Automatic Structuring of Neural Networks for Spatio-Temporal Real-World Applications

Zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften

der Fakultät für Informatik

der Universität Karlsruhe (Technische Universität)

vorgelegte

Dissertation

von

Ulrich Bodenhausen

aus Korbach

Tag der mündlichen Prüfung: 13.6.1994

Erster Gutachter: Prof. Dr. Alexander Waibel

Zweiter Gutachter: Prof. Dr. Scott E. Fahlman

Lebenslauf

Persönliche Daten

Geburtstag:	18.11.1963
Geburtsort:	Korbach
Familienstand:	ledig

Schulbesuch

1970 - 1974	Grundschule: Westwallschule in Korbach
1974 - 1983	Gymnasium: Alte Landesschule Korbach

Bundeswehr

Wehrdienstuntauglichkeit wegen Kurzsichtigkeit

Studium

1983 - 1989	Studium der Physik an der Philipps-Universität Marburg Studienschwerpunkte im Hauptstudium: Informatik, Physik der Energieversorgung, Angewandte Physik, Vordiplom 1985, Abschlußdiplom1989: Diplom-Physiker

Berufliche Tätigkeit

08/89 bis 12/89	Forschungsassistent (Visiting Research Professional) bei Prof. J. McClelland, Carnegie Mellon University, Pittsburgh, USA
01/90 bis 04/91	Forschungsassistent (Visiting Research Professional) bei Dr. A. Waibel, Carnegie Mellon University, Pittsburgh, USA
seit 04/91	Wissenschaftlicher Mitarbeiter bei Prof. Waibel an der Universität Karlsruhe, Fachbereich Informatik

Promotion

04/91 bis 06/94	Thema: "Automatic Structuring of Neural Networks for Spatio-Temporal Real-World Applications" Betreuer: Prof. A. Waibel, Universität Karlsruhe und Prof. S. Fahlman, Carnegie Mellon University, Pittsburgh, USA

Sonstige Kenntnisse/Aktivitäten

- sehr gute Englischkenntnisse
- gute Französischkenntnisse
- Hobbies: - Musik
  - Fotografie
  - Modellflugsport

PUBLICATIONS:

Philipps Universität Marburg:

“Content-Addressable Storage in Asymmetric Recirculation Networks”, Ulrich Bodenhausen, Diplom Thesis (M.S. Thesis), June 1989

“The Tempo Algorithm: Learning in a Neural Network with Adaptive Time-Delays”, Ulrich Bodenhausen, Proceedings of the International Joint Conference on Neural Networks, Washington D.C., January 1990.

Carnegie Mellon University:

“Learning Internal Representations of Pattern Sequences in a Neural Network with Adaptive Time Delays”, Ulrich Bodenhausen, Proceedings of the International Joint Conference on Neural Networks, San Diego, June 1990.

“The Tempo 2 Algorithm: Adjusting Time-Delays By Supervised Learning”, Ulrich Bodenhausen and Alex Waibel, NIPS 90, Denver, November 1990.

“Learning the Architecture of Neural Networks for Speech Recognition”, Ulrich Bodenhausen and Alex Waibel, ICASSP 91, Toronto, April 1991.

Universität Karlsruhe:

“Application Oriented Automatic Structuring of Time-Delay Neural Networks for High Performance Character and Speech recognition”, Ulrich Bodenhausen and Alex Waibel, In: Proceedings ICNN 93, San Francisco, March 1993.

“Connectionist Architectural Learning for High Performance Character and Speech Recognition”, Bodenhausen, U., and Manke, S., In: Proceedings ICASSP-93, Minneapolis, April 1993.

“Flexibility Through Incremental Learning: Neural Networks for Text Categorization”, Geutner, P., Bodenhausen, U., and Waibel, A., In: Proceedings World Congress on Neural Networks, Portland, Oregon, 1993

“Automatically Structured Neural Networks For Handwritten Character and Word Recognition”, Bodenhausen, U., and Manke, S., In: Proceedings ICANN 93, Amsterdam, September 1993

“Tuning By Doing: Flexibility Through Automatic Structure Optimization”, Bodenhausen, U., and Waibel, A. , In: Proceedings Eurospeech 93, Berlin, September 1993

“A Connectionist Recognizer For On-Line Cursive Handwriting Recognition”, S. Manke, and U. Bodenhausen, In: Proceedings ICASSP 94, Adelaide, April 1994.

Contents

Abstract

The successful application of speech recognition (SR) and on-line handwriting recognition (OLHR) systems to new domains greatly depend on the tuning of a recognizer's architecture to a new task. Architectural tuning is especially important if the amount of training data is small because the amount of training data limits the number of trainable parameters that can be estimated properly using an automatic learning algorithm. The number of trainable parameters of a connectionist SR or OLHR is dependent on architectural parameters like the width of input windows over time, the number of hidden units and the number of state units. Each of these architectural parameters provides different functionality in the system and can not be optimized independently. Manual optimization of these architectural parameters is time-consuming and expensive. Automatic optimization algorithms can free the developer of SR and OLHR applications from this task.

In this thesis I develop and evaluate novel methods that allocate connectionist resources for spatio-temporal classification problems automatically. The methods are evaluated under the following **evaluation criteria**:

- Suitability for small systems (~ 1,000 parameters) as well as for large systems (more than 10,000 parameters): Is the proposed method efficient for various sizes of the system?
- Ease of use for non-expert users: How much knowledge is necessary to adapt the system to a customized application?

- Final performance: Can the automatically optimized system compete with state-of-the-art well engineered systems?

Several algorithms were developed and evaluated in this thesis. The Automatic Structure Optimization (ASO) algorithm performed best under the above criteria. ASO **automatically optimizes**

- the width of the input windows over time which allow the following unit of the neural network to capture a certain amount of temporal context of the input signal;
- the number of hidden units which allow the neural network to learn non-linear classification boundaries;
- the number of states that are used to model segments of the spatio-temporal input, such as acoustic segments of speech or strokes of on-line handwriting.

The ASO algorithm uses a constructive approach to find the best architecture. Training starts with a neural network of minimum size. Resources are added to specifically improve parts of the network which are involved in classification errors. ASO was developed on the recognition of spoken letters and improved the performance on an independent test set from 88.0% to 92.2% over a manually tuned architecture. The performances of architectures found by ASO for different domains and databases are also compared to architectures optimized manually by other researchers. For example, ASO improved the performance on on-line handwritten digits from 98.5% to 99.5% over a manually optimized architecture. It is also shown that ASO can successfully adapt to different sizes of the training database and that it can be applied to the recognition of connected spoken letters.

The ASO algorithm is applicable to all classification problems with spatio-temporal input. It was tested on speech and on-line handwriting, as two instances of such tasks. The approach is new, requires no domain specific knowledge by the user and is efficient. It is shown for the first time that

- fully automatic tuning of all relevant architectural parameters of speech and on-line handwriting recognizers (window widths, number of hidden units and states) to the domain and the available amount of training data is actually possible with the ASO algorithm
- automatic tuning by ASO is efficient, both in terms of computational effort and final performance.

1. Introduction to the Problem

The design of better human-computer interfaces has become an important area of Computer Science. Speech recognition, on-line handwriting recognition, gesture recognition and lipreading are promising approaches towards better multi-modal interfaces because they should provide more efficient and natural communication between humans and machines.

Artificial neural networks have become an important paradigm for these systems. All of the systems mentioned above require the processing of dynamically changing patterns, which are often called “pattern sequences”. Section 1.1 gives a short introduction to the processing of pattern sequences with neural networks.

The design of these systems can be either special or general purpose. Figure 1 on page 4 shows the differences in the case of a speech recognition system. The general purpose system is usable by all possible users and has a very large vocabulary that makes it usable in many applications. The special purpose system is specialized on one application and may even be trained for one speaker only. While the general purpose system is the ultimate goal, there still remains a performance gap for them to become truly “general purpose”. On the other hand it is now possible to design special purpose systems for well defined and small domains with high recognition accuracies. The reason why they are not used more frequently is that these systems are far from being “automatically adaptable” to new, customized domains. Despite the use of powerful learning techniques, they still require time-consuming tuning by experts, which is often too expensive for the limited number of customers interested in that particular application.

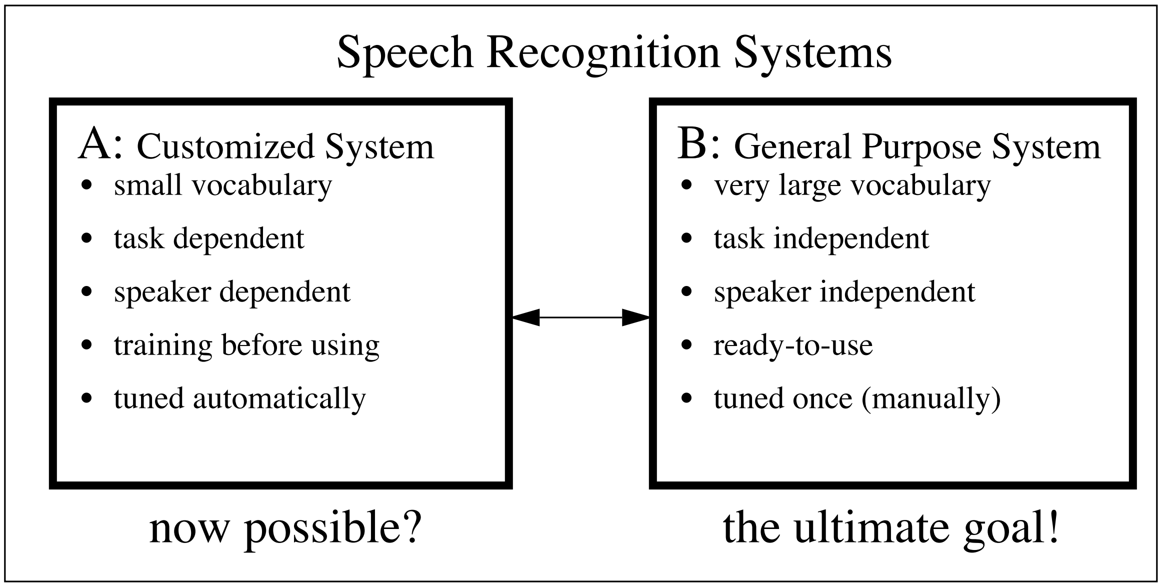

Figure 1. A comparison of possible speech recognition system options. System A has to be adapted for a customized application. System B is a general purpose system that needs no adaptation, but it is harder to build. System A can be realized earlier, but in order to be practical it needs an automatic tuning algorithm.

In this thesis I will propose and evaluate several algorithms that do the tuning of neural network based speech and on-line handwriting recognition systems automatically. The techniques also apply to gesture recognition and lipreading, but they were tested on speech recognition (SR) and on-line handwriting recognition (OLHR). The next sections give a short introduction to the following problems:

- How can pattern sequences be processed with neural networks?
- Why do these systems need tuning to a particular task?
- Manual tuning vs. automatic tuning and their feasibility depending on the size of the domain.

1.1 The Processing of Pattern Sequences with Neural Networks

Many real-world applications like speech recognition or on-line handwriting recognition require the processing of pattern sequences. Several neural network architectures have been proposed for this problem so far. A very successful approach to the processing of temporal context with neural networks is the implementation of time-delays [Waibel, 1987], [Tank, 1987], [Waibel, 1989], [Lang, 1990], [Bodenhausen, 1990a], [Bodenhausen, 1990b]. This approach is not only technically plausible, but also motivated by knowledge about the brain[1].

1. Real axons have a limited conduction speed (which is dependent on the diameter of the axon and whether it is myelinated or not). Additionally, the length of most axons is much greater than the euclidean distance between the connected neurons. This leads to a great variety of different time-delays in the brain.

In contrast to fully connected general purpose neural networks, networks with time-delays can be seen as highly structured networks that are specialized on certain spatio-temporal tasks. In real-world applications like speech recognition and on-line handwriting recognition, these highly structured neural networks have been shown to be superior to fully connected networks [Waibel, 1989], [Guyon, 1991]. The importance of the network structure has also recently been examined by using the *nonparametric statistical inference framework* by Geman et al. [Geman, 1992]. They come to the conclusion that "dedicated machines are harder to build but easier to train" and suggest that important properties of the task have to be built into the architecture of the network. Similar conclusions have been made by Minsky and Papert [Minsky, 1988].

1.2 Feasibility of Manual Tuning

Manual design of the network structure can be very time-consuming for real-world applications. This may be feasible for general purpose systems like speaker independent, very large vocabulary speech recognition systems that are optimized once and then applied without further adaptation to a special task. Despite the great effort that is still needed to improve these general purpose systems, there is a considerable number of applications that require the best possible performance on a small, well defined, and customized domain. Achieving the best possible performance for these applications greatly depends on the tuning of the architecture to the particular task, especially if the amount of training data is small (which is often true for customized applications). While it is possible to build these systems, the required manual tuning is not tolerable for a system that has to be trained for each application that it is used for. The importance of adaptability and customization of speech recognition systems has also recently been pointed out by Lee [Lee, 1993] and Wilpon [Wilpon, 1993]. Lee emphasizes that the adaptation process should be "transparent, real-time, and incremental" and that speech recognition systems should be "customizable for ultra-high performance". Additionally, speech recognition systems "should be intuitive and easy to use for the developer - it should be possible to develop applications on top of the technology without having to understand speech algorithms". Wilpon includes "quick prototyping and development of new products and services" among the challenges that have to be solved before universal use of automatic speech recognition can be achieved in a telephone network [Wilpon, 1993]. He points out that "the technology must support creation of new products and service ideas based on speech in an efficient and timely fashion. Users should not be required to wait weeks or months for new products or services".

Multi-modal systems combining speech recognition, on-line handwriting recognition, lipreading, gesture recognition etc. can benefit even more from automatic tuning because many of these modalities can in principle be realized by similar algorithms, but need different architectures due to different characteristics of the domains. This means that the required tuning effort of multi-modal systems grows with the number of input modes (speech, gesture, lipreading, etc.). This thesis proposes and evaluates several algorithms that do the tuning of all relevant architectural parameters automatically.

1.3 Structured vs. Fully Connected Neural Networks

There are two reasons for the superior generalization ability of highly structured over fully connected architectures for specialized applications. Both are related to the phenomenon of overfitting which means that a network does not only learn the training data, but can also learn the noise in this data. These two explanations are:

1. There is a relationship between the number of trainable parameters, amount of training data and generalization (see [Denker, 1987], [Baum, 1989], [Solla, 1989], [Moody, 1991] and others)[1]. Networks with too many trainable parameters for the given amount of training data learn well, but do not generalize well because they have too many degrees of freedom and can learn to fit the noise in the training data. With too few trainable parameters, the network does not have enough capacity to learn the training data and also performs very poorly on the testing data. Imposing structure on the network can increase the generalization performance by reducing the number of trainable parameters [Waibel, 1987], [Lang, 1989], [Waibel, 1989a], [Waibel, 1989b], [Le Cun, 1989].

2. Imposing structure on the network can also highlight important features of the data and decrease the ability of the network to learn unimportant features like noise. Other unimportant if not undesirable features are task specific. In speech recognition, phonemes can have varying durations depending on the speakers or the words that they appear in. It would be undesirable that the neural network learns these durations from the training data. Thus structures that are time-shift invariant highlight features that are independent of position in time [Waibel, 1989a].

From the neuroscience viewpoint it is well-known that the brain is highly structured and not fully connected [Kandel, 1981]. Performance comparisons between fully connected and highly structured brains are not possible because the are no fully connected brains. However, it is believed that the human brain has developed such a highly structured connectivity for *some* reason. Although the processes that are involved in the developing stages of the brain (induction, proliferation, cell migration, axonal outgrows, dendrite elaboration, cell death, and neural recognition) are well understood [Kandel, 1981], very few connectionist models have been build that incorporate a connectionist growth (structuring) process that is also applicable to real-world problems.

Aside from the number of parameters and the structure of the network, the amount of training (the number of training epochs) is important for generalization. If a network with many parameters and many degrees of freedom is trained too long it will start to fit the noise in the training data. This can be avoided by terminating the training when the performance on an independent validation set starts to decline. This method is important for both general purpose and specially structured networks, but with lesser importance for the specially structured networks because of their limited ability to fit the noise in the training data.

1. Hidden Markov Models (HMMs) and hybrid HMM/neural network systems also underly similar generalization constraints

1.4 The Contribution of this Thesis

The following thesis proposes and evaluates algorithms that optimize the structure of neural networks for spatio-temporal tasks automatically. The goal of the thesis is an automatic resource allocation model that finds a good structure for the given task and performs well under the following evaluation criteria:

- Suitability for small systems (~ 1,000 parameters) as well as for large systems (more than 10,000 parameters): Is the proposed method efficient for various sizes of the system?
- Ease of use for non-expert users: How much knowledge is necessary to adapt the system to a customized application?
- Final performance: Can the automatically optimized system compete with state-of-the-art well engineered systems?

The algorithm developed in this thesis (the *Automatic Structure Optimization* algorithm) fulfills these criteria very well.

1.5 Outline

Related work is summarized and discussed in the second chapter. The analysis of the advantages and disadvantages of other architectural optimization methods leads to a clear definition of desired properties of the algorithm. The databases that are used for performance evaluations are described in chapter 3. In chapter 4, initial experiments with the *Tempo 2* algorithm are described. The most successful algorithm developed in this thesis, the *Automatic Structure Optimization* (ASO) algorithm is described in chapter 5. Chapter 6 summarizes the performances of the ASO algorithm on segmented data. In chapter 7, an *Automatic Validation Analyzing Control System* (AVACS) for extremely small databases is proposed. Chapter 8 describes the application of the ASO algorithm to the recognition of connected letters. The ASO/AVACS combination is evaluated under the criteria of this thesis in Chapter 9. Chapter 10 concludes this thesis with summary and conclusions.

2. Related Work

This thesis can be seen in the context of four research areas: Spatio-temporal processing with neural networks, speech recognition, on-line handwriting recognition and architectural selection/ optimization of neural networks. A short review of the current research in these fields is attempted in the following sections. Due to the amount of research in these fields it is only possible to mention research that is directly related to the topic of this thesis.

2.1 Spatio-Temporal Processing with Neural Networks

Many real-world applications require the processing of patterns that evolve over time. In the following it is assumed that time is quantized into discrete steps. This is possible because many time series are intrinsically discrete, and continuous series can be assumed as locally stationary and can be sampled at a fixed interval. There are two basic concepts for the processing of spatio-temporal patterns with artificial neural networks:

- Recurrent networks
- Networks using time-delays.

The following sections summarize the key ideas of both concepts an discuss the differences that are relevant in the context of this thesis. A recent article by Mozer [Mozer, 1993] gives a more general overview about temporal sequence processing with neural networks.

2.1.1 Spatio-Temporal Processing with Recurrent Neural Networks

The basic principle of a recurrent network is shown in Figure 2 on page 10. In this example of a network with two units only, the activation of unit U_1 at time t activates unit U_2 at time t+1 via the connection w_{21} etc. This recurrent network can be replaced by a much larger feedforward network. Thus the learning rules for the recurrent network can be derived from the learning rules for the equivalent feedforward network [Rumelhart, 1986]. Although these fully recurrent networks are very powerful, they are not used very often due to the complexity of training. A simpler version of the fully recurrent network was proposed by Elman [Elman, 1988] (see Figure 3 on page 10). The Simple Recurrent Network (SRN) uses both a copy of the previous hidden representation (at time t-1) and the current input (at time t) as input to the network. The SRN is probably the most frequently used recurrent network. It could be shown that the SRN is able to learn finite state grammars [Cleeremans, 1989].

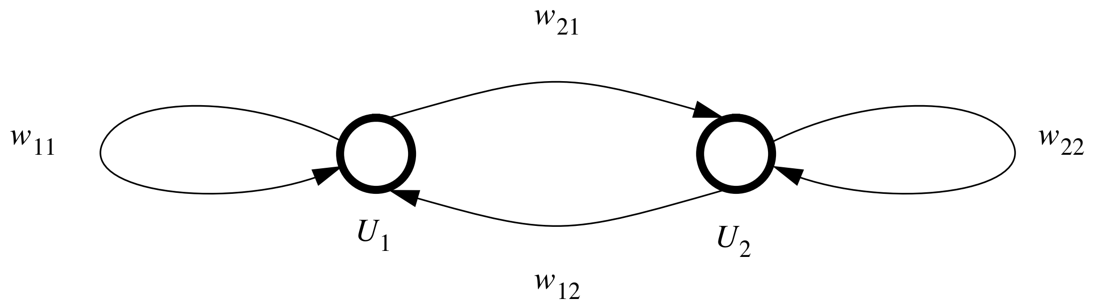

Figure 2. A recurrent network with two units. All connections have a time-delay of one. The activation of unit U_1 at time t activates unit U_2 at time t+1 via the connection w_{21} etc.

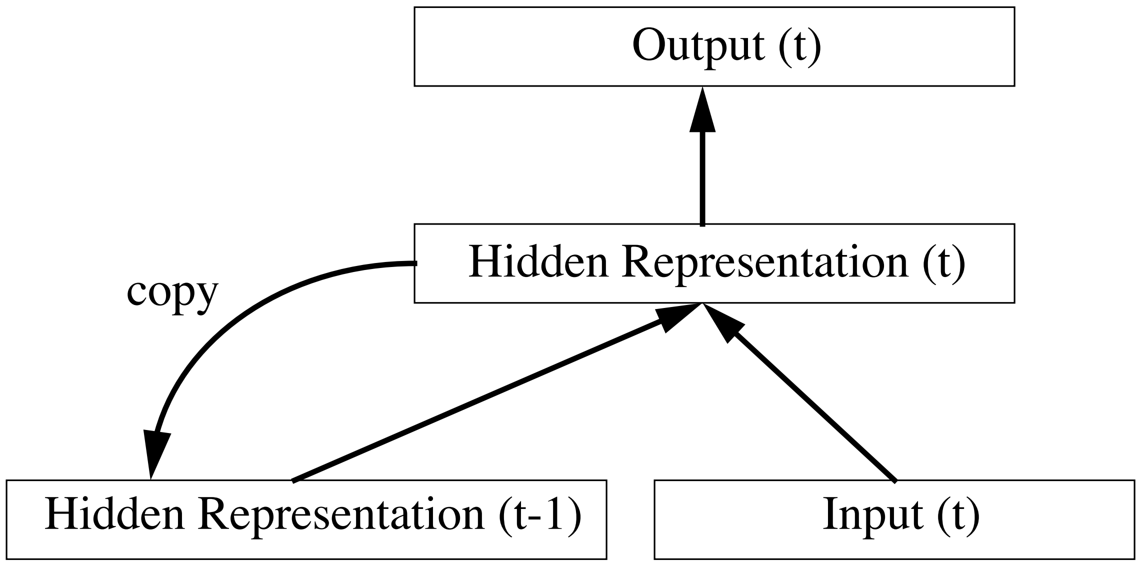

Figure 3. The Simple Recurrent Network (SRN): The SRN proposed by Elman uses both a copy of the previous hidden representation (t-1) and the current input (t) as input to the network.

One of the most interesting features of recurrent networks is the general ability to learn the required memory depth automatically from the training data [Cleeremans, 1989]. However, there are two important drawbacks:

- The network has no direct information about the history of the pattern sequence. The history has to be learned by the recurrent weights of the network, which takes many epochs of training. Franzini reported a significant reduction in training time when he removed recurrent connections from his speech recognition system [Franzini, 1990].
- Due to the nature of recurrence, the recent past is always favored over the distant past. A recent paper by Bengio et al. studies the problem of learning long-time dependencies with recurrent networks [Bengio, 1993].

These two drawbacks may be the reason that recurrent networks are not used for speech recognition very often. Especially the long training time of recurrent networks is too critical for real-world applications. Although Franzini got a small increase in the error rate when he removed the recurrent connections from his speech recognition system, the reduced training time allowed some improvements that were much more effective and improved the system much more than the recurrent connections [Franzini, 1990].

2.1.2 Spatio-Temporal Processing using Time-Delays

The basic principle of spatio-temporal processing with time-delayed connections is shown in Figure 4 on page 12. The decision of the neural network at time t is based on the inputs at time *(t-n), (t-n+1) .. (t)*. This way, a well defined history of the temporal sequence is considered. The sequential processing with time-delays is different from the processing with recurrent connections in the following ways:

- The internal memory of a network using time-delays is limited to a temporal context that is strictly limited by the maximum time-delay *n*. It is not possible to take information into account that occurred before *(t-n)*. Although this seems to be a serious disadvantage because the structure has to be predefined before training, this can also be an advantage: The neural network does not attempt to extract too much contextual information from the training data which may be helpful for learning of the training data, but might decrease the test performance.
- Each time-delayed feature can be considered equally. In contrast, recurrent networks always favor the recent past over the distant past. This “weighting” can also be learned by the weights of time-delayed connections if it is desirable, but other “weightings” of the past can be learned, too.
- Sequential learning with time-delays is generally fast compared to learning with recurrent connections because an important information about the task (i.e. temporal relationships) is already build into the structure of the network and needs not be inferred indirectly from numerous examples.

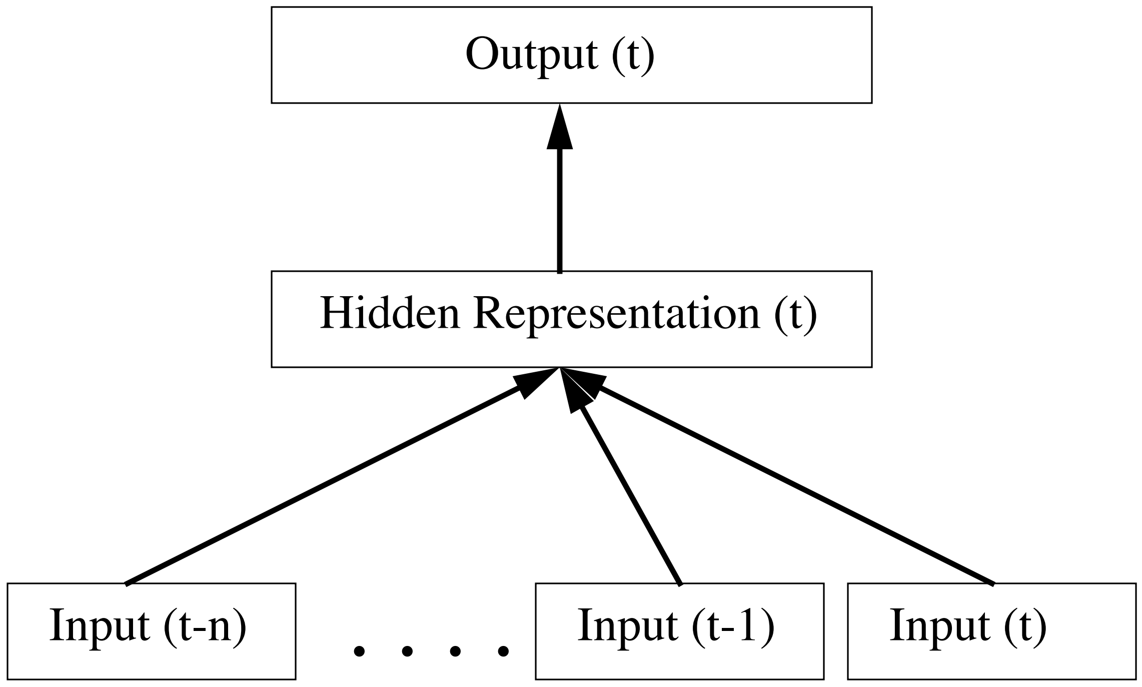

Figure 4. Spatio-temporal Processing using time-delays: N patterns from a sequence of patterns are simultaneously fed into the neural network.

Several interesting architectures using time-delays have been proposed:

2.1.2.1 Spatio-Temporal Processing using Smooth Time-Delay Functions

Several approaches have been proposed that use smooth delay functions over time [Tank, 1987], [Unnikrishnan, 1991], [Unnikrishnan, 1992]1 or Gamma functions [de Vries, 1991] and [de Vries, 1992] instead of the rectangular hat-shaped function. These smooth functions have the advantage that it is possible to learn various parameters of these functions, such as the time-delays [Unnikrishnan, 1991] or Gamma memory parameters [de Vries, 1992]. The approach by Unnikrishnan et al. was tested on the Texas Instruments (TI) digits database. The approach by de Vries and Principe awaits evaluation.

2.1.2.2 The Time-Delay Neural Network (TDNN)

The Time-Delay Neural Network (TDNN) architecture was originally designed for speech recognition [Waibel, 1987], [Waibel, 1989a], but was also later used for on-line handwriting recognition [Guyon, 1991]. The network uses time-delays between all layers to represent temporal relationships between events in time. Besides that, the primary goal was to design an architecture that provides non-linear classification invariant under translation in time. This has been realized

1. The approach is very similar to the Tempo 2 approach described in Chapter 4 on page 37.

by sliding input-windows over time and linking of connection weights (see Figure 5 on page 13). In contrast to other implementations of multi-layer networks using time-delays, the concept of the sliding input window over time is also used for the connections from the first hidden layer to the second hidden layer, which are called "phoneme units" since they represent the score[1] of a certain phoneme or sub-phoneme at a given time.

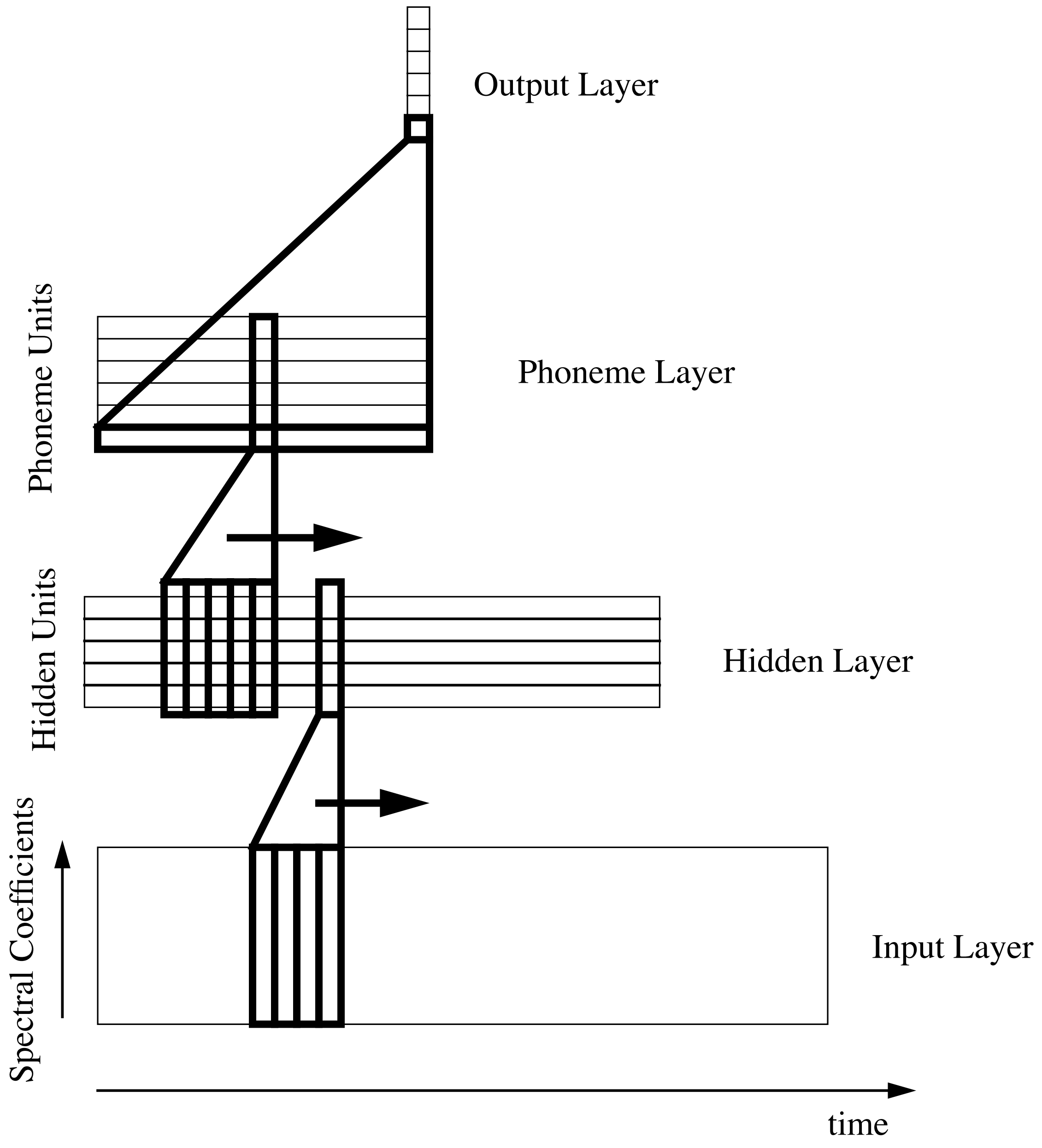

Figure 5. The Time-Delay Neural Network (see text for explanation).

1. or the a posteriori probability at a given time, depending on the training method (see Section 2.4.5 on page 23)

The output of the network is computed by adding all of these scores over time and applying a non-linear function like the sigmoid function to the sum. The TDNN architecture with two hidden layers using sliding input windows over time leads to a relatively small number of independently trainable parameters. Together with temporal shift invariance this leads to good generalization results with this architecture.

The evaluation of the TDNN on a /b/, /d/, and /g/ phoneme classification task were very promising. The application of the original TDNN to word recognition [Bottou, 1990] could be achieved by a straightforward extension of the original concept: The output units represented words instead of phonemes. In consequence, the network grew considerably and many independent weights had to be trained. Although quite successful, this concept was not pursued very long because phonemes were not explicitly expressed. This was the reason for the incorporation of non-linear time alignment into the TDNN architecture, which is described in the next section.

2.1.2.3 The Multi State Time-Delay Neural Network (MS-TDNN)

TDNNs have been extended to Multi-State Time-Delay Neural Networks (MS-TDNNs) [Haffner, 1991a], [Haffner, 1991b], [Haffner, 1992a], [Haffner, 1992b] that allow the recognition of sequences of ordered events that have to be observed jointly Figure 6 on page 15. The extension of the original TDNN architecture is the incorporation of non-linear time alignment between the phone units and the output units. The time alignment is realized by the *Dynamic Time Warping* (DTW) algorithm proposed by Sakoe and Chiba [Sakoe, 1978].

In the MS-TDNN architecture the powerful classification properties of the first two hidden layers of the original TDNN (with the sliding input windows over time) are used to compute frame-level scores for sub-word units like phonemes or phones. For clarity let us assume that the frame-level scores of phonemes are computed. Then words are modeled by a sequence of states *(1, ..., s, ...N)* that can have variable durations. The score of a word in the vocabulary accumulates frame-level phoneme scores, which are a function of the output

$$\vec{Y}(t) = (Y_1(t), Y_2(t), \ldots, Y_l(t))$$

of the front end TDNN.

