The Free Speech Journal, Issue # 5(1997)
Published 10/22/97
® 1997 All rights reserved.

Phoneme Probability Estimation with Dynamic Sparsely Connected Artificial Neural Networks

Nikko Ström, (nikko@speech.kth.se)

Department of Speech, Music and Hearing, KTH, Stockholm, Sweden

Centre for Speech Technology, KTH, Stockholm, Sweden

Abstract

This paper presents new methods for training large neural networks for phoneme probability estimation. An architecture combining time-delay windows and recurrent connections is used to capture the important dynamic information of the speech signal. Because the number of connections in a fully connected recurrent network grows super-linear with the number of hidden units, schemes for sparse connection and connection pruning are explored. It is found that sparsely connected networks outperform their fully connected counterparts with an equal number of connections. The implementation of the combined architecture and training scheme is described in detail. The networks are evaluated in a hybrid HMM/ANN system for phoneme recognition on the TIMIT database, and for word recognition on the WAXHOLM database. The achieved phone error-rate, 27.8%, for the standard 39 phoneme set on the core test-set of the TIMIT database is in the range of the lowest reported. All training and simulation software used is made freely available by the author, and detailed information about the software and the training process is given in an Appendix.

Table of contents

1 Introduction

Speech recognition was one of the first problems that artificial neural networks (ANNs) were applied to during the rapid spread of ANN models in the 1980's. ANNs have been used for recognition of large units like words directly (e.g. Ström, 1992; English and Boggess, 1992; Li, Naylor, and Rossen, 1992), but attempts to recognize smaller units like phonemes have been more successful. Several different methods exist for combining the ANN classification of sub-units into sequences that constitutes words. A few of the more well-known methods are: "Hybrid HMM/ANN Architecture" (Bourlard and Wellekens, 1988), "Linked Predictive Neural Networks" (Tebelskis and Waibel, 1990), "Hidden Control Neural Architecture" (Levin, 1990), and "Stochastic Observation HMM" (Mitchel, Harper, and Jamieson, 1996). Of the mentioned methods, the hybrid HMM/ANN architecture is the most wide-spread today.

ANN systems undisputedly solve some problems better than all other methods in automatic speech recognition. The phoneme recognition rate on the TIMIT database reported by Robinson (1994) is several percent higher than that of all other systems – a large difference for this type of test (a comparison of different approaches is given in Table 3, section 5.1). ANN solutions are also typically compact, i.e., have a small number of parameters, and offer fast decoding compared to standard HMM systems. Still, for the last years, the research activities on ANNs for speech recognition have not by far been as intensive as for the prevailing HMM paradigm. Preference for a well-established technology and the relatively few equally excellent results reported on the word-level are possible reasons for the mild interest in ANN solutions. However, the evaluation of the SQALE-project (Steeneken and van Leeuwen, 1995) is an example of a case where hybrid HMM/ANN technology significantly outperforms state-of-the-art HMM systems provided by leading research sites for large vocabulary tasks.

A different reason for the limited spread of ANN solutions can be problems with the network training, i.e., efficiently and robustly determining the parameters of large networks. For example, the recurrent network used by Robinson is trained with special parallel hardware, and a rather complex training heuristic is used with several *ad hoc* parameters to be determined empirically. Recurrent connections are of course not the only path to good results, but other existing solutions have different problems. An ANN architecture without recurrent connections is used with good results by Bourlard and Morgan (1993). Instead they use time-delay windows (Waibel *et al.*, 1987) to capture the temporal cues of the speech signal. The lack of recurrent connections make the training algorithm more robust, but very large networks are used to achieve good results, and therefore the available computing resources limit the performance of the system.

