeton University Press.

[Bell89] Bellagarda J.R., and Nahamoo D. (1989). Tied Mixture Continuous Parameter Models for Large Vocabulary Isolated Speech Recognition. *Proceedings of the International Conference on Acoustics, Speech and Signal Processing*, Glasgow, Scotland, May 89, pp. 13-16.

[Beng88] Bengio Y. and De Mori R. (1988). Use of neural networks for the recognition of place of articulation, *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing* (ICASSP-88), New-York, pp. 103-106.

[Ben89a] Bengio Y., Cardin R., De Mori R. & Merlo E. (1989). Programmable execution of multi-layered networks for automatic speech recognition. *Communications of the Association for Computing Machinery*, vol. 32, no. 2, pp. 195-199.

[Ben89b] Bengio Y., Cosi P., Cardin R., De Mori R. (1989). Use of multi-layered networks for coding speech with phonetic features. *Advances in Neural Information Processing Systems 1*, ed. D.S. Touretzky, Morgan Kaufmann Publishers, pp.224-231.

[Ben90a] Bengio Y. and De Mori R. (1990). Connectionist models and their application to automatic speech recognition. *Artificial Neural Networks and Statistical Pattern Recognition: Old and New Connections*, ed. I.K. Sethi & A.K. Jain, Elsevier, Machine Intelligence and Pattern Recognition Series. In press.

[Ben90b] Bengio Y. (1990). Radial Basis Functions for speech recognition. *Speech Recognition and Understanding: Recent Advances, Trends and Applications*, (NATO Advanced Study Institute Series F: Computer and Systems Sciences). In Press.

[Ben90c] Bengio Y., Cardin R., De Mori R. (1990). Speaker-independent speech recognition with neural networks and speech knowledge. *Advances in Neural Information Processing Systems 2*, ed. D.S. Touretzky, Morgan Kaufmann Publishers, pp.218-225.

[Ben90d] Bengio Y., Cardin R., De Mori R. and Normandin Y. (1990). A hybrid coder for hidden Markov models using a recurrent neural network. *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, Albuquerque, NM, April 90, pp. 537-540.

[Ben90e] Bengio Y., Pouliot Y., Bengio S. and Agin P. (1990). A neural network to detect homologies in proteins. *Advances in Neural Information Processing Systems 2*, ed. D.S. Touretzky, Morgan Kaufmann Publishers, pp.423-430.

[Ben90f] Bengio Y., Pouliot Y. (1990). Efficient recognition of immunoglobulin domains from amino-acid sequences using a neural network. *Computer Applications in the Biosciences*, vol.6, no.4, pp.319-324.

[Ben90g] Bengio Y. and Bengio S. (1990). Learning a synaptic learning rule. Technical Report #751. Computer Science Department. Université de Montréal.

[Ben91a] Bengio Y., De Mori R., Flammia G. and Kompe R. (1991). Phonetically motivated acoustic parameters for continuous speech recognition using artificial neural networks. To appear in *EuroSpeech 91*, Genova, Italy.

[Ben91b] Bengio Y., De Mori R., Flammia G. and Kompe R. (1991). A comparative study of hybrid acoustic phonetic decoders based on artificial neural networks. To appear in *EuroSpeech 91*, Genova, Italy.

[Ben91c] Bengio Y., De Mori R., Flammia G. and Kompe R. (1991). Global Optimization of a Neural Network - Hidden Markov Model Hybrid. To appear in *Proceedings of the International Joint Conference on Neural Networks 91*, Seattle, WA.

[Blum79] Blumstein S.E. and Stevens K.N. (1979), Acoustic invariance in speech production: Evidence from measurements of the spectral characteristics of stop consonants. *Jour. of Acoust. Soc. Amer. (J.A.S.A.)* Vol. 66, No. 4, pp. 1001-1018.

[Blum87] Blumer A., Ehrenfeucht A., Haussler D. and Warmuth M. (1987). Occam's razor. *Inf. Proc. Let.* 24, pp. 377-380.

[Bohr88] Bohr H., Bohr J., Brunak S., Cotteril R.M.J., Lautrup B., Norsokov L., Olsen O.H., and Petersen S.B. (1988). Protein secondary structure and homology by neural networks. *FEBS Letters*, **241**, pp.223-228.

[Bott90] Bottou L., Fogelman Soulie F., Blanchet P. and Lienard J.S. 1990. Speaker-independent isolated digit recognition: multilayer perceptrons *vs* dynamic time warping. *Neural Networks* 3 (4) : 453-465.

[Bott91] Bottou L. (1991). Une approche théorique de l'apprentissage connexioniste; applications à la reconnaissance de la parole. PhD thesis, Université de Paris XI.

[Bour88] Bourlard, H. and Wellekens, C.J. (1988). Links between Markov models and multi-layer perceptrons. *Advances in Neural Information Processing Systems 1*, (ed. D.S. Touretzky) Morgan Kauffman Publ., pp. 502-510.

[Bour89] Bourlard, H. and Wellekens, C.J. (1989). Speech pattern discrimination and multilayer perceptrons. *Computer, Speech and Language*, vol. 3, pp.1-19.

[Brei84] Breiman L., Friedman J.H., Olsten R.A. and Stone C.J. (1984). Classification and Regression Trees. Waldsworth, Belmont, CA.