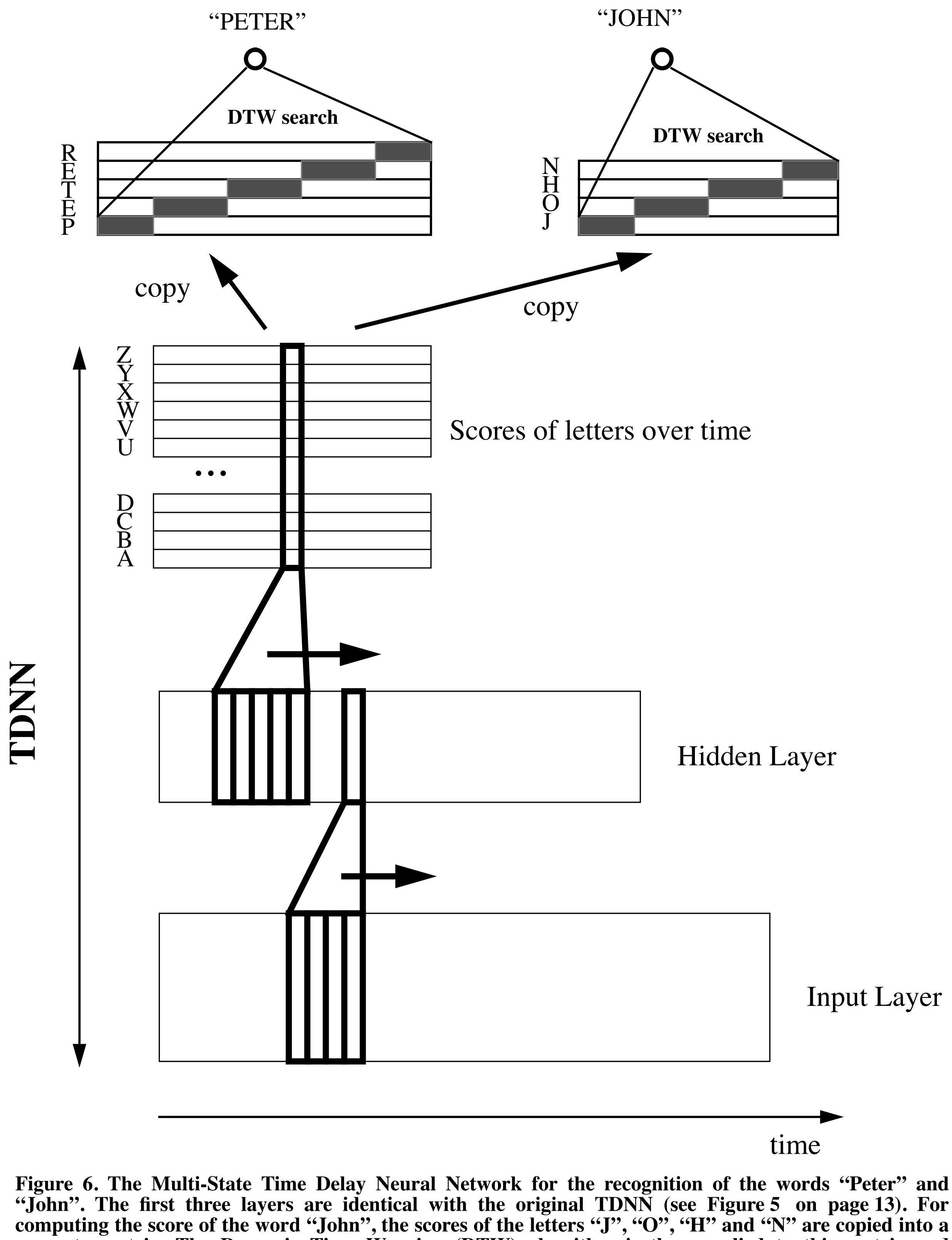

Figure 6. The Multi-State Time Delay Neural Network for the recognition of the words "Peter" and "John". The first three layers are identical with the original TDNN (see Figure 5 on page 13). For computing the score of the word "John", the scores of the letters "J", "O", "H" and "N" are copied into a separate matrix. The Dynamic Time Warping (DTW) algorithm is then applied to this matrix and computes the score for the word "John" by adding the letter-scores along the path with the highest probability.

The score of a word is found by

$$O = Max_{\{T_1, \ldots, T_{N+1}\}} \sum_{s=1}^{N} \sum_{t \geq T_s}^{t < T_{s+1}} Score_s(\vec{Y}(t))$$

The DTW algorithm finds the optimal alignment $\{T_1, \ldots, T_{N+1}\}$ which maximizes this word score. The $Score_s(\vec{Y}(t))$ can be computed using several functions like

$$Score_s(\vec{Y}(t)) = \log(Y_{i(s)}(t))$$

or

$$Score_s(\vec{Y}(t)) = weight_i \cdot Y_{i(s)} + Bias_i$$

Further details about the training procedure of the MS-TDNN can be found in [Tebelskis, 1993], [Hild, 1993a], and [Hild, 1993b]. The MS-TDNN has been successfully applied to a variety of tasks so far:

- Speaker dependent recognition of connected English letters [Haffner, 1991a], [Haffner, 1992a], [Haffner, 1992b], [Hild, 1993a], [Hild, 1993b].
- Speaker independent recognition of connected English letters [Haffner, 1992a], [Haffner, 1992b], [Hild, 1993a], [Hild, 1993b]
- Speaker independent recognition of telephone-quality French digits [Haffner, 1991b].
- Word spotting [Zeppenfeld, 1992], [Zeppenfeld, 1993].
- On-line handwriting recognition [Bodenhausen, 1993a], [Bodenhausen, 1993b], [Bodenhausen, 1993c], [Manke, 1994].

A recent modification of the original MS-TDNN incorporates the Forward-Backward training procedure [Haffner, 1993a], [Haffner, 1993b].

2.2 Speech Recognition

Despite considerable work over the last three decades, speech recognition is still considered an unsolved problem. The reason is that it is currently not possible to build a SR system which can be called "general purpose" in the sense that it handles

- continuous and/or spontaneous speech
- a very large vocabulary
- speaker independence
- acoustic ambiguity

- environmental noise.

Many sophisticated techniques for speech recognition have been proposed. Most state-of-the-art systems are based on Hidden Markov Models (HMMs), Neural Networks or hybrids of both. In large vocabulary systems, the recognition of words is generally decomposed into the recognition of small subunits of words like phonemes or phones. The simplest is the use of context-independent phone models. These models are not sensitive to the phonetic context in which a phone occurs. More advanced systems use context-dependent phone models. Early work on this concept is reported by researchers at IBM [Bahl, 1980] and BBN [Schwartz, 1984]. Triphone models condition a model on both the left and right context of a phone. A simple calculation shows that for 50 phones there are $50^3 = 125,000$ possible triphones. Such a high number of triphone models requires an immense amount of training data. Clustering can be used to group similar triphones together to so-called generalized triphones [Lee, 1990].

Compared to the great amount of research on speech recognition in general, the number of publications on automatic architectural optimization is rather limited. This is not surprising because it was necessary to demonstrate the feasibility of powerful recognizers before questions like automatic architectural optimization could be approached. The Successive State Splitting (SSS) algorithm was the first algorithm which optimizes the number of states of a context dependent Hidden Markov Model (HMM) automatically:

2.2.1 The Successive State Splitting Algorithm

The Successive State Splitting (SSS) algorithm was developed by Takami and Sagayama [Takami, 1992] to generate an efficient network of context dependent Hidden Markov Models (HMMs) automatically. The algorithm works as follows:

1. Training of an initial model: The algorithm starts with the training of an initial model per phone which consists of one state having a diagonal-covariance 2-mixture Gaussian density distribution.

2. Calculation of the distribution size: For every state existing at this time, the size of the distribution d_i is calculated for each state S_i by

$$d_i = \sum_k^K \frac{\sigma_{ik}^2}{\sigma_{Tk}^2} n_i$$

   where n_i is the number of training samples for state i, K is the parameter dimension, σ_{Tk}^2 is the variance coefficients of all samples and σ_{ik}^2 is a variance coefficient of state i (see [Takami, 1992] for details). The state with the largest distribution size, S_m, will be split in the next step.

3. Split of the state: The state S_m is split into two states S'_m and S_M, each of which has a single Gaussian density distribution corresponding to one of the original two Gaussian density distributions. At this time, there are two possibilities on the split domain. One is a split in the contextual domain and the other is a split in the temporal domain. The maximum likelihood P_t obtained by a split on the temporal domain and the maximum likelihood P_c obtained by a split on the contextual domain are computed for both possible splits (see [Takami, 1992] for details). The split with the higher likelihood for all samples is chosen.

4. Retraining of the model: At this time, each of S'_m and S_M has still a single Gaussian density distribution. The model is retrained in order to form a 2-mixture Gaussian density distribution. The steps from 2) to 4) are repeated until a prescribed total number of states is reached.
5. Change of the distributions: The complete HMM is retrained in order to change each distribution to a voluntary on, e. g. a single Gaussian density distribution.

The SSS algorithm was initially tested with the recognition of 6 Japanese consonants (/b/, /d/, /g/, /m/, /n/ and /N/) and performed slightly better than standard mixture Gaussian density distribution HMMs with varying number of mixtures.

2.3 On-Line Handwriting Recognition

Pen interfaces are an interesting input device for computers. The handwriting is recorded on a touch-sensitive tablet and processed with an on-line handwriting recognizer (OLHR). The ideal OLHR should be able to handle

- single digits/characters as well as cursive handwritten words
- cursive handwritten words with missing characters
- writer independence
- varying writing styles.

Research on on-line cursive script recognition goes back to the early 1960's. Good reviews on the different approaches can be found in [Lindgren, 1965], [Berthod, 1990], [Tappert, 1990] and [Lecolinet, 1993].

Preliminary writer dependent results with a neural network/ Hidden Markov Model (HMM) hybrid have been published by Flann and Shekhar [Flann, 1993]. The data set consists of 1000 words from a 250 word lexicon written by three writers. The reported word accuracies vary between 86.7% and 94.8%.

Writer-dependent results on a 20,000 word vocabulary have been obtained with a Multi-State Time-Delay Neural Network (Section 2.1.2.3 on page 14) [Manke, 1994]. The results vary from 97.7% word accuracy for a 400 word vocabulary to 83.0% for a 20,000 word vocabulary. Writer-independent results have been reported by Schenkel et al. using a Time-Delay Neural Network/ HMM hybrid [Schenkel, 1993]. Results vary from 85% for a 500 word vocabulary to 77% for a 25,000 word vocabulary using first-best search in both cases. A second-best search improved the results to 87% and 80%, respectively, but required considerable additional processing time.

These results show that word recognition accuracy is reduced considerably with both increasing vocabulary size and writer (user) independence. Customized systems can still outperform the general purpose systems on a limited domain. An additional comparison of word recognition results over the last 30 years shows that most reported systems recognize 80 - 90% of the words correctly. The real progress in all those years has not been made in recognition rate but rather in the degree of distortion of the writing the system can deal with, the size of the vocabulary and the degree of writer independence. This leads to the following two conclusions:

- Progress towards the ultimate general purpose system (very large vocabulary, writer and writing style independent) has actually been made. More work is necessary for further improvement of the systems.
- Very high word accuracies on limited domains are possible now, or have already been possible for some time. The reason for the limited use (or even non-existing use) of customized, small domain on-line handwriting recognition is due to the fact that the optimization of the system was/is not efficient because no fully automated methods are available.

It is the goal of this thesis to investigate and develop methods that can automate the optimization that is required for best possible performance on small, customized domains.

2.4 Architectural Selection/Optimization of Neural Networks

Several approaches have been proposed for the automatic selection/optimization of neural network architectures:

- Architectural selection techniques that select the best architecture from a set of architectures based on the performances on an independent data set.
- Stopped training
- Regularization techniques that avoid overfitting by limiting the computational power of fixed architectures.
- Pruning algorithms that start with a reasonably large architecture and remove resources during training.
- The Bayesian framework for architecture selection.
- Constructive algorithms that start with a minimal architecture and increase the resources of the network until a good/best architecture for the task is found.
- Optimization of neural network architectures by Genetic algorithms.
- Boosting of neural network performance.

2.4.1 Architectural Selection

Architectural selection deals with the following question: Given a set of observations that are learned by a set of independently trained architectures, how can the architecture be identified that performs best on new observations, i.e the architecture with the best generalization performance?

Let D be a set of observations with

$$D = \{ (\vec{x}_j, t_j), j = 1 \dots N \}$$

D is assumed to be generated as

$$t_j = \mu(x_j) + \varepsilon_j$$

where $\mu(x)$ is an unknown function, the inputs are drawn independently with an unknown stationary probability density function $p(x)$, the ε_j are independent random variables with zero mean and variance σ_ε^2, and t_j are the observed target values.

The learning problem is to find an estimate $\hat{\mu}_\lambda(x;D)$ of $\mu(x)$ given the data set D from a class of models $\mu_\lambda(x)$ indexed by λ. The basic principle of architectural selection is based on the evaluation of the prediction risk $P(\lambda)$ on data that was not used to train the system:

$$P(\lambda) = \int (dxp(x)\,[\mu(x) - \hat{\mu}(x)]^2 + \sigma_\varepsilon^2)$$

The number of testing examples is usually limited so that the prediction risk has to be approximated by a finite data set:

$$P(\lambda) \approx E\{\frac{1}{N}\sum_{j=1}^{N}(t_j - \hat{\mu}_\lambda(x_j))^2\}$$

The basic principle of architectural selection is to search for the architecture with the lowest prediction risk among a set of possible architectures. The simplest possibility is to separate the original training data into training data and validation data. The validation set can then be used to calculate an estimate of the prediction risk. This method has the disadvantage that the number of training examples that can be used for training is reduced which is critical if the original training set is already very small. Utans and Moody [Utans, 1991] discuss several approaches (Cross Validation [Mosteller, 1968], Generalized Cross Validation [Wahba, 1990] and others) which avoid this problem and evaluate them on a corporate bond rating task.

2.4.2 Stopped Training

In real-world applications with large neural networks (with thousands of connections) it is often not practical to train a set of architectures and select the best architecture according to the prediction risk. In these cases the training process (i.e. the search for the best parameters within a particular architecture) of a single architecture can be stopped when the best performance on an independent validation set is reached. This method is usually called “stopped training” or “non-convergent” because training is not completed. Various theoretical results ([Stone, 1977], [Baldi, 1991], [Finnoff,1991a] and [Finnoff, 1991b]) and empirical results ([Hergert, 1992], [Finnoff, 1992], [Weigend, 1990b], [Morgan, 1990] and [Renals, 1993b]) have provided strong evidence for the efficiency of stopped training. A simple explanation why early stopped training improves gerneralization was given by Renals in a talk at Eurospeech 93: The networks are usually initialized with small random weights. Thus early stopping has an effect similar to regularization methods which avoid learning of large weights (See “Regularization Techniques” on page 21.).

2.4.3 Regularization Techniques

Regularization techniques like weight-decay or weight-elimination [Rumelhart, 1988], [Weigend, 1990], [Chauvin, 1989], [Krogh, 1992] are another method to optimize neural network architectures automatically. These methods add some complexity measure to the cost function that is optimized by the learning rule. The original weight elimination proposed by Rumelhart [Rumelhart, 1988] uses the following error measure:

$$E = \sum_k (t_k - o_k)^2 + \lambda \sum_{i,j} \frac{w_{ij}^2}{1 + w_{ij}^2}$$

where t_k is the target output of unit k, o_k is the actual output of unit k, λ is the regularization parameter and w_{ij} is a weight in the network. This method can reduce the risk of overfitting the data with too many trainable parameters in the network [Weigend, 1990], but the decay parameter λ has to be well adjusted for optimal results. If λ is too small the weight decay will not be as effective as necessary. If λ is a bit too large the learning will be slowed down significantly. If λ is much too large, the network will not learn the training examples [Bodenhausen, 1992].

A more complicated penalty term was proposed by Nowlan and Hinton [Nowlan, 1992]. They proposed a penalty term in which the distribution of weight values is modeled as a mixture of multiple Gaussians. A set of weights is simple if the weights have high probability density under the mixture model. This is achieved by clustering the weights into subsets with the weights in each cluster having very similar values. Since the appropriate means or variances of the clusters are not known in advance, the parameters of the mixture model are adapted at the same time as the network learns the data. Simulations on a toy problem and a time-series prediction task provide evidence that the use of a more sophisticated model for the distribution of weights in a network can lead to getter generalization performance than a simpler form of weight decay [Nowlan, 1992].

2.4.4 Pruning Algorithms

Pruning algorithms start with a reasonable number of parameters (weights, units) and remove some of the resources during the training phase. The simplest method removes a certain number of weights with the smallest magnitude [Hertz, 1991]. This simple method is very easy to implement and can improve generalization on selected tasks [Svarer, 1993], but it often leads to the elimination of the wrong weights [Hassibi, 1993]. More elaborate pruning algorithms are described in the next sections:

2.4.4.1 Optimal Brain Damage

Optimal Brain Damage (OBD) [LeCun, 1990] is an algorithm that selectively removes unimportant weights from a network. The basic idea is to use second-derivative information to trade-off network complexity and training set error. The goal is to find a set of weights w_{ij} whose deletion will cause the least increase of E, the output error of the network. These weights have a low “saliency” and are removed to increase the generalization performance. OBD works as follows [LeCun, 1990]:

1. Choose a reasonable network architecture.

2. Train the network until a reasonable solution is obtained.

3. Compute the second derivatives h_{kk} for each parameter:

$$h_{kk} = \sum_{(i,j) \in V_k} \frac{\partial^2 E}{\partial w_{ij}^2}$$

where V_k is the set of index pairs of all the weights that are controlled by the parameter u_k.

4. Compute the saliencies s_k for each parameter:

$$s_k = h_{kk} \frac{u_k^2}{2}$$

5. Sort the parameters by saliency and delete some low-saliency parameters.

6. Iterate to step 2.

OBD uses only the diagonal from the Hessian matrix H with the elements h_{ij}:

$$h_{ij} = \frac{\partial^2 E}{\partial u_i \partial u_j}$$

Using all elements of the Hessian matrix can be computationally very demanding for large networks as they are used for real-world application. Typical networks for on-line handrwriting recognition or speech recognition have more than $n = 5000$ connections. Computing all second derivatives would require the computation of n^2 second derivatives. Although OBD does only require the diagonal elements, it is still necessary to implement routines for the computation of second derivatives.

2.4.4.2 Optimal Brain Surgeon

Optimal Brain Surgeon (OBS) [Hassibi, 1993a] has been proposed recently and uses all second derivatives. One advantage over OBD is that theoretically no retraining is required after weights have been removed from the network because the required adjustment of the remaining weights can be computed from the Hessian matrix. However, using OBS for large networks can be computationally prohibitive for large networks. The authors suggest that large networks are split into smaller subnetworks and that the full Hessian matrices are only computed for these subnetworks [Hassibi, 1993 b]. This method seems plausible, but the advantage of OBS over OBD (no retraining after the removal of weights) is lost.

Although OBD and OBS can improve reasonable architecures considerably [LeCun, 1990], it is not clear whether a non-reasonable architecture can be optimized to state-of-the art performance. Additionally, routines for computing the second derivatives have to be implemented and the final computational cost may be beyond an acceptable limit, especially for OBS. Another disadvantage of both OBD and OBS is the inability to recover when an important connection was accidentally removed, as no connections can be added back in.

Ramachandran and Pratt proposed a "sceletonization" method for neural networks that is based on the Information Measure (IM) of hidden units [Ramachandran, 1992]. "Superfluous" hidden units are removed if their decision hyperplane does not contribute to the separation of the training data. This contribution is measured by the Information Measure that is described in [Quinlan, 1986]. This approach sounds very reasonable, but it does not avoid the problem of overfitting to the training data which is more due to the fact that noise is learned by the network. The sceletonization algorithm does only remove hidden units which are not needed to learn the training patterns plus the noise in the training patterns. The generalization performances confirm this: The sceletonized networks performed only as well as networks with a generous number of hidden units [Ramachandran, 1992].

2.4.5 The Bayesian Framework for Architecture Selection

The Bayesian school of statistics can both be used for interpretation of classifier outputs as well as for neural network architecture selection. The following sections give a short introduction to both areas:

2.4.5.1 Pattern Classification and Bayesian Probabilities

In many pattern classification problems, the task is to assign one of M classes $\{C_i; i = 1 \ldots M\}$ to the input vectors X with the elements $\{x_i; i = 1 \ldots D\}$. Classes might represent different phonemes for speech recognition or different letters for handwritten characters. Input values might be continuous or binary. Minimum-error Bayesian classifiers perform this task by calculating the Bayesian probability $p(C_i|X)$ for each class, and assigning the input to the class with the highest Bayesian probability. The Bayesian probability $p(C_i|X)$ represents the conditional probability of class C_i given the input X. The Bayesian rule allows it to be expressed as follows:

$$p(C_i|X) = \frac{p(X|C_i)\,p(C_i)}{p(X)}$$

where $p(X|C_i)$ is the likelihood or conditional probability of producing the input vector X if the class is C_i, $p(C_i)$ is the *a priori* probability of class C_i, and $p(X)$ is the unconditional probability of the input. Conventional Bayesian classifiers estimate the Bayesian probability for each class by separately estimating the factors in the above equation. The likelihoods $p(X|C_i)$ are estimated by assuming that they can be well-modeled by specific parameter distributions, such as Gaussian or Gaussian mixture distributions. Training involves estimating the parameters of the assumed likelihood distributions and estimating the a priori class probabilities $p(C_i)$ from the training data. The unconditional probability $p(X)$ is common to all classes, it is usually omitted.

2.4.5.2 Neural Network Classifiers as Bayesian *a posteriori* Probability Estimators

Neural network outputs can estimate Bayesian *a posterioi* probabilities in a more direct way than conventional Bayesian classifiers. The relationship between minimizing a squared-error cost function and estimating Bayesian probabilities was established for the two-class case already in 1973 [Duda, 1973]. Many recent papers have provided new derivations for the two-class and multiclass case for either the squared error cost function or the cross entropy cost function [Bourlard, 1989], [Gish, 1990], [Hampshire, 1990], [Richard, 1991], [Ruck, 1990], [Shoemaker, 1991], [Wan, 1990], and [White, 1989].

2.4.5.3 Bayesian Back-Propagation

A new approach to connectionsit model comparison based on a Bayesian framework was introduced by MacKay [MacKay, 1991], [MacKay, 1992a-d], [Buntine, 1991] and [Wolpert, 1993]. The Bayesian analysis computes the evidence of a model given the training data D. The evidence of a certain architecture α can be written as follows:

$$p(\alpha|D) = \frac{p(D|\alpha)\,p(\alpha)}{p(D)}$$

The prior belief $p(\alpha)$ is assumed to be equal for each architecture, so the architectures can be compared based on $p(D|\alpha)$. This can be computed by integrating over all tunable parameters w of the model:

$$p(D|\alpha) = \int p(D|w,\alpha)\,p(w|\alpha)\,dw$$

Empirically, the correlation between a high Bayesian evidence of an architecture and a high test performance is very high, so the evidence can be used as a selection criterion for architectures. It is currently unclear whether it is possible to proof this correlation [Thodberg, 1993]. However, the evidence of an architecture can be computed based on the training set and no validation set is necessary. The Bayesian framework can also be used for the adjustment of the weight decay parameter or pruning [Thodberg, 1993]. The adjustment of the weight decay parameter is made during training and the tedious and computer-intensive search for weight decay parameters can be avoided.

The theory behind Bayesian Back-Propagation can be found in [MacKay, 1991] or [MacKay, 1992a-d]. Although MacKay's aim was to develop a method for real-world applications, he did only study learning problems with one or two input units. A more real-world oriented recipe for Bayesian Back-Propagation was presented by Thodberg [Thodberg, 1993]. Thodberg simplified the framework and made it suitable for neural networks with 8 - 10 input units and up to approximately 1,000 weights. The results presented on the prediction of the fat content of meat based on spectroscopic data are very convincing, but the framework is still not applicable for (even small) speech recognition or on-line handwriting recognition problems. The following arguments show why the current Bayesian framework is not suitable for these problems right now:

- The Bayesian framework itself is computationally demanding and requires a considerable additional programming effort because the Hessian matrix of all parameters (and their inverse) has to be computed. In addition, the evidences have to be computed for a large number of alternative architectures before the best architecture can be selected.
- The correlation between a high evidence of an architecture and high test performance has not been proven yet. Thodberg recommends to check this relation in all applications with a separate validation set. When the above correlation is validated on the particular task, he suggests to merge both training and validation data and retrain the system and select the most evident architecture. This method can be extremely tedious since training times for real-world applications can be in the order of weeks.

The Bayesian framework for architecture selection, weight decay adjustment and pruning seems to be a very accurate tool for experts working on very small domains with few parameters. However, it is not usable in the current form for speech recognition and on-line handwriting recognition.

2.4.6 Constructive Algorithms

Constructive algorithms start with a small network and increase the resources during the training phase. For example, Cascade-Correlation [Fahlman, 1990 a], [Fahlman, 1990 b] starts with no hidden units and tries to solve the classification problem without them. If this fails, additional hidden units are added one after the other.

2.4.6.1 The Cascade-Correlation Learning Architecture

The Cascade Correlation learning architecture [Fahlman, 1990 a], [Fahlman, 1990 b] is one of the best known constructive neural network architectures. It combines two new ideas: 1. A cascaded architecture in which hidden units are added to the network one after the other and do not change after they have been added. 2. A learning algorithm which installs hidden units automatically depending on the training database.

Cascade-Correlation begins the training phase without hidden units and trains the direct connections from the input to the output until either the training is completed or no further improvement is possible with the current architecture. Hidden units are added one at a time if the performance is not acceptable. Each new hidden unit receives a connection from each of the network's original inputs and also from each pre-existing hidden unit. This leads to the construction of 'deep' neural network architectures, meaning neural networks with a high number of hidden layers and powerful high-order feature detectors.

If it is necessary to add a hidden unit to the network, a set of candidate units is created. All candidate units receive a trainable input from all input units and all pre-existing hidden units. The outputs of the candidate units are not yet connected to the output units. The trainable input weights of the candidate units are adjusted to maximize S, the sum over all units o of the magnitude of the correlation[1] between V, the candidate unit's value, and E_0, the residual error observed at unit o. S is defined as

$$S = \sum_o \left| \sum_p (V_p - \bar{V})(E_{p,o} - \bar{E}_o) \right|$$

where o is the network output at which the error is measured and p is the index of the training pattern. $\bar{V}$ and $\bar{E}_o$ are the values of V and E_o averaged over all patterns. The input weights of the candidate units are trained using $\partial S / \partial w_i$, the partial derivative of S with respect to each of the candidate unit's incoming weights w_i. All weights of all candidate units are trained to maximize S using a gradient ascent method. When S stops improving, the best candidate unit is installed in the network and its input weights are frozen. The connections from the candidate unit to the output units are then trained by the gradient descent method that was used to train the direct connections from the input units to the output units (for example the standard delta rule or the Quickprop algorithm [Fahlman, 1988]).

Cascade-Correlation was tested with a large number of tasks, including difficult artificial tasks such as N-Input Parity and the "Two-Spirals-Problem" [Fahlman, 1990 a], [Fahlman, 1990 b] and classification tasks (for example [Yang, 1991]). Cascade-Correlation was also successfully applied in a speech recognition system [Sorensen, 1992]. In most reported applications Cascade-Correlation networks performed at least equally well or much better than manually tuned Back-Propagation networks. The advantages of Cascade-Correlation can be summarized as follows:

- There is no need to tune the number of hidden units and/or the number of hidden layers manually. Potential extensions include the automatic use of different activation functions for the hidden units which could increase the computational power of the network.
- Cascade-Correlation learns fast compared to standard Back-Propagation. Hidden units are assigned for special tasks and only one layer of connections is trained at any given time, avoiding back-propagation of errors through several layers of connections.
- Cascade-Correlation can build deep networks, allowing high order feature-detectors without the dramatic slowdown of learning due to back-propagation of the errors through several layers.
- Cascade-Correlation can build networks with a mixture of activation functions of the hidden units (sigmoid or Radial Basis Functions, see Section 2.4.6.3 on page 28).

2.4.6.2 Meiosis Networks

Hanson proposed a completely different approach for the allocation of hidden units in a multilayer perceptron [Hanson, 1990b]: Each connection in the network has two parameters, the mean of the weight $\mu_{w_{ij}}$ and the standard deviation of the weight $\sigma_{w_{ij}}$. Each time the weight is used in a forward pass, the actual weight w_{ij} is computed by

$$w_{ij} = \mu_{w_{ij}} + \sigma_{w_{ij}} \Phi(w_{ij}, 0, 1)$$

1. The shown version of S is a covariance because the formula leaves out some normalization terms. Early versions of the system used a true correlation, but the shown version of S works better in most situations.

where $\Phi(w_{ij}, 0, 1)$ is a random variate with mean zero and standard deviation one. The learning rules change the mean of the weights as well as the standard deviation as a function of the error signal ("stochastic delta rule, [Hanson, 1990a]. The idea is to model the uncertainty of a weight such that a high uncertainty of a connection is represented by a high standard deviation. The number of hidden units is initialized at one. The splitting policy is fixed for all problems to occur when both the standard deviation relative to the mean (the *composite variance* of the connections) for the ingoing and the outgoing connections of a hidden unit exceeds 100%, that is when both:

$$\frac{\sum_i \sigma_{ij}}{\sum_i \mu_{ij}} > 1.0$$

and

$$\frac{\sum_k \sigma_{jk}}{\sum_k \mu_{jk}} > 1.0$$

Meiosis then proceeds as follows:

- A forward stochastic pass is made, producing an output
- The output is compared to the target producing errors which are then used to update the mean and the variance of the weight
- The composite input and output variance and means are computed for each hidden unit
- For those hidden units whose composite variances are larger than 1.0 node splitting occurs. Half of the variance is assigned to each new node with a jittered mean centered on the old mean.

There is no explicit stopping criterion. The network stops creating nodes based on the prediction error.

Although the Meiosis network performed very well on some initial tests (XOR, 3Bit parity and blood NMR data), my own experiments with this algorithm on the classification of the phonemes /b/, /d/, and /g/ show that:

- The initial training phase can be very unstable due to the fact that
  - training starts with one hidden unit only and
  - there are no direct connections from the input to the output like in the Cascade Correlation algorithm that could enable the network to learn some of the training patterns with an insufficient number of hidden units

  With just one hidden unit, training progress on many problems is not possible. This means that all of the first epochs are just used for the allocation of hidden units and no useful features are extracted from the training data.

- The quality of the Gaussian noise is very critical for a stable learning phase, especially if the total number of connections is very small (for example at the beginning of the resource allocation process). If the network grows and many connections are allocated, the computational expense of a good Gaussian noise generator can become significant since a random number has to be generated for each connection at each presentation of a training example.

2.4.6.3 Constructive Neural Networks using Radial Basis Functions

Neural Networks using radial basis function (RBF) units instead of sigmoid hidden units have been proposed by Moody and Darken ([Moody, 1988]) and others. RBF outputs estimate minimum-error Bayesian *a posteriori* probabilities [Richard, 1991]. These classifiers have the advantage of short training times and high classification accuracy if the problem is well suited for modelling by Gaussian functions in feature space. However, there are two important drawbacks: First, the number of (Gaussian) hidden units has to be chosen a priori as in standard Back-Propagation networks. The second problem stems from the fact that usually the k-means algorithm is used to position the Gaussians in input vector space. K-means locates the Gaussians at those locations where many input vectors can be found which may not be optimal. Constructive versions of this idea have been proposed by [Reilly, 1982], [Moody, 1989], [Chang, 1993], [Fritzke, 1993] and others that solve either the first or both problems. Two approaches that solve both problems are described in the next sections:

The Boundary Hunting Radial Basis Function (BH-RBF) Classifier recently developed by Chang and Lippmann [Chang, 1993] allocates the centers of the radial basis functions constructively near class boundaries. The algorithm creates complex decision regions only in regions were confusions occur and the corresponding outputs are similar. A predicted square error measure is used to determine how many centers to add and to determine when to stop adding centers.

The basic idea of the algorithm is as follows: Classification with RBFs requires selecting the output which is highest for each input pattern. In regions where one class dominates, the a posteriori probability for that class will be uniformly high (near 1.0). In these regions it is not necessary to model the variation of the a posteriori probability in detail. The accurate Bayesian *a posteriory* probability is only necessary at the boundary between different classes. The BH-RBF algorithm allocates RBF units to do accurate Bayesian *a posteriori* estimation only in regions where it is required. Overfitting by adding too many centers at a time is avoided by using the predicted squared error (PSE) as the criterion for choosing new centers [Barron, 1984]:

$$PSE = RMS + \frac{C\sigma^2}{n}$$

where RMS is the root mean squared error on the training set, σ^2 estimates the variance of the error, C is the total number of centers in the RBF classifier, and n is the total number of patterns in the training set. The error variance σ^2 is selected empirically using an independent validation set. Different values of σ^2 are tried and the value which leads to the best validation performance is used. In evaluations with an artificial task and a seismic database with seven classes and 14 input features the BH-RBF performed slightly better than conventional RBF, Gaussian mixture, or MLP classifiers [Chang, 1993].

Fritzke’s approach is very similar to the previous one. A resource variable τ_s is computed for all units with the index s.The resource variable is changed after each presentation of a training example by computing

$$\tau_s = \|t - o\|^2$$

where t is the target output vector and o is the actual output vector. At the end of a training epoch the RBF unit q with the maximum resource value is determined. The center of the new RBF unit r is placed between the center of the unit q and the direct neighbor f with maximum distance in input vector space. The resource variables τ_c and output weight vectors w_c of all direct neighbors of RBF unit r are redistributed according to

$$\Delta\tau_c = \frac{\left|F_c^{(new)}\right| - \left|F_c^{(old)}\right|}{\left|F_c^{(old)}\right|}\tau_c$$

$$\Delta w_c = \frac{\left|F_c^{(new)}\right| - \left|F_c^{(old)}\right|}{\left|F_c^{(old)}\right|}w_r$$

where F_c is the Voronoi region of unit c, the region in input vector space that have the same nearest reference vector. $|F_c|$ is the n-dimensional volume of F_c. The resource variable τ_r and the output weight vector w_r of the new RBF unit are initialized as

$$\tau_r = -\sum_{c \in N_r} \Delta\tau_c$$

$$w_r = -\sum_{c \in N_r} \Delta w_c$$

where N_r are all neighbors units of unit r. Fritzke’s constructive algorithm is also formulated for self-supervised neural networks. The supervised version was tested with the “Two-Spiral Problem” and a vowel recognition task [Robinson, 1989] and performed very well.

2.4.6.4 Constructive Time-Delay Radial Basis Function Networks

A constructive Time-Delay Radial Basis Function (TDRBF) network was recently developed and tested on a small phoneme recognition task [Berthold, 1993]. The approach combines the idea of the sliding input window from the standard Time-Delay Neural Network [Waibel, 1989] and the constructive RCE training procedure [Reilly, 1982]. The RCE training procedure introduces new prototypes (i.e. RBF units) when necessary and adjusts the radii of existing prototypes when necessary. The TDRBF network was tested on the classification of the phonemes /b/, /d/ and /g/ and achieved good recognition performances (97.7% on test data) with manual adjustment of the input window sizes. These results are promising, but a final evaluation of the algorithm is not possible based on this simple task.

2.4.6.5 Constructive Recurrent Networks

The Recurrent Cascade-Correlation (RCC) architecture combines the Cascade-Correlation algorithm with a simplified version of Elman's Simple Recurrent Network (SRN) [Elman, 1988] (see chapter 2.1). The simplification was necessary to avoid a violation of the Cascade-Correlation concept. Normally, in the SRN there is total connectivity between the state variables (i. e. the previous output of the hidden units) and the hidden unit layer. The Cascade-Correlation concept is to install new hidden units and to freeze their weights once their training is completed. This would have been impossible with the total connectivity mentioned before. The solution was to reduce the full connectivity between the state variables and the hidden unit layer to recurrent self loops.

The RCC architecture was initially tested on a simple finite-state grammar that is frequently used for evaluations of recurrent architectures [Cleeremans, 1989] and performed better than the standard SRN. Additional tests with morse code were successful, too. Recently, Chen et al. showed that RCC networks with sigmoid activation functions are not capable of representing *all possible* finite state grammars [Chen, 1993]. They propose an alternative constructive recurrent network model. However, the question whether it is important to be able to learn *all possible* finite state grammars is beyond the scope of this thesis.