Contemporary high performing ANN solutions typically require more computation for training than the wide-spread maximum likelihood (ML) training for the standard HMM (Rabiner and Juang, 1993; Lee, 1989). Comparing standard HMM and hybrid HMM/ANN training in more detail, a difference in computational complexity can be noted. The amount of computation increases faster for ANN training when the size of the model is increased. When more training data is available for training an HMM system, typically more context-dependent models are introduced. However, in a somewhat simplified view[1], the training time is independent of the number of models, and proportional to the amount of training data. This is because each model is trained only on a partition of the data. In the case of an ANN system, the capacity is determined by the number of hidden units, and when more training data is

[1] This is not true if the free parameters of the HMM are increased by some other means, e.g., increasing the number of components per Gaussian mixture. We have also made the approximation that the Baum-Welsh algorithm converges in the same number of iterations independent of the size of the training data, and on the number of context dependent models.

available, more hidden units are typically introduced. The computation for training increases faster than for HMM training, because the entire ANN is trained on all data. Again in a simplified view[2], the training time is proportional to the product of the amount of training data and the size of the ANN (the number of connections). Thus, in comparison with HMM training, the amount of computation for ANN training is more dependent on the model size. In particular for recurrent networks this is a problem because the number of recurrent connections grows quadratically with the number of hidden units.

In part, the scaling properties of ANN training originate from the inherent discriminative nature of the training, but this is not the whole truth. In this paper we propose new methods for robust training of large, high performance ANNs based on sparsely connected networks. In a sparsely connected ANN, the hidden layer can be allowed to grow without necessarily increasing the number of connections proportionally. The problem with the quadratic relation between units and connections in recurrent networks is addressed by introducing local connectivity for the recurrent connections. The local connectivity has the effect that units close to each other have higher likelihood of being connected. Thus, it promotes the development of groups of units performing sub-tasks of the problem. The sparse connection schemes used, and a related concept, connection pruning, are discussed in section 4. In section 2, a review of the theory of feed-forward ANNs is given together with the definition of the particular network architecture used in this paper, and details of the training algorithm. Recognition results on the TIMIT and WAXHOLM databases are reported in section 5.

The HMM paradigm has currently the advantage of a large mature body of easily available software (e.g., Young *et al*., 1995). To promote further development in the hybrid HMM/ANN field, and to make reproduction of our results easier, the software toolkit used for training and running the neural networks of this study is made freely available. In the appendix, the software is described and instructions on how to obtain a copy are given. Information about the simulations of this study is given in detail to make reproduction of the results possible.

2 Basic theory

This section describes the basic theory behind the dynamic feed-forward artificial neural networks used in the study. Because we wish to make replication of the results straight-forward, the description is rather detailed. Most of the material is well-known background-knowledge, covered in textbooks on ANN computing (e.g., Bishop, 1995 or Ripley, 1996). Details of the unified implementation of recurrent units and time-delay windows (Section 2.2) as well as the introduction of multiplication units and units with several different non-linearities can be of interest also for readers with some experience in the field.

2.1 Feed-forward networks

Feed-forward networks are *directed acyclic graphs*, i.e., the activities of all artificial neurons, hereafter called *units*, can be computed in one iteration, and there is no feedback that would make the activation of a unit depend recursively on its own value. A network consists of input units, hidden units and output units and connections between them. In the popular analogy with a biological nerve-system, the input units are “sensors” whose values are determined by “the environment”, the hidden units corresponds to internal neurons and the output units can be thought of as neurons of a motor system, controlling the organism’s responses to the input from the environment. This analogy can be very inspiring, but in practice feed-forward ANNs are used simply to approximate complex input/output maps, with no known explicit formula

[2] This is true only if the number of training epochs needed for convergence of the training is independent of the ANN’s size, and on the amount of training data. This matter is discussed further in section 5.1.5.

but a great deal of characterizing data. The strength of ANN models is the weak constraints they put on the mapping; It has been shown that, given a sufficient number of hidden units and characterizing data, feed-forward ANNs with one layer of hidden units can approximate any bounded function on a compact set with arbitrary accuracy (e.g., Hornik, Stinchcombe and White 1989). In practice however, the performance is limited due to: *i)* problems finding the set of connection weights that gives the optimal network and *ii*) the size of the available database.

In this paper we concentrate on mappings defining a "1-of-N" classification of the input patterns. Each input pattern of the training database is assigned to one of *N* classes. The target values for the *N* output units are -1.0 for all units except the one corresponding to the correct class whose target is +1.0. The *N* classes are the phonemes and the input patterns are feature representations of the short-time spectra of the sound wave. This will be covered more thoroughly in section 3.