[Brid90] Bridle J.S. Bridle, J.S. 1990. Training stochastic model recognition algorithms as networks can lead to maximum mutual information estimation of parameters. *Advances in Neural Information Processing Systems 2*, (ed. D.S. Touretzky) Morgan Kauffman Publ., pp. 211-217.

[Brid91] Bridle J.S. and Cox S.J. (1991). RECNORM: simultaneous normalisation and classification applied to speech recognition. To appear in *Advances in Neural Information Processing Systems 3*, (ed. D.S. Touretzky) Morgan Kauffman Publ.

[Broo88] Broomhead D.S., Lowe D. (1988). Multivariable functional interpolation and adaptive networks. *Complex Systems*, 2, pp. 321-355.

[Brow87] Brown P.F. (1987). The acoustic-modeling problem in automatic speech recognition. PhD Thesis, Dept. of Computer Science, Carnegie-Mellon University.

[Byrn87] Byrne J.H. (1987). Cellular analysis of associative learning. *Physiological Review*, **67**, pp. 329-439.

[Byrn89] Byrne J.H., Gingrich K.J. and Baxter D.A. (1989). Computational capabilities of single neurons: relationship to simple forms of associative and nonassociative learning in *Aplysia. Computational Models of Learning in Simple Neural Systems*, Hawkins R.D. & Bower G.H. eds. Academic Press. pp. 31-63.

[Carl87] Carley L.R. & Raymond S.A. (1987). Comparison of the after-effect of impulse on threshold at nodes of Ranvier at single frog sciatic axons. J.Physiology, **386**:503-527.

[Cash90] Cashman N.R. and Pouliot Y. (1990). EBV Ig-like domains. *Nature*, **343**, pp. 319.

[Chal90] Chalmers D.J. (1990). The evolution of learning: An experiment in genetic connectionism. *Connectionist Models: Proceedings of the 1990 Summer School*, pp.81-90.

[Chau90] Chauvin Y. (1990). Dynamic behavior of constrained back-propagation networks. *Advances in Neural Information Processing Systems 2*, (ed. D.S. Touretzky) Morgan Kauffman Publ., pp. 642-649.

[Chig88] Chigier B. and Brennan R.A. (1988). Broad Class Network Generation Using a Combination of Rules and Statistics for Speaker Independent Continuous Speech *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing, ICASSP-88*, pp. 449-452.

[Cole88] Cole R.A., and Hou L. (1988). Segmentation and Broad Classification of Continuous Speech. *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing ICASSP-88*, pp. 453-456.

[Cohe89] Cohen J.R. (1989). Application of an auditory model to speech recognition. Journal of the Acoustical Society of America, 85 (6), pp. 2623-2629.

[Coll89] Collins E., Ghosh S., and Scofield C. (1989). An application of a multiple neural network learning system to emulation of mortgage underwriting judgements. Nestor Inc., Providence, RI.

[Cosi90] Cosi P., Bengio Y. and De Mori R. (1990). Phonetically-based multi-layered networks for acoustic property extraction and automatic speech recognition. *Speech Communication* special issue on neurospeech, vol. 9, no. 1, pp. 15-30.

[Cove65] Cover T. (1965). Geometrical and statistical properties of systems of linear inequalities with applications to pattern recognition. *IEEE Trans. Elect. Comp.* vol. 14, pp.326-334.

[Crow89] Crow T. (1989). Associative learning, memory and neuromodulation in *Hermissenda*. In *Neural Models of plasticity*, J.H. Byrne & W.O. Berry, eds., pp. 1-21.

[Cybe89] Cybenko G. (1989). Approximation by superposition of sigmoidal functions. *Mathematics of Control, Signal, and Systems*, 2:303-31-1.

[Davi89] Davis L. (1989). Mapping neural networks into classifier systems. *Proc. Third International Conference on Genetic Algorithms*, J.D. Shafer ed., Morgan Kaufmann, pp. 375-378.

[Delg80] Delgutte B. (1980). Representation of speech-like sounds in the discharge patterns of auditory nerve fibers. *Journal of the Acoustical Society of America*, vol. 68, no. 3, pp.843-857.

[Delg84] Delgutte B. and Kiang N.Y.S. (1984). Speech coding in the auditory nerve. *Journal of the Acoustical Society of America*, vol. 75, no. 3, pp.866-907.

[DeMo85a] De Mori R., Laface P. and Mong Y. (1985). Parallel algorithms for syllable recognition in continuous speech. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 7:56-69.

[DeMo85b] De Mori R. and Palakal M. (1985). On the use of taxonomy of time-frequency morphologies for automatic speech recognition. *Proc. Ninth International Joint Conference on Artificial Intelligence*, Los Angeles, CA. pp. 877-879.

[DeMo87] De Mori R., Lam L. and Gilloux M. (1987). Learning and plan refinement in a knowledge-based system for automatic speech recognition. *IEEE Trans. Pattern Analysis and Machine Intelligence, PAMI-2*, **2**, pp. 289-305.

[Deve84] Devereux J., Haeberli P. and Smithies O. (1984). A comprehensive set of sequence analysis programs for the VAX. *Nucleic Acids Research*, **12**, pp. 387-395.