2.4.7 Genetic Algorithms

Hybrids of neural network learning and genetic algorithms (GA) for optimization seem an appealing way to construct artificial cognitive systems because both techniques are inspired by nature. The GA by Holland [Holland, 1975] has been explored for almost 20 years now. The basic concept is to consider a population of *individuals* that each represent a potential solution to a problem. The relative success of each individual on this problem is considered his *fitness*, and used to selectively reproduce the most fit individuals to produce similar but not identical offspring of the next generation. By iterating this process, the population efficiently samples the space of potential individuals and eventually converges on the most fit. The process starts with an initial population g_0 which is constructed randomly. Each individual is evaluated by some *environment* function that returns the fitness

$$\mu(x_i) \in \Re$$

of each individual in g_0.The evolutionary algorithm then performs two operations: First, its *selection* algorithm uses the population's N fitness measures to determine how many offspring each member of g_o contributes to g_1. Second, some set of *genetic operators* are applied to these offspring to make them different from their parents. The resulting population is now g_1. These individuals are again evaluated, and the cycle repeats itself. The iteration is terminated by some measure suggesting that the population has converged.

Within this basic concept of GA there are many options for optimization of neural network architectures. A good overview can be found in [Belew, 1990]. In general, each *individual* in the above concept is a certain neural network architecture that has to be trained and then evaluated. The computational effort of such a procedure can become quite excessive considering many real-world applications. The time-complexity of the system is the product of the number of generations the GA is run, times the size of the population of each generation times the training time taken by each individual:

$$\Gamma = Generations \times PopulationSize \times TrainingTime$$

GA can find very powerful neural network architectures [Belew, 1990], but the number of generations is typically in the order of 100 and the number of individuals is typically in the order of 50. Although the time complexity can be often reduced significantly [Belew, 1990], the total computational effort is not feasible for speech or on-line handwriting tasks where one training run can take a week or more.

An interesting GA approach to neural network optimization was proposed by Braun and Weisbrod [Braun, 1993]. Instead of training all offspring from scratch, weights from the ancestors can be re-used (by weight transmission). This can reduce the computational effort considerably. However, GA tend to generate a number of equally good solutions where one would be enough. The computational effort for each solution is tolerable, but the overall effort for the one solution that is finally needed is still very large.

2.4.8 Boosting

Boosting converts a learning machine with a finite error rate into an ensemble of learning machines with, in principle, arbitrarily low error rate. It was originally described by Schapire [Schapire, 1990] in the context of Valiant's "probably approximately correct" learning model [Valiant, 1984]. It was applied to neural networks designed for optical character recognition by Drucker et al. [Drucker, 1993a], [Drucker, 1993b]. The boosting algorithm proceeds as follows: Assume an oracle that generates an unlimited supply of independent training examples.

- First, generate a set of training examples and train a first network.
- After the first network is trained it is used in combination with the oracle to produce a second training set. Flip a fair coin.
  - If the coin is heads, pass outputs from the oracle (new training patterns) through the first network until the first network misclassifies a pattern and add this pattern to a second training set.
  - If the coin is tails, pass outputs from the oracle (new training examples) through the first learning machine until the first network finds a patterns that it classifies correctly and add it to the second training set.

  This process is repeated until enough patterns have been collected. These patterns, half of which the first network classifies correctly and half incorrectly, constitute the training set that is used to train a second network.

- The first two networks are then used to produce a third training set in the following manner: Pass the outputs from the oracle (new training examples) through the first two networks: If the networks disagree on the classification, add this pattern to the third training set. Otherwise, toss the pattern. Continue this until enough patterns are generated for the third training set. A third network is then trained with the third data set.

In the testing phase, the test patterns are passed through all three networks and the final output of the system is determined as follows:

- If the first two networks agree, that is the label.
- If the first two disagree, use the output of the third network.

The proof, that boosting works depends on the assumption that there is enough training data to generate the three data sets in the above manner. This may be quite difficult depending on the performance of the first network. For example, suppose there are 9,000 training examples and 3,000 of them are used for training of the first network and that network achieves a 5% error rate. To get 3,000 training patterns for the second network (meaning 1,500 patterns that the first network classifies correctly and 1,500 that it misclassifies), approximately 30,000 new patterns are needed to get the 1,500 patterns that the first network misclassifies (assuming an error rate of 5% for the first network). Further training data is used to train the third network. The solution to this problem is to generate additional training data by using small deformations around the finite training set based on techniques of Simard [Simard, 1992].

More details of the training procedure (including the use of a validation set for stopping training and the deformation method) can be found in [Drucker, 1993a]. The results on a database of segmented ZIP codes from the United States Postal Service (divided into 9,709 training patterns and 2,007 validation samples) are very good, although 153,000 deformed patterns were needed to generate the second training set and 195,000 deformed patterns were needed to generate the third training set.

An open question is the choice of the architecture for each successive network. The training examples for the second and third network get successively harder to train on [Drucker, 1993]. This suggests that boosting could benefit from constructive or pruning algorithms, with a great advantage for the constructive approach since training gets harder and not easier.

Boosting and architectural optimization are two techniques that seem to complement each other very well rather than compete with each other. The intention is similar to the Automatic Validation Analyzing Control System (AVACS).

2.4.9 Limitations of the Algorithms

The previous sections have shown that there is a lot of research going on in the field of architectural selection, stopped training, regularization techniques, pruning algorithms, and constructive algorithms. There are even variants of constructive algorithms for spatio-temporal tasks. The reason for the development of a new algorithm is based on empirical results from a workshop on manual optimization of neural network based speech recognition systems, organized at Vail, Colorado [Bodenhausen, 1991c] (see Appendix). The workshop and the most important results are summarized in the Appendix. The following conclusions can be extracted from the workshop for the design of an automatic optimization algorithm:

- Stopped training based on the performance on an independent validation set is a reliable and easy method to avoid overfitting.
- The number of hidden units is an important, but not the most important architectural parameter for neural network based speech recognition systems. Haffner and Franzini report that there is a certain range of number of hidden units where generalization ability is similar. This may only be true for the large training databases that they used. For small training databases the number of hidden units may be more critcal because the system has to learn a mapping of similar difficulty with less training patterns. The conclusion for an automatic optimization algorithm is that the number of hidden units should be included in the optimization process, but should not be the only architectural parameter to be modified.
- The optimization of the input windows leads to significant improvements and should be included into the automatic optimization process.
- The number of states per acoustic event and the type of acoustic event that is modeled (words or phonemes) seem to be the most important architectural parameters. Some presenters did not want to present their current state topolgy exactly, but they admitted that optimization at this level of the system is very effective. This suggests that automatic optimization of architectural parameters for speech recognition should include optimization of the state topology.
- The total number of parameters in such systems can grow very large (50.000 parameters and more). The optimization algorithm should be able to deal with that.
- Human optimizers tend to perform optimization steps sequentially, which may not be optimal. However, an intelligent human optimizer is able to recover from wrong paths, partly due to technical exchange at conferences. Obviously, automatic optimization procedures can not do this. A solution to this problem is the development of optimization algorithms that optimize synergetically.

A comparison of the conclusions from the workshop and the description of the algorithms in this section shows that all of the algorithms do not fit the requirements perfectly. *Architectural selection methods* are a very useful tool to finally select the best architectures among a set of decent candidates. *Stopped training methods* are simple and effective and should be used in conjunction with other methods anyway. But the method of how to come up with a good architecture automatically and efficiently (where the former two methods can then be applied) has still to be discussed under the criteria collected at the workshop. The following sections summarize the reasons why a new algorithm had to be developed:

2.4.9.1 Algorithms not Tailored for Spatio-Temporal Processing

- Standard *Cascade-Correlation* is stable in the initial learning phase, fast and effective. However, it does not include the optimization of relevant parameters like the number of states or the time-delays.

2.4.9.2 Algorithms not Suitable for Large Systems

- *Genetic Algorithms* are computationally too demanding.
- *Meiosis Networks* are unstable in the initial constructive phase and the computational effort for the Gaussian noise generator is considerable.
- *Optimal Brain Damage* and *Optimal Brain Surgeon* are very powerful, but the computational effort for the second derivatives and the matrix inversion is very large.
- The *Bayesian Framework* is not suitable for the same reason.

2.4.9.3 Algorithms not Suitable for Small Amounts of Training Data

- Boosting needs too much training data for small, customized systems.

2.4.9.4 Algorithms not Suitable for other Reasons

- *Recurrent networks* and their *constructive variants* were ruled out because their training takes too long.
- *Constructive Time-Delay Radial Basis Function Networks* were not around at the beginning of the thesis and have not been evaluated completely.
- *Constructive Radial Basis Function* approaches do not optimize all relevant architectural parameters.

3. Databases for Performance Evaluations

The aim of this work is the development and evaluation of connectionist resource allocation models for spatio-temporal real-world applications. The proposed algorithms were first evaluated with segmented data (two speech recognition tasks and two handwritten character recognition tasks). These tasks are small enough to allow a reasonable number of experiments but are also large enough to be relevant for an application oriented algorithm. These tasks are:

- Classification of the voiced stops /b/, /d/ and /g/ from a single japanese speaker. The phonemes were extracted from japanese words recorded in a sound-proof booth with a sampling rate of 12kHz. The speech was Hamming windowed and a 256-point FFT computed every 5 ms. 16 normalized melscale coefficients were computed from the power spectrogram by computing log energies in each melscale energy band, where adjacent coefficients in frequency overlap by one spectral sample and are smoothed by reducing the shared sample by 50%. Adjacent coefficients in time were collapsed for further data reduction resulting in an overall 10 ms frame rate. All coefficients were then normalized. The phonemes were hand-labeled and extracted from the spoken words. This task will be denoted by **SEG_SR_BDG** (segmented /b/, /d/, /g/ task) in the following.
- Recognition of the spelled English alphabet (approx. 3,000 words spoken by a single speaker (DBS) taken from the CMU-ALPH database. The coefficients are computed in the same way as for the SEG BDG classification task, except that they were automatically labeled and segmented. Speaker DBS (Dave Sanner) is known to be a particular hard speaker for SR systems. This task will be denoted **SEG_SR_ALPH_DBS** in the following.

- Recognition of the digits 0, 1, 2, ..., 9 written on a touch sensitive tablet (approx. 1,000 digits, recorded as described in [Guyon, 1991]: During writing, the position and the pressure of the pen are recorded from the tablet. Resampling is used to reduce the temporal variations of the digits. From these data points, the directions and the curvatures of the pen strokes are computed and are added to the data. This task will be denoted **SEG_OLHR_DIGIT**.
- Recognition of the capital letters A, B, ..., Z written on a touch sensitive tablet (approx. 2,500 capital letters, recorded as described above. This task will be denoted **SEG_OLHR_A_Z**.

The most promising algorithm will be tested with a speaker dependent connected letter recognition task (connected spelled letters from the English alphabet) from the CMU-ALPH database. Speaker MJMT (Joe Tebelskis) was chosen because his data is frequently used for performance comparisons. The coefficients are computed in the same way as for SEG_SR_BDG. The database consists of 1,000 spelled words per speaker. This task will be denoted **CONNECTED_SR_ALPH_MJMT**.

4. Initial Experiments: The Tempo 2 Approach

Initial experiments included the evaluation of the Tempo 2 approach, which was originally developed as a neurophysiologically motivated model of cognitive resource allocation [Bodenhausen, 1990a], [Bodenhausen, 1990b]. In this approach, a neural network is trained with a learning algorithm that adjusts time-delays and the width of input-windows automatically. The learning rules require input-windows over time that can be described by a smooth function. With these input-windows it is possible to derive learning rules for adjusting the center and the width of the window. During training, new connections are added if they are needed by splitting already existing connections and training them independently.

Adaptive time-delays in neural networks could have significant advantages for the processing of pattern-sequences, especially if the relevant information is distributed across non-consecutive patterns. A typical example for this kind of pattern sequences are rhythms (relevant in music and speech). In a rhythm, there are many events but also many gaps between these events. Another example is speech, where some parts of an utterance are more important for understanding than others (example: 'hat', 'fat', 'cat'..). Therefore a network that allocates existing and new resources to the parts of the input sequence that are most helpful for the task could be more compact and efficient for various tasks.

4.1 The Tempo 2 Learning Algorithm

The Tempo 2 learning algorithm [Bodenhausen, 1990a], [Bodenhausen, 1990b] was designed to train weights, delays and widths of input windows in a neural network. It is a generalization of the Back-Propagation network proposed by Rumelhart, Hinton and Williams [Rumelhart, 1986]. Adapting delays and widths of input-windows in a neural network is possible because a Gaussian shaped input-window over time is used. In the network, a unit j at time t is activated by input from a Gaussian shaped input-window centered around (t-d) and standard deviation σ, where d (the time-delays) and σ(the width of the input-window) are to be learned. (Other windows are possible. The function describing the shape of the window has to be smooth.) The input of unit j at time t, $x_j(t)$,

$$x_j(t) = \sum_{\tau=0}^{t} \sum_{k} y_k(\tau)\, \theta(\tau,t,d_{jk},\sigma_{jk})\, w_{jk}$$

with $\theta(\tau,t,d_{jk},\sigma_{jk})$ representing the Gaussian input window given by

$$\theta(\tau,t,d_{jk},\sigma_{jk}) = \frac{1}{\sigma_{jk}\sqrt{2\pi}} \cdot e^{-\left(\frac{(\tau - t + d_{jk})^2}{2\sigma_{jk}^2}\right)}$$

where $y_k$is the output of the previous sending unit and w_{jk}, $d_{jk}$and $\sigma_{jk}$are the weights, delays and widths on its connections, respectively.

This approach is partly motivated by neurophysiology and mathematics. In the brain, a spike that is sent by a neuron via an axon is not received as a spike by the receiving cell. Rather, the postsynaptic potential has a short rise and a long tail. Let us assume a situation with two neurons. Neuron A fires at time t-d, where d is the time that the signal needs to travel along the connection and to activate neuron B. Neuron B is activated mostly at time t, but the postsynaptic potential will decrease slowly and neuron B will get some input at time t+1, some smaller input at time t+2 and so on. Functionally, a spike is smeared over time and this provides some “local memory”.

For our simulations we simulate this behavior by allowing the receiving unit to be activated by the weighted sum of activations around an input centered at time t-d. If the sending unit (“neuron A”) was activated at time t-d, then the receiving unit (“neuron B”) will be activated mostly at time t, will be less activated at time t+1, and so on (see Figure 7 on page 39 and Figure 8 on page 39). In our case, the input-window function also allows the receiving unit to be (less) activated at times t-1, t-2 etc.. This symmetric ‘behavior’ enables us to formulate a learning rule that can increase and decrease time-delays.

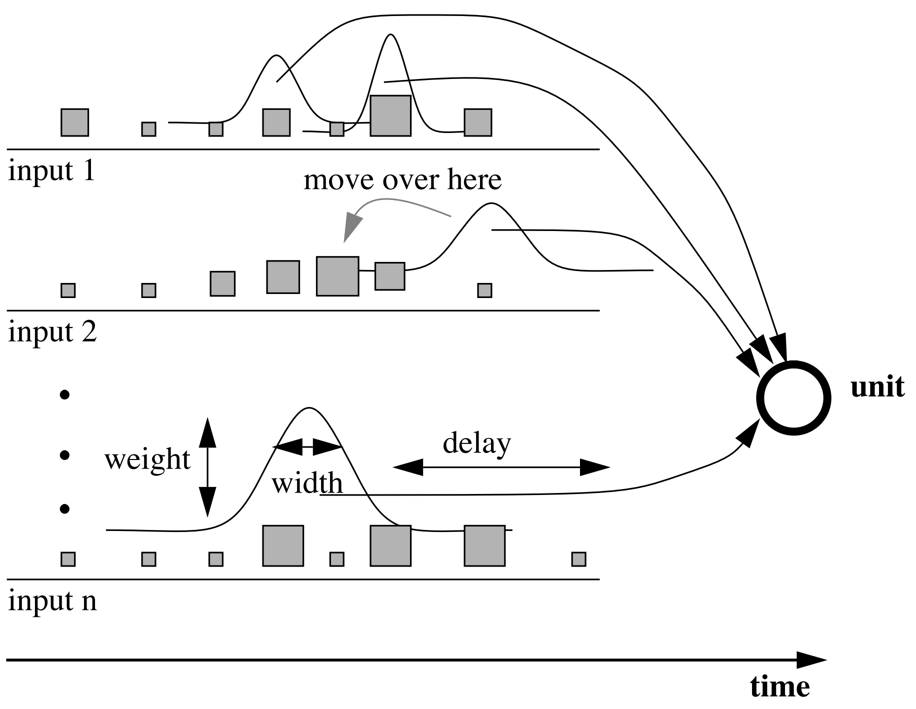

Figure 7. The input to one hidden unit in the Tempo 2 network. The boxes represent the activations of the sending units. A large box represents a high activation and a small box represents a small activation.

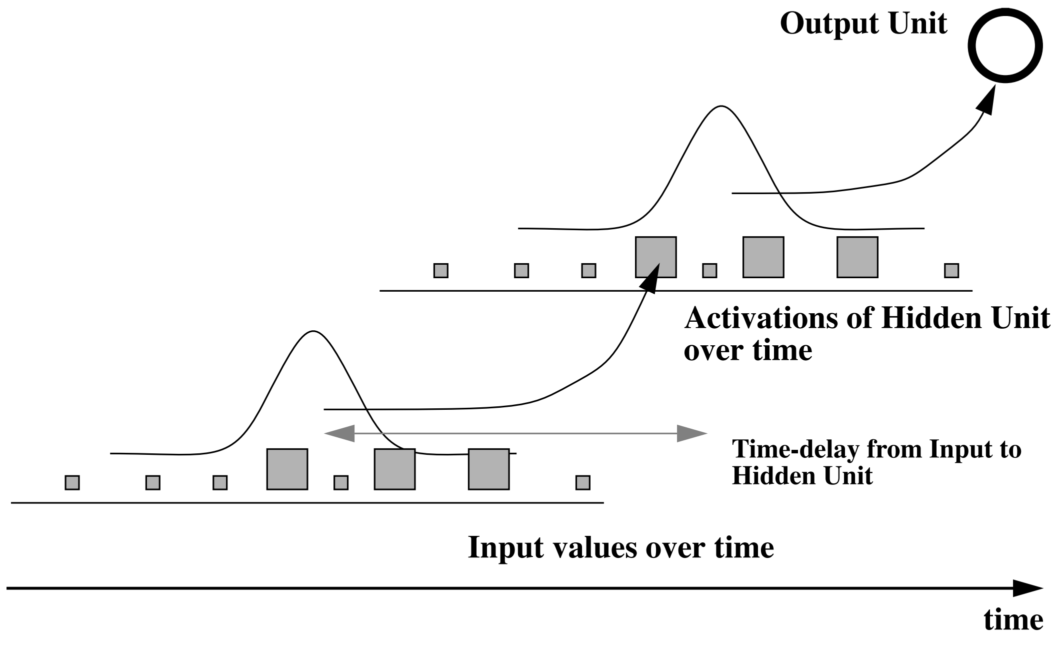
Output Unit
Activations of Hidden Unit over time
Time-delay from Input to Hidden Unit
Input values over time
time

Figure 8. The flow of activation from input over one hidden unit to one output unit. Only one Gaussian connection between the units is shown.

The Gaussian input-window has two advantages:

1.) It provides some robustness against temporally misaligned input tokens. By looking at Figure 7 on page 39 it is obvious that small misalignments of the input signal do not change the input of the receiving unit significantly. The robustness is dependent on the width of the window. Therefore a wide window would make the input of the receiving unit more robust against signals shifted in time, but would also reduce the time-resolution of the unit. This is another reason for the implementation of a learning rule that adjusts the width of the input-windows of each connection.

2.) With this Gaussian input-window over time, it is possible to compute how the input of unit j would change if the delay of a connection or the width of the input-window were changed. The formalism is the same as for the derivation of the learning rules for the weights in a standard Back-Propagation network. The change of a delay is proportional to the derivative of the output error with respect to the delay. The change of the width is proportional to the derivative of the error with respect to the width of the input-window. As in all Back-Propagation networks, the error is propagated back to the hidden layer. The learning rules for weights w_{ji}, delays d_{ji} and widths σ_{ji} were derived from

$$\Delta w_{ji} = -\varepsilon_1 \frac{\partial E}{\partial w_{ji}}$$

$$\Delta d_{ji} = -\varepsilon_2 \frac{\partial E}{\partial d_{ji}}$$

$$\Delta \sigma_{ji} = -\varepsilon_3 \frac{\partial E}{\partial \sigma_{ji}}$$

where ε_1, ε_2 and ε_3 are the learning rates and E is the error. As in the derivation of the standard Back-Propagation learning rules, the chain rule is applied (z = w, d, σ):

$$\frac{\partial E}{\partial z_{ji}} = \frac{\partial E}{\partial x_j(t)} \cdot \frac{\partial x_j(t)}{\partial z_{ji}}$$

where $\frac{\partial E}{\partial x_j(t)}$ is the same in the learning rules for weights, delays and widths. The partial derivatives of the input with respect to the parameters of the connections are computed as follows:

$$\frac{\partial}{\partial w_{ji}} x_j(t) = \sum_{\tau=0}^{t} y_i(t)\,\theta(\tau, t, d_{ji}, \sigma_{ji})$$

$$\frac{\partial}{\partial d_{ji}} x_j(t) = \sum_{\tau=0}^{t} y_i(t)\, w_{ji} \frac{\partial}{\partial d_{ji}} \theta(\tau, t, d_{ji}, \sigma_{ji})$$

$$\frac{\partial}{\partial \sigma_{ji}} x_j(t) = \sum_{\tau=0}^{t} y_i(t)\, w_{ji} \frac{\partial}{\partial \sigma_{ji}} \theta(\tau, t, d_{ji}, \sigma_{ji})$$

The concept of the learning rules for time-delays and widths is shown graphically in Figure 9 on page 41.

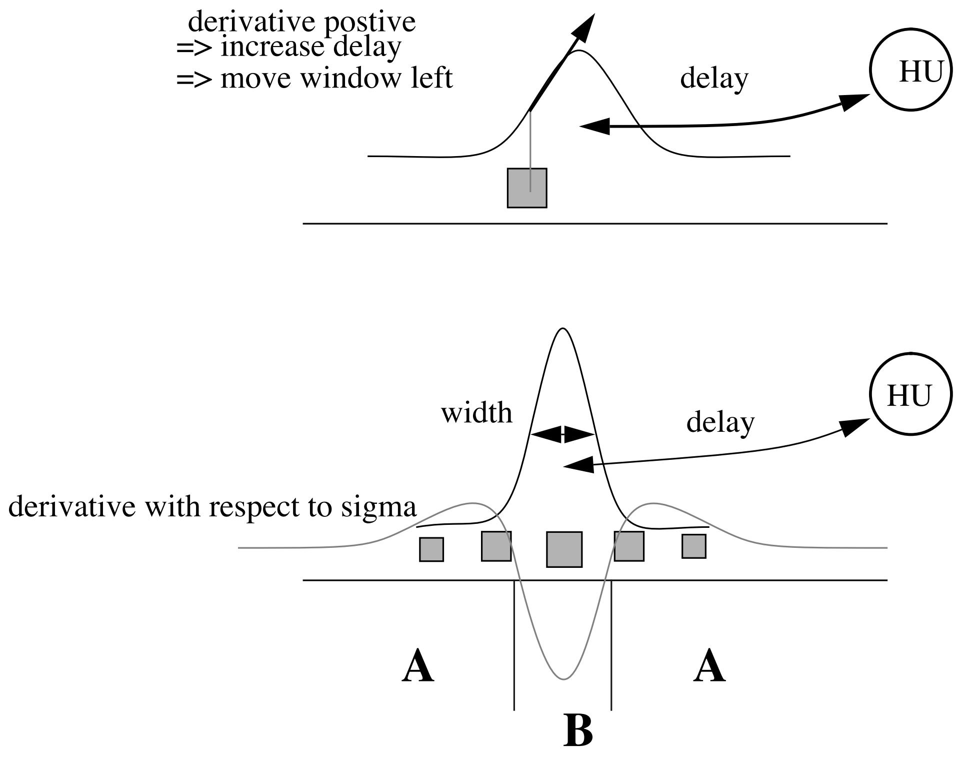

Figure 9. A graphical explanation of the learning rules for delays and widths: The derivative of the Gaussian input-window with respect to time is used for adjusting the time-delays (upper picture). The derivative with respect to the standard deviation (sigma) is used for adjusting the width of the window (dotted line, lower picture). A majority of activation in area A will cause the window to grow, a majority of activation in area B will cause the window to shrink.

4.2 Adding New Connections

Learning algorithms for neural networks that add hidden units have recently been proposed [Fahlman, 1990], [Hanson, 1990]. In the Tempo 2 network connections are added to the already existing ones in a similar way as it is used by Hanson for adding units [Hanson, 1990]. During learning, the network starts with one connection between two units. Depending on the task this may be insufficient and it would be desirable to add new connections where more connections are needed. New connections are added by splitting already existing connections and afterwards training them independently. The rule for splitting a connection is motivated by observations during training runs. It was observed that input-windows started moving backwards and forwards (that means the time-delays changed) after a certain level of performance was reached. This can be interpreted as inconsistent time-delays which might be caused by temporal variability of certain features in the samples of speech. During training we compute the standard deviations of all delay changes and compare them with a threshold: If

$$\sum_p \left(\Delta d_{ji}(p) - \frac{\sum_p |\Delta d_{ji}|}{D} \right)^2$$

is larger than a certain threshold, then connection ij is split (where Δd_{ji} is the change of delay ji computed for pattern p and D is the total number of training patterns).

4.3 Recognition Performances with the Tempo 2 Approach (Segmented Data)

The Tempo 2 network was initially tested with rhythm classification. The results were encouraging and evaluation was carried out on a phoneme classification task. In this application, adaptive delays can help to find important cues in a sample of speech. Units should not accumulate information from irrelevant parts of the phonemes. Rather, they should look at parts within the phonemes that provide the most important information for the kind of feature extraction that is needed for the classification task. The network was trained on the phonemes /b/, /d/ and /g/ from a single speaker (SEG_SR_BDG). 783 tokens were used for training and 759 tokens were used for testing.

In order to evaluate the usefulness of each adaptive parameter, the network was trained and tested with a variety of combinations of constant and adaptive parameters (see Table 1 on page 43). In all cases the network was initialized with random weights and delays and constant widths σ of the input windows. All results were obtained with 8 hidden units in one hidden layer[1]. Performances of 64% on training data and 63% on testing data are obtained if weights and widths of the connections are fixed and only delays are learned. Keeping weights and delays fixed and only learning the widths of the connections leads to recognition performances of 63.5%

1. Networks with a different number of hidden units were also tested. The best performances were obtained with 8 hidden units.

(training data) and 61.8% (testing data). These results show that the main parameters in the network are the weights. Delays and widths seem to be of importance if they are trained together with the weights.

The above results are only interesting for an evaluation of the usefulness of each parameter. Learning combinations of these parameters yields much higher recognition performances (see Table 1 on page 43): The network learns to classify 93.2% of the training samples and 89.3% of the testing samples correctly, if the delays and the widths of the connections are fixed. If the delays are learned in addition to the weights, the performance increases to 98.3% (training data) and 97.8% (testing data). A further increase of the performance can be obtained by also learning the widths of the input-windows (98.8% on training data and 98.0% on testing data).

TABLE 1. :b/, /d/ and /g/ classification performance with the Tempo 2 approach with 8 hidden units in one hidden layer. The network is initialized with random weights and constant widths (SEG_SR_BDG).

adaptive parameters	**constant parameters**	**training set performance**	**testing set performance**
weights	delays, widths	93.2%	89.3%
delays	weights, widths	64.0%	63.0%
widths	weights, delays	63.5%	61.8%
delays, widths	weights	70.0%	68.6%
weights, delays	widths	98.3%	97.8%
weights, widths	delays	98.1%	97.7%
weights, delays, widths	-	98.8%	98.0%

Additional simulations were conducted without hidden units. The results are worse than the results obtained with 8 hidden units (see Table 2 on page 43) but training was faster and easier (the choice of the learning rate was easier and no simulations got stuck in local minima). The higher performances with hidden units suggest that the nonlinear classification capabilities are important for this classification task.

TABLE 2. /b/, /d/ and /g/ classification performance with the Tempo 2 approach with no hidden units. The network is initialized with random weights, random delays and constant widths (SEG_SR_BDG).

adaptive parameters	**constant parameters**	**training set performance**	**testing set performance**
weights, delays	widths	96.8%	95.7%
weights, delays, widths	-	97.2%	96.1%

The Tempo 2 algorithm was also applied to segmented spelled alphabet recognition. 2210 letters spoken by a single speaker were used for training and 520 letters were used for testing. Networks without hidden units and with a varying number of hidden units were simulated. The results without hidden units are summarized in Table 3 on page 44.

The same database was used to train and test a network with varying numbers of hidden units. The best results were obtained with 40 hidden units. Training of such a large network is computationally very expensive. The results of a limited number of training runs are reported in Table 4 on page 44. Surprisingly, the results without hidden units are better than the results with hidden units .

TABLE 3. Alphabet recognition performance without hidden units. The network is initialized with random weights, constant delays and constant widths (SEG_SR_ALPH_DBS.

adaptive parameters	constant parameters	training set performance	testing set performance
weights	delays, widths	71.4%	65.6%
weights, delays	widths	90.3%	83.8%
weights, delays, widths	-	91.4%	84.4%

TABLE 4. Alphabet recognition performance with 40 hidden units in one hidden layer. The network is initialized with random weights, random delays and constant widths (SEG_SR_ALPH_DBS).

adaptive parameters	constant parameters	training set performance	testing set performance
weights, delays	widths	90.1%	82.7%

4.4 Discussion of the Tempo 2 Approach

In these initial experiments the advantages of an artificial neural network that can automatically learn important structural parameters by gradually changing time-delays and widths of the Gaussian input windows are explored. In this approach, the learning rules for the time-delays and the width of the windows were derived in the same way as for the learning weights. The results show that time-delays and widths of input windows in artificial neural networks can be learned automatically by a gradient descent method. The proposed learning rule is able to improve performance significantly compared to a fixed structure. This means that the Tempo 2 algorithm is very useful in situations where no knowledge about the temporal characteristics of the task is available.

How can the structural parameters of the Tempo 2 network be interpreted? The width of an input window determines how much local temporal context is captured by a single connection. Additionally, a large window means increased robustness against temporal misalignments of the input tokens. A large window also means that the connection transmits with a low temporal resolution. The learning rule for the widths of the windows has to compromise between increased robustness against misaligned tokens and decreased time-resolution. With the proposed Tempo 2 algorithm, this is successfully done by a gradient descent method.

The receptive fields of the output units before and after training are shown in Figure 11 on page 46 and Figure 12 on page 47. A comparison of both figures shows that the Tempo 2 algorithm is able to adapt the time-delays and the widths of the input windows effectively.

A comparison between the widths before and after training reveals that 70 - 80% of the windows in the network get smaller during training if the network is initialized with too large input windows (meaning a small temporal resolution). These results show that it is possible to automatically compromise between high temporal resolution and increased robustness against temporal distortions in artificial neural networks.

The adaptive time-delays can be interpreted as an adaptive depth of the internal memory. A long memory is realized by a long delay and vice versa.

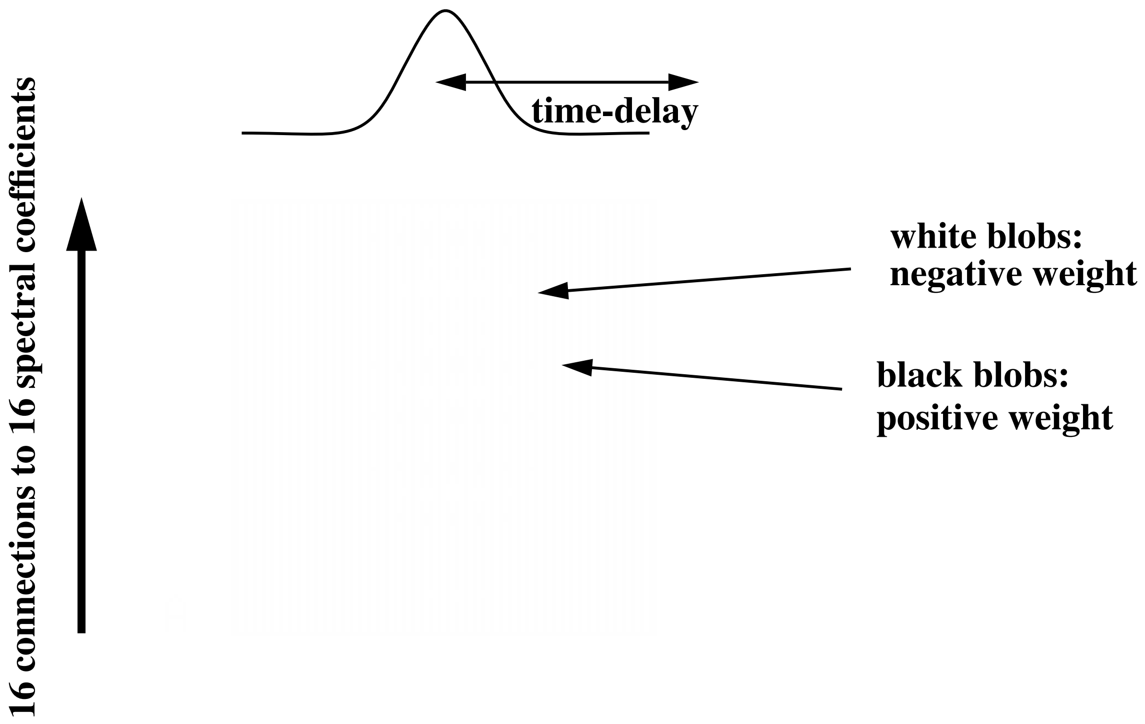
time-delay
white blobs: negative weight
black blobs: positive weight
16 connections to 16 spectral coefficients

Figure 10. Explanation for Table 11 on page 46 and Table 12 on page 47. The receptive field of the output unit representing the letter "A" of a Tempo 2 network without hidden units. Negative weights are indicated by white blobs and positive weights are indicated by black blobs. The Gaussian window over time is displayed by the set of blobs from left to right. The time-delay of a connection is indicated by the position of the set of blobs within the boxes. A center of the Gaussian on the left side of the box represents a long time-delay. A center on the right side represents a short time-delay. The picture shows the receptive field before training where the time delays are all initialized by 7.0 (corresponding to 70 ms). In each box the weights connecting to the input units which represent the lower spectral frequencies are shown at the bottom.

Figure 11. The receptive fields of the output units in the Tempo 2 network without hidden units before training. See Table 10 on page 45 for explanation

.

Figure 12. The receptive fields of the output units of a Tempo 2 network without hidden units after training (SEG_SR_ALPH_DBS). The initialized network is shown in Table 11 on page 46. See Table 10 on page 45 for explanation.

The comparison of the performances with one adaptive parameter set (either weights, delays or widths) shows that the main parameters in the network are the weights. Delays and widths seem to be of a lesser importance, but in combination with the weights the delays can improve the performance, especially generalization. A Tempo 2 network with trained delays and widths and *random* weights can classify 70% of the phonemes correctly. This suggests that learning temporal parameters is effective.

The network achieves results comparable to a similar network with a very powerful handtuned architecture (the TDNN[1]). Additionally, the network is more compact. The Tempo 2 network performs the classification task with one third of the number of connections of a standard TDNN. This suggests that the kind of learning rule could be helpful in applying time-delay neural networks to problems where no knowledge about optimal time windows is available. At higher levels of processing such adaptive networks could be used to learn rhythmic (prosodic) relationships in fluent speech and other tasks.