Except for some special units, activities in our framework are computed in the same way as the classic ANNs of (Rumelhart, Hinton and Williams, 1986), but the sigmoid function is replaced by the computationally more convenient tanhyp function. Thus, for the tanhyp units, the activation a_i of unit i is defined by:

$$a_i = \tanh(net_i) \tag{1}$$

where

$$net_i = \sum_j w_{ji} a_j \tag{2}$$

and w_{ji} is the connection weight from unit j to unit i. It is easy to show that besides the change to an activation function with a symmetric range [-1;1], this is equivalent to a linear transformation of the weight space. The tanhyp function can be expanded as follows:

$$\frac{\tanh\left(\frac{x}{2}\right)+1}{2} = \frac{1}{2}\left(\frac{e^{\frac{x}{2}} - e^{-\frac{x}{2}}}{e^{\frac{x}{2}} + e^{-\frac{x}{2}}} + 1\right) = \frac{e^{\frac{x}{2}}}{e^{\frac{x}{2}} + e^{-\frac{x}{2}}} = \frac{1}{1+e^{-x}} = \text{sigmoid}(x) \tag{3}$$

Thus, the transformation,

$$\begin{cases} a_i = 2a_{i,RHW} - 1 \\ w_{ji} = 2w_{ji,RHW} \end{cases} \tag{4}$$

where *RHW* indicates parameters of the classic network, transforms a classic network based on sigmoid units to a network with tanhyp non-linearity.

It is sometimes convenient to work also with units without any non-linearity and units with other non-linearities. Because these units are not defined in the classical networks, the weights of connections to them are not affected by the transformation between our activities and the classical *RHW* domain of (4). In the experiments of this study we use linear units, exponential units, inverter units (1/x) and multiplication units in addition to the usual tanhyp and input units. Moreover, each network has one special-purpose unit that has a constant activity of 1.0. Connections are by default added from this unit to all tanhyp units – the effect is identical to that of a unit-bias. Multiplication units are slightly more complicated than the other types and require the computation of (6) instead of (2). In summary, unit activities are computed by

$$\begin{cases} a_i = \tanh(net_i) & \text{if unit } i \text{ is a tanhyp unit} \\ a_i = net_i & \text{if unit } i \text{ is a linear unit} \\ a_i = 1.0 & \text{if unit } i \text{ is the bias unit} \\ a_i = clamped & \text{if unit } i \text{ is an input unit} \\ a_i = \exp(net_i) & \text{if unit } i \text{ is an exponential unit} \\ a_i = 1/net_i & \text{if unit } i \text{ is an inverter unit} \\ a_i = prod_i & \text{if unit } i \text{ is a multiplication unit} \end{cases} \tag{5}$$

where

$$prod_i = \prod_{j,\text{unit } j \text{ feeds into unit } i} a_j \tag{6}$$

in analogy with (2).

The target values for the output units are 1.0 for the unit corresponding to the correct class and -1.0 for all other units. The objective function for the back-propagation training is based on the cross-entropy distance in the *RHW* domain (Solla, Levin, and Fleisher, 1988). If the target output activation for unit *i* is $\tau_{i,RWH}$ in the *RHW* domain, the contribution, e_i, to the cross entropy transformed to the tanhyp domain can be computed from (4):

$$\begin{aligned} e_i &= (1-\tau_{i,RWH})\log(1-a_{i,RWH}) + \tau_{i,RWH}\log(a_{i,RWH}) = \\ &= \left(1-\left(\frac{\tau_i+1}{2}\right)\right)\log\left(1-\frac{a_i+1}{2}\right) + \left(\frac{\tau_i+1}{2}\right)\log\left(\frac{a_i+1}{2}\right) = \\ &= \left(\frac{1-\tau_i}{2}\right)\log\left(\frac{1-a_i}{2}\right) + \left(\frac{\tau_i+1}{2}\right)\log\left(\frac{a_i+1}{2}\right) = \begin{cases} \log\left(\dfrac{1-a_i}{2}\right) & \text{if } \tau_i = \text{-}1 \\ \log\left(\dfrac{a_i+1}{2}\right) & \text{if } \tau_i = 1 \end{cases} \end{aligned} \tag{7}$$

The objective function, *E*, is the sum of e_i for all units and input patterns, so the derivative with respect to the activity is

$$\frac{dE}{da_i} = \begin{cases} 0 & \text{when the unit is not an output unit} \\ 1/(a_i+1) & \text{when the unit corresponds to the correct class} \\ 1/(a_i-1) & \text{otherwise} \end{cases} \tag{8}$$