[Devr88] Devroye L. (1988). Automatic pattern recognition, a study of the probability of error. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol.10, no. 4, pp.530-543.

[Done89] Donegan N.H., Gluck M.A. and Thompson R.F. (1989). Integrating behavioral and biological models of classical conditioning. *Computational Models of Learning in Simple Neural Systems*, Hawkins R.D. & Bower G.H. eds. Academic Press. pp. 109-156.

[Dria91] Driancourt X., Bottou L. and Gallinari P. (1991). Comparison and cooperation of several classifiers. Tech. Report no. 652, Universite de Paris-Sud, Laboratoire de Recherche en Informatique.

[Duda73] Duda R.O. and Hart P.E. (1973). *Pattern Classification and Scene Analysis*, John Wiley & Sons.

[Dzwo91] Dzwonczyk M. (1991). Quantitative failure models of feed-forward neural networks. M.Sc. Thesis, MIT.

[Fahl83] Hinton G.E., Sejnowski T.J. (1983). Massively parallel architectures for AI: NETL, Thistle, and Boltzmann machines. *Proceedings of the National Conference on Artificial Intelligence AAAI-83.*

[Fahl91] Fahlman S.E., and Lebiere C. (1990). The Cascade-Correlation learning architecture. *Advances in Neural Information Processing Systems 2*, (ed. D.S. Touretzky) Morgan Kauffman Publ., pp. 524-532.

[Fant73] Fant G. (1973) *Speech Sounds and Features*, MIT press, Cambridge, Mass.

[Flam91] Flammia G. (1991). *Speaker Independent Consonant Recognition in Continuous Speech with Distinctive Phonetic Features*, M.Sc. Thesis, McGill University, School of Computer Science.

[Fran90] Franzini, M., Lee K-F. and Waibel A. (1990). Connectionist Viterbi training: a new hybrid method for continuous speech recognition. *Proceedings of the International Conference on Acoustics, Speech and Signal Processing*, Albuquerque, NM, April 90, pp. 425-428.

[Funa89] Funahashi K.I. (1989). On the approximate realization of continuous mappings by neural networks. *Neural Networks*, 2:183-192.

[Gall88] Gallinari P., Thiria S. and Fogelman F. (1988). Multilayer perceptrons and data analysis. *Proc. IEEE ICNN 88* (1), pp. 391-399.

[Gall89] Galland C.C. and Hinton G.E. (1989). Deterministic learning in networks with asymmetric connectivity. Tech. Rep. CRG-TR-89-6.

[Gard87] Gardner D. (1987). Synaptic diversity characterizes Biological Neural Networks. *Proc. IEEE First International Conference on Neural Networks*, San Diego, CA, pp. IV-17 - IV-22.

[Gema84] Geman S. and Geman D. (1984). Stochastic relaxation, Gibbs distribuions, and the Bayesian restoration of images. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. PAMI-6, no. 6, pp.721-741.

[Gema91] Geman S., Bienenstock E. and Doursat R. (1991). Neural networks and the bias/variance dilemma. Preprint, Brown University, Providence, RI.

[Gill81] Gill P.E., Murray W. and Wright M.H. (1981). *Practical Optimization*, Academic Press.

[Gold85] Goldhor R.S. (1985). Representation of consonants in the peripheral auditory system: A modeling study of the correspondance between response properties and phonetic features. RLE technical report no. 505 (Cambridge, MA: MIT Press).

[Gold88] Goldberg D. (1988). *Genetic Algorithms in Machine Learning, Optimization, and Search.* Addison-Wesley.

[Gori89] Gori M., Bengio Y. and De Mori R. (1989). BPS: a learning algorithm for capturing the dynamic nature of speech. *Proc. IEEE Int. Joint Conf. on Neural Networks*, Washington DC. pp. II.417-II.424.

[Gluc87] Gluck M.A. and Thompson R.F. (1987). Modeling the neural substrate of associative learning and memory: a computational approach. *Psychological Review* 94, p.176.

[Gray84] Gray R.M. (1984). Vector quantization. *IEEE ASSP Magazine*, April 84, pp. 4-29.

[Gree60] Greenwood T.A., Durand D. (1960). in *Technometrics*, **2**, pp. 55-56.

[Grib87] Gribskov M., McLachlan M. and Eisenber D. (1987). Profile analysis: detection of distantly related proteins. *Proc. Natl. Acad. Sci. USA*, **84**, pp. 4355-4358.

[Gros89] Grossman T., Meir R., and Domany E. (1989). Learning by choice of internal representation. *Advances in Neural Information Processing Systems 1*, (ed. D.S. Touretzky) Morgan Kauffman Publ., pp. 73-80.

[Haff89] Haffner P., Waibel A.H. and Shikano K. (1989). Fast back-propagation learning methods for large phonemic neural networks. *Proceedings of Eurospeech*, September 1989.

[Haff91] Haffner P., Franzini M. and Waibel A. (1991). Integrating time alignment and neural networks for high performance continuous speech recognition. *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing* (ICASSP-91), Toronto, paper# 8.S2.17, pp. 105-108.