Although the overall results with the Tempo 2 algorithm are encouraging it should be mentioned that the flexibilty of the approach (automatically optimized temporal resolution and memory depth) can lead to some instabilities during training. In various training runs it was observed that the learning rates for weights, delays and widths can not be chosen independently of each other. This can be explained by an interaction between updates of these parametersets. For example, if the time-delay of a connection is changed the optimal weight changes, too. Several learning schedules were tried to explore interaction between the updates of the parametersets:

- Updating all parameters after a forward pass (change all mode): This is the fastest way to update the parameters in the network. However, a large learning rate for the weights requires a large learning rate for delays and widths, too (and vice versa).
- Updating one parameter after each forward pass: This mode is computationally very expensive because more forward passes in the network are needed. The performance was worse than in the change all mode.
- Updating the most important parameterset more frequently than the less important parametersets: The evaluation of this mode was carried out by changing the weights at each epoch and changing delays and widths at each third epoch. Training time was comparable to the change all mode, but the performance was a little bit worse.

4.5 Conclusions of the Tempo 2 Approach

The Tempo 2 algorithm has been shown to effectively adjust time-delays and widths of input windows. A technique for spliting connection has been proposed, too. Various training runs with the Tempo algorithm have shown that the weights, time-delays and widths of the connections can not be trained independently from each other. This effect becomes more important in networks with hidden layers and two or more layers of Tempo 2 connections. This could explain why the network without hidden units outperformed the network with various numbers of hidden units for

1. The TDNN has been shown to be a very powerful approach to phoneme recognition. The fixed time-delays and the kind of time-window were chosen partly because they were motivated by results from earlier studies [Blumstein, 1980], [Kewley-Port, 1983] and because they were successful from an engineering point of view.

the big alphabet recognition task. The huge network with hidden units suffers too much from the interaction between weights, delays and widths. The small network for the bdg task (with 8 hidden units) did not suffer from this effect. A technique that controls the interaction between weights, delays and width could be based on a computation of derivatives like

$$\frac{\partial^2 E}{\partial w_{ji} \partial d_{ji}}$$

Adding these terms to the learning rules could account for the interactions between the updates of the parametersets. However, the learning rules would become computationally significantly more expensive and training would take much longer. This is a serious disadvantage and reduces the merits of the algorithm under the criteria defined in Section 1.4 on page 7. Although the Tempo 2 algorithm is a very powerful model of cognitive connectionist resource allocation, it is not competitive in view of real-world applications. These observations with the Tempo 2 approach led to the development of a hybrid approach with one layer of Tempo 2 connections and one layer of TDNN connections, which is described in the next section.

4.6 Adapting the Tempo 2 Approach: The TDNN/Tempo 2 Hybrid

The problems with the interactions between the Tempo 2 learning rules for weights, delays and widths in large neural networks with many hidden units lead to the development of a hybrid network with only one layer of Tempo 2 connections (see Figure 13 on page 50). The connections from the input to the hidden units are realized as TDNN-type connections as introduced in [Waibel, 1989]. The main idea of this type of connection is the use of a shifted input window over time. In contrast to the Tempo 2 input window, the TDNN-type input connections are realized by a set of consecutive hat-shaped input windows over time. This set of input-windows is shifted over the whole spectrogram and training is performed using weight sharing across shifts. This enforces time-shift invariant recognition of acoustic features.

In the hybrid approach described in this section, the connections from the hidden units to the output units are the Tempo 2 connections described in the previous section. The idea behind this approach is to use one layer of TDNN-type connections to get a time-shift invariant distributed internal representation in the hidden layer. The ability of the Tempo 2 connections to adapt to the temporal characteristics of the task is then used for the mapping from the hidden units to the output units. This hybrid approach was tested with the same alphabet recognition task (segmented data) that was used in the previous experiments and achieved slightly better recognition results (90% on training data and 85.8% on testing data). This approach was tested with a varying number of hidden units. I found that 25 hidden units are sufficient to reach these results, but almost the same results could be obtained with 50 hidden units (91.4% on training data and 85.6% on testing data)

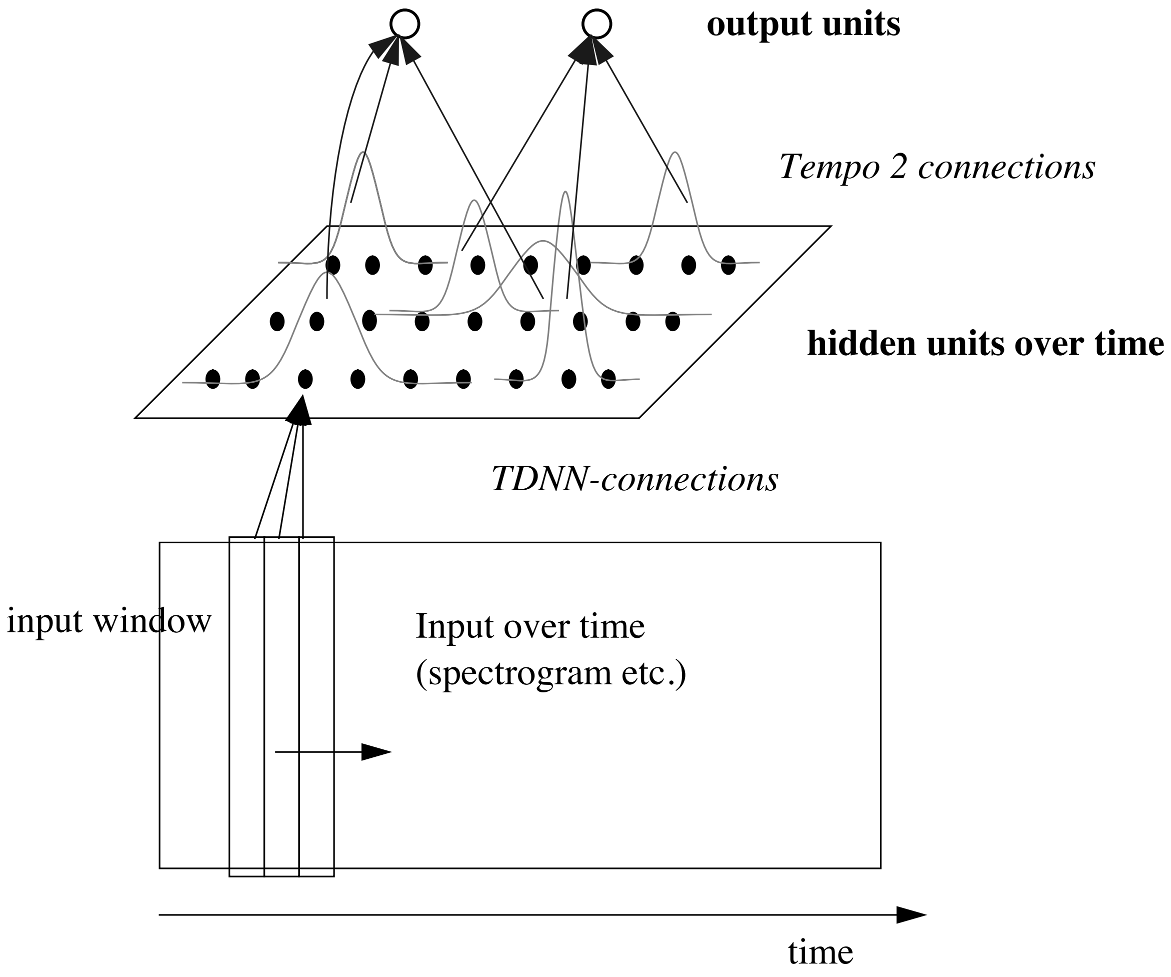

Figure 13. The TDNN/Tempo 2 hybrid network. The connections between the input units and the hidden units are TDNN-type connections (see text for details) and the connections between the hidden units and the output units are Tempo 2 connections. Only two output units are shown for simplicity.

The performance on the testing data is better than using the Tempo 2 approach alone (with or without hidden units) and choosing the learning rates was much easier. However, the TDNN-type connections from the input to the hidden layer had to be optimized manually for these results. An algorithm that also optimizes this type of connections would be highly desirable. This led to the development of the Automatic Structure Optimization (ASO) algorithm described in the next chapter.

5. The Automatic Structure Optimization (ASO) Algorithm

One reason for the introduction of structure to the network is the relationship between the number of trainable parameters, amount of training data and generalization (see [Denker, 1987], [Baum, 1989], [Solla, 1990], [Moody, 1991] and others). Networks with too many trainable parameters for the given amount of training data learn well, but do not generalize well. This phenomenon is usually called overfitting. With too few trainable parameters, the network fails to learn the training data and performs very poorly on the testing data. Imposing structure into the network can increase the generalization performance by reducing the number of trainable parameters [Waibel, 1987], [Waibel, 1989a], [Lang, 1989].

Unfortunately, highly structured networks can be optimized in many more ways than fully connected networks. In order to achieve optimal performance without time-consuming manual optimization, an Automatic Structure Optimization (ASO) algorithm is proposed that automatically optimizes the structure and the total number of parameters synergetically and also considers the current amount of training data. Rather than starting with a distributed internal representation, the structure of the network is constructed by adding units and connections in order to selectively improve certain parts of the network. At the beginning of the training run the internal representation is completely local and gets more and more distributed in the following optimization process. Only a concept for structuring the network has to be specified before training. The concept for structuring the network is derived from (simple) knowledge about the task, such as invariances. A constructive learning approach is used to find a network structure that is specifically tailored for the task and the current amount of training data.

5.1 Concept of the Automatic Structure Optimization Algorithm (ASO)

The proposed algorithm is based on five principles, which are explained in more detail in the following sections:

- built-in invariances
- *automatic* model decomposition
- *constructive* resource allocation
- *classification dependent* resource allocation
- *early* resource allocation

5.1.1 Built-in Invariances

If there is any knowledge about the task, it should be built into the structure of the network. For speech and handwritten character recognition, a classifier that is robust against temporal distortions is highly desirable. This can be achieved by using shifted input windows over time as in the Time-Delay Neural Network (TDNN) [Waibel, 1987], [Waibel, 1989a], [Lang, 1989]. Shifting the window and weight sharing reduces the number of independently trainable parameters and ensures that the hidden abstractions that are learned are invariant under translations in time.

5.1.2 Automatic Model Decomposition

Instead of learning very complex decision surfaces for the classification of events, it may be better to decompose the classification into the recognition of subevents that have to be observed jointly. In many cases the decision surfaces for the recognition of these subevents are much easier to learn. This method is used in many speech recognition systems. For example, the recognition of words can be decomposed into the recognition of sequences of phonemes or phoneme like units. TDNNs have recently been extended to Multi State Time-Delay Neural Networks (MS-TDNNs) [Haffner, 1991a], [Haffner, 1991b] (see chapter 2) that allow the recognition of sequences of ordered events that have to be observed jointly. Unfortunately, this also means that another architectural parameter (i.e. the number of states) has to be optimized.

5.1.3 Constructive Resource Allocation

One of the key requirements for the algorithm is the suitability for small systems (~ 1,000 parameters) as well as for large systems (more than 10,000 parameters). While many algorithms have been developed that work very well for small systems (see chapter 2), the efficient application of these algorithms to large systems requires considerable modifications or simplifications. Pruning algorithms (see chapter 2) require the choice of a "reasonably large" network architecture before the architectural optimization by network pruning can start. If the “reasonably large” network architecture is too large, the computational cost/memory requirements for the training of the first epochs before pruning may be prohibitive. If the “reasonably large” network architecture is too small, the best architecture can not be found at all. To avoid the choice of a “reasonably large” network architecture, a constructive approach is preferable.

A constructive approach also has the advantage that the computational requirement for an epoch is very small at the beginning of the training run when the network is still very small. The computational requirement for each epoch grows with the allocation of additional resources. If training has to be stopped before the optimal architecture is found (due to time constraints), a smaller-than-optimal architecture performs already well on many patterns and is probably superior to a very large network that was only trained for very few epochs.

5.1.4 Classification Dependent Resource Allocation

The choice of a suitable criterion for the allocation of resources is of critical importance. The popular Cascade Correlation algorithm (see chapter 2) allocates resources depending on the training error. Since the main focus of this thesis is on applications using classification, it is feasible to use the classification output of the network directly instead of the training error. Simply put, the network is improved where it performs badly. The training error is still used to train the weights within a given architecture.

It was frequently observed that application-oriented researchers using neural networks use the confusion matrix of the training data for manual optimization of the network architectures. A certain architecture is trained until the stopping criterion and then the confusion matrix is evaluated. If a structured approach is used (as in many speech recognition systems), the modeling can be refined if too many errors in a certain class are observed. This kind of approach is also very useful for an automatic optimization procedure and is used in the ASO algorithm.

5.1.5 Early Resource Allocation

Waiting for a whole training run and then making decisions on the further optimization of the network is computationally very expensive. Experience with neural network classifiers shows that it is possible to detect the most important misclassifications very early in the training run if a “normal” learning rate is used[1]. In this case it is most efficient to also change the architecture early in the training run. Starting the training run again is not necessary.

For practical use of the algorithm, early resource allocation (meaning the successive allocation of resources from early in the training run) is of critical importance. In most of the experiments with the ASO algorithm, 5,000 to 25,000 connections had to be allocated before the best architecture was reached. This means that early resource allocation is of critical importance lest training/ architectural optimization time can get too large.

1. A very large learning rate leads to oscillations. In this case the confusions will obviously change, too.

5.2 Tools of the ASO Algorithm

5.2.1 The Confusion Matrix

The confusion matrix will be the main tool for the analysis of the network architectures in the following sections. Table 5 on page 54 shows an example. The target outputs are shown on the horizontal axis and the actual outputs are shown on the vertical axis.

TABLE 5. The confusion matrix for the segmented spelled alphabet database (SEG_SR_ALPH_DBS) at an intermediate training stage (75 training epochs). The performance on the training set is 87.5%. The target outputs ("A" to "Z") are shown on the horizontal axis. The actual outputs of the networks are shown on the vertical axis.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	74	1			2					6															1	
B	1	65		2							2						1				1	2				
C			74																3							3
D		3		80		1	2													2			2			1
E	1				73					1								1			4					
F					1	77	1						1	1		1			2	1		3	1	1		
G			1				78	19									1									
H	3	1			3			63		1							1			1		1				
I	1								73					2	3			3								
J							1			76	1				1											1
K		1									79		2			1								1	1	
L			1		1				1			79			2											
M		2			1				1				74	37	2			3						1	1	
N				1	1								4	44	1											
O	2								6			5	1		74			1							2	
P		2		1		1		1			1					81	2	1					1	1		
Q			2				2	2			2						78			1				2		
R									3				2	1	1			74				1			2	
S			3			1	1					1							78				1			
T			1														1			80				1		
U					2																79	1				
V	3	9			1	4							1				1	1			1	75	3	1	4	
W		1		1		1										2						1	76			
X																						1		77		1
Y									1	1					1			1							74	
Z			3																2				1			79

5.2.2 Confusion Symmetries

Although the confusion matrix shows all confusions, it is not possible to see directly whether two classes are confused pairwise (class “A” is confused with class “B” and class “B” is confused with class “A”). The following matrix computes all of these symmetries: Let c_{ij} be an element of the confusion matrix C. The elements of the confusion-symmetry matrix S are computed as follows:

$$s_{ij} = c_{ij}c_{ji}$$

Table 6 on page 55 shows an example for such a matrix. Because $s_{ij} = s_{ji}$ only the elements below the diagonal are shown for clarity.

TABLE 6. The confusion-symmetry matrix of the confusion matrix shown in Chapter 5 on page 54 reveals that the spelled letters “M” and “N” are the most frequently pairwise confused letters (SEG_SR_ALPH_DBS).

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	*	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
B	1	*	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
C			*	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
D		6		*	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
E	2				*	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
F						*	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
G							*	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
H								*	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
I									*	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
J										*	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
K		2									*	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
L												*	-	-	-	-	-	-	-	-	-	-	-	-	-	-
M													*	-	-	-	-	-	-	-	-	-	-	-	-	-
N													148	*	-	-	-	-	-	-	-	-	-	-	-	-
O									18			10	2		*	-	-	-	-	-	-	-	-	-	-	-
P						1					1					*	-	-	-	-	-	-	-	-	-	-
Q							2	2									*	-	-	-	-	-	-	-	-	-
R									9				6		1			*	-	-	-	-	-	-	-	-
S			9			2													*	-	-	-	-	-	-	-
T																	1			*	-	-	-	-	-	-
U					8																*	-	-	-	-	-
V		18				12												1			1	*	-	-	-	-
W				2		1										2						3	*	-	-	-
X																						1		*	-	-
Y															2			2							*	-
Z			9																							*

Why does ASO use the confusion-symmetry matrix? In various training/optimization runs it was observed that it is advantageous to use different criteria for different types of resources. The confusion-symmetry matrix directly shows pairwise confusions and is advantageous as an allocation criterion for resources that solve these pairwise confusions (see Section 5.5.3 on page 74).

5.3 Assumptions based on Empirical Observations

The design of the ASO algorithm is based on two assumptions:

- Neural networks, that are too small for the given task, do not change the distribution of the misclassifications very much if the learning rate is set normal and oscillations are avoided. This means that the most important confusions can be detected very early in the training run and that the constructive process can be started very early.
- Confusions are not equally distributed over the whole confusion matrix. The confusions in typical speech and on-line handwriting applications are rather local and can easily be detected by analysis of the confusion matrix.

These two assumptions are based on empirical evidence from many training runs of neural network based speech and on-line handwriting recognizers. The following sections attempt to present the empirical evidence:

Table 7 on page 57 shows the confusion matrix on the training set of a network that is too small to learn the task (on the SEG_SR_ALPH_DBS task). After 200 epochs there are still too many confusions. The performance on the training data was 50.0%. Further training will only solve very few of these confusions. One of the major ideas behind the ASO algorithm is based on the assumption that the distribution of the confusions at the end of the training run is similar to the distribution of the confusions earlier in the training run if the network is too small to learn the task. This assumption is true in training runs with a normal learning rate. A very large learning rate will lead to oscillations which will change the distribution of the confusions significantly. The confusion matrix of this network on the test set is shown in Table 8 on page 58. Many confusions are the same as for the training set. Test performance was only 42.1%.

TABLE 7. The confusion matrix of a small network on the training set after 200 epochs. The network is too small to learn the task (SEG_SR_ALPH_DBS).

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	31	1	3		2		1		18		1				17		6				4	1				
B		14	1	1			3		1		7					9	1					3	12		9	
C			37	1	4	3	2	1	1			1							2	3			1		6	7
D		10	2	13	2						4					2	3			6	1		7			4
E			4	2	11		1								1		2			2	3					3
F	3		1	2	1	82	1		1				2	1	2	4	2		2		5	3	4	2	2	1
G		1	6		4		24		2	1	1						6			3	6	3	2		1	5
H	26		1	9	29		13	83		1	4					2	4			10	21	8	2		1	
I	6	3			1		1		29					1	1	1	1						2		3	
J	1		4		2		1		4	82	5				1		3				5		2			2
K	1	5	1		1		5		1	1	9					7	10			3	1	2	1		2	
L	1		1		1				2			82			7											
M	4	1			6				3		1		74	15	5			1			1	1	2		1	
N			1	3	3								6	68	1						1					
O	5	2		2	2		2		7		2	2	1		32	1	3				2		2		3	3
P		8		6	4		1				4				1	30	5			11		3	2		4	
Q		5	1	3			5				10				1	5	10			7	2	3	10			1
R		2							5				2		4			80				1			3	
S	2		9												1				79				1			
T		2	2	1	2						5					4	4			12		3	3			2
U		7		10	2		9				4					6	5			7	13	3	7		8	
V	1	2	2	6	6		1				3					2	8			5	10	39	3		4	8
W		13	1	9	1		8		1		11				1	2	3			8	6	3	17		3	6
X	2								1		1					1	1		1					83		3
Y		7	1	5			2	1	9		8				10	9	7	4		5	4	6	2		35	4
Z	2	2	7	12	1		5				5						1		1	3		3	3			36

TABLE 8. The confusion matrix on the test set of the same network as in Table 7 on page 57 (SEG_SR_ALPH_DBS). Many confusions are the same as for the training set.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	4	2			4		1		4		1	1			3		1			1	1					
B		1																				1	2		1	
C			4	3		1		1				1				1				4						1
D				4			1				2				1	3	1			3			2			2
E			1	1	2												1			1	1					
F	1			1		18							3		1		2				1	1	1	1		
G		1	2		1		5								1	1	4				1		1			
H	7		2	1	10		1	16		1	2									2	7		1			
I	1	2					1		3			1				1										
J							1		3	19	2															
K				1							3				1	2	1					1				
L	1								1			14														
M	3								1				14	6	2	1		1				1				
N					1								1	14												
O	2	1			1				1		1	1			5								1			
P		3			1		2	1			2					0	3			1						
Q		5	2				3		1		2					5	2			2	3	3	4			1
R									1			1	1		2			19				3			2	
S							1						1		1	1			20							
T		1	1								1						2			2						1
U		2		4			2									2	1				0	1	6		1	
V		1	1	2											1	1	1			1	4	6			1	5
W				1			1				2					1	1			2	1		2		1	
X	1																					2		19		
Y		1	1	1		1		2	5		2	1			2	1				1	1	1			14	1
Z			6	1			1																			9

The use of a table to display the confusion matrices is not very convenient because many numbers have to be read for the interpretation of the matrix. The use of a graphic display (where the number of confusions is proportional to the size of little blobs) makes the distribution of the confusions much more obvious. Each of the following figures shows two different confusion matrices simultaneously:

- the confusion matrix on the training data on the lower left
- the confusion matrix on the validation data on the lower right.

Figure 14 on page 59 and Figure 15 on page 60 show a training run with a network that is too small to achieve good recognition results on this task (SEG_SR_ALPH_DBS). A comparison of the confusion matrices on training data after one, two, three, five, 50 and 250 epochs shows clearly that the distribution of the misclassifications does not change significantly over the training run. The figures also show that the misclassifications are not equally distributed over the whole confusion matrix.

Figure 14. The confusion matrices for both training and validation data from a network that is too small to learn the task well (continuously spelled alphabet recognition with 500 training sentences). The matrices are shown after the first, the second and third epoch. A comparison of these matrices shows that the distribution of the misclassifications does not vary much from the first to the third epoch. Additionally, the misclassifications are not equally distributed over the whole matrices. Chapter 15 on page 60 shows samples from the confusion matrices from the rest of the training run.

Figure 15. The confusion matrices on both training and validation data from a network that is too small to learn the task (continued from the last page). Here, the confusion matrices after the five, 50 and 250 epochs are shown. A comparison reveals that the distribution of the errors does not change significantly. The network reached 78.6% on training data, 77.4% on validation data and 78.7% on testing data, which is not good for this task (continuous spelled alphabet recognition).

5.4 Resource Allocation by the ASO Algorithm

The ASO algorithm tries to optimize the architecture of neural classification networks for best possible generalization performance. According to Moody [Moody, 1992], the expected error on the test set can be approximated as follows:

$$\langle E_{test}(\lambda) \rangle_{\xi\xi'} \approx \langle E_{train}(\lambda) \rangle_{\xi} + 2\sigma_{eff}^2 \frac{p_{eff}}{n}$$

where n is the number of training exemplars in the training set ξ, σ_{eff}^2 is the effective noise variance in the response variable(s), λ is a regularization or weight decay parameter, and p_{eff} is the effective number of parameters in a nonlinear model [Moody, 1992][1].

The idea of the ASO algorithm is to start with a small number of parameters for the given number of training exemplars (leading to a small second summand on the right side of the above equation) and increasing this number to decrease the expected error on the training set (the first summand in the above equation). The goal is to increase the second summand and to decrease the first summand until the best possible compromise between a low training error and a high number of parameters is reached.

Resources are allocated to lead to the smallest possible number of effective parameters, which is the relevant parameter for generalization capability. The total number of parameters is only relevant for computational cost.

5.4.1 The Criteria for the Allocation of Resources

The criterion for the constructive allocation of resources is derived from the confusion matrix and the confusion-symmetry matrix on the training set. At each epoch the confusion matrix C with the elements c_{ij} and the confusion-symmetry matrix S with the elements s_{ij} are computed.

Several criteria for the allocation of resources from these matrices were tried. The simplest criterion is derived from the sum of all matrix elements of one column (i.e. counting all mistakes of a particular unit). Let C be the confusion matrix.

$$C = \begin{bmatrix} c_{11} & c_{12} & \dots & c_{1n} \\ c_{21} & c_{22} & & \\ \dots & & & \\ c_{n1} & & & c_{nn} \end{bmatrix}$$

1. The effective number parameters p_{eff} is smaller or equal than the total number of parameters p_{tot}.

All elements of the columns are added:

$$A_j = \sum_{i=1}^{m} c_{ij}$$

New resources are allocated if

$$A_j > \bar{A} \cdot f(p, n)$$

where $\bar{A}$ is the average of all A_j and $f(p, n)$ is a function that limits the allocation depending on the number of parameters p and the number of training patterns n (see Section 5.4.3 on page 63).

A different criterion emphasizes the distribution of the confusions. The idea is to weight the same total number of confusions higher if they are distributed over many classes than if they are only in few classes. This can be achieved by the following product:

$$B_j = \prod_{i=1}^{m} (c_{ij} + 1)$$

One criterium has been derived based on the confusion-symmetry matrix. Let S be the confusion-symmetry matrix (note that the elements of the diagonal and above the diagonal are not computed [see 7.1.2]):

$$S = \begin{bmatrix} s_{11} & s_{12} & \cdots & \cdots & s_{1n} \\ s_{21} & & & & \\ s_{31} & & & & \\ \cdots & & & & \cdots \\ s_{n1} & s_{n2} & \cdots & s_{n(n-1)} & s_{nn} \end{bmatrix}$$

So far, only the maximum of all s_{ij} is considered. The criterium for the allocation of new resources is reached if

$$max\{s_{ij} | s_{ij} \in S, i > j\} > f(p, n) \cdot (average\{s_{ij} | s_{ij} \in S, i > j\})$$

where $f(p, n)$ is a function that limits the allocation of resources (see Section 5.4.3 on page 63).

5.4.2 The Frequency of Resource Allocation

One of the key principles of the ASO algorithm is early resource allocation, meaning that the allocation of resources is started very early in the training run. An additional constraint comes from the fact that the algorithm should be able to allocate many parameters efficiently. Typical network sizes for the tasks that are used as a benchmark in this thesis (see Chapter 3 on page 35) require networks with up to 30,000 independently trainable connections. The minimal size of the architecture can be very small, approximately 500 independently trainable weights. That means that the algorithm should be able to allocate such a high number of weights efficiently. There are two possible ways for doing this:

- Allocating large groups of resources occasionally during the training run.
- Allocating smaller groups of resources frequently.

Since the number of resources that are allocated at a given time is also determined by the criteria from Section 5.4.1 on page 61, it is preferable to allocate resources as frequently as possible. In the current implementations, new resources can be allocated at the end of each training epoch[1]. Experiments with resource allocation after three or five epochs did require considerably longer training runs (because it took longer to construct the network) and did not perform better.

5.4.3 Limiting the Allocation of Resources Depending on Training Set Size

All constructive resource allocation algorithms need a method to decide when no further resources are necessary. The most obvious method stops the allocation of resources when all training patterns are learned. This is unrealistic for real-world applications like speech or handwriting recognition because a) it may not be possible to learn all training examples (for example because they may have the wrong label) and b) training until all patterns are learned may lead to overfitting to the training data.

Stopping resource allocation when the performance on a separate validation set goes down does not solve the above problem. The performance tends to decrease slightly when new resources are added to the network[2] and increases again when training is continued. For best generalization performance it is necessary to have two different criteria, one that limits the allocation of resources and another that terminates the training process.

Constructive algorithms that allocate different types of resources (like the width of the input windows, the number of states and the number of hidden units in the given architectural optimization problem) also have to distinguish between a global limitation criterion (that is valid for all types of resources together) and local limitation criteria (that are only valid for one type of resource).

1. depending on whether the criterion for resource allocation is fulfilled or not.

2. depending on the initialization of the resources that are added to the network

The simplest method is to use fixed upper bounds for all of these criteria. This is very simple and easy to implement, but this also means that at least some knowledge about the required resources and a good distribution over the different types of resources should be available.

A more elegant solution is the use of a soft limiting function which as described in the next section. An even better solution is the *Automatic Validation Analyzing Control System* (AVACS) described in Chapter 7 on page 91.

5.4.3.1 Global Resource Limitation Depending on Training Set Size

The ASO algorithm uses a simple scheme that modifies the global criterion for the allocation of resources depending on the value of the function f which depends on the number of training patterns and the total number of parameters in the network:

$$f(p, n) = \frac{\left(\frac{p}{n}\right)^2}{\alpha} + 1$$

where p is the number of parameters and n is the number of training exemplars and α is a constant. The value of α is not critical. Values between 5 and 20 give good results. The effect is that adding more resources is made easy if the number of connections is small compared to the number of training patterns and gets harder with an increasing number of connections. This avoids hard upper bounds for the network resources. The validation set was used to test the dependence of the generalization performance for different values of α. The results are summarized in Table 9 on page 64.

TABLE 9. Validation Performance Depending on the choice of the Resource Limiting Function (constant α). The system was trained with 520 patterns from the segmented speech corpus (SEG_SR_ALPH_DBS).

constant α	performance on validation data
3	79.8%
5	81.5%
10	81.2%
15	81.2%
20	80.0%

The quadratic function works well in all simulations that were tried. However, a simple constant (β) leads to almost the same results:

$$f(p, n) = \beta$$

This simple alternative achieves almost the same results with $\beta \approx 1.1$ .. $\beta \approx 1.5$. The validation set was used to determine the best value for β and to show that the ASO algorithm is robust against the exact choice of β . The results are shown in Table 10 on page 65. The results show that the choice of β is not very critical. However, a large β leads to “slower” resource allocation in the early stage of a training run, which generally leads to a higher number of training epochs.

TABLE 10. Validation Performance Depending on the Choice of the Resource Limiting Constant β (after training with 520 patterns from the segmented speech corpus (SEG_SR_ALPH_DBS)

constant β	performance on validation data
1.0	78.2%
1.1	81.5%
1.3	81.0%
1.5	81.5%
1.7	80.7%
2.0	80.4%

5.4.4 The Initialization of New Resources

New weights can be initialized in three ways:

- small random numbers
- zero
- non-random, non-zero.

The ASO algorithm uses all of these possibilities depending on the weight that is initialized. Weights are initialized by small random numbers when no knowledge about the task of the connection is available (like all weights from the inputs to state units or hidden units. Random weights have the advantage that they do not disturb the learning process very much, i. e. the training error does not change much after the allocation of these weights.

Weights are initialized by certain non-random values if the connection is installed for a special pairwise confusion (like the weights from the hidden units to the state units, see Section 5.5.3 on page 74). Weights are initialized by zero if the weights are not directly needed to solve a certain confusion, but should be usable for other confusions if the Back-Propagation algorithm adjusts the weight significantly (see Section 5.5.3 on page 74).

5.5 Application of the ASO Algorithm to MS-TDNNs

Multi State Time Delay Neural Networks (MS-TDNNs) (see Section 2.1.2 on page 11) conform with the first two principles of the ASO algorithm, build-in invariances and task decomposition. Additionally, they are very powerful classifiers [Haffner, 1991a], Haffner, 1992a], [Haffner, 1922b], [Hild, 1993] and the architecture of these highly structured networks can be optimized in many ways. For best performance, the size of the input windows, the number of hidden units and the (word specific) state sequence topologies are of critical importance for optimal performance. This makes MS-TDNNs a suitable candidate for the demonstration of capabilities of the ASO algorithm.

The ASO algorithm optimizes all relevant parameters of MS-TDNN type network structures for a given amount of training data. The minimal configuration of a MS-TDNN consists of an input layer, a state layer and an output layer (see Section 2.1.2 on page 11). Let us consider a word recognition task where each output unit represents a word. Each state unit represents a small piece of the utterance, phonemes or sub-phonemic states. The network is initialized with a window size of one (one connection between an input unit and a unit of the following layer) and one state unit per output unit (see Figure 18 on page 68). The net input of the output units is computed by integrating the weighted activity of the single or multiple state unit(s) over time. The activation of the output units is given by the sigmoid of the net input. The state units can also be regarded as a special type of hidden units because of their very constrained connectivity to the output units.

The ASO algorithm optimizes the size of the input windows of the state units and the number of states based on the confusion matrix. The number of hidden units and the size of the input windows of these hidden units is optimized based on the confusion-symmetry matrix. (see Figure 17 on page 67).

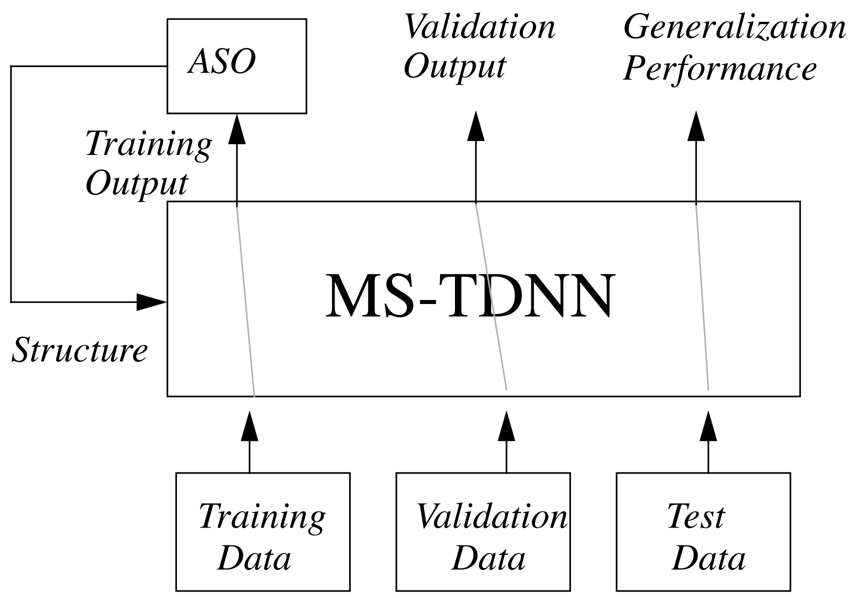

Figure 16. System overview.