[Haus89] Haussler D. (1989). Generalizing the PAC model: sample size bounds from metric dimension-based uniform convergence results. *Proc. of the 30th Annual Symposium on the Foundations of Computer Science*, IEEE.

[Hawk83] Hawkins R.D., Abrams T.W., Carew T.J. and Kandel E.R. (1983). A cellular mechanism of classical conditioning in *Aplysia*: Activity-dependent amplification of presynaptic facilitation. *Science*, **219**, pp.400-404.

[Hawk89a] Hawkins R.D., Bower G.H. (eds.) (1989). *Computational Models of Learning in Simple Neural Systems*. Academic Press.

[Hawk89b] Hawkins R.D. (1989). A biologically based computational model for several simple forms of learning. *Computational Models of Learning in Simple Neural Systems*, Hawkins R.D. & Bower G.H. eds. Academic Press. pp. 65-108.

[Hawk89c] Hawkins R.D. (1989). A simple circuit model for higher-order features of classical conditioning. In *Neural Models of plasticity*, J.H. Byrne & W.O. Berry, eds., pp. 74-93.

[Hint84] Hinton G.E., Sejnowski T.J. and Ackley D.H. (1984). Boltzmann machines: Constraint satisfaction networks that learn. Tech. Rep. TR-CMU-CS-84-119, Carnegie-Mellon University, Dept. of Computer Science.

[Hint87] Hinton G.E. (1987). Connectionist Learning Procedures. Technical Report CMU-CS-87-115 (version 2).

[Hint89] Hinton G.E. (1989). Deterministic Boltzmann learning performs steepest descent in weight-space. *Neural Computation 1*, 1, pp. 143-150.

[Hint90] Hinton G.E. and Nowlan S.J. (1990). The bootstrap Widrow-Hoff rule as a cluster-formation algorithm. *Neural Computation*, **2**, pp.355-362.

[Holl75] Holland J. (1975). *Adaptation in Natural and Artificial Systems*. University of Michigan Press.

[Holl89] Holley L.H. and Karplus M. (1989). Protein secondary structure prediction with a neural network. *Proc. Natl. Acad. Sci. USA*, **86**, pp.152-156.

[Hopf82] Hopfield J.J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proc. National Academy of Sciences, USA*, 41, pp.115-123.

[Hopf83] Hopfield J.J., Feinstein D. and Palmer R. (1983). Unlearning has stabilizing effects in collective memories. *Nature 304*, pp. 158-159.

[Hopf84] Hopfield J.J. (1984). Neurons with graded response have collective computational properties like those of two-state neurons. *Proc. Natl. Acad. Sci. U.S.A* , **81**, pp. 3088-3092.

[Hopf89] Hopfield J.J. and Tank D.W. (1989). Neural architectures and biophysics for sequence recognition. In *Neural Models of plasticity*, J.H. Byrne & W.O. Berry, eds., pp. 363-377.

[Horn89] Hornik K., Stinchcombe M., and White H. (1989). Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks. *Neural Networks*, 2:359-368.

[Huan89] Huang, X.D. and Jack, M.A. (1989). Semi-continuous hidden Markov models for speech signals. *Readings in Speech Recognition*, Waibel & Lee eds., Academic Press.

[Irin90] Irino T. and Kawahara H. (1990). A method for designing neural network multivariate analysis: application to speaker-independent vowel recognition. *Neural Computation 2*, 3, pp. 386-397.

[Jaco88] Jacobs R.A. (1988). Increased rates of convergence through learning rate adaptation. *Neural Networks*, 1, pp.295-307.

[Jac91a] Jacobs R.A., Jordan M.I., Nowlan S.J., and Hinton G.E. (1991). Adaptive Mixtures of Local Experts. *Neural Computation 3*, 1, pp. 79-87.

[Jac91b] Jacobs R.A., Jordan M.I., and Barto A.G. (1991). Task decomposition through competition in modular connectionist architecture: The what and where vision task. *Cognitive Science*, in press.

[Joha90] Johansson E.M., Dowla F.U. and Goodman D.M. (1990). Back-propagation learning for multi-layer feed-forward neural networks using the conjugate gradient method. UCRL-JC-104850 Preprint, Lawrence Livermore National Laboratory.

[Jord88] Jordan M.I. (1988). Supervised learning and systems with excess degrees of freedom. COINS tech. report 88-27.

[Jord90] Jordan M.I. (1990). Motor learning and the degrees of freedom problem. In M. Jeannerod, (ed.), *Attention and Performance*, XIII. Hillsdale, NJ: Erlbaum.

[Jouv88] Jouvet D. (1988). *Reconnaissance de Mots Connectes Independamment du Locuteur par des Methodes Statistiques.* Doctorate Thesis, ENST-88E006, Ecole National Superieure des Telecommunications, France.

[Judd88] Judd S. (1988). On the complexity of loading shallow neural networks. *Journal of Complexity*, 4.

[Karm90] Karmin E.D. (1990). A simple procedure for pruning back-propagation trained neural networks. *IEEE Trans. Neural Networks*, 1(2), pp.239-242.

[Kian65] Kiang N.Y.S., Watanabe T., Thomas E.C. and Clark L.F. (1965). *Discharge patterns of single fibers in the cat's auditory nerve fibers.* Cambdrige, MA: MIT Press.