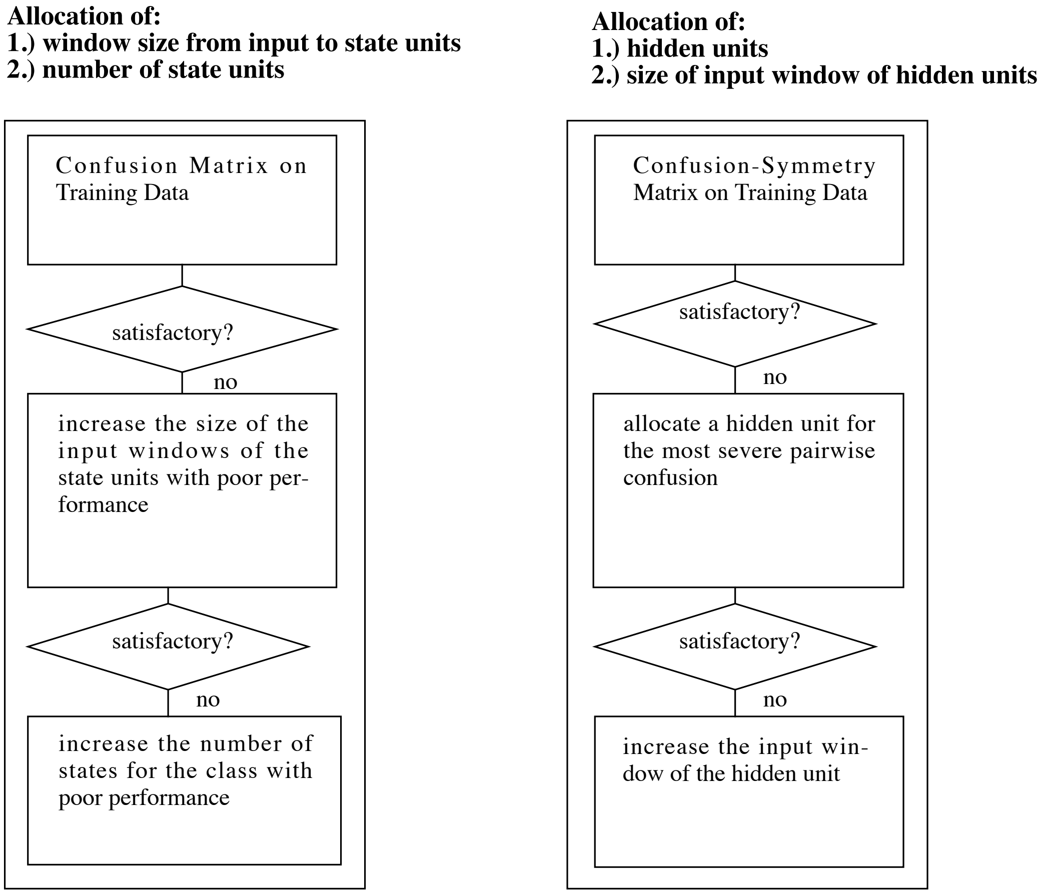
Allocation of:
1.) window size from input to state units
2.) number of state units
Confusion Matrix on Training Data
satisfactory?
no
increase the size of the input windows of the state units with poor performance
satisfactory?
no
increase the number of states for the class with poor performance
Allocation of:
1.) hidden units
2.) size of input window of hidden units
Confusion-Symmetry Matrix on Training Data
satisfactory?
no
allocate a hidden unit for the most severe pairwise confusion
satisfactory?
no
increase the input window of the hidden unit

Figure 17. The default order in which resources are allocated by the ASO algorithm. The elements of the confusion symmetry matrix s_{ij} are computed from the elements c_{ij} of the confusion matrix as $s_{ij} = c_{ij}c_{ji}$ The "satisfactory?" decision refers to the criteria from Section 5.4.1 on page 61. In principle, there are two independent allocation schemes: 1.) One for the optimization of the window size of the direct connections from input to state units (shown on the left side) and 2.) one for the allocation of hidden units and optimization of the input window size of these hidden units (shown on the right side).

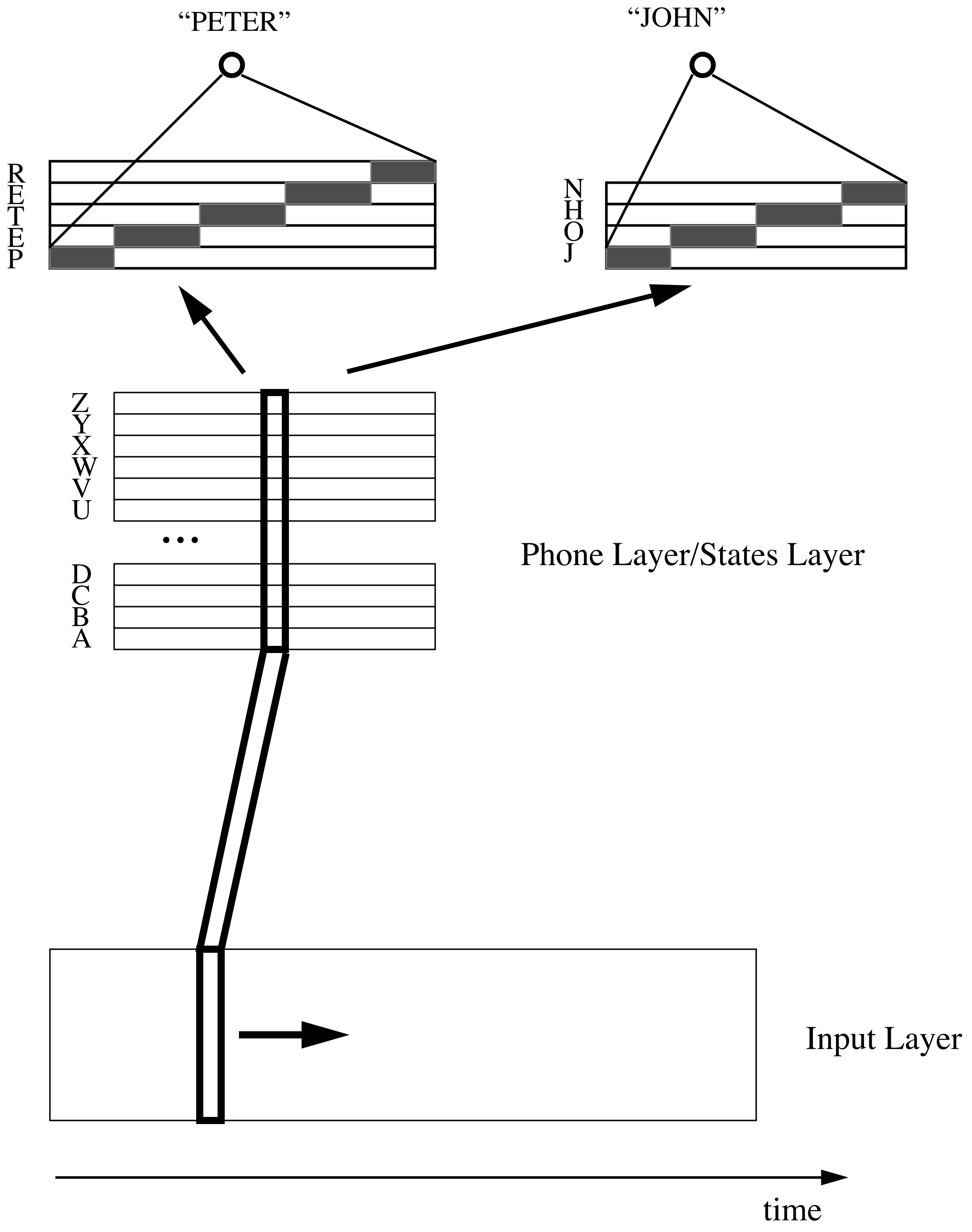

Figure 18. The initial architecture of the MS-TDNN for spelled letter recognition: no hidden units, one state per letter, input windows of one frame width.

5.5.1 Increasing the Size of the Input Windows

The ASO algorithm starts with a window size of one (no temporal context), one state unit and no hidden units. The size of the input windows is increased depending on either A_j or B_j. At first, the size of the input window from the input to the state unit is increased by adding one set of random connections (see Figure 19 on page 70). In the next epoch, these new connections are trained together with the already existing connections.

5.5.1.1 Limiting the Size of the Input Windows

The maximum size of the input windows depends on the number of states that model a word. If a word is modeled by many states then the state units do not need such a large input window as a state unit that models a whole word. In the ASO algorithm the average duration a_i of the event to be modeled (for example a spelled letter) is computed and the maximal size of the input window m_i is the average duration of the event divided by the actual number of states s_i:

$$m_i = int\left(\frac{a_i}{s_i}\right)$$

where int rounds the value of the quotient to an integer.

5.5.2 The Allocation of State Units

If the size of the input window of a state unit converges and the corresponding output unit still makes more mistakes than the average unit, then a new state unit is added. The size of the input window of the 'old' state unit is halved to avoid a dramatic increase of the number of trainable parameters. The 'new' state unit receives input from an input window of the same size as the 'old' state unit after halving, but with random connections. From now on, the output unit receives input from both state units.

Figure 19. The growth of the input windows of the direct connections from input to the state units. In the first epoch (indicated by number 1) all windows have a width of one frame. The 16 spectral coefficients are shown on the vertical axis of each box. The weights are displayed by small blobs inside the boxes. The value of a weight is proportional to the size of these little blobs. Positive weights are displayed by black blobs, negative weights by white blobs. In the second epoch, the input windows of the models '@' (silence), 'A', 'B', 'C', 'E' and 'F' are two frames each etc.

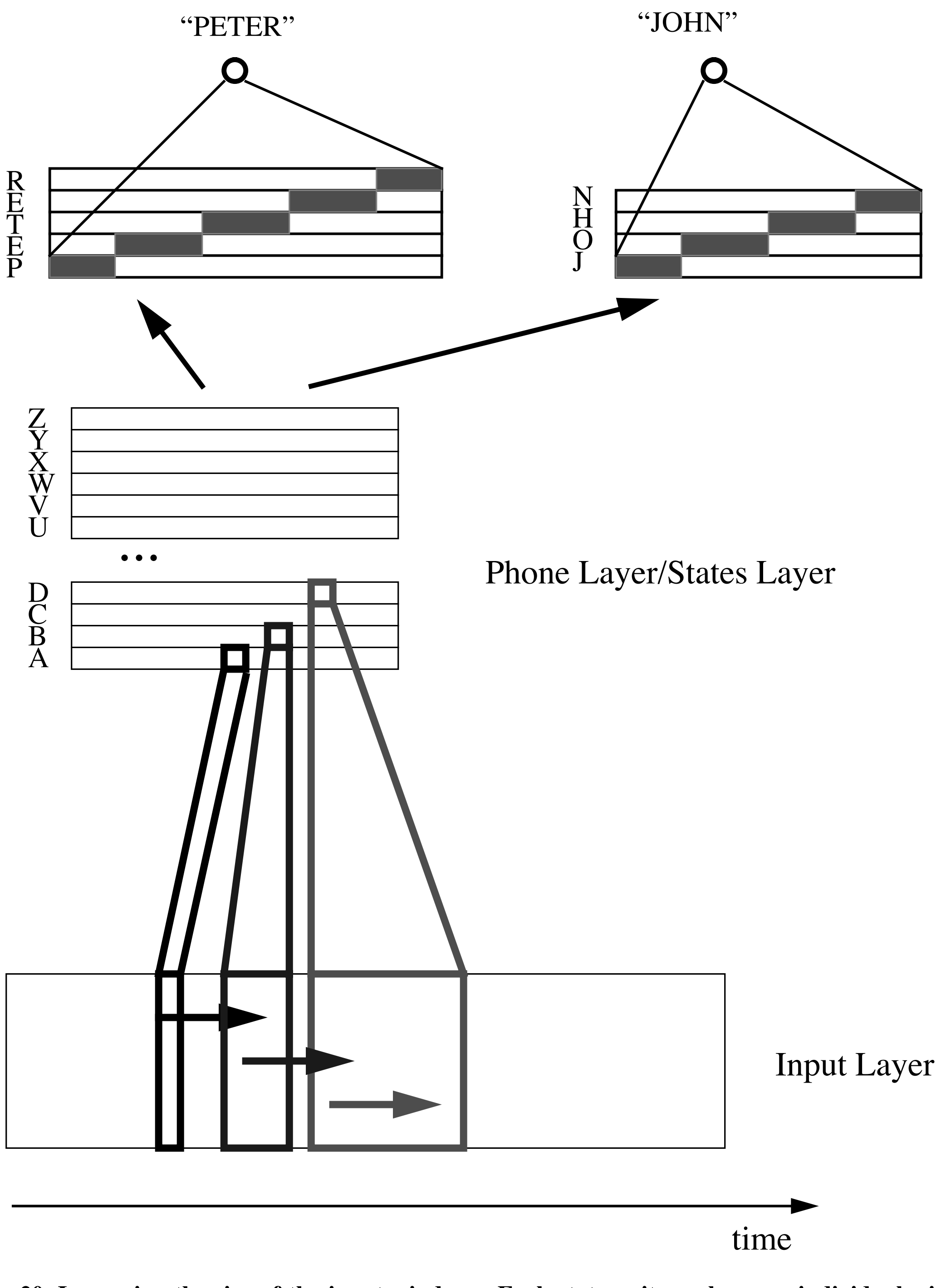

Figure 20. Increasing the size of the input windows: Each state unit can have an individual window size. Only the input windows of the state units representing "A", "B" and "D" are shown.

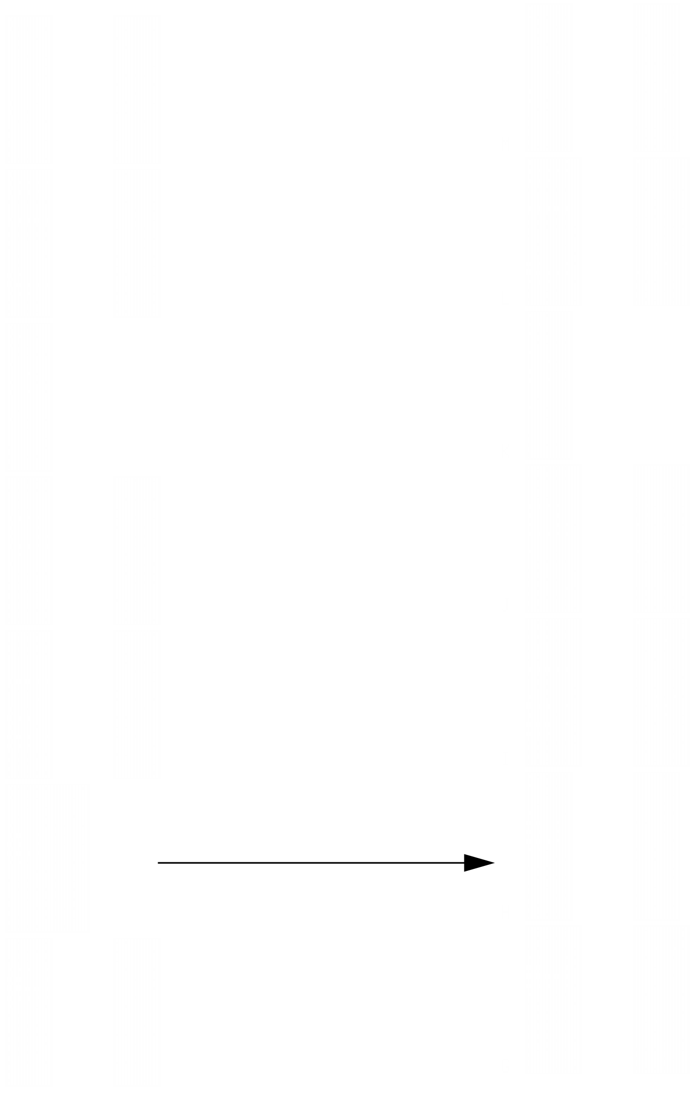

Figure 21. Adding a state for the model of the letter 'H'. Left column: The input windows of the models of the letters 'G' to 'M' are shown on the right side. The letters 'G', 'I', 'J', 'L' and 'M' are modeled by two states already and the letters 'H' and 'K' are modeled by one state only. Right column: After the next epoch, the model for the letter 'H' is split into two states with smaller input windows. Some of the weights from the first state (the left of both) are kept and used after the split, too. The weights of the second state (the right one of both states) are initialized randomly.

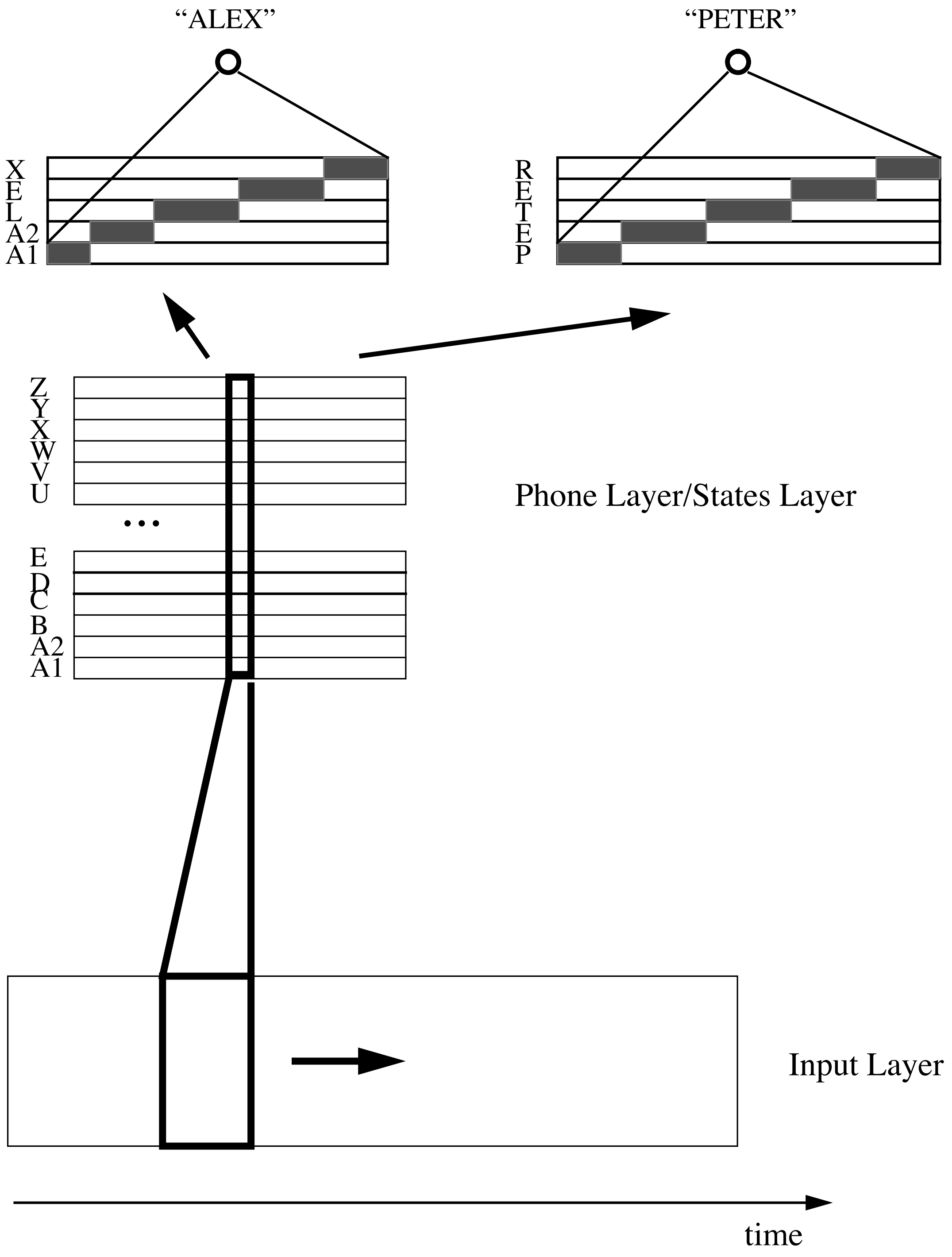

Figure 22. Increasing the number of states: "A" is modeled by two different states "A1" and "A2". All other letters are modeled as before.

5.5.3 The Allocation of Hidden Units

The confusion matrix C was used in initial experiments as a criterion for the allocation of hidden units. Performance results on SEG_SR_ALPH_DBS (2210 training patterns) are shown in Table 11" on page 74. The baseline was a system which automatically optimized the width of the input windows according to the confusion matrix C and classified 85.2% of the validation set correctly (with one state unite per letter and no hidden units). If hidden units were allocated according to C in addition to the window widths, the performance increased to 88.5%. If window widths and the number of state units were allocated according to C, then the performance was 88.5%. If all three types of resources were allocated according to C, then the validation performance was 88.5%. This means that the benefit of automatic allocation of state and hidden units does not combine when the same criterion is used for all types of resources. If window widths and the number of state units were allocated according to C and the number of hidden units was allocated according to S, then the validation performance increased to 90.2%.

TABLE 11. Speech recognition results (SEG_SR_ALPH_DBS). Three types of resources (window widths, state units, and hidden units) are allocated automatically according to the standard confusion matrix C and/or the confusion-symmetry matrix S (as indicated in the table).

window width allocated according to	state units allocated according to	hidden units allocated according to	validation performance
C	- (1 state unit, fixed)	- (no hidden units)	84.2%
C	- (1 state unit, fixed)	C	88.7%
C	C	- (no hidden units)	88.5%
C	C	C	88.5%
C	C	S	90.2%

My conclusion from these experiments was that state units and hidden units have to be allocated for different types of problems. They should specialize on different things and provide different functionality in the system. What functionality could be provided by hidden units? Hidden units are needed if a non-linear mapping from the input to the state units is required [Minsky, 1988]. An example of a non-linear decision boundary is shown in Figure 23 on page 75. With a given linear boundary, many pairwise confusions like the confusion of "A" with "B" and "B" with "A" appear. Hidden units are needed to avoid these pairwise confusions. The confusion matrix S (see Section 5.2 on page 54) was chosen because it directly represents these pairwise confusions.

Other alternatives would be possible, too. For example, computing the sum

$$c_{ij} + c_{ji}$$

would be possible, too. However, the criterion for the allocation of hidden units should emphasize as much as possible on learning problems which are not already detected (and attacked by other resource allocations) by the standard confusion matrix C. The product

$$c_{ij} \cdot c_{ji}$$

is maximal if the confusion is equally distributed on both sides of the boundary (see Figure 23 on page 75) and goes to zero if the boundary is in one direction or another. This means that hidden units are not allocated if the (linear) boundary is not adjusted such that the confusions are equally distributed on both sides of the boundary. Hidden units are installed only when there is no further possibility to move the given boundary.

The weights from the hidden units to the state units are initialized to solve one pairwise confusion[1]. The idea is to initialize the two weights form the new hidden unit to the two state units (which are involved in the pairwise confusion) with different initial values, for example with 0.3 and -0.3. The weights from the input to the hidden units are initialized randomly. This way the hidden unit starts to learn to distinguish between these two classes right from the first Back-Propagation of the errors from the output. All other weights form the hidden units to the other state units do exit, but they are initialized by 0.0. Back-Propagation can adjust these weights such that the other state units can benefit from the hidden unit, but the primary goal of a hidden unit is to solve a certain pairwise confusion.

Figure 24 on page 76 shows how hidden units are added to the MS-TDNN architecture. Hidden units are always added to the already existing direct connections from the input to the state units (like in the Cascade Correlation architecture, [Fahlman, 1990 a and b]). Figure 25 on page 77 shows the allocation of the first hidden unit in a training run on segmented speech data (SEG_SR_ALPH_DBS). The letters "I" and "R" were pairwise confused. Figure 26 on page 78 shows the growth of the input window of the first hidden unit (because the letters "I" and "R" were again the pairwise confused in the next training epoch) and the allocation of a second hidden unit (because the letters "I" and "O" were pairwise confused).

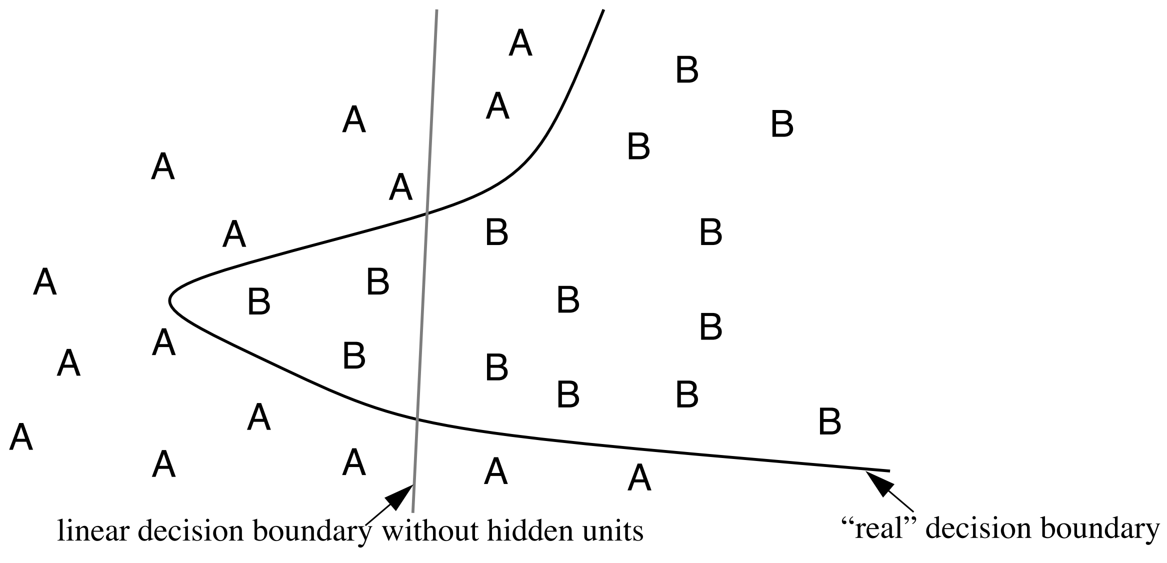

Figure 23. An example where the classes 'A' and 'B' are confused because only linear decision boundaries can be learned without hidden units. The dotted line shows the best possible linear separator. In this case the pairwise confusions ("A" with "B" and vice versa) will be very frequent.

1. Hidden units are always installed in addition to the direct connections between the input and the state units.

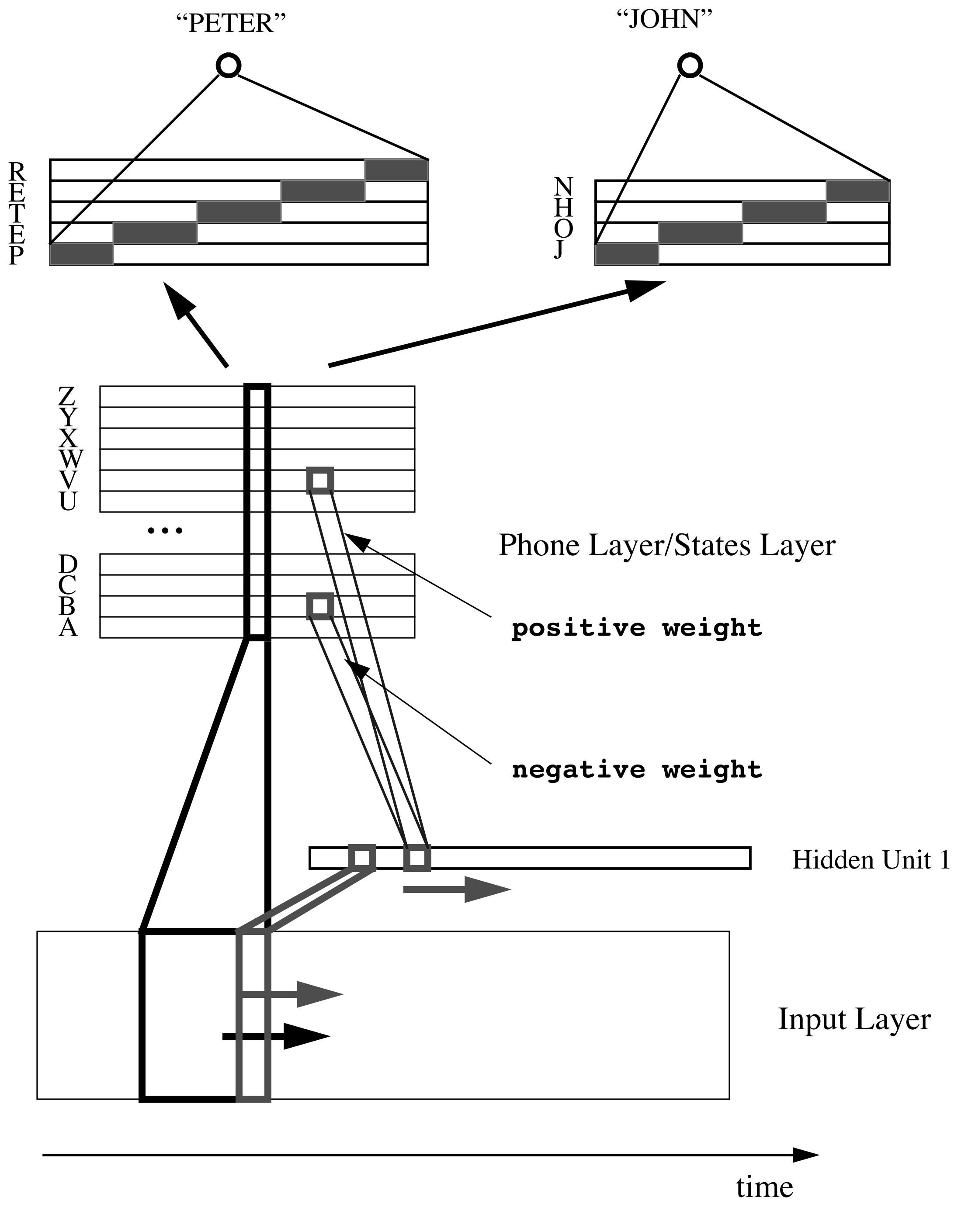

Figure 24. Adding hidden units: Depending on the confusion-symmetry matrix, hidden units are allocated to solve the most severe pairwise confusions. In this example a hidden unit is allocated to solve the pairwise confusion of "B" and "V". The hidden units receive input from a a sliding input window over time (like in the original MS-TDNN). The size of the input windows (from input to the hidden unit and from the hidden unit to the state units) is initialized as one and grows if the same pairwise confusion appears again in the following epochs.

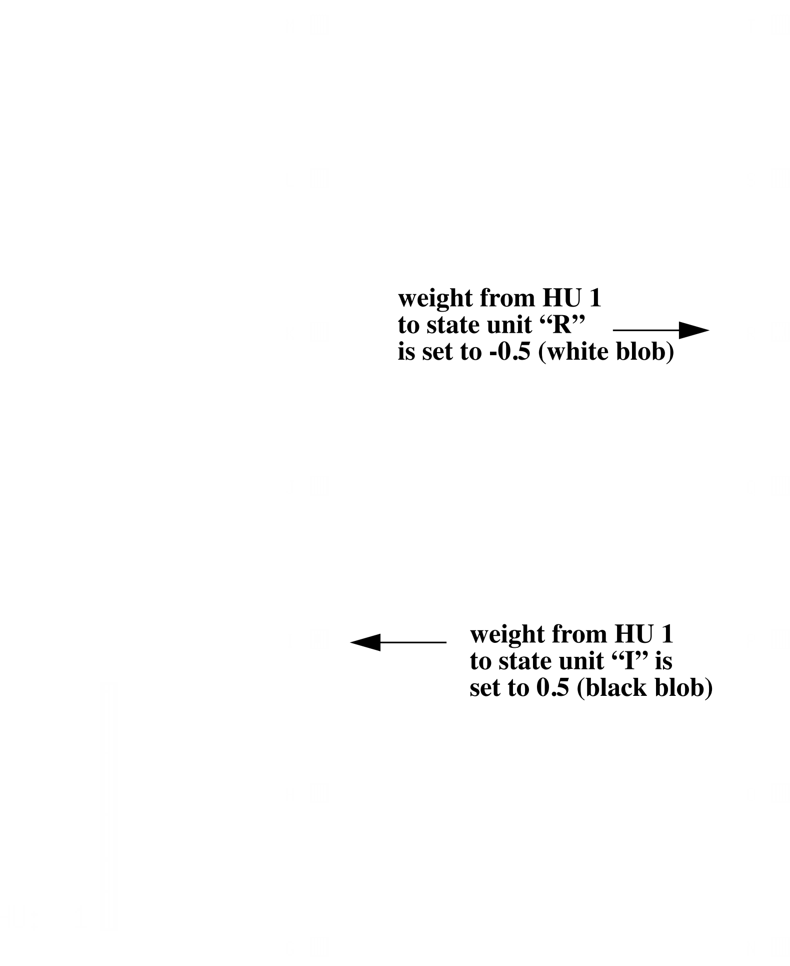

Figure 25. The allocation of hidden units: In this training run, the criterion for the allocation of a hidden unit was fulfilled for the letters “I” and “R” because they were frequently pairwise confused. The input window of the new hidden unit (HU 1) is shown on the left side. Its width is one and the weights are initialized by small, random numbers. Some of the weights from the hidden unit to the state units are shown on the right side: The weight to the state unit representing letter ‘I’ is initialized by 0.5 (indicated by a black blob) and the weight to the state unit is initialized by -0.5 (indicated by a white blob). All other weights to the other state units exist, but they are initialized by 0.0 (no blobs are visible). In the following epochs, all weights are trained by Back-Propagation.

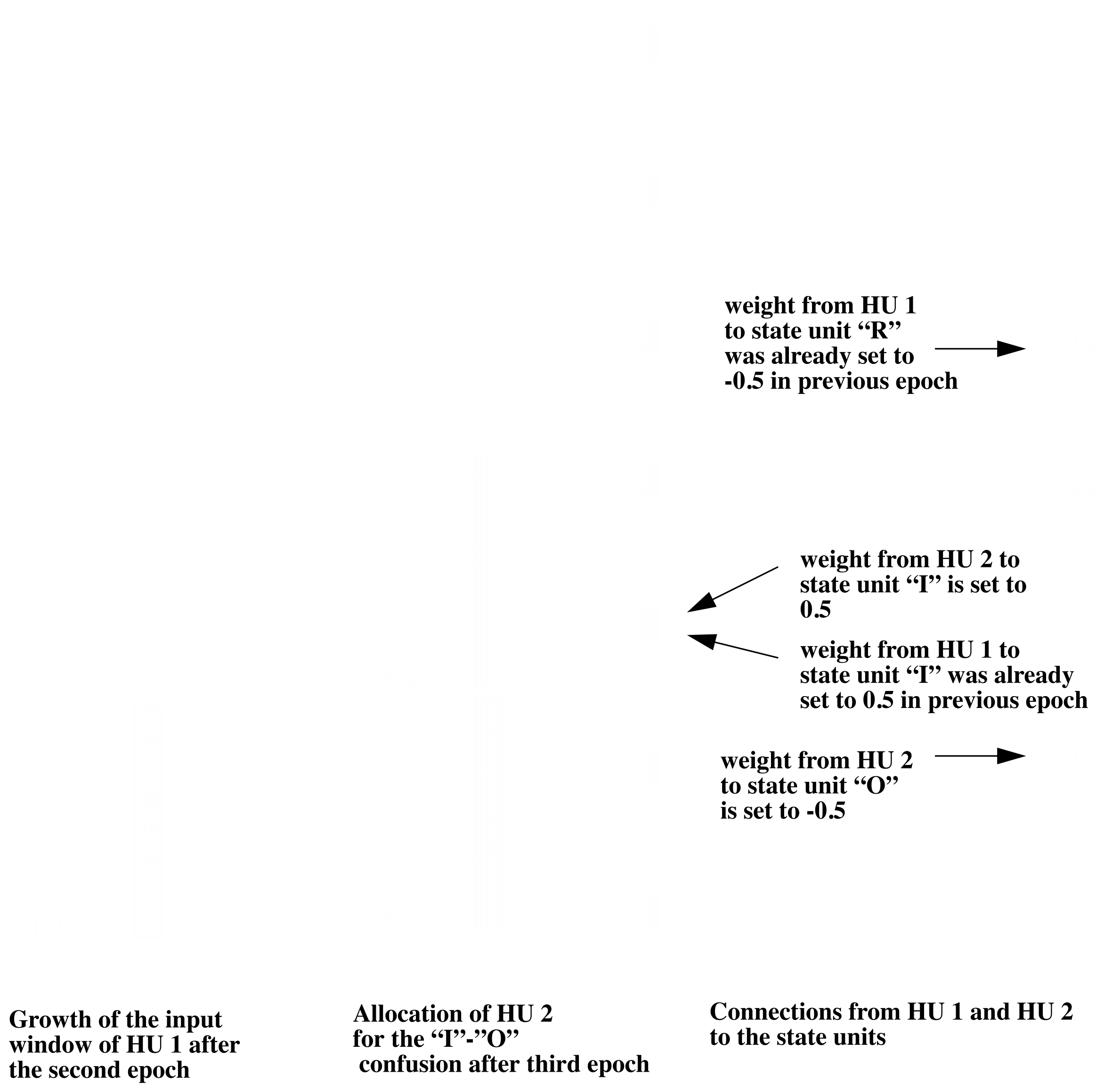

Figure 26. Continued form Figure 25 on page 77. The growth of the input windows of the hidden units and the allocation of a second hidden unit: Right: After the second epoch, the input window of the first hidden unit is increased. Middle: After the third epoch, a second hidden unit is allocated with an input width of one frame. The connections from the hidden units to the state units are shown on the right side: The second hidden unit is allocated because the letters “I” and “O” were pairwise confused. The initial weight from the second hidden unit to the state unit representing “I” is set to 0.5 (as indicated by the second black blob) and the weight to the state unit representing “O” is set to -0.5 (as indicated by the white blob).