[Kirk83] Kirkpatrick S., Gelatt C.D. Jr., Vecchi M.P. (1983). Optimization by simulated annealing. *Science* 220, pp. 671-680.

[Klat82] Klatt D. (1982) Prediction of perceived phonetic distance from critical-band spectra: a first step, *Proc. International Conference on Acoustics, Speech and Signal Processing 82*, pp. 1278-1281.

[Klop89] Klopf A.H. (1989). Classical conditioning phenomena predicted by a drive-reinforcement model of neuronal function. In *Neural Models of plasticity*, J.H. Byrne & W.O. Berry, eds., pp. 104-132.

[Koho88] Kohonen T. (1988). *Self-Organization and Associative Memory.* Springer-Verlag, New-York (second edition).

[Koho89] Kohonen T., Barna G., and Chrisley R. (1989). Statistical pattern recognition with neural networks: benchmarking studies. Tech. Report, Laboratory of Computer and Information Science, Helsinky University of Technology, Finland.

[Kole90] Kolen, J.F. and Pollack, J.B. 1990. Backpropagation is sensitive to initial conditions. *Complex Systems*, vol. 4, no. 3, pp. 269-280.

[Kuhn87] Kuhn G. (1987). A first look at phonetic discrimanation using connectionist models with recurrent links. CCRP IDA SCIMP working paper No.4/87, Institute for Defense Analysis, Princeton, NJ.

[Kuh90a] Kuhn G. and Herzberg N. (1990). Variations on training of recurrent networks. *Proc. 24th Conference on Information Sciences and Systems*, Princeton University, NJ.

[Kuh90b] Kuhn G., Watrous R.L. and Ladendorf B. (1990). Connected recognition with a recurrent network. *Speech Communication*, vol. 9, pp. 41-49.

[Lang88] Lang K.J. and Hinton G.E. (1988). The development of the Time-Delay Neural Network architecture for speech recognition. Tech. Report CMU-CS-88-152, Carnegie-Mellon University.

[Lars86] Larson J., Wong D. and Lynch G. (1986). Patterned stimulation at the theta frequency is optimal for induction of long-term potentiation. *Brain Research*, **368**, pp. 7-35.

[LeCu85] Le Cun Y. (1985). Une procedure d'apprentissage pour reseau a seuil assymetrique. [A learning procedure for assymetric threshold network]. Proc. of Cognitiva 85, pp. 599-604, Paris.

[LeC89c] Le Cun Y., Boser B., Denker J.S., Henderson D., Howard R.E., Hubbard W. and Jackel L.D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural Computation*, vol. 1, no. 4, pp.541-551.

[LeC89a] Le Cun Y. (1989). Generalization and network design strategies. Technical report CRG-TR-89-4, Department of Computer Science, University of Toronto.

[LeC89b] Le Cun Y. 1989. Generalization and network design strategies. *Connectionism in Perspective*, (Pfeifer, Schreter, Fogelman, Steels eds.), North Holland. pp. 143-155.

[LeC89d] Le Cun Y. et al. (1989). Handwritten digit recognition: Application of neural network chips and automatic learning. *IEEE Communications Magazine*, 27(11):41-46.

[LeC90] Le Cun Y., Kanter I., Solla S. (1990). Second order properties of error surfaces, learning time, generalization. Presented at the *Neural Networks for Computing* conference, Snowbird, April 1990.

[LeeH 89] Lee K.F., and Hon H.-W. (1989). Speaker-Independent Phone Recognition Using Hidden Markov Models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. ASSP-37, pp. 1641-1648.

[Lee 89] Lee K.F. (1989). *Automatic Speech Recognition: the development of the SPHINX system.*, Kluwer Academic Publ.

[Levi83] Levinson S.E., Rabiner L.R. and Sondhi M.M. (1983). An introduction to the application of the theory of probabilistic functions of a Markov process to automatic speech recognition. *Bell System Technical Journal*, vol. 64, no. 4, pp. 1035-1074.

[Levi90] Levin, E. (1990). Word recognition using hidden control neural architecture. *Proceedings of the International Conference on Acoustics, Speech and Signal Processing*, Albuquerque, NM, April 90, pp. 433-436.

[Lipp87] Lippman R. (1987). An introduction to computing with neural nets. *IEEE ASSP Magazine*, April 87, pp. 4-22.

[Lipp89] Lippman R. (1989). Review of neural networks for speech recognition. *Neural Computation* vol. 1, no. 1., pp. 1-38.

[Ljun83] Ljung and Soderstrom (1983). *Theory and Practice of recursive identification*, MIT Press.

[MacQ67] MacQueen J. (1967). Some methods of classification and analysis of multivariate observations. in LeCam L.M. and Neyman J. eds., *Proc. 5th Berkeley Symposium on Math., Stat. and Prob.*, University of California Press, Berkeley CA, p.281.

[Makh90] Makhoul, J. and El-Jaroudi, A. (1990). Posterior Probability Estimation with Neural Nets. Presented at the *Neural Networks for Computing* conference at Snowbird, April 1990.

[Mead89] Mead C.A. (1989). *Analog VLSI and Neural Systems*. Addison-Wesley, Reading, MA.

[Mel90] Mel B. and Koch C. (1990). Sigma-pi learning: On radial basis functions and cortical associative learning. In D. Touretzky, ed., *Advances in Neural Information Processing Systems 2*, pp. 474-481, Morgan Kaufmann.

[Merl86] Merlo E., De Mori R., Mercier G, and Palakal M. (1986). A continuous parameter and frequency domain based Markov model. *Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP-86)*, April 86, pp. 1597-1600.

[Mill83] Miller M.I. and Sachs M.B. (1983). Representation of stop consonants in the discharche patterns of auditory nerve fibers. *Journal of the Acoustical Society of America*, vol. 74, no. 2, pp.502-517.

[Mins69] Minsky M. and Papert S. (1969). *Perceptrons*, Cambridge MA: MIT Press.

[Moor88] Moore W.R. (1988). Conventional fault-tolerance and neural computers. NATO ASI Series vol.F41, *Neural Computers*, Eckmiller & v.d. Marlsburg eds. pp.29-37.

[Mor90a] Morgan, N. and Bourlard, H. (1990). Continuous speech recognition using multilayer perceptrons with hidden Markov models. In *Proceedings of the International Conference on Acoustics, Speech and Signal Processing*, Albuquerque, NM, April 90, pp. 413-416.

[Mor90b] Morgan, N. and Bourlard, H. (1990). Generalization and parameter estimation in feedforward nets: some experiments. D. Touretzky, ed., *Advances in Neural Information Processing Systems 2*, pp. 630-637, Morgan Kaufmann.

[Moze88] Mozer M.C. (1988). A focused back-propagation algorithm for temporal pattern recognition. Tech. Report CRG-TR-88-3, University of Toronto.

[Moze89] Mozer M.C., Smolensky S. (1989). Skeletonization: a technique for trimming the fat from a network via relevance assessment. *Advances in Neural Information Processing Systems 1*, (ed. D.S. Touretzky) Morgan Kauffman Publ., pp. 107-115.

[Nada88] Nadas, A., Nahamoo, D. and Picheny, M.A. (1988). On a model-robust training method for speech recognition. *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. ASSP-36, no. 9., pp. 1432-1436.

[Need70] Needleman S.B., Wunsch C.D. (1970). A general method applicable to the search of similarities in the amino acid sequence o two proteins. *Journal of Molecular Biology*, 48, pp.443-453.

[Nils65] Nilsson N.J. (1965). Learning Machines: Foundations of Trainable Pattern-Classifying Systems. McGraw-Hill, New-York.

[Nowl90] Nowlan S. (1990). Competing experts: an experimental investigation of associative mixture models. Technical Report CRG-TR-90-5, Dept. Computer Science, University of Toronto.

[Park85] Parker D.B. (1985). *Learning-logic*, TR-47, Cambridge, MA: MIT Center for computational research in economics and management science.

[Pavl88] Pavlides C., Greenstein J., Grudman M. and Winson J. (1988). Long-term potentiation in the dendate gyrus is induced preferentially on the positive phase of Θ-rhythm. *Brain Research*, vol. 439, pp. 383-387.

[Pear89] Pearlmutter B.A. (1989). Learning state space trajectories in recurrent neural networks. *Neural Computation* vol. 1, no. 2, pp. 263-269.

[Penr55] Penrose R. (1955). A generalized inverse for matrices. *Proc. Cambridge Philos. Soc.*, **51**, pp. 406-513.

[Pete90] Peterson C. (1990). Parallel distributed approaches to combinatorial optimization: benchmark studies on traveling salesman problem. *Neural Computation*, vol. 2, no. 3, pp. 261-269.

[Plat91] Platt J. (1991). A resource-allocating network for function interpolation. To appear in *Neural Computation*, vol.3, no. 2.

[Plau86] Plaut D.C., Nowlan S.J. and Hinton G.E. (1986). Experiments on learning by back-propagation. Tech. Report CMU-CS-86-126, Carnegie-Mellon University, Pittsburgh.

[Pogg89] Poggio T. and Girosi F. (1989). A theory of networks for approximation and learning. MIT AI Memo No. 1140.

[Poll84] Pollard D. (1984). *Convergence of stochastic processes*. New-York: Springer-Verlag.

[Pome89] Pomerleau D.A. (1989). ALVINN: an autonomous land vehicle in a neural network. *Advances in Neural Information Processing Systems 1*, ed. D.S. Touretzky, Morgan Kaufmann, pp. 305-313.

[Psal90] Psaltis D., Brady D. and Hsu K. (1990). Learning in optical neural computers. *Proc. Int. Joint Conf. on Neural Networks, IJCNN-90*, Washington DC. pp. II.72-II.75.

[Qian88] Qian N. and Sejnowski T.J. (1988). Predicting the secondary structure of globular proteins using neural network models. *Journal of Molecular Biology*, **202**, pp.865-884.

[Quin86] Quinlan J.R. (1986). Induction of Decision Trees. *Machine Learning*, 1(1), pp. 81-106.

[Rabi85] Rabiner L.R. and Levinson S.E. (1985). A speaker-independent, syntax-directed, connected word recognition system based on hidden Markov models and level building. *IEEE Trans. Acoust. Speech Signal Processing*, vol. ASSP-33, no. 3, pp.561-573.