5.5.4 How to Initialize new Resources?

All connections from either the input to a state unit or the input to a hidden unit are initialized randomly. This reduces the risk that the new connections disturb the learning process. A side-effect is that noise is added which prevents the network from getting stuck in local minima. This noise is reduced afterwards because the new connections are trained together with the already existing connections.

Some experiments were also conducted with a non-random initialization of the weights from the input to the state unit after the allocation of a new state unit (see Section 5.5.2 on page 69). Since the size of the input window of the "old" state unit is halved (to avoid a dramatic increase in the number of trainable parameters), it would also be possible to use the values of the removed weights as the initialization of the connections of the "new" state. Both methods were compared under exactly the same conditions on the alphabet recognition task with 520 training patterns. The results are shown in Table 12 on page 79. The random initialization is slightly better.

TABLE 12. A comparison of two different initialization methods for the weights from the input to the state units after the allocation of a new state (SEG_SR_ALPH_DBS, 520 training patterns)

Initialization of the weights from input to the "new" state unit after the allocation of a new state unit	validation performance
random	81.5%
with the values of the weights that were removed from the "old" state unit	78.3%

Additional experiments were conducted with a random initialization of the weights from the hidden units to the state units instead of the symmetric/zero non-random initialization (see Section 5.5.3 on page 74). The results are shown in Table 13 on page 79. The random initialization is only slightly worse than the non-random initialization.

TABLE 13. A comparison of two different initialization methods for the weights from the newly allocated hidden units to the state units (SEG_SR_ALPH_DBS, 520 training patterns)

Initialization of the weights from the newly allocated hidden unit to the state units	validation performance
symmetric/zero (see Section 5.5.3 on page 74)	81.5%
random	80.6%

5.5.5 Should all Connections be Trained Equally?

Many connections in the network are added when the error at the output units is quite low already. This means that they do not receive the same amount of training as connections that were trained from the beginning. This can be an advantage, because there is less danger that the 'young' connections contribute to overfitting, especially if they are initialized with small random values.

The small amount of training of the young connections could also be a disadvantage because these connections are not used to their full potential. Additional simulations with an adaptive learning rate that reduces the learning rate depending on the age of a connection were done. The following methods were tried:

$$\varepsilon 1_{ij} = \varepsilon_0 \left(\frac{minage}{age_{ji}}\right)$$

$$\varepsilon 2_{ij} = \varepsilon_0 \left(\frac{1}{age_{ji} + 1 - minage}\right)$$

where $minage$ is the age of the youngest connection in the network, age_{ij} is the age of a connection and ε_0 is the standard learning rate. Both methods reduce the learning rate for old connections when new connections are added. Training of these networks took a little bit longer and the results were worse (see table x). This suggests that unequal training is indeed an advantage of constructive learning algorithms.

TABLE 14. Speech Recognition Results (Segmented Spelled Alphabet Recognition, SEG_SR_ALPH_DBS)

	performance training data	performance validation data
without adaptation of the learning rate depending on the age of a connection	99.8%	90.2%
adapting the learning rate ($\varepsilon 1_{ij}$)	97.8%	88.7%
adapting the learning rate ($\varepsilon 2_{ij}$)	94.6%	84.4%

5.5.6 Smoothing of the DTW Path

In case of more than one state unit per output unit the inputs of the output units can be computed in three different ways: The simplest way is to give each state unit an equal share of the time slice that the output unit represents. The second possibility is to use *Dynamic Time Warping* (DTW) [Sakoe, 1978] to find the best path through the activation matrix of the state units. The third possibility is to smooth the DTW path by Gaussian functions positioned according to the DTW segmentation. Smoothing of the DTW path allows the states to model the transitions between two states more accurately. If each state specializes on different parts of the spectrogram, then the transition between these parts may not be modeled by any of them. Smoothing allows both states to partially represent the transition (see Figure 27 on page 81).

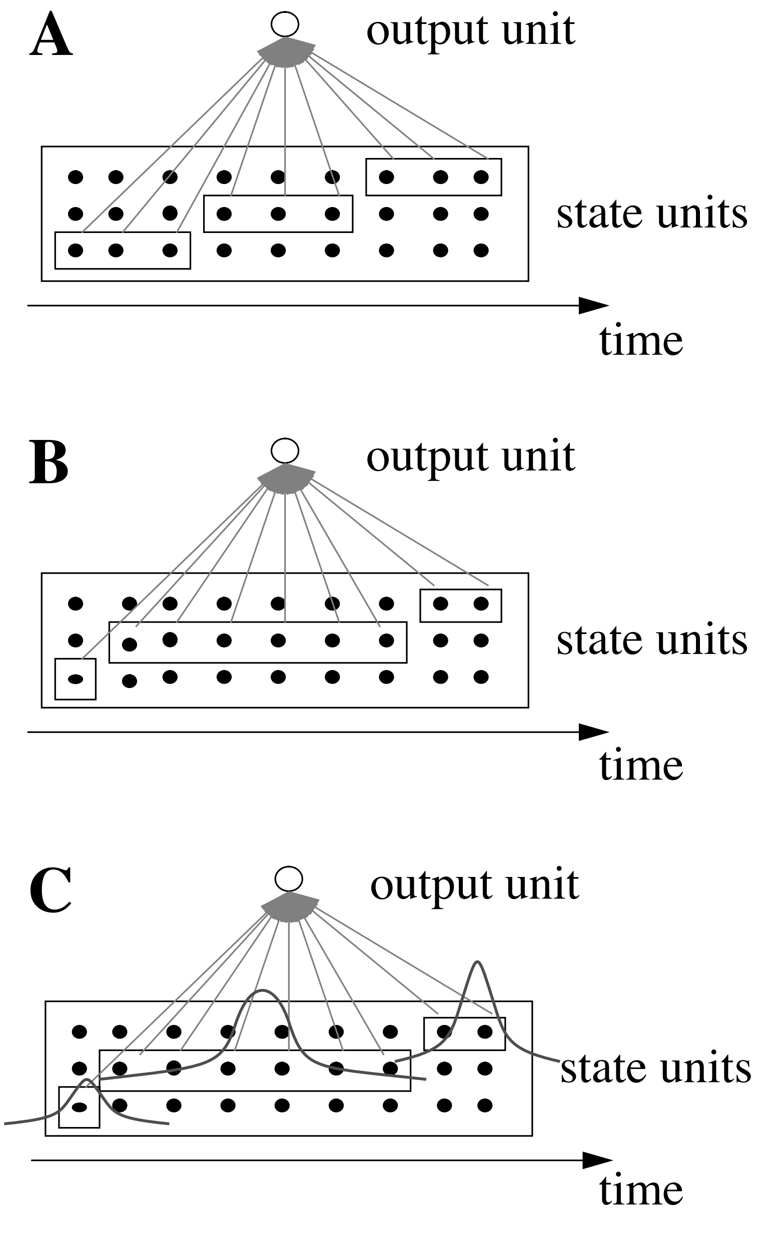

Figure 27. The activation of the output units can be computed in three different ways: A.) Each state unit gets the same time-slice B.) Using DTW to find the best path through the activation matrix of the state units and C.) Using Gaussian functions positioned according to the DTW segmentation to smooth the DTW path.

5.6 Typical Training Runs with the ASO Algorithm

The following pictures show the initial architecture of the MS-TDNN for the recognition of segmented spelled letters (SEG_SR_ALPG_DBS). The network is initialized with one state per letter, an input window width of one frame and no hidden units (see Figure 28 on page 82). Figure 29 on page 83 shows the weights from input to the state units after training. As can be seen, the ASO algorithm constructed an architecture that is far from homogeneous. The number of states varies considerably.

Figure 28. The initialized network for the alphabet recognition task (SEG_SR_ALPH_DBS). All words are modeled by one state. The window size for each state is one. The values of the weights are too small to be seen as black and white blobs on this scale. See Figure 29 on page 83 for the trained and optimized network.

Figure 29. The weights from the input units to the state units for the segmented alphabet recognition task (SEG_SR_ALPH_DBS) after training. Negative weights are displayed by white blobs, positive weights by black blobs. The weights for the "A" model are displayed at the left bottom. "A" is modeled by two state units. Each of these state units gets input from input windows with size 8. The words "B", "C", "D", and "E" are modeled by three states each, etc.

6. Final Results with the ASO Approach (Segmented Data)

The final evaluation of the ASO algorithm on segmented data was carried out on the segmented alphabet recognition task (SEG_SR_ALPH_DBS) and two handwritten character recognition tasks (SEG_OLHR_DIGIT and SEG_OLHR_A_Z). The databases were cut into training data, validation data and testing data. The validation data was used to determine the stopping criterion for the training phase.

Evaluation of the ASO algorithm requires the comparison with manually optimized MS-TDNN architectures. In many of these comparisons, the handtuning was done by myself. To avoid potential bias, the ASO system is also compared with handtuned architectures that were proposed and implemented by other researchers. In retrospective it is hard to quantify the effort for the manual tuning done by myself. The performances of the manual tuned architectures were achieved after at least 20 manual optimization steps (meaning at least 20 training runs for each task, each requiring up to 50 hours processing time on a DEC 5000/200 workstation).

6.1 Experimental Methodology

The following methods were used to determine the performance on the test set:

- The MS-TDNN architecture was trained with standard Back-Propagation [Rumelhart, 1986].
- The network was trained and optimized for a generous number of epochs. Resources were limited according to the function introduced in Section 5.4.3 on page 63. The validation set was run through the network after each fifth epoch. At these intermediate evaluations, the architecture, the weights and the validation performance were stored on disk.
- After training/optimization, the best architecture was selected according to the validation performance.
- The best architecture was loaded from disk and the test data was run through this architecture.

6.2 Segmented Speech Data: Alphabet Recognition

The results on the segmented speech database for both manually optimized architectures and automatically optimized architectures are summarized below (see Table 15 on page 86, Table 16 on page 87). The first table shows the general ability of the ASO algorithm to match the performance of a handtuned system. In the case of Gaussian smoothing of the DTW path (see Section 5.5.6 on page 80), the results are even superior to the manually tuned version (also with Gaussian smoothing).

TABLE 15. Speech Recognition Results (SEG_SR_ALPH_DBS, 2210 training patterns)

	performance training data	performance testing data
manually optimized MS-TDNN architecture with DTW	94.3%	85.0%
manually optimized MS-TDNN with Gaussian smoothing of the DTW path	98.9%	88.0%
automatically optimized MS-TDNN architecture with standard DTW	97.1%	85.0%
automatically optimized MS-TDNN with Gaussian smoothing of the DTW path	99.5%	92.2%

Table 16 on page 87 shows that the system is much more flexible with the ASO algorithm. The MS-TDNN architecture was manually tuned for a training set of 2210 training patterns. Additional training runs with exactly the same architecture were conducted with a reduced training set of 520 and 1040 training examples. All systems were then evaluated by the same test set.

The ASO algorithm was used to optimize the MS-TDNN for all of these three training set sizes. The table shows that the system with ASO is much more flexible because the performances on the “new” training set sizes (520 and 1040 training patterns) are much better. This experiment shows clearly that the ASO algorithm is able the adapt to system to the size of the database.

TABLE 16. Segmented Alphabet Recognition Results Depending on Training Set Size (SEG_SR_ALPH_DBS

number of training patterns	test performance with MS-TDNN architecture manually optimized for 2210 training patterns	test performance with automatically optimized MS-TDNN architecture
520	75.7%	81.5%
1040	79.5%	88.5%
2210	88.0%	92.2%

6.3 Single Handwritten Digits and Letters

The results for the recognition of handwritten digits and capital letters are summarized in Table 17 on page 87 and Table 16 on page 87. Results with different manually optimized architectures (single state TDNNs) are added for comparison.

The results of these tests are especially important since most of the development of the ASO algorithm was done with segmented speech data. The results show that the architectural optimization by the ASO algorithm is equally effective for handwriting data.

TABLE 17. Handwritten Character Recognition Results (SEG_OLHR_DIGIT)

	performance training data	performance testing data
manually optimized MS-TDNN architecture *without* hidden units	98.3%	96.5%
automatic optimization of the window size, 1 state unit per output unit	97.2%	97.0%
automatic optimization of the window size and the number of state units	99.6%	98.0%
automatically optimized architecture with Gaussian smoothing of the DTW path	100%	99.5%
TDNN by [Guyon, 1991] applied to SEG_OLHR_DIGIT	100%	95.5%
TDNN architecture manually optimized by Stefan Manke (University of Karlsruhe, Germany) for SEG_OLHR_DIGIT	100%	98.5%

TABLE 18. Handwritten Character Recognition Results (SEG_OLHR_A_Z) Depending on Training Set Size

number of training patterns	test performance with TDNN architecture manually optimized for 1170 training patterns	test performance with automatically optimized MS-TDNN architecture
520	no convergence	81.5%
1170	88.5%	88.5%
1560	90.5%	91.3%

6.4 Conclusions ASO Algorithm on Segmented Data

The results on three different tasks show that the ASO algorithm can achieve equal or better results than handtuned architectures without any tuning to the particular task.Table 16 on page 87 and Table 16 on page 87 show that the MS-TDNN network optimized by ASO can adapt to different amounts of training data. The handtuned architecture performed equally well for the amount of data that it was optimized for, but did not generalize as well for more data and failed to learn a small subset completely for various learning rates and momentums.

Figure 29 on page 83 shows the connections from the input layer to the state layer after constructing the network with the ASO algorithm for the alphabet recognition task (SEG_SR_ALPH_DBS). 16 letters are modeled by three states, 6 words are modeled by two states and four words are modeled by one state only. The architecture constructed by the ASO algorithm is very heterogeneous in the sense that it would be very time-consuming to find such an architecture by hand.

The results suggest that the ASO algorithm is able to optimize MS-TDNN type networks for real world applications with varying amounts of training data effectively.

The design principles of the ASO algorithm (as described in Section 5.1 on page 52) should also allow the design of automatic structuring algorithms for other tasks. Combinations of the ASO algorithm with Optimal Brain Damage [Le Cun, 1990] did require significantly more processing time and did not lead to better recognition performances. This suggests that the ASO algorithm alone is very robust against overallocation of resources and that no pruning algorithm is necessary.

6.5 Comparison with the Tempo 2 Algorithm

In this thesis, three different approaches to automatic resource allocation for spatio-temporal tasks have been evaluated on the same segmented data so far:

- The Tempo 2 algorithm
- The Tempo 2/TDNN hybrid
- The ASO algorithm

The goal of this thesis is an automatic connectionist algorithm that can reach the same performance as manually tuned state-of-the-art systems. The following sections summarize the results from the last chapters and compare the recognition performances. The most promising algorithm was tested with continuous speech (see Chapter 8 on page 103).

The models that were proposed are very different in design:

- The Tempo 2 algorithm optimizes a connectionist architecture by small gradual changes. A supervised gradient descent method is used to learn important architectural parameters. The internal representation in a network with hidden units is distributed both over units and time.
- The Automatic Structure Optimization (ASO) algorithm is a completely different approach. Instead of slow gradual changes of the architecture, units and connections are added using a more performance-oriented constructive approach. During training, the network starts with a local internal representation that gets more and more distributed the more state units are added.
- The TDNN/Tempo 2 hybrid is in-between these two approaches.

These first three models have been extensively tested on the same data sets. After many training runs the following conclusions can be made:

- The application of the Tempo 2 algorithm with hidden units to large tasks is computationally expensive and suffers from convergence problems. The performance was very good on a small /b/, /d/ and /g/ classification task, but a large network for alphabet recognition suffered from local minima and did not perform as well as the same network without hidden units. Additionally, training took very long with large networks.
- The Tempo 2 network without hidden units did remarkably well. It seems as if the adaptation of time-delays and the widths of the input windows reduces the need for a nonlinear mapping that can only be realized with hidden units.
- The TDNN/Tempo 2 Hybrid with hidden units did even better than the Tempo 2 approach.
- The ASO approach did best on the same task.

These results suggest that starting with a local representation and then slowly changing to a more distributed representation (ASO approach) is more effective than starting with a completely distributed representation from the beginning (Tempo 2 approach with hidden units). The superiority of the ASO approach over the Tempo 2 approach could also be due to the different resource allocation techniques. The Tempo 2 approach is based on a learning rule that does small gradual changes to the network structure while the ASO approach uses a performance oriented decision rule to add connections and units to the network. The superiority of the ASO approach over the Tempo 2 approach conforms with Gemans conclusions that important properties of the task should be build into the architecture and should not be learned in any statistical meaningful way [Geman, 1991]. These results also support a relatively strict constructionist view of brain development where the major structure is established primarily under genetic and developmental control [Kandel. 1981].

Overall, the ASO algorithm seems to be the most promising approach for the application to continuous speech.

7. The Automatic Validation Analyzing Control System (AVACS)

In previous chapters, a constructive algorithm was described that successfully allocates different types of architectural resources. The constructive process is stopped when the performance on an independent validation set decreases. Although the algorithm was shown to construct efficient architectures, the use of a whole validation set to just measure one number (i. e. the performance) can be considered wasteful. If a whole data set is already set apart for evaluation purposes, the result could possibly be analyzed more carefully.

The scenario is similar to a teaching situation in a school: A teacher spends a lot of time teaching a class. At the end of the class, an exam is written. The teacher evaluates the exam and gives grades. If the teacher just looks at the grades, can he improve his own teaching? The grades just tell him something about the intelligence of the students and the overall quality of his own teaching, but the grades do not tell him *which mistakes* were made. Of course, the information is right there in the written exams, it is very easy to detect *what kind of mistakes* the students made and *which question* they could not answer. The exams do not tell the teacher *why* the students could not answer a question correctly, but knowing *which* question they could not answer is certainly preferable to only knowing that they could not answer n% of the questions correctly.

The same thoughts should also hold for artificial learning machines. A learning algorithm should get more information from the validation run than just a single number. Detailed information about *which* mistakes were made is often available from a validation set, so why should it not be used?

The Automatic Validation Analyzing Control System (AVACS) analyzes the output of the system carefully and **controls** the resource allocation of the ASO algorithm with this information (see Figure 33 on page 100). The tools of the AVACS algorithm are described in the next section. The tools are then used to show the reader that overfitting can be detected early in a training run.

7.1 Tools of the AVACS Algorithm: The Confusion-Difference Matrix

The confusion-difference matrix is used to selectively display overfitting. The elements d_{ij} of this matrix are computed as:

$$d_{ij} = \bar{c}_{ij}(train) - \bar{c}_{ij}(validation) \qquad \text{if j <> i}$$

and

$$d_{ij} = -(\bar{c}_{ij}(train) - \bar{c}_{ij}(validation)) \qquad \text{if j=i}$$

where $\bar{c}_{ij}$ are the elements of the confusion matrices normalized by the number of appearances of a particular class in the data. The sign of the elements on the diagonal (j=i) has to be changed because in these cases the $\bar{c}_{ij}$ do represent the right classifications and no misclassifications. The interpretation of the difference matrix is straightforward:

- Small numbers or positive numbers indicate that the network generalizes well on the validation data. This means that the network should also generalize well on the final test set if the validation set is representative for the task. In this case there is no need to limit the allocation of further resources to further increase the performance on the training data.
- Negative numbers indicate that the performance on the validation data is worse than the performance on the training data, which is quite normal depending on the number of effective parameters, the number of training patterns and the noise variance of the data [Moody, 1991]. However, it is possible to detect those classes that generalize worse than other classes.
- Large negative numbers indicate serious overfitting. This information is used by AVACS to selectively decelerate/stop further resource allocation.

Why does AVACS use the confusion-difference matrix? AVACS is designed to **control** the ASO algorithm when too many trainable parameters were allocated. Large negative numbers in the confusion-difference matrix are a direct indicator of overfitting. This class-specific overfitting information can be used to limit resource allocation by the ASO algorithm. Because of the structured approach that is used by ASO to construct the network, it is possible to assign resources to classes or pairs of classes that have to be classified. If the allocation of parameters has led to overfitting (as directly indicated by the confusion-difference matrix) then it is possible to select all resources that are involved in this. In case of serious overfitting it is probably best to stop further resource allocation at all. It is also possible to distinguish between local overfitting (some classes overfit, but not many) and global overfitting (most classes overfit) and control the allocation of resources by ASO appropriately.

Small negative numbers or positive numbers in the confusion-difference matrix are also important information for the ASO algorithm because it can continue (or even accelerate) resource allocation for these classes.

7.2 Application of the Confusion-Difference Matrix

The careful analysis of the validation results is especially important in situations where a complex task has to be learned from a very small database. The complexity of the task requires a complex algorithm with many trainable parameters, but these parameters can not be estimated well due to the small amount of training data. The following examples show such training runs. Just looking at the confusion-difference matrix reveals that

- overfitting problems can be detected easily
- overfitting problems are often not equally distributed over the whole problem. Rather, there are some very local problems.

7.2.1 Typical Training Runs with large Networks and a Small Amount of Training Data

Table 19 on page 94 shows the confusion matrix of a network with many parameters that is trained by a small amount of training data. The network learns almost all of the 520 training patterns within 200 epochs. However, the network does not generalize well. The confusion matrix on the test set is shown in Table 20 on page 95. The performance on the test set is only 77.7%

The use of a table to display the confusion-difference matrices is not very convenient because many numbers have to be read for the interpretation of the matrices. The use of a graphic display (where the value of the elements of the matrix is proportional to the size of little blobs) makes the distribution of the generalization capability much more obvious. Each of the following figures (for example Figure 30 on page 96) shows four different confusion matrices simultaneously:

- the confusion matrix on the training data on the left bottom
- the confusion matrix on the validation data on the right bottom
- the confusion-difference matrix on the upper left
- the confusion matrix on the testing data on the upper right.

TABLE 19. The confusion matrix of a MS-TDNN with a large number of parameters (17164 independent parameters, width of the input windows = 10 frames, 10 hidden units). The network could learn the training data (520 patterns, SEG_SR_ALPH_DBS) much better than shown in this confusion matrix. However, further training would decrease test performance. The confusion matrix on the test data is shown in Table 20 on page 95.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	20																									
B		16		1																	3	1				
C			20																							
D		2		17												1										
E					18																					
F						20																				
G							18										1									
H							1	20																		
I									20																	
J										20																
K				1							20						2									
L												20			1											
M					1								20	1												
N														19												
O															19											
P		1		1												19	1						2			
Q							1										14									
R																		20								
S																			20							
T																	2			20						
U					1																15					
V		1																			2	18	1			
W																						1	17			
X																								20		
Y																									20	
Z																										20

TABLE 20. The confusion matrix of a very large network on the test data (test performance is 77.7%, SEG_SR_ALPH_DBS)

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	15				2				2									1								
B		14		1	1								1			3					3	1	2			
C			18														1		2							3
D		2		10			1	2			3					1						2				
E	1			2	16										1						2					
F		1		1		20										1				1		2				
G							13	3								1	1									
H							4	13																		
I									18						1											
J	2									20																
K											14										1					
L												20													1	
M													14		1											
N	2												4	20	4			1				2				
O															12						1				1	
P		3									1					8						1				
Q			1	1			1	2									13									
R																		18				1				
S																			17							
T				2							2					5	5			19						
U					1																11					
V				1			1									1					2	8				
W				2									1									2	18			
X																								20		
Y															1										18	
Z			1																1			1				17

The interpretation of the figures is simple: For the confusion matrices on training, validation and testing data the size of the black blobs is proportional to the number of confusions. The scaling is such that the full blob represents all appearances of a particular class in the data set. The elements of the confusion-difference matrix can have both positive and negative values. Positive numbers are indicated by black blobs, negative numbers are indicated by white blobs. This means that large white blobs indicate a poor generalization performance since a large negative value for d_{ij} indicates that there were more confusions on the validation set than on the training set.

The confusion matrix on the testing data is only displayed to show the relevance of the confusion matrix of the validation data. All conclusions drawn from the confusion matrix on the validation data are correct if the confusion matrices on validation and testing data are similar or almost equal[1]. The testing-performance on the confusion matrix on the testing data is, of course, **never** used as a stopping criterion or as a criterion for resource allocation.

1. The relevance of the confusion matrix on the validation set is of course dependent on the size of the validation set.

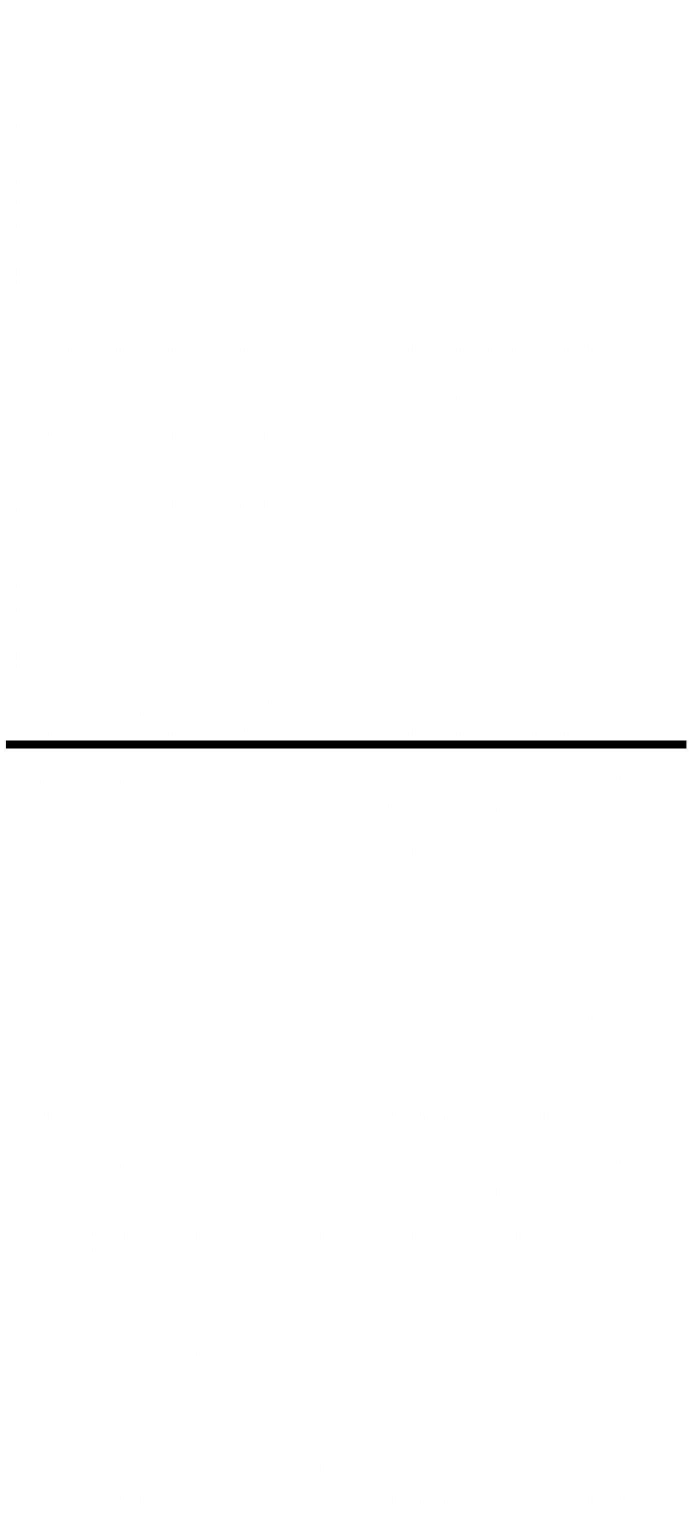

Figure 30. A typical training run of a very large network (with fixed architecture) trained with a small amount of training data (CONNECTED_SR_ALPH_MJMT). The confusion matrices on training, validation and testing data and the confusion-difference matrix are shown after the first epoch (top) and the tenth epoch (bottom). After ten epochs, many of the initial errors still remain, but first learning progress is evident.

Figure 31. Continued from Figure 30 on page 96. The capabilities of the network after 25 epochs (top) and 50 epochs (bottom). Some serious misclassifications remain on the training data. Additionally, the large white blobs in the confusion-difference matrices show that 1.) the network already has some overfitting problems and 2.) that these overfitting problems are not equally distributed over the whole confusion-difference matrix.

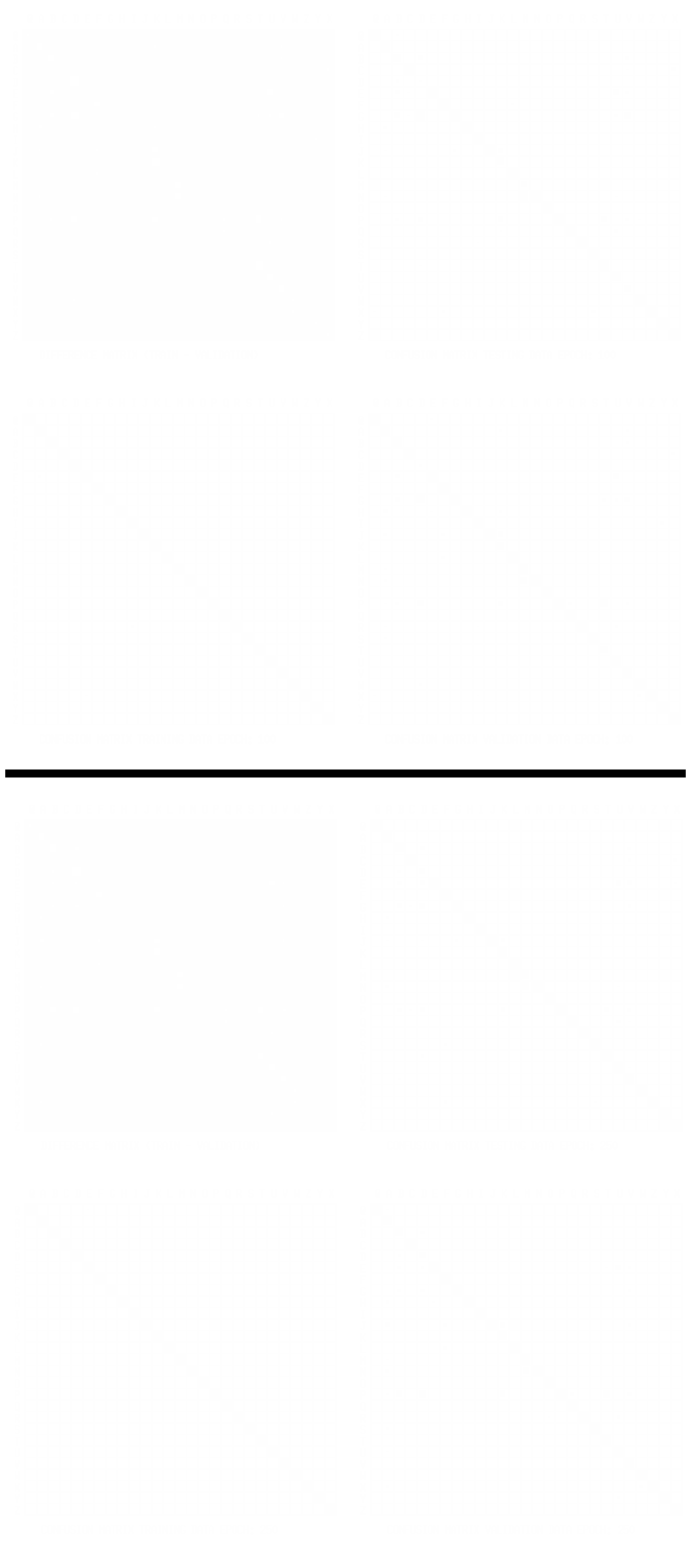

Figure 32. Continued from Figure 31 on page 97. The capabilities of the network after 100 epochs (top) and after 250 epochs (bottom). In both cases, the training data is learned well, but there are still some serious misclassifications on both the validation and testing data. A comparison with Figure 31 on page 97 shows, that the most serious generalization problems (as indicated by large white blobs in the confusion-difference matrix) are the same as after 50 or even 25 epochs (see Figure 31 on page 97).

7.3 The Automatic Validation Analyzing Control System (AVACS)

AVACS monitors the learning and tuning process and is designed to detect poorly generalizing models on a class by class basis as early as possible. A validation set is used to test the generalization ability of the system frequently in the training run. The confusion matrices are computed for both the training and the validation data. From these matrices a new confusion-difference matrix with the elements d_{ij} is computed. The interpretation of the confusion-difference matrix is straightforward:

- Small numbers or positive numbers indicate that the network generalizes well on the validation data.
- Negative numbers indicate that the performance on the validation data is worse than the performance on the training data, which is quite normal depending on the number of effective parameters, the number of training patterns and the noise variance of the data [Moody, 1991]. However, it is possible to detect those classes that generalize worse than other classes. This could indicate four possible problems:

  1.) The ASO algorithm allocated too many parameters.

  2.) The particular model does not fit because the gradient descent training did not reach the global minimum because of initial conditions.

  3.) The particular model does not fit because the architecture of the network does not fit for the task.

  4.) The particular model does not fit because of inconsistent training/validation data. More examples of this particular class are needed for consistent training of the system.

There are many options for recovery from poor generalization. The simplest option is to contaminate all weights of a certain class with a certain amount of noise. This method, although very simple, performed very well in our experiments (with 10 - 30% noise). Changing the weight decay parameter λ is also very simple and effective.

More sophisticated methods for recovery from poor generalization were tried, too. For example, it is possible to change the default order in which resources are allocated (see Figure 17 on page 67). Another option is to completely re-initialize the poorly generalizing parts of the network and to retrain them. In all experiments none of these methods performed better than the contamination with noise. It is probably best to try a certain number of these options automatically and, if none of these helped, tell the user to collect more training data or to accept the current generalization capability. Figure 33 on page 100 shows the complete system of a MS-TDNN that is automatically structured by the ASO module, which itself is controlled by the AVACS module.

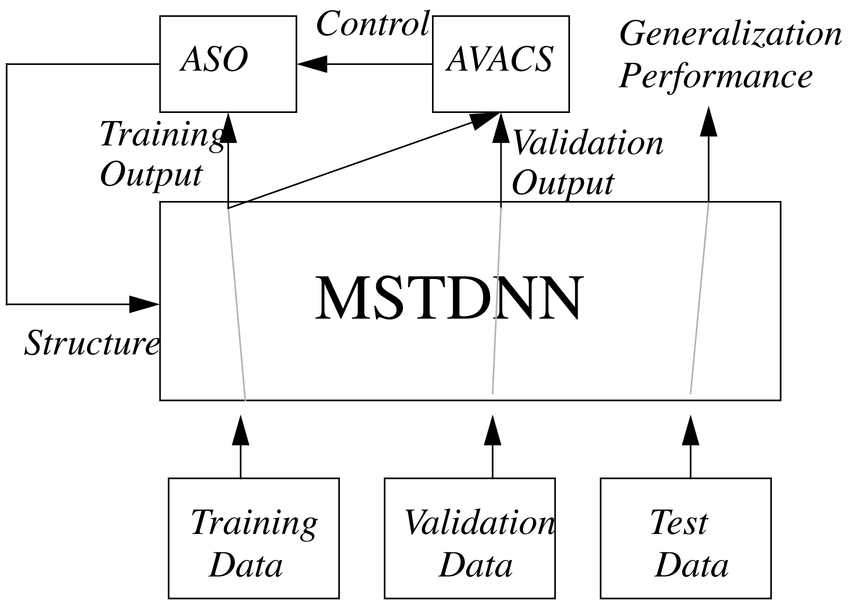

Figure 33. System overview including the ASO and the AVACS modules.