[Rabi89] Rabiner L.R. (1989). A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, vol. 77, no. 2, Feb. 89., pp. 257-285.

[Robi90a] Robinson, T. and Fallside, F. (1990). Phoneme recognition from the TIMIT database using recurrent error propagation networks. Engineering Dept., Cambridge University, CUED/F-INFENG/TR 42.

[Robi90b] Robinson T., Holdsworth J., Patterson R. and Fallside F. (1990). A comparison of preprocessors for the Cambridge recurrent error propagation network speech recognition system. *Proceedings of the 1990 International Conference on Spoken Language Processing*, Kobe, Japan, pp. 1033-1036.

[Rose57] Rosenblatt F. (1957). *The Perceptron – a perceiving and recognizing automaton.* Report 85-460-1, Cornell Aeronautical Laboratory, Ithaca, N.Y.

[Rose62] Rosenblatt F. (1962). *Principles of Neurodynamics.* New York: Spartan.

[Ross89] Rossen M.L. (1989). *Speech Syllable Recognition with a Neural Network.* Ph.D. Thesis, Brown University.

[Rume86a] Rumelhart D.E., McClelland J.L. (eds.) (1986). *Parallel Distributed Processing*, volume 1. Bradford Books, MIT Press.

[Rume86b] Rumelhart D.E., Hinton G.E. and Williams R.J. (1986). Learning internal representation by error propagation. *Parallel Distributed Processing* volume 1. Rumelhart D.E. and McClelland J.L. (eds.), Bradford Books, MIT Press, pp. 318-362.

[Sach80] Sachs M.B. and Young E.D. (1979). Effects of nonlinearities on speech encoding in the auditory nerve. *Journal of the Acoustical Society of America*, vol. 68, no. 3, pp.858-875.

[Sejn86] Sejnowski T. and Rosenberg C.R. (1986). NETtalk: a prallel network that learns to read aloud. Technical report JHU/EECS-86/01, John Hopkins University, Electrical and Computer Science Department.

[Sene84] Seneff S. (1984). Pitch and spectral estimation of speech based on an auditory synchrony model. *Proc. International Conference on Acoustics, Speech and Signal Processing* (ICASSP 84), San Diego.

[Sene85] Seneff S. (1985). Pitch and spectral estimation of speech based on an auditory synchrony model. RLE Technical report no. 504, Cambridge, MA: MIT Press.

[Sene86] Seneff S. (1986). A joint synchrony/mean-rate model of auditory speech processing. *Proc. International Conference on Acoustics, Speech and Signal Processing* (ICASSP 86), Tokyo, pp. 37.8.1-37.8.4.

[Sene88b] Seneff S. (1988). A joint synchrony/mean-rate model of auditory speech processing. *Journal of Phonetics*, vol. 16, no. 1, pp. 55-76.

[Sen+88] Seneff S. and Zue V. (1988). Transcription and alignment of the TIMIT database, in J.S. Garofolo ed. *Getting started with the DARPA TIMIT CD-ROM : An acoustic phonetic continous speech database*. National Inst. of Stand. and Tech. (NIST), Gaithersburgh, MD.

[OSha87] O'Shaughnessy, D. (1987). *Speech Communication Human and Machine*. Addison Wesley.

[Sine83] Sinex D.G. and Geisler C.D. (1983). Response of auditory nerve fibers to consonant-vowel syllables. *Journal of the Acoustical Society of America*, vol. 73, no. 2, pp.602-615.

[Smit81] Smith T.F. and Waterman W.S. (1981). Identification of common molecular subsequences. *Journal of Molecular Biology*, **147**, pp.195-197.

[Stev75] Stevens K.N. (1975) The potential role of properties detectors in the perception of consonants, in *Auditory analysis and perception of speech* G. Fant and M.A. Tatham ed., Academic Press, London, pp. 303-330.

[Stev81] Stevens K.N. and Blumstein S.E. (1981). The search for invariant acoustic correlates of phonetic features. *Perspectives on the study of speech* P.D. Eimas and J.L. Miller ed., Lawrence Erlbaum ass. pp. 1-38.

[Stev90] Stevenson M., Winter R. and Widrow B. (1990). Sensitivity of feedforward neural networks to weight errors. *IEEE Trans. on Neural Networks*, vol. 1, pp.71-80.

[Stew73] Stewart G.W. (1973). Introduction to matrix computations. *Academic Press.*

[Stin89] Stinchcombe M. and White H. (1989). Universal approximation using feedforward networks with non-sigmoid hidden layer activation function. *Proceedings of the International Joint Conference on Neural Networks 89*, Washington D.C. (IEEE), vol. I, pp. 613-617.

[Stor82] Stormo G.D., Schneider T.D., Gold L. and Ehrenfeucht A. (1982). Use of the perceptron algorithm to distinguish translational initiation sites in *E. Coli. Nucleic Acid Research*, **10**, pp.2997-3010.

[Tam89] Tam D.C. and Perkel D.H. (1989). Quantitative modeling of synaptic plasticity. *Computational Models of Learning in Simple Neural Systems*, Hawkins R.D. & Bower G.H. eds. Academic Press. pp. 1-30.

[Tebe91] Tebelskis J., Waibel A., and Petek B. (1991). Continuous speech recognition using linked predictive networks. To appear in D. Touretzky, ed., *Advances in Neural Information Processing Systems 3*, Morgan Kaufmann.

[Thom86] Thompson R.F. (1986). The neurobiology of learning and memory. *Science*, **233**, pp. 941-947.

[Tikh77] Tikhonov A.N. and Arsenin V.Y. (1977). *Solutions of Ill-posed Problems.*, W.H. Winston, Washington D.C.

[Tish89] Tishby N., Levin E., and Solla S.A. (1989). Consistent inferences of probabilities in layered networks: Predictions and generalization. Proc. International Joint Conference on Neural Networks, IJCNN-89 vol. II, pp. 103-109.

[Tsoi91] Tsoi A.C., and Pearson R.A. (1991). Comparison of three classification techniques: CART, C4.5, and multi-layer perceptron. To appear in D. Touretzky, ed., *Advances in Neural Information Processing Systems 3*, Morgan Kaufmann.

[Vapn71] Vapnik V.N. and Chervonenkis A.Ya. (1971). On the uniform convergence of relative frequencies of events to their probabilities. *Th. Prob. and its Applications*, V17:N2, pp. 264-280.

[Vapn82] Vapnik V.N. (1982). *Estimation of Dependences Based on Empirical Data.*, New-York , Springer-Verlag.

[Wahb82] Wahba G. (1982). Constrained regularization for ill-posed linear operator equations, with applications in meteorology and medecine. *Statistical Decision Theory and Related Topics III*, vol. 2, Gupta and Berger eds., Academic Press.

[Waib87] Waibel A., Hanazawa, Hinton G., Shikano K. and Lang. K. (1987). Phoneme recognition using Time-Delay Neural Networks. TR-I-0006, ATR Interpreting Telephony Research Laboratories.

[Waib88] Waibel A., Hanazawa T. and Shikano K. (1988). Phoneme recognition: neural networks *vs* hidden Markov models. *Proceedings of the International Conference on Acoustics, Speech and Signal Processing*, (ICASSP88), New-York NY, April 88, pp.107-110.

[Waib89] Waibel A., Sawai H. and Shikano K. (1989). Modularity and scaling in large phonemic neural networks. *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. ASSP-37, pp. 1888-1898.

[Wang89] Wang H., Wu J. and Tang P. (1989). Superfamily expands. *Nature*, **337**, pp.514.

[Watr89] Watrous R. (1989). Context-modulated discrimination of similar vowels using second-order connectionist networks. CRG-TR-89-5, University of Toronto.

[Werb74] Werbos P.J. (1974). *Beyond regression: New tools for prediction and analysis in the behavioral sciences.* Ph.D. thesis Harvard University.

[Werb88] Werbos P.J. (1988). Generalization of back-propagation with application to a recurrent gas market model. *Neural Networks*, vol. 1, no. 4, pp. 339-356.

[Whi89a] Whitley D. and Hanson T. (1989). Optimizing neural networks using faster, more accurate genetic search. *Proc. Third International Conference on Genetic Algorithms*, J.D. Shafer e l., Morgan Kaufmann, pp. 391-396.

[Whi89b] White H. (1989). Learning in artificial neural networks: A statistical perspective. *Neural Computation*, vol. 1, no. 4, pp. 425-464.

[Whi91] White H. (1991). An overview of representation and convergence results for multilayer feedforward networks. To appear in *Advances in Neural Information Processing Systems 3*, (ed. D.S. Touretzky) Morgan Kauffman Publ.

[Wilb83] Wilbur W.J. and Lipman D.J. (1983). Rapid similarity searches of nucleic acids and protein data banks. *Proc. Natl. Acad. Sci. USA*, **80**, pp.726-730.

[Will88] Williams R.J. and Zipser D. (1988). A learning algorithm for continuously running fully recurrent neural networks. ICS report 8805, Institute for Cognitive Science, University of California at San Diego.

[Wil88b] Williams A.F. and Barclay A.N. (1988). The immunoglobulin superfamily domains for cell surface recognition. *Annual Review of Immunology*, **6**, pp.381-405.

[Wins90] Winson J. (1990). The meaning of dreams. *Scientific American*, Nov. 1990, pp. 86-96.

[Youn79] Young E.D. and Sachs M.B. (1979). Representation of steady-state vowels in the temporal aspects of the discharge pattern of population of auditory nerve fibers. *Journal of the Acoustical Society of America*, vol. 66, no. 5, pp.1381-1403.

[Zue90a] Zue V., Seneff S. and Glass J. (1990). Speech database development: TIMIT and beyond. *Speech Communication*, Vol. 9. No. 4. august 1990, pp. 351-356.

[Zue90b] Zue V., Glass J., Goddeau D., Goodine D., Leung H., McCandless M., Phillips M., Polifroni J., Seneff S. and Whitney D. (1990). Recent progress on the MIT VOYAGER spoken language system. *Proc. Int. Conf. Spoken Languague Processing 1990*, Kobe, Japan, p.29.6.1.

[Zwic80] Zwicker E. and Terhardt E. (1980). Analytical expressions for critical band rate and critical bandwidths as a function of frequency. *Journal of the Acoustical Society of America*, vol. 68, no. 5, pp.1523-1525.