7.4 Performances

AVACS was evaluated on the segmented spoken letter recognition task (SEG_SR_ALPH_DBS) with two training set sizes (520 and 2200 training patterns). Table 21 on page 100 shows the results on this task of a handtuned MS-TDNN (tuned for a training set size of 2200 training patterns), a MS-TDNN optimized by ASO and a MS-TDNN optimized by ASO with AVACS. The results show a small improvement for the large training set (2200 patterns) and a significant improvement for the small training set (520 patterns) using AVACS. These results suggest that AVACS is advantageous when the training set is extremely small for the complexity of the task.

TABLE 21. Alphabet Recognition Results Depending On Training Set Size (SEG_SR_ALPH_DBS). AVACS uses contamination with 25% noise for all weights of a poorly generalizing class.

	test performance (520 training patterns)	test performance (2200 training patterns)
handtuned MSTDNN	75.7%	88.0%
MSTDNN with ASO	81.5%	91.7%
MSTDNN with ASO + AVACS	83.5%	92.3%

7.5 Discussion

In the previous section it was shown that AVACS can improve generalization performance, especially if the database is extremely small for the complexity of the problem. Although the validation set is never **directly** used for architectural optimization, it is used to **control** resource allocation by ASO, which means that it is **indirectly** used for architectural optimization. One might argue that the validation set becomes part of the training set and that its value as an independent test of overfitting is destroyed. This can only be avoided by using two independent validation sets. This may not always be practical. My experiments suggest that the results achieved by AVACS with one validation set are better than the results without AVACS. These results suggest that the advantage from AVACS is greater than the loss of independence of the validation set.

8. Extension to Continuous Speech Recognition Systems

The recognition of continuous speech is considerably harder than the recognition of isolated words. There are two reasons: First, word boundaries are typically not detectable in continuous speech. This results in additional confusable words and phrases (for example “youth in Asia” and “Euthanasia”) as well as a larger search space. The second problem is that there is much greater variability in continuous speech due to stronger coarticulation (or inter-phoneme effects) and poorer articulation (“did you” becomes “didja” etc.).

In contrast to continuous speech, connected speech is usually limited to small vocabularies and single sentences, but features the same difficulties (coarticulation/articulation effects, unknown word boundaries, and search space). The recognition of connected letters is a very demanding task for automatic speech recognition systems because strings of spelled letters are highly confusable. Even human beings have often difficulty to understand strings correctly if no domain knowledge can be used to repair recognition errors. The high confusability of spelled letters has often led to the use of keywords for certain letters (“beta” for “b” or “delta” for “d” etc.). The high confusability of continuously spelled letters makes this task a suitable candidate for the final evaluation of the ASO algorithm. As it is a non-trivial task for state-of-the-art SR systems [Hild, 1993], [Hild, 1993b], advanced architectural optimization is necessary for best possible performance and it should be a challenge for an automatic optimization algorithm.

8.1 One-Stage Dynamic Time Warping

The recognition of connected words requires two operations:

- nonlinear time-alignment
- recognition.

In case of connected letters, each letter is treated as a word, and a sequence of spelled letters is treated as a sentence. The system developed for the recognition of segmented spoken letters or single written characters already performs nonlinear time-alignment and recognition simultaneously, but does not incorporate word boundary detection as the individual words handsegmented from the continuous string. The extension to connected letter recognition can be efficiently done by the use of the one-stage Dynamic Programming algorithm [Ney, 1984]. The search for the optimal path is constrained by two types of transition rules: transition rules inside the word (letter) and across word (letter) boundaries. Carrying out the optimization using these path constraints and dynamic programming is done simultaneously, hence the name "one-stage" algorithm.

A detailed description of the one-stage algorithm is omitted as the application of Multi-State Time Delay Neural Networks to connected letter recognition can be found elsewhere (see [Haffner, 1991a], [Haffner, 1992a], [Haffner, 1992b], [Hild, 1993a], [Hild, 1993b]) and is not the primary topic of this thesis. However, it provides a state-of-the-art system, a challenge for performance comparison of the automatic structuring algorithm developed in this thesis. The application of the ASO algorithm to connected letter recognition is simple and straightforward: The confusion matrices and confusion-symmetry matrices are computed for letters that are labeled for the first training epochs (bootstrapping, see [Haffner, 1991a], [Haffner, 1992a], [Haffner, 1992b], [Hild, 1993a], [Hild, 1993b]). The results presented in this chapter suggest that no further extension of the ASO algorithm to connected letter recognition is necessary.

8.2 Substitutions, Insertions and Deletions

Continuous speech recognizers are harder to evaluate than isolated word recognizers because insertion and deletion errors can occur in addition to substitution errors which can occur in both systems. Typical examples of these three classes of errors are shown in the next sections. The reference string is shown in the first row and the hypothesis of the speech recognizer is shown in the second row:

8.2.1 Substitutions

A typical substitution error of automatic letter recognizers is shown in the first example. The letter "b" is recognized as a "d":

REF: v a g a B o n d

HYP: v a g a D o n d

8.2.2 Insertions

The following example shows an insertion error: The speech recognizer adds a letter at the end of the sentence:

REF: h h j p q *

HYP: h h j p q O

8.2.3 Deletions

A typical deletion error is shown in the next example. The speech recognizer does not recognize the second “r” in the sentence:

REF: c a r R y

HYP: c a r * y

8.2.4 The Word Accuracy

The final performance of the system is computed by a small program which compares both the reference file and the hypothesis file and computes the percentage of each type of error. The “word accuracy” wa is computed as follows:

$$wa = 1 - \left(\frac{sub + del + ins}{N}\right)$$

where sub is the number of substitutions, del is the number of deletions, ins is the number of insertions and N is the number of words (letters) in the database.

8.3 Duration Constraints

The search of the optimal path by the one-stage algorithm can be constrained in many ways. Duration constraints for states or words (letters) are simple and can improve the recognition performance significantly. Simply put, these duration constraints allow the search algorithm to stay in one state for a minimal or maximal number of frames. Minimum constraints are very effective to avoid insertions, which otherwise appear frequently in connected letter recognition (consider the sequence “B - E - E - E”). Since optimal duration constraints greatly depend on the number of states, it is necessary to adjust the duration constraints automatically because the number of states is optimized by the ASO algorithm. A very elaborate technique that models the duration of the phonemes by a Gaussian distribution has been developed by Hild [Hild, 1993a], [Hild, 1993b], but the following simple technique works very well, too: The average duration a_i of each letter i is computed from the training data. The minimum state duration m_i is computed by

$$m_i = int\left(\frac{a_i}{s_i \cdot const}\right)$$

where $const$ is an empirical constant factor, int rounds the value of the quotient to an integer, and s_i is the actual number of states that are used to model a letter. A good value for $const$ was found empirically by using a network with a fixed architecture and running the validation set (100 sentences) through the system. The results of these experiments are summarized in Table 22 on page 106. The table shows that the number of deletions increases significantly for $const \leq 1.2$ (leading to large minimum durations m_i). Insertions are totally avoided in this case. The number of insertions grows significantly for $const \geq 1.4$ (leading to small minimum durations). The range in-between leads to a good balance between deletion and insertion errors.

TABLE 22. Connected letter recognition results (CONNECTED_SR_ALPH_MJMT) of a fixed architecture depending on duration constraint constant. The constant was set to 1.25 in the training run and the validation set was used to determine the stopping criterion.

const	% Substitutions	% Deletions	% Insertions	% Word Accuracy
1.0	2.4	11.8	0.0	85.5
1.1	1.8	3.2	0.0	94.9
1.2	1.5	1.8	0.1	96.6
1.25	1.9	1.4	0.3	96.4
1.3	2.0	1.1	0.4	96.5
1.4	1.6	0.9	0.7	96.8
1.5	1.8	0.9	0.9	96.4
1.6	2.0	0.8	1.8	95.4
2.0	2.3	0.6	4.1	93.0

It is also possible to use constant minimum durations m_i. The results for the handtuned architecture[1] are similar to the results presented in Table 22 on page 106 and are shown in Table 23 on page 107. For small minimum durations ($m_i \leq 3$) the high number of insertion errors reduces the word accuracy significantly. For large minimum duration constraints ($m_i \geq 6$) the large number of deletion errors reduces the word accuracy. The optimum is in-between.

1. The number of states of each letter does not vary that much in the handtuned architecture. Silence was modeled by two states and all other letters were modeled by four states.

TABLE 23. Connected letter recognition results (CONNECTED_SR_ALPH_MJMT) of a fixed architecture depending on a fixed minimum duration. The minimum duration was set to 4 in the training run and the validation set was used to determine the stopping criterion.

m_i	% Substitutions	% Deletions	% Insertions	% Word Accuracy
1	2.9	0.4	11.3	85.4
3	2.7	0.6	4.0	92.7
4	2.4	0.9	0.9	95.8
5	1.8	1.6	0.2	96.4
6	1.4	4.6	0.0	93.9
8	3.2	32.0	0.0	64.8

8.4 Experimental Methodology

There are several limitations when a database of this size and complexity is used for training. In the database there is one sentence per file. The files of such databases are usually not directly on the disk of the machine that is used for training. Rather, they have to be read from disks on other machines via the local data network. If this is done frequently (i.e. each epoch), the processing time is increased significantly. In this case it is advantageous to load the whole database into the memory once due to speed increase. This reduces the memory available for the speech recognizer and the freedom for the allocation of architectural resources. In the following training run, the resources were limited due to memory constraints in the following way:

- The number of hidden units was limited to 25. This seems to be a rather low upper bound compared to other MSTDNN implementations ([Haffner, 1991a], [Haffner, 1992a], [Haffner, 1992b], [Hild, 1993a], [Hild, 1993b]), but the direct connections from the input to the state units reduce the required number of hidden units because there is a shortcut between the input and the state units and not all information has to be pumped through the hidden units. Figure 36 on page 113 shows an example where 9 hidden units were allocated. In all training runs up to 21 hidden units have been allocated. This means that 25 hidden units is a generous upper limit which does not affect the behavior of ASO.
- The width of all input windows was limited to 10 frames, which responds to a temporal context of 100 ms. Although the ASO algorithm did sometimes allocate larger windows for other tasks (SEG_SR_ALPH_DBS), 100 ms of temporal context should be really sufficient. It was never observed that the generalization performance changed significantly when the window width was also limited to 10 frames on other speech tasks (SEG_SR_BDG, SEG_-SR_ALPH_DBS).
- The number of states was limited to four states per letter. This may sound like a s serious limitation, especially for long letters like “W”. It should be noted that together with adaptive input window size of state and hidden units there are many possibilities to cover “long” letters like “W”.

The above limitations avoided swapping on the available machines (DEC 5000/200 with 24 MB of memory) and allowed reasonable processing times. The architecture shown in the following figures was optimized by ASO without AVACS. As mentioned before, AVACS is not necessary if such a large database as CONNECTED_SR_ALPH_MJMT is available for training.

The following methods were used to determine the word accuracy:

- The MS-TDNN architecture was trained with standard Back-Propagation [Rumelhart, 1986].
- The network was trained and optimized for a generous number of epochs. Resources were limited according to the function introduced in Section 5.4.3 on page 63. The validation set was run through the network after each fifth epoch. At these intermediate evaluations, the architecture, the weights and the validation performance were stored on disk.
- After training/optimization, the best architecture was selected according to the validation performance.
- The best architecture was loaded from disk and the test data was run through this architecture.

8.5 Performance

Table 24 on page 108 shows a comparison of the recognition results on speaker dependent connected letter recognition. 500 sentences were used for training, 100 sentences for validation and 400 sentences for testing. A MS-TDNN architecture handtuned by myself reaches 96.8%, the same system tuned by ASO reaches 97.4% and a MS-TDNN system developed by Hermann Hild reaches 97.5%. See “Evaluation” on page 119 for further discussion of these results.

TABLE 24. Performance comparisons on the connected spelled alphabet recognition task (500 training sentences, 100 validation sentences, and 400 test sentences, CONNECTED_SR_ALPH_MJMT).

System description	Word Accuracy
MSTDNN manually optimized by Hermann Hild at UKA, trained with phoneme labels, handtuned number of states per phoneme, number of hidden units and window sizes, Gaussian modeling of duration constraints	97.5%
MSTDNN manually optimized by myself, trained with letter labels only, handtuned number of states per letter, number of hidden units and window sizes, simple duration control	96.8%
MSTDNN optimized by ASO, trained with letter labels only, automatic optimization of the number of states per letter, number of hidden units and window sizes, simple duration control	97.4%

8.6 A Typical Training Run with the ASO Algorithm

The following figures show the architecture that was optimized by the ASO algorithm for the full training set (500 training sentences) of the connected letter recognition task (CONNECTED_-SR_ALPH_MJMT). Figure 34 on page 111 and Figure 35 on page 112 show the widths of the input windows, the number of states and all weights for the direct connections from the input to the state units. As can be seen, the structure optimized by ASO is very complex. The number of states varies from one (letter “R”) to four (for example the letters “A”, “B”, “D” etc.). The values of the weights are indicated by the size of the little blobs. A good example for the effect of unequal training can be seen at the fourth state of letter “D” or the first state of letter “S”: The blobs are very small compared to the other blobs, indicating that these weights have not been trained a lot.

The number of states allocated for letter “W” is surprising, too. Normally, one would expect that a letter with such a long spelling would require many states. The ASO algorithm allocated only three states with comparatively small input windows. A look at the confusion matrices (see Figure 40 on page 117) shows that the letter “W” is really no problem in the training data. The confusion matrix on the validation data shows that the network occasionally classifies an “A”, B” or “D” as a “W”, which are reasonable confusions[1] considering the pronunciation of “W”. In the test set, only confusions with “D” occur. Figure 38 on page 115 shows that the weights from the hidden units to the state units representing letter “W” are very strong. This means that the letter “W” makes use of the non-linearities provided by the hidden units. However, the input windows from the hidden units to the state units representing “W” are very small (only one frame), indicating that “W” was not often involved in serious pairwise confusions.

Figure 37 on page 114 and Figure 38 on page 115 show the weights from the hidden units to the state units. Interestingly, many state units have learned rather strong connections to most of the hidden units, especially if their input window (to the hidden units) is very small. Classes with large input windows for these connections are typical candidates for pairwise confusions, for example “B”, “D”, and “E” or “A” and “H”.

Figure 36 on page 113 shows the weights from the input to the 9 hidden units. Hidden unit number 8 and 9 were allocated shortly before the best performance on the validation set was reached. All input weights to these hidden units are very small (as indicated by small blobs) because the training error was already very low at the time when they were installed. In contrast, the hidden units 1 to 7 were allocated very early when the error (and the error gradient) was very large. Interestingly, the size of the input window of the first hidden unit is only one frame, which indicates that this hidden unit was installed for a pairwise confusion that could be repaired by such a small input window.

1. The results shown were obtained with architectural optimization by the ASO algorithm only. The AVACS module could probably have solved these confusions. AVACS was not used because I thought it would not make much difference for such a large database (500 training sentences).

Overall, Figure 34 on page 111, Figure 35 on page 112, Figure 36 on page 113, Figure 37 on page 114, and Figure 38 on page 115 show that

- the ASO algorithm constructs a rather heterogeneous architecture, meaning that the sizes of the input windows, the number of states, and the number of hidden units can vary considerably for each letter.
- the allocation of resources is sometimes intuitively reasonable, but sometimes not.
- it is hard to display and interpret a network with more than 20,000 independent parameters.

Figure 34. The direct weights from the input to the state units for the state units representing “@” (silence), “A” to “M”. Each of the boxes represents all ingoing weights of a state unit. The index of the 16 spectral coefficients is on the vertical axis of each box. The width of the input window is given by the horizontal width of the boxes. The values of the weights are displayed by small blobs inside the boxes. The value of a weight is proportional to the size of these little blobs. Positive weights are displayed by black blobs, negative weights by white blobs.

Figure 35. The direct weights from the input to the state units for the state units representing “N” to “Z”. See Figure 34 on page 111 for explanation.

Figure 36. The input weights of the hidden units. The index of the 16 spectral coefficients is on the vertical axis of each box. The width of the input window is given by the horizontal width of the boxes. The values of the weights are displayed by small blobs inside the boxes. The value of a weight is proportional to the size of these blobs. Positive weights are displayed by black blobs, negative weights by white blobs.

Figure 37. The weights from the hidden units to the state units “@” to “M”. The index of the hidden units is on the vertical axis and the width of the window is given by the horizontal width of the box.

Figure 38. The weights from the hidden units to the state units “N” to “Z”. See Figure 37 on page 114 and Figure 34 on page 111 for explanation.

Figure 39. The confusion matrices on training data (left bottom), validation data (right bottom), testing data (right top) and the confusion-difference matrix on the left top of a connected letter recognizer tuned by the ASO algorithm after 10 epochs of training. The confusion matrices show that the network performs surprisingly well for the low number of training epochs. The confusion matrices show that the misclassification errors are not equally distributed over the whole matrices. The big white blobs in the confusion-difference matrix show that the network generalizes poorly. The reason is that many misclassifications are not the same in the training data and the validation data, although the performances are very similar (79% and 71%). The confusion matrices at the end of the training run are shown in Figure 40 on page 117.

Figure 40. The confusion matrices on training data (left bottom), validation data (right bottom), testing data (right top) and the confusion-difference matrix on the left top of a connected letter recognizer tuned by the ASO algorithm after training. The confusion matrices are almost perfect. The ASO algorithm constructed an architecture that generalizes equally well for all classes.

9. Evaluation

As stated in the introduction, the algorithms developed in this thesis are evaluated under the following criteria:

- Suitability for small systems (~ 1,000 parameters) as well as for large systems (more than 10,000 parameters): Is the proposed method efficient for various sizes of the system?
- Ease of use for non-expert users: How much knowledge is necessary to adapt the system to a customized application?
- Final performance: Can the automatically optimized system compete with state-of-the-art well engineered systems?

The following sections discuss the ASO/AVACS combination under these criteria.

9.1 Performance Comparison with other Systems

Performance comparisons of the ASO system with handtuned systems have already been discussed in chapters 6 and 7. The ASO algorithm achieved similar or better recognition results as handtuned systems. In many of these comparisons, the handtuning was done by myself. To avoid potential bias, the ASO system is also compared with handtuned architectures that were proposed and implemented by other researchers and that represent state-of-the-art systems.

The first benchmark is the classification of single on-line handwritten digits (SEG_OLHR_-DIGITS). The data set was recorded in summer 1991 at IKA in the same way as described in [Guyon, 1991]. The TDNN architecture proposed by Guyon et al. was implemented by Stefan Manke from UKA and achieved a performance of 95.5% on the test data. Manke improved the performance to up to 98.5% on the test set by his own manual tuning [Manke, 1992]. This tuning was necessary because the original architecture from Guyon et al. was developed for the recognition of all single on-line characters (digits, lower case letters, and upper case letters) and was trained with a larger database (12,000 characters from 250 writers instead of 780 digits from approximately 50 writers). The result after manual tuning of the architecture was 98.5%. The ASO system achieved 99.5% under the same conditions. All results are summarized in Table 25" on page 120.

TABLE 25. Handwritten On-Line Character Recognition Results (SEG_OLHR_DIGIT)

System	Performance Testing Data
TDNN by [Guyon, 1991], implemented by Manke (UKA) on SEG_OLHR_DIGIT [Manke, 1992]	95.5%
TDNN architecture manually optimized by Manke (UKA) in summer 1992 for this data base [Manke, 1992]	98.5%
Automatically optimized architecture with Gaussian smoothing of the DTW path	99.5%

The ASO system was also compared with a system developed by Haffner and Hild for speaker dependent connected alphabet recognition task [Haffner, 1991a], [Haffner, 1992a], [Haffner, 1992b], [Hild, 1993a], [Hild, 1993b]. Some of the recent improvements like sentence level training were switched off to allow for comparable experimental conditions[1]. However, some differences between the systems remain:

- The MS-TDNN with fixed architecture was trained with labels for phonemes. This is possible because the actual number of phonemes per letter is known at the beginning of the training run. It is not possible to use these labels for the training optimized by the ASO algorithm because the ASO algorithm changes the number of phonemic states per letter significantly during training.
- Hermann Hild from UKA developed a better duration control based on Gaussian modeling of letter/phoneme durations. This improved duration modeling was not included in the ASO system.

Both systems were trained, validated and tested with exactly the same data sets. The results are summarized in Table 26" on page 121. The MS-TDNN system by Hermann Hild performs comparably (97.5% vs. 97.4%) as the ASO-MSTDNN system. The small difference between these two results is surprising given the fact that

1. The performance of Hermann Hild's MS-TDNN system with sentence level training is 98.5% on the test set (speaker MJMT).

- 1.) no phoneme labels were used
- 2.) a simpler duration control was used
- 3.) the great amount of manual tuning that was applied to the other system over several years.

TABLE 26. Performance comparisons on the connected alphabet recognition task (500 training sentences, 100 validation sentences, and 400 test sentences) under comparable conditions.

System Description	%Word Accuracy
MSTDNN manually optimized by Hermann Hild at UKA, trained with phoneme labels, handtuned number of states per phoneme, number of hidden units and window sizes, Gaussian modeling of duration constraints	97.5%
MSTDNN optimized by ASO, trained with letter labels only, automatic optimization of the number of states per letter, number of hidden units and window sizes, simple duration control	97.4%

9.2 Suitability for small and large Systems

The ASO algorithm was developed and tested with classification tasks with 10 (on-line handwritten digits) to 27 classes (connected letter recognition) so far. The smallest system was initialized with 10 state units, no hidden units and an input window of one frame, each consisting of 8 features. This makes a total of ((8 * 10) + 10) = 90 independently trainable weights for the initial architecture[1]. After optimization of the architecture, the average architecture had an average number of 2.6 states per digit, 3.4 hidden units and a window width of 13.2 for the direct weights from the input to the state units, a width of 8.7 for the windows from the input to the state units, and an average width of 7.8 for the weights from the hidden units to the state units. The average number of independently trainable parameters was 3956 in 10 different optimization runs. This was the smallest problem that the ASO algorithm was applied to so far.

When the ASO algorithm was applied to the connected letter recognition task with 27 classes (26 letters plus silence), the system was again initialized with one state per letter, no hidden units and input windows of one frame, each consisting of 16 features (spectral coefficients). This makes a total of ((27 * 16) + 27) = 459 independently trainable connections. In an average of three optimization runs, the ASO algorithm constructed networks from 17,000 connections to 26,000 connections, with an average of 19,824 connections. This was the largest problem that the ASO algorithm was applied to so far.

1. The bias weights are left out in this comparison for simplicity.

For an evaluation of the efficiency of the ASO algorithm it is necessary to compare the total computational effort for both optimization of the architecture and the training of the weights with the effort for training of an already handtuned architecture. This comparison can only be based on the total computation time on a certain machine. The number of training epochs is not useful, because the ASO algorithm starts with a small number of trainable parameters. This means that each training epoch at the beginning of the training run is much shorter, but also more training epochs are required. Table 27" on page 122 shows the relative computational effort for both optimization and training with ASO compared to the training of a handtuned architecture (which is assumed to be known). The relative computational effort varies from 146% for the classification of single on-line handwritten digits to 122% for connected spoken letter recognition. Fortunately, the relative computational effort is high (146%) for the smallest task and lower (122%) for the largest task. This could be explained by the reduced effort per epoch at the beginning of a training run in case of constructive learning. However, the relative effort for the first epoch is very similar in both cases. For example, in the first epoch of the handwritten digits task only 90 of the approximately 4000 connections have to be computed/updated:

$$\frac{90}{4000} = 0.0225$$

In case of connected letter recognition, only 459 instead of approximately 20,000 connections have to be updated:

$$\frac{459}{20,000} = 0.02295$$

These two numbers are very close together and can not explain the lower relative computational effort for the large task. However, it is also important to note how fast the resources were allocated in a certain training run. Unfortunately, the actual number of connections at each training epoch was not stored and no further analysis is possible.

TABLE 27. The relative computational effort for the optimization by the ASO algorithm and training compared to training of a handtuned architecture. The effort for the handtuned architecture (assumed to be known in advance) is set to 100%

Task	Number of simulation runs	Relative computational effort under the assumption that the best architecture is known
single on-line handwritten digits	10	146%
single on-line handwritten capital letters	5	138%
segmented spoken alphabet recognition	5	133%
connected spoken alphabet recognition	3	122%

How efficient is the ASO algorithm? The total computational effort for both optimization and training is always greater than the effort for training of a known optimal handtuned architecture. This is not surprising because in this comparison it is assumed that the optimal handtuned architecture is already known, which hardly ever is true. How many training runs does an experienced engineer need to find an equivalent architecture? The exact answer to this question is beyond the topic of this thesis. Many experiments with human subjects would be necessary to find an answer. However, observations of colleagues suggest that the number of trials can grow as large as 50, even if the training runs need several days of computation time[1]. Such a high number of trials is only feasible in a research environment and not in a development environment. If we assume that a well-trained engineer needs 5 trials to develop a good architecture and that each trial requires the same computational effort as the best architecture, then the comparison strongly favors the ASO algorithm (see Table 27" on page 122).

TABLE 28. The relative computational effort for the optimization by the ASO algorithm and training compared to training of a handtuned architecture assuming that 5 trials are needed by the human developer to develop an equivalent architecture.

task	relative computational effort of ASO optimization/training under the assumption that the human developer needs 5 trials to develop an equivalent architecture
single on-line handwritten digits	29.2%
single on-line handwritten capital letters	27.6%
segmented spoken alphabet recognition	26.6%
connected spoken alphabet recognition	24.4%

These results (together with the performance comparisons, see above) suggest that the ASO algorithm can efficiently optimize the architectures for a given task.

9.3 Ease of Use for non-Expert Users

How much knowledge is necessary to adapt the system to a customized application? To answer this question, the following two questions have to be discussed:

- Is the ASO algorithm easily implementable?
- Is the optimization by the ASO algorithm robust against the choice of internal (ASO) parameters?

1. No doubt, the implementation of the ASO algorithm is additional work that has to be invested. Most of this work has to be invested in the implementation of variable numbers of hidden units and state units, variable widths of the input windows for the connections from the input to the state units, from the input to the hidden units and the hidden units to the state units. However, the computation of confusion, confusion-symmetry, and confusion-difference matrices as well as computation of the allocation criteria is simple and straightforward. No complex operations like matrix inversions are necessary.

1. The MS-TDNN system that was used for performance comparison (see Table 26" on page 121) was probably trained more than 50 times. However, no detailed records are available.

2. It was already shown in Section 5.4.3 on page 63 that the ASO algorithm is robust against the choice of the resource limiting function $f(p, n)$. Several functions with different internal constants yield similar generalization performances (see Table 9” on page 64 and Table 10” on page 65). This may be explained by the positive effect of unequal training (see Section 5.5.5 on page 80). Resources that are allocated very late in a training run are not trained by large error gradients any more. In contrast, their random initialization may neutralize already existing signs of overfitting. That suggests that in constructive learning the exact number of parameters is less important than the point in time in the training run when a resource was allocated. It is obvious that this point in time is not as dependent on the resource limiting function as the actual number of parameters. This may explain the robustness of the ASO algorithm to different resource limiting functions.

It should be mentioned that there are some cases when knowledge of the user is advantageous: In some applications the allowable upper limits for certain resources due to memory constraints may be a severe problem. This can partly be avoided by a sophisticated dynamic memory allocation, which increases the implementation effort. However, the ASO algorithm is designed for customized applications (see Section 1.2 on page 5) for a limited domain[1]. Resource allocation should be limited by the functions from Section 5.4.3 on page 63 and not by the available resources of the computer. If it is necessary to set tight upper bounds for certain resources due to severe memory constraints, some knowledge is advantageous to weight the upper limits of the different types of resources appropriately.

The overall answer to the question “How much knowledge is necessary to adapt the system to a customized application?” is: Implementation of the ASO algorithm is straightforward, but not trivial. If it is implemented and if it is not constrained by other limitations (like severe memory constraints which require the setting of tight upper bounds for certain resource types) it does not require any knowledge about the application.

9.4 Other Advantages

In the previous sections, the ASO algorithm has been discussed and evaluated under the criteria that were put forward in the introduction. ASO was extended by AVACS (see Chapter 7 on page 91) The ASO/AVACS combination also offers additional advantages:

9.4.1 Equalization of Misclassifications/Generalization Ability

As pointed out before, the misclassifications of real-world classifiers are hardly ever uniformly distributed across the whole confusion matrix (see Section 5.3 on page 56). This particular “feature” was used for the design of the ASO algorithm, and is also implicitly used by AVACS[2]. ASO uses an unequal distribution of misclassifications as a criterion for the allocation of new resources.

1. ASO is not designed to optimize a 5,000 word vocabulary speaker independent continuous speech recognizer to fit on a 640 KB PC. Rather, the goal is to achieve the best possible performance on a limited customized domain with little consideration of the memory requirements.

2. AVACS uses the “feature” that the generalization capability is not equally distributed across the whole confusion-difference matrix.

An analysis of the confusion matrices after training/optimization shows that this method tends to balance the misclassifications across classes as well as the generalization ability. This is a new and highly desirable feature of the ASO algorithm, which is even amplified by AVACS. Adjustment of the resource limiting function for each class could even allow control of the distribution of misclassifications.

In this context it is interesting to investigate whether the popular weight decay (see Section 2.4.3 on page 21) offers the equalization of generalization ability, too. Simulations of fixed architectures with and without weight decay (see Section 2.4.3 on page 21) show that weight decay can improve generalization, but does not balance the generalization ability across classes (see Figure 41 on page 126).

A comparison of the confusion-difference matrices of a fixed architecture (tuned manually) trained with weight decay and an adaptive architecture optimized by the ASO/AVACS combination (both computed at the peak of the validation performance) is shown in Figure 42 on page 127. A comparison of both confusion-difference matrices shows that the automatically optimized architecture generalizes better and that it does have fewer serious generalization problems than the fixed architecture (that are indicated by large white blobs in Figure 42 on page 127).

Figure 41. A comparison of the confusion-difference matrices of fixed architectures without weight decay (left column) and with constant weight decay of 0.0001 (right column). The architecture was rather generous for the given number of training patterns (520 training patterns from SEG_SR_ALPH_DBS) to explore the benefit of weight decay. The run without weight decay reached the best performance on the validation set after 150 epochs. Test performance was 71.9%. The run with weight decay performed exactly the same at 200 epochs of training, but reached an even better performance of 74.7% on validation data and 73.1% on test data after 385 epochs. A comparison shows that 1.) training with weight decay can take much longer but 2.) it can improve the generalization performance and 3.) that weight decay does not change the distribution of errors. It does not equalize the generalization ability.

Figure 42. A comparison of the confusion-difference matrices for a constant architecture trained with weight decay ($\lambda = 0.0001$) on top and an architecture optimized by ASO with AVACS (AVACS added 20% noise for all connections that are involved in poorly generalizing models) after training with 520 patterns from the segmented spoken alphabet database. The fixed architecture was adapted for the number of training patterns manually and reached the best performance on the validation set after 385 epochs, test performance was 75.7%. The architecture optimized by ASO/AVACS combination reached the best validation performance after 175 epochs. Test performance was 83.5%. The confusion-difference matrices show that 1. the system optimized by ASO/AVACS generalizes better; 2. that there are fewer serious generalization errors (that are indicated by large white blobs); and 3. that the errors are better distributed.

10. Summary

The successful application of speech recognition (SR) and on-line handwriting recognition (OLHR) systems to

- new domains
- differing task complexities
- new recognition modalities

greatly depends on the tuning of the architecture to each new task, especially if the amount of training data is small. In this thesis I have developed and evaluated methods that allocate connectionist resources (connections, time-delays, hidden units, state units) automatically and avoid manual optimization of the architecture. These methods are especially interesting for the flexible application of speech recognition (SR) and on-line handwriting recognition (OLHR) systems to customized applications with a small amount of available training data.

Initial experiments were conducted with the Tempo 2 approach (see Section 4 on page 37), which was originally developed as a neurophysiologically motivated model of cognitive resource allocation. Initial experiments on a small phoneme classification task were promising, but larger tasks revealed scaling problems of this approach.

In the next step, the Tempo 2 approach was adapted and a hybrid of the Time-Delay Neural Network (TDNN) and the Tempo 2 approach was developed (see Section 4.6 on page 49). Although results were promising, a new algorithm motivated by human manual tuning strategies was developed. The concept behind this strategy can be summarized as follows: If the classifier misrecognizes class x out of n classes, could this problem be cured by additional resources for class x?

This algorithm, the *Automatic Structure Optimization* (ASO) algorithm was indeed a big improvement from the previous algorithms under the evaluation criteria of this thesis, namely

- suitability for small systems (~ 1,000 parameters) as well as for large systems (more than 10,000 parameters)
- ease of use for non-expert users
- competitive performance with state-of-the-art systems.

The ASO algorithm described in Section 5 on page 51, fulfills these criteria very well (see Section 9.1 on page 119, Section 9.2 on page 121, and Section 9.3 on page 123). In addition to the ASO algorithm, a new algorithm was developed that controls the optimization steps of the ASO algorithm. This algorithm, the *Automatic Validation Analyzing Control System* (AVACS) was motivated by modern human teaching strategies. The concept behind this strategy can be summarized as follows: If the average student can not solve question x, could it be that there was something wrong with the teaching of how to solve question x? This concept was shown to be also successful with an artificial learning system. As in real life it is not always true that the learning machine is “just too stupid”, sometimes it is the teaching method which has to be modified. AVACS controls ASO and accelerates/decelerates resource allocation for each class independently.

The combination of ASO with AVACS was shown to be a reliable tool for the automatic optimization of SR and OLHR systems. The criteria that were put forward at the beginning of this thesis were fulfilled. In addition the ASO/AVACS combination was shown to offer the property of “generalization balancing across classes”, meaning that the distribution of generalization errors is changed and fewer serious gerenalization mistakes occur compared to fixed architectures. The architectures constructed by ASO/AVACS are very heterogeneous compared to most manually optimized architectures. Resources are only allocated where they are actually needed and where they can be trained reliably by the available amount of training data.

10.1 Significance of this work

Despite the aim to develop a general purpose, speaker independent, very large vocabulary, spontaneous speech recognizer there is a considerable number of applications that require the best possible performance on small, well defined, and customized domains. For these applications, manual tuning of the architecture (i. e. optimizing the number of states, the number of hidden units and the width of the input windows) is too costly and not tolerable because each application requires its own optimization. Many of todays speech recognition systems are very powerful, but the complexity of these systems is such that these systems ar far from being intuitive and easy to use for the developer. In general it is not possible to develop applications on top of the technology without understanding the underlying speech algorithms. This makes quick prototyping impossible, which is very important for the creation of new products and services. Developers of software that includes speech recognition should not be required to invest months or years in an understanding of details of speech recognition technology or in the tuning of these systems.

The current situation of speech technology is rather peculiar: Complex systems have been developed, but the final user is demanding even more general systems (with a much larger vocabulary, even higher word accuracy and less restrictions concerning speaking style) and developers of customized applications are demanding simpler systems that are much easier adaptable to small domains and allow quicker prototyping. The major research effort currently aims at general purpose systems. This thesis presents an important step in the direction of easily adaptable, customized systems. I will explain in the next sections why automatic optimization algorithms, which free developers from internal details of speech algorithms and their time-consuming tuning, are an important bridge towards greater distribution of speech recognition technology.

10.2 Why is this important?

As mentioned before, speech recognition technology has recently made significant advances towards an important breakthrough. However, the general purpose systems are currently not general enough and the specialized systems are generally far from being easily adaptable. The result is that end users do not accept the current off-the-shelf general purpose systems and that developers of customized applications do not dare to include speech recognition into their products.

How can this situation be improved? Further development of general purpose systems demands a considerable effort and expertise in research (acoustics, search, language, ...) as well as the availability of considerable computational resources. It is important to pursue these developments because the current results suggest that real general purpose speech recognition will actually be possible in the future. In the meantime it is important to make end users accustomed to this new technology, both to its advantages and peculiarities. This can be done by small, customized applications or applications where speech is not the primary input modality. This thesis shows that it is possible to develop automatic optimization algorithms on top of state-of-the-art speech technology which make this technology easily usable for developers of customized applications. The effort for the development of such algorithms is much smaller than the effort that still has to be invested in real general purpose systems. Automatic optimization algorithms on top of todays speech recognizers can bridge the gap between the current speech recognition technology and the general purpose technology of tomorrow and can introduce users to the advantages of this exciting new input modality. The same applies to other new input modalities like on-line handwriting, gesture recognition, or lipreading.

10.3 Key contributions

In this thesis, an automatic algorithm for the tuning of architectural parameters of speech and on-line handwriting recognizers has been developed. The algorithm is applicable to all classification problems with spatio-temporal input. It was tested on speech and on-line handwriting, as two instances of such tasks. The approach is new, requires no domain specific knowledge by the user and is efficient (see Chapter 9 on page 119). It was shown for the first time that

- fully automatic tuning of all relevant architectural parameters of speech and on-line handwriting recognizers (widths of input windows, number of hidden and state units) to the domain and the available amount of training data is actually possible
- the automatic tuning can be done efficiently, both in terms of computational effort and final performance.

10.4 Future Work

The “generalization balancing across classes” property would be an interesting topic for further research. In my opinion, the measurement of performance by a single number is not sufficient for todays advanced systems. More consideration should be given to the distribution of errors. Further research could be devoted to algorithms that attempt to control

- the distribution of misclssifications according to user specifications
- classification risk.

Acknowledgments

I gratefully acknowledge the support of the McDonnel-Pew Foundation (Cognitive Neuroscience Program). Many thanks to my advisors, Alex Waibel and Scott Fahlmann who have been a continuous source of both constructive ideas and challenging critique. Many thanks to my previous advisors, Jay McClelland from CMU, H. Reitbök and Reinhard Eckhorn from the University of Marburg, for broadening my view of connectionist research.

I would also like to thank Stefan Manke, Hermann Hild, Patrick Haffner, Joe Tebelskis, Thorsten Zeppenfeld, Paul Duchnowsky and Michael Finke for lots of helpful discussions about MS-TDNNs. Special thanks to Stefan Manke for recording the on-line handwriting database, providing results from manually tuned TDNN's and answering numerous questions about the Unix environment at UKA. Special thanks to Hermann Hild for help with the connected letter database and providing results from his MS-TDNN. Many thanks to Arthur McNair for his pioneering work at UKA in April 1991. Additional thanks to Dave Sanner and Gilette Elvgren for the introduction to the Unix environment at CMU.

Special thanks to Ingrid Gemen, Silke Dannenmaier, Stefan Manke, Thorsten Minkwitz, Ed Pillai, Michael Philippsen, Stephan Teiwes, Michael Berthold, Finn Dag Buo, Bojan Petek, Markus Höhfeld, Otto Schmidbauer, and Gerd Völksen.

Appendix

The Post-NIPS Workshop "Optimization of Neural Network Architectures for Speech Recognition" (December 1991)

The workshop was announced as follows:

"A variety of neural network algorithms have recently been applied to speech recognition tasks. Besides having learning algorithms for weights, optimization of the network architectures is required to achieve good performance. Also of critical importance is the optimization of neural network architectures within hybrid systems for best performance of the system as a whole. Parameters that have to be optimized within these constraints include the number of hidden units, number of hidden layers, time-delays, connectivity within the network, input windows, the number of network modules, number of states and others. The proposed workshop intends to discuss and evaluate the importance of these architectural parameters and different integration strategies for speech recognition systems.Participating researchers interested in speech recognition are welcome to present short case studies on the optimization of neural networks, preferably with an evaluation of the optimization steps.The workshop could also be of interest to researchers working on constructive/destructive learning algorithms because the relevance of different architectural parameters should be considered for the design of these algorithms."

There were 26 participants and seven talks by invited speakers. The talks given by K. Iso, M. Franizini and P. Haffner did include quantifiable evaluations of optimization steps. Iso presented results on the speaker independent recognition of 10 Japanese digits with a prediction based recognition system. A system without hidden units achieved an error of 0.9%. The error could be reduced to 0.2% with the optimal number of hidden units. Iso also presented an evaluation of different input windows for a speaker dependent system that was trained on a 100 word vocabulary. A system using the input frames with the time-delays 2 and 1 to predict the current frame (time-delay = 0) recognized 97.5% correctly while the same system using the frames with the time-delays 3 and 0 to predict the frame with time-delay 1 achieved 96.5% only.

Franzini presented some optimization results on speaker independent isolated english letters. He evaluated a Connectionist Viterbi Training (CVT) network [Franzini, 1990] with one hidden layer (19.2% error), two hidden layers (15.6% error), and three hidden layers (15.9%). The number of epochs needed for training were 746 (one hidden layer), 2,933 (two hidden layers), and 5,401 (three hidden layers). Franzini also varied the number of hidden units in each hidden layer. His results indicated that 15 to 25 hidden units lead to similar error rates. Franzini also reported that according to his experiments an input window of seven frames (with each frame representing 10 ms) worked best. He also replaced the recurrent version of his system [Franzini, 1990] with a simple feed-forward system and found an insignificant reduction of the error rate (6%) but a significant reduction in computational cost. Finally, he reported a 40% lower string error rate for a system using word models instead of phone models.

Haffner reported some optimization results from his Multi State Time-Delay Neural Network (MS-TDNN) [Haffner, 1991a], [Haffner, 1991b]. According to his experiments on speaker independent digit recognition[1], 10 to 20 hidden units in one hidden layer are sufficient. Experiments with up to 70 hidden units did lead to the same recognition performances when training was stopped according to best performance on an independent validation set. Haffner also did some experiments with the input windows from the hidden layer to the phone layer. He found that a MSTDNN using input windows with the time-delays -3, -1, 0, 1, and 3 instead of the standard input window with -2, -1, 0, 1, 2 performed equally well, but learned much faster. He did not want to present detailed information about the number of states that were used in the system. However, he confirmed that the number of states for each phoneme is one of the most important parameters to optimize.

There were many other interesting comments and discussions at the workshop that can not be summarized in detail. However, the following conclusions can be extracted for the design of an automatic optimization algorithm:

- Stopped training based on the performance on an independent validation set is a reliable and easy method to avoid overfitting.
- The number of hidden units is an important, but not the most important architectural parameter for neural network based speech recognition systems. Haffner and Franzini report that there is a certain range of number of hidden units where generalization ability is similar. This may only be true for the large training databases that they used. For small training data-

1. a huge database collected at CNET with telephone quality speech

bases the number of hidden units may be more critical because the system has to learn a mapping of similar difficulty with less training patterns. The conclusion for an automatic optimization algorithm is that the number of hidden units should be included in the optimization process, but should not be the only architectural parameter to be modified.

- The optimization of the input windows leads to significant improvements and should be included into the automatic optimization process.
- The number of states per acoustic event and the type of acoustic event that is modeled (words or phonemes) seem to be the most important architectural parameters. Some presenters did not want to present their current state topology exactly, but they admitted that optimization at this level of the system is very effective. This suggests that automatic optimization of architectural parameters for speech recognition should include optimization of the state topology.
- The total number of parameters in such systems can grow very large (50.000 parameters and more). The optimization algorithm should be able to deal with that.
- Human optimizers tend to perform optimization steps sequentially, which may not be optimal. However, an intelligent human optimizer is able to recover from wrong paths, partly due to technical exchange at conferences. Obviously, automatic optimization procedures can not do this. A solution to this problem is the development of optimization algorithms that optimize synergetically.

References

[Bahl, 1980]	Bahl, L. R., Bakis, R., and Cohen, P. S. Further Results on the Recognition of a Continuously Read Corpus Proceedings ICASSP, Denver, Colorado, 1980
[Baldi, 1991]	Baldi, P., and Chauvin, Y. Temporal Evolution of Generalization During Learning in Linear Networks Neural Computation, 3, 589 - 603, 1991
[Barron, 1984]	Barron, A. Predicted Squared Error: A Criterion For Automatic Model Selection In: Farlow, S., Editor, Self-Organizing Methods In Modeling New York, Marcel Dekker, 1984
[Baum, 1989]	Baum, E.B., and Haussler, D. What Size Net Gives Valid Generalization?, Neural Computation, 1: 151-160

[Belew, 1990] Belew, R. K., McInerney, J., and Schraudolph, N. N.
Evolving Networks: Using the Genetic Algorithm with Connectionsist Learning
Technical Report, CS90-174, University of California at San Diego
La Jolla, CA 92093, 1990

[Bengio, 1993] Bengio, Y., Frasconi, P., and Simard, P.
The Problem of Learning Long-Term Dependencies in Recurrent Neural Networks
Proceedings of the IEEE International Conference on Neural Networks
San Francisco, 1993

[Berthod, 1990] Berthod, M.
On-line Analysis of Cursive Handwriting
In C. Y. Suen and R. De Mori, editors, Computer Analysis and Perception
Vol. 1, Visual Signal, pp. 55 - 81, CRC Press, 1990

[Berthold, 1993] Berthold, M.
A Time-Delay Radial Basis Function Network for Phoneme Recognition
private communication, to be published

[Blumstein, 1980] S.E. Blumstein and K.N. Stevens.
Perceptual Invariance And Onset Spectra For Stop Consonants In Different Vowel Environments. Journal of the Acoustical Society of America, 67:648--662, 1980.

[Bodenhausen, 1990a]Bodenhausen, U.
The Tempo Algorithm: Learning In A Neural Network With Adaptive Time-Delays.
Proceedings IJCNN 90, Washington D.C., January 1990.

[Bodenhausen, 1990b]Bodenhausen, U.
Learning Internal Representations of Pattern Sequences In A Neural Network With Adaptive Time-Delays.
Proceedings IJCNN 90, San Diego, June 1990.

[Bodenhausen, 1991a]Bodenhausen, U., and Waibel, A.
The Tempo 2 Algorithm: Adjusting Time-Delays By Supervised Learning
In Advances in Neural Information Processing Systems
Morgan Kaufmann, 1991.

[Bodenhausen, 1991b] Bodenhausen, U. and Waibel, A.
Learning The Architecture Of Neural Networks For Speech Recognition.
In Proceedings ICASSP 91, Toronto, May 1991.

[Bodenhausen, 1991c]Bodenhausen, U., and Waibel, A.
Optimization of Neural Network Architectures for Speech Recognition
post-NIPS workshop, Vail, Colorado, USA, December 6 and 7, 1991

[Bodenhausen, 1992] Bodenhausen, U.
unpublished results

[Bodenhausen, 1993a] Bodenhausen, U., and Waibel, A.
Application Oriented Automatic Structuring of Time-Delay Neural Networks for High Performance Character and Speech Recognition.
In: Proceedings ICNN 93, San Francisco, March 1993.

[Bodenhausen, 1993b] Bodenhausen, U., and Manke, S.
Connectionist Architectural Learning for High Performance Character and Speech Recognition.
In: Proceedings ICASSP-93, Minneapolis, April 1993.

[Bodenhausen, 1993c] Bodenhausen, U., and Manke, S.
Automatically Structured Neural Networks For Handwritten Character and Word Recognition.
In: Proceedings ICANN 93, Amsterdam, September 1993

[Bodenhausen, 1993d] Bodenhausen, U., and Waibel, A.
Tuning By Doing: Flexibility Through Automatic Structure Optimization
In: Proceedings Eurospeech 93, Berlin, September 1993

[Bottou, 1990] Bottou, L.
private communication

[Bourlard, 1989] Bourlard, H., and Wellekens, C.
Links Between Markov Models and Multilayer Perceptrons
In Advances in Neural Information Processing Systems 1,
Ed.: D. Touretzky, pp. 502 - 510, Morgan Kaufmann, San Mateo, 1989

[Braun, 1993] Braun, H., and Weisbrod, J.
Evolving Neural Networks for Application Oriented Problems
Proceedings of the 2nd Annual Conference on Evolutionary Programming
La Jolla, CA, 1993

[Buntine, 1991] Buntine, W. L., and Weigend, A. S.
Bayesian Back-Propagation
Complex Systems 5, pp. 603 - 643, 1991

[Chang, 1993]	Chang, E. I., and Lippmann, P. A Boundary Hunting Radial Basis Function Classifier Which Allocates Centers Constructively In: Advances in Neural Information Processing Systems 5, 1993
[Chauvin, 1989]	Chauvin, Y. A Back-Propagation Algorithm with Optimal Use of Hidden Units In; Touretzky, D. editor, Neural Information Processing Systems 1, Denver, 1988, Morgan Kaufmann, 1989
[Chen, 1993]	Chen, D., Giles, C. L., Sun, G. Z., Chen, H. H., Lee, Y.C., Goudreau, M. W. Constructive Learning of Recurrent Neural Networks Proceedings ICNN 93, San Francisco, March 1993
[Cleeremans, 1989]	Cleeremans, A., Servan-Schreiber, D., and McClelland, J. Finite-State Automata and Simple Recurrent Networks in: Neural Computation 1, pp. 372 - 381, 1989
[Deller, 1993]	Deller, J. R., Proakis, J. G., and Hansen, J. H. L. Discrete-Time Processing of Speech Signals Macmillan Publishing Company, New York, 1993
[Denker, 1987]	Denker, J., Schwartz, D., Wittner, B., Solla, S., Howard, R., Jackel, L., and Hopfield, J. Automatic Learning, Rule Extraction and Generalization Complex Systems, 1, pp. 877 - 922, 1987
[de Vries, 1991]	de Vries, B., and Principe, J. C. A Theory for Neural Networks with Time Delays In: Lippmann, Moody and Touretzky (Eds.), Advances in Neural Information Processing Systems 3 (pp. 162 - 168), San Mateo, CA, Morgan Kaufmann, 1991
[de Vries, 1992]	de Vries, B., and Principe, J. C. The Gamma Model - A new Neural Net Model for Temporal Processing Neural Networks, 5, 565 - 576, 1992
[Drucker, 1993a]	Drucker, H., Schapire, R., and Simard, P. Boosting Performance in Neural Networks to be published in International Journal of Pattern Recognition
[Drucker, 1993b]	Drucker, H., Schapire, R., and Simard, P. Improving Performance in Neural Networks using a Boosting Algorithm In: Neural Information Processing Systems 5, 1993

[Duda, 1973]	Duda, R. O., and Hart, P. E. Pattern Classification and Scene Analysis John Wiley, New York, 1973
[Elman, 1988]	Elman, J. L. Finding Structure in Time CRL Tech Report 8801, University of California at San Diego, Center for Research and Language
[Fahlman, 1988]	Fahlman, S. E. Faster-Learning Variations of Back-Propagation; An Empirical Study In: Proceedings of the 1988 Connectionist Summer School, Morgan Kaufmann, 1988
[Fahlman, 1990 a]	Fahlmann, S.E., and Lebiere, C. The Cascade-Correlation Learning Architecture In: Touretzky, D. editor, Neural Information Processing Systems 2, Denver, 1989, Morgan Kaufmann, 1990
[Fahlman, 1990 b]	Fahlman, S. E., and Lebiere, C. The Cascade-Correlation Learning Architecture Technical Report CMU-CS-90-100 School of Computer Science, Carnegie Mellon University, Pittsburgh
[Fahlman, 1991]	Fahlman, S. E. The Recurrent cascade-Correlation Architecture In: Lippmann, Moody and Touretzky (Eds.), Advances in Neural Information Processing Systems 3 (pp. 162 - 168), San Mateo, CA, Morgan Kaufmann, 1991
[Finnoff, 1991a]	Finnoff, W. Complexity Measures for Classes of Neural Networks with variable Weight Bounds in Proceedings International Conference on Neural Networks, Singapore, 1991
[Finnoff, 1991b]	Finnoff, W., and Zimmermann, H. G. Detecting Structure in small Datasets be Network Fitting under Complexity Constraints in Proceedings of 2nd Annual Workshop on Computational Learning Theory and Natural Learning Systems, Berkley, 1991
[Finnoff, 1992]	Finnoff, W., Hergert, F., and Zimmermann, H. G. Improving Model Selection by Nonconvergent Methods private communication from Siemens AG, Corporate Research and Development, Otto-Hahn-Ring 6, 8000 München 83

[Flann, 1993] Flann, N. S., and Shekhar, S.
Recognizing On-Line Cursive Handwriting Using a Mixture of
Cooperating Pyramid-Style Neural Networks
In: Proceedings World Congess on Neural Networks
1993

[Franzini, 1990] Franzini, M., Lee, K. F., and Waibel, A
Connectionist Viterbi Training: A New Hybrid Method For Continuous
Speech Recognition
Proceedings ICASSP 1990, Albuquerque, April 1990

[Frasconi, 1992] Fransconi, P., Gori, M., and Soda, G.
Injecting Nondeterministic Finite State Automata into Recurrent
Neural Networks
private communication, 1992

[Fritzke, 1993] Fritzke, B.
Growing Cell Structures- A Self-Organizing Network for Unsupervised
and Supervised Learning
Technical Report 93-026, International Computer Science Institute (ICSI)
Berkeley, available from neuroprose via anonymous ftp, 1993

[Galler, 1993] Galler, M., and De Mori, R.
Using Search to Improve Hidden Markov Models
Proceedings ICASSP 93, Minneapolis, Minnesota,
Vol. 2, pp. 303 - 306, 1993

[Geman, 1992] Geman, S., Bienenstock, E., Doursat, R.
Neural Networks and the Bias/Variance Dilemma
Neural Computation 4, 1-58, 1992

[Geutner, 1993] Geutner, P., Bodenhausen, U. , and Waibel, A.
Flexibility Through Incremental Learning: Neural Networks for
Text Categorization
in: Proceedings World Congress on Neural Networks,
Portland, Oregon, 1993

[Gish, 1990] Gish, H.
A Probabilistic Approach to Understanding and Training of
Neural Network Classifiers
In Proceedings ICASSP, pp. 1361 - 1364, April 1990

[Guyon, 1991] Guyon, I., Albrecht, P., Le Cun, Y., Denker, W., Hubbard, W.
Design of a Neural Network Character Recognizer for a Touch
Terminal, Pattern Recognition, 24(2), 1991

[Haffner, 1991a] Haffner, P., Franzini, M., and Waibel, A.
Integrating Time Alignment and Neural Networks for High Performance Continuous Speech Recognition, ICASSP 91

[Haffner, 1991b] Haffner, P., and Waibel, A.
Time-Delay Neural Networks Embedding Time Alignment: A Performance Analysis, Eurospeech 91

[Haffner, 1992a] Haffner, P., and Waibel, A.
Connectionist Word-Level Classification in Speech Recognition
Proceedings ICASSP 92, San Francisco , March 1992

[Haffner, 1992b] Haffner, P., and Waibel, A.
Multi-State Time Delay Neural Networks for Continuous Speech Recognition
Neural Information Processing Systems 4, 1992

[Haffner, 1993a] Haffner, P.
Connectionist Speech Recognition with a Global MMI Algorithm
In Proceedings EUROSPEECH 93, Berlin, 1993

[Haffner, 1993b] Haffner, P.
$\alpha\beta$-Implement "Fuzzy" Connectionist Time-Alignment in Speech Recognition
Proceedings ICANN, Amsterdam, September 1993

[Hampshire, 1990] Hampshire, J. B II., and Pearlmutter, B.
Equivalence Proofs for Multilayer Perceptron Classifiers and the Bayesian Discriminant Function
In Proceedings of the 1990 Connectionsit Models Summer School,
Eds. D. Touretzky, J. Elman, T. Sejnowski, and G. Hinton
Morgan Kaufmann, San Mateo, CA, 1990

[Hanson, 1990a] Hanson, S. J.
A Stochastic Delta Rule
Physica D, 1990

[Hanson, 1990b] Hanson, S. J.
Meiosis Networks.
In: Touretzky, D. editor, Neural Information Processing Systems 2,
Denver, 1989, Morgan Kaufmann, 1990

[Hassibi, 1993a] Hassibi, B., and Stork, D. G.
Second Order Derivative for Network Pruning: Optimal Brain Surgeon
In: Neural Information Processing Systems 5, 1993

[Hassibi, 1993b] Hassibi, B., and Stork, D. G.
Optimal Brain Surgen and Generaal Network Pruning
In: Proceedings ICNN, San Francisco, pp. 293 - 299, March 1993

[Hergert, 1992] Hergert, F., Finnoff, W., and Zimmermann, H.G.
A Comparison of Weight Elimination Methods for Reducing Complexity in Neural Networks
Proceedings International Joint Conference on Neural Networks, Baltimore, 1992

[Hild, 1993a] Hild, H., and Waibel, A.
Connected Letter recognition with a Multi-State Time Delay Neural Network
Neural Information Processing Systems 5, 1993

[Hild, 1993b] Hild, H., and Waibel, A.
Multi-Speaker/Speaker-Independent Architectures for the Multi-State Time Delay Neural Network
Proceedings ICASSP 93, Minneapolis, 1993

[Holland, 1975] Holland, J.
Adaptation in Natural and Artificial Systems
University of Michigan Press, Ann Arbor, MI

[Kamm, 1990] Kamm, C. E.
Effects Of Neural Network Input Span On Phoneme Classification.
In Proceedings of the International Joint Conference on Neural Networks, June 1990.

[Kewley-Port, 1983] Kewley-Port, D.
Time Varying Features As Correlates Of Place Of Articulation In Stop Consonants. Journal of the Acoustical Society of America, 73:322--335, 1983.

[Kohonen, 1982] Kohonen, T.
Self-Organized Formation of Topologically Correct Feature Maps.
Biological Cybernetics, 43:59-69, 1982.

[Kohonen, 1988a] Kohonen, T.
The “Neural” Phonetic Typewriter. *IEEE Computer,* (3):11-22, 1988.

[Kohonen, 1988b] Kohonen, T., Barna, G., Chrisley, R.
Statistical Pattern Recognition with Neural Networks: Benchmarking Studies, *Technical Report,* No. SF-02150, Helsinki University of Technology, Espoo, Finland, 1988.

[Krogh, 1992] Krogh, A., and Hertz, J. A.
A Simple Weight Decay Can Improve Generalization
In: Neural Information Processing Systems 4, 1992

[Lang, 1989] Lang, K.
PhD Thesis, Carnegie Mellon University, PA, 15213

[Lang, 1990] Lang, K.J. , Hinton, G.E., and Waibel, A.H.
A Time-Delay Neural Network Architecture For Speech Recognition.
Neural Networks Journal, 1990.

[Lecolinet, 1993] Lecolinet, E., and Baret, O.
Cursive Word Recognition: Methods and Strategies
In: Fundamentals in Handwriting Recognition, Bonas,
NATO-ASI workshop, 1993

[Le Cun, 1989] Le Cun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E.,
Hubbard, W., and Jeckel, L. D.
Back-Propagation Applied to Handwritten Zip Code Recognition
Neural Computation 1, pp. 541 - 551, 1989

[Le Cun, 1990] Le Cun, Y., Denker, J.S., Solla, S.A.
Optimal Brain Damage
In: Touretzky, D. editor, Neural Information Processing Systems 2,
Denver, 1989, Morgan Kaufmann, 1990

[Lee, 1990] Lee, K. F., Hon, H. W., and Reddy. D. R.
An Overview of the SPINX Speech Recognition System
IEEE Transactions on Signal Processing, Vol. 38, pp. 35 - 45, Jan. 1990

[Lee, 1993] Lee, K. F.
Speech Recognition for Personal Computing
In: Proceedings 1993 IEEE Workshop on Automatic Speech Recognition
Snowbird, Utah, USA, December 1993

[Leonard, 1984] Leonard, R. G.
A Database for Speaker-Independent Digit Recognition
Procceeddings ICASSP, March 1984

[Lindgren, 1965] Lindgren, N.
Machine Recognition of Human Language,
Part III - Cursive Script Recognition
IEEE Spectrum, pp. 104 - 116, May 1965

[MacKay, 1991] MacKay, D. J. C.
Bayesian Methods for Adaptive Models
PhD Thesis, California Inst. Tech., Pasadena, 1991

[MacKay, 1992a] MacKay, D. J. C.
Bayesian Interpolation
Neural Computation 4, pp. 415 - 447, 1992

[MacKay, 1992b] MacKay, D. J. C.
A Practical Bayesian Framework for Backpropagation Networks
Neural Computation 4, pp. 448 - 472, 1992

[MacKay, 1992c] MacKay, D. J. C.
Information Based Objective Functions for Active Data Selection
Neural Computation 4, pp. 590 - 604, 1992

[MacKay, 1992d] MacKay, D. J. C.
The Evidence Framework Applied to Classification Networks
Neural Computation 4, pp. 720 - 736, 1992

Manke, 1992] Manke, S.
unpublished results, private communication

[Manke, 1994] Manke, S., and Bodenhausen, U.
A Connectionist Recognizer For On-Line Handwriting Recognition
In: Proceedings ICASSP 94, 1994

[Minsky, 1988] Minsky, M.L., Papert, S.A.
Perceptron-Expamded Edition
MIT Press, 1988

[Mosteller, 1968] Mosteller, F., and Tukey, J. W.
Data Analysis, Including Statistics
In: G. Lindzey and E. Aronson, editors, Handbook of Social Psychology, Vol. 2, Addison-Wesley, 1968 (first edition 1954)

[Moody, 1988] Moody, J., and Darken, C.
Fast Learning in Networks of Locally Tuned Processing Units
In: Touretzky, Hinton and Sejnowsky, editors, Connectionist Models Summer School, Morgan Kaufmann Publishers, 1988, pp. 133 - 143

[Moody, 1989] Moody, J.
Fast Learning in Multi-Resolution Hierarchies
in: Touretzky, D. S., Advances in Neural Information Processing Systems 1 1989

[Moody, 1991] Moody, J.
The Effective Number of Parameters: An Analysis of Generalization and Regularization in Nonlinear Learning Systems
in: Advances in Neural Information Processing Systems 4, 1991

[Morgan, 1990] Morgan, N., and Bourlard, H.
Generalization and Parameter Estimation in Feedforward Nets: Some Experiments
in D. Touretzky, (Ed.), Advances in Neural Information Processing Systems 2, (pp. 598 - 605), San Mateo, Morgan Kaufmann, 1990

[Mozer, 1993] Mozer, C.
Neural net architectures for temporal sequence processing
in: A. Weigend and N. Gershenfeld (Eds.), Predicting the future and understanding the past, redwood City, CA: Addison-Wesley Publishing

[Ney, 1984] Ney, H.
The Use of a One-Stage Dynamic Programming Algorithm for Connected Word Recognition
IEEE Transactions on Acoustics, Speech, and Signal Processing
Vol. ASSP 32, NO 2, April 1984

[Nowlan, 1992] Nowlan, S. J., and Hinton, G. E.
Simplifying Neural Networks by Soft Weight-Sharing
Neural Computation, 1992

[Quinlan, 1986] Quinlan, J. R.
Introduction of decision trees
Machine Learning 1(1): 81 - 106, 1986

[Ramachandran, 1992]Ramachandran, S., and Pratt, L.Y.
Information Measure Based Skeletonisation
In: Advances in Neural Information Processing Systems 4, 1992

[Reilly, 1982] Reilly, D., Cooper, L., and Elbaum, C.
A Neural Model for Category Learning
Biol. Cybernetics, Vol. 45, pp. 35 - 41, 1982

[Renals, 1993a] Renals, S., and MacKay, D.
Bayesian Regularisation Methods in a Hybrid MLP-HMM System
Proceedings of Eurospeech, Berlin, 1993

[Renals, 1993b] Renals, S.
Bayesian Regularisation Methods in a Hybrid MLP-HMM System
additional information given in talk at Eurospeech, Berlin 1993

[Richard, 1991]	Richard, M. D., and Lippmann, R. P. Neural Network Classifier Estimates Bayesian a posteriori Probabilities Neural Computation, Volume 3, Number 4, 1991
[Robinson, 1989]	Robinson, A. J. Dynamic Error Propagation Networks PhD Thesis, Cambridge University, 1989
[Ruck, 1990]	Ruck, D. W., Roger, S. K., Kabrisky, M., Oxley, M. E., and Suter, B. W. The Multilayer Perceptron as an Approximation to a Bayes Optimal Discriminant Function IEEE Transactions Neural Networks 1(4), pp. 296 - 298, 1990
[Rumelhart, 1986]	Rumelhart, D.E., Hinton, G.E., and Williams, R.J. Learning Internal Representations By Error Propagation. In J.L. McClelland and D.E. Rumelhart, editors, Parallel Distributed Processing; Explorations in the Microstructure of Cognition, chapter 8, pages 318--362. MIT Press, Cambridge, MA, 1986.
[Rumelhart, 1988]	Rumelhart, D.E. Learning and Generalization Plenary Address, IEEE International Conference on Neural Networks, San Diego, 1988
[Sakoe, 1978]	Sakoe, H., Chiba, S. Dynamic Programming Algorithm Optimization for Spoken Word Recognition. *IEEE Transaction on Acoustics, Speech and Signal Processing,* (26):43-49, 1978.
[Schapire, 1990]	Schapire, R. The Strength of Weak Learnability Machine Learning, Vol. 5, #2, pp. 197 - 227, 1990
[Schenkel, 1993]	Schenkel, M., Guyon, I., and Henderson, D. On-Line Cursive Script Recognition using Time-Delay Neural Networks and Hidden Markov Models private communication, paper submitted
[Schwartz, 1984]	Schwartz, R. M., Chow, Y. L., Roucos, S. et al. Improved Hidden Markov Modelling of Phonemes for Continuous Speech Recognition Proceedings ICASSP, San Diego, California, 1984

[Shoemaker, 1991] Shoemaker, P. A.
A Note on Least-Squares Learning Procedures and Classification by Neural Network Models
IEEE Transactions Neural Networks, 2(1), pp 158 - 160, 1991

[Simard, 1992] Simard, P., Victorri, B., and LeCun, Y.
Tangent Prop- A Formalism for Specifying Selected Invariances in an Adaptive Network
In: Advances in Neural Information Processing Systems 4, 1992

[Solla, 1989] Solla, S.
Learning and Generalization in Layered Neural Networks. The Contiguity Problem.
In: Neural Networks: From Models to Applications
Ed.: L. Personnaz and G. Dreyfus, pp. 168 - 177, IDSET Paris, 1989

[Sorensen, 1992] Sorensen, H. B. D., and Hartmann, U.
Self-Structuring Hidden Control Neural Networks for Speech Recognition
Proceedings of ICASSP 92, San Francisco, 1992

[Stone, 1977] Stone, C. J.
Cross-Validation: A Review
Math. Operationsres. Statist. Ser., 9, 1 - 51, 1977

[Svarer, 1993] Svarer, C., Hansen, L.K., and Larsen, J.
On Design and Evaluation of Tapped-Delay Neural Network Architectures
Proceedings ICNN, San Francisco, pp. 46 - 51, March 1993

[Takami, 1992] Takami, J., and Sagayama, S.
A Successive State Splitting Algorithm for Efficient Allophone Modeling
Proceedings ICASSP, 1992

[Tank, 1987] Tank, D.W. and Hopfield, J.J.
Neural Computation By Concentrating Information In Time.
In Proceedings National Academy of Sciences, pages 1896--1900, April 1987.

[Tappert, 1990] Tappert, C. C., Suen, C. Y., and Wakahara, T.
The State of Art in On-Line Handwriting Recognition
IEEE Transactions on Pattern Analysis and Machine Intelligence 12(8), pp. 787 - 808, 1990

[Tebelskis, 1993] Tebelskis, J., and Waibel, A.
Performance Through Consistency: MS-TDNN's for Large Vocabulary Continuous Speech Recognition
Neural Information Processing Systems 5, 1993

[Thodberg, 1993] Thodberg, H. H.
Ace of Bayes: Application of Neural Networks with Pruning
posted in Neuroprose, available via anonymous ftp

[Unnikrishnan, 1991] Unnikrsihnan, K. P., Hopfiel, J. J., and Tank, D. W.
Connected Digit Speaker Dependent Speech Recognition using a Neural Network with Time-Delayed Connections
IEEE Transactions on Signal Processing, 39, 698 -713, 1991

[Unnikrshnan, 1992] Unnikrsihnan, K. P., Hopfiel, J. J., and Tank, D. W.
Speaker-Independent Digit Recognition Using a Neural Network with Time-Delayed Connections
Neural Computation 4, 108 - 119, 1992

[Utans, 1991] Utans, J., and Moody, J.
Selecting neural Network Architectures via the Prediction Risk: Application to Corporate Bond rating Prediction
Proceedings of the First International Conference on Artificial Intelligence Applications on Wall Street
IEEE Computer Society Press, Los Alamos, CA, 1991

[Valiant, 1984] Valiant, L. G.
A Theory of the Learnable
Communication of the ACM, Vol. 27, # 11, pp. 1134 - 1142, 1984

[Wahba, 1990] Wahba, G.
Spline Models for Observational Data
Volume 59 of Regional Conference Series in Applied Mathematics
SIAM Press, Philadelphia, 1990

[Waibel, 1989a] Waibel, A., Hanazawa, T., Hinton, G., Shikano, K., and Lang, K.
Phoneme Recognition using Time-Delay Neural Networks
IEEE, Transactions on Acoustics, Speech and Signal Processing, March 1989

[Waibel, 1989b] Waibel, A., Sawai, H., and Shikano, K.
Modularity and Scaling in Large Phonemic Neural Networks
IEEE Transactions on Acoustoics, Speech and Signal Processing, December 1989

[Waibel, 1990] Waibel, A., and Lee, K.F.
Readings in Speech Recognition
Morgan Kaufmann Publishers, San Mateo, California, 1990

[Wan, 1990] Wan, E. A.
Neural Network Classification: A Bayesian Interpretation
IEEE Transaction Neural Networks 1(4), pp. 303 - 305, 1990

[Weigend, 1990a] Weigend, A.S., Hubermann, B.A., Rumelhart, D.E.
Prediciting the Future: A Connectionist Approach
Tech Report SSL-90-20, Xerox System Science Laboratory,
Palo Alto, CA, 1990

[Weigend, 1990b] Weigend, A.S., Hubermann, B.A., Rumelhart, D.E.
Prediciting the Future: A Connectionist Approach
Int. J. Neural Systems 1, pp. 193 - 209, 1990

[White, 1989] White, H.
Learning in Artificial Neural Networks: A Statistical Perspective
Neural Computation 1, pp. 425 - 464, 1989

[Wilpon, 1993] Wilpon, J. G.
Applications of Speech Recognition Technology in Telecommunications
In: Proceedings 1993 IEEE Workshop on Automatic Speech Recognition
Snowbird, Utah, USA, December 1993

[Wolpert, 1993] Wolpert, D. H.
Use of Evidence in Neural Networks
In: Neural Information Processing Systems 5, 1993

[Yang, 1991] Yang, J., and Honovar, V.
Experiments with the Cascade-Correlation Algorithm
Technical Report 91-16, Department of Computer Science,
Iowa State University, Ames, Iowa 50011, USA,
available via anonymous ftp from neuroprose

[Zeppenfeld, 1992] Zeppenfeld, T., and Waibel, A.
A Hybrid Neural Network, Dynamic Programming Word Spotter
In Proceedings ICASSP 92, 1992

[Zeppenfeld, 1993] Zeppenfeld, T., Houghton, R., and Waibel, A.
An Improved MSTDNN for Word Spotting
In Proceedings ICASSP 93, Minneapolis