Now we compute the derivative with respect to the connection weights in the standard way using the chain rule to get the recursive equations:

$$\delta_i = -\frac{\partial E}{\partial net_i} = \begin{cases} (1-a_i)(1+a_i)backnet_i & \text{if } i \text{ is a tanh unit} \\ -\frac{1}{a_i^2}backnet_i & \text{if } i \text{ is a inverter unit} \\ \exp(a_i)backnet_i & \text{if } i \text{ is a exponential unit} \\ backnet_i & \begin{array}{l}\text{if } i \text{ is an input, linear,} \\ \text{multiplication or the bias unit}\end{array} \end{cases} \quad \textbf{(9)}$$

where

$$backnet_i = -\frac{dE}{da_i} + \sum_{\substack{j \\ \text{unit } j \text{ is not a} \\ \text{multiplication unit}}} \delta_j w_{ij} + \sum_{\substack{j \\ \text{unit } j \text{ is a} \\ \text{multiplication unit}}} \delta_j \frac{a_j}{a_i} \quad \textbf{(10)}$$

and the derivatives with respect to the weights are:

$$\frac{\partial E}{\partial w_{ij}} = -\delta_j a_i \quad \textbf{(11)}$$

The derivatives of (11) can be used for gradient decent type minimization of *E*. The details of this are filled out in section 2.4.

Note that if an output unit has no out-flowing connections, then dE/da_i cancels one of the factors of δ_i in (9), and the resulting δ_i is simply the difference between a_i and the target (+1.0 for the correct class and -1.0 otherwise).

2.2 Recurrent connections and time-delay

Dynamic features of speech such as formant movements, that are known to be of importance for phoneme classification (e.g., Fant 1969), are not captured by the short time spectrum representation used as input to the network. Therefore, phonetic classification of short-time spectra can be greatly enhanced by considering also the context of neighboring spectra. A step in this direction was taken by Waibel *et al.* (1987) when they introduced *time-delay neural networks* (TDNN). In this paper we denote by TDNN, all network architectures where the units are connected to units in lower layers with time-delayed connections so that the activities depends on the activities of lower layer units in a finite time-delay window (Figure 1 (left)). The first experiments with TDNN successfully showed an improved classification of stop consonants where it is well known that the dynamic formant patterns are of great importance. Later the architecture has been successfully applied to complete phoneme inventories and used in hybrid HMM/ANN speech recognition systems with good results (e.g., Bourlard and Morgan, 1993; Cohen *et al*., 1992).

A different course to include the context in the classification is to connect units in the same layer with a delay of one time-step – so called recurrent connections (Figure 1 (middle)). The network still remains a feed-forward network because the recurrent connections are delayed. This approach differs from TDNN in that the activity of a unit at a particular time depends recursively on activities in its layer and lower layer at all previous times. Networks with recurrent connections are called *recurrent neural networks* (RNN) (Rumelhart, Hinton and Williams, 1986) or dynamic neural networks (Pearlmutter, 1990) and this is currently the most successful architecture for phoneme recognition (Robinson and Fallside, 1991; Robinson, 1994).

TDNNs and RNNs have much in common; in particular, both use time-delayed connections to incorporate context into the classification. In fact, if the connections of RNNs are allowed to have multiple time-delays instead of just one time-step, the resulting network has all the modeling power of both architectures. This unified architecture, RTDNN (Recurrent Time-Delay Neural Network) introduced by Ström (1992), is used in this study (Figure 1 (right)).

The delayed connections have the effect that the response of an input pattern is delayed several time-points in the output units. One feasible way to tackle this problem is to delay also the target values for the output units (Robinsson and Fallside, 1991; Robinsson 1994). In the RTDNN framework, we have chosen to use look-ahead connections instead of delaying the targets. Look-ahead connections let a unit depend on the activity of other units at *future* times (indicated by a positive superscript, e.g., z^{+1}, in Figure 1). This concept is often used in TDNN architectures, but not possible to implement in RNN architectures because recurrent connections must be delayed in feed-forward networks. Look-ahead connections force the computation of some unit activities to be delayed, but the network is still a feed-forward network as long as no unit's activity at a particular time depends on its own activity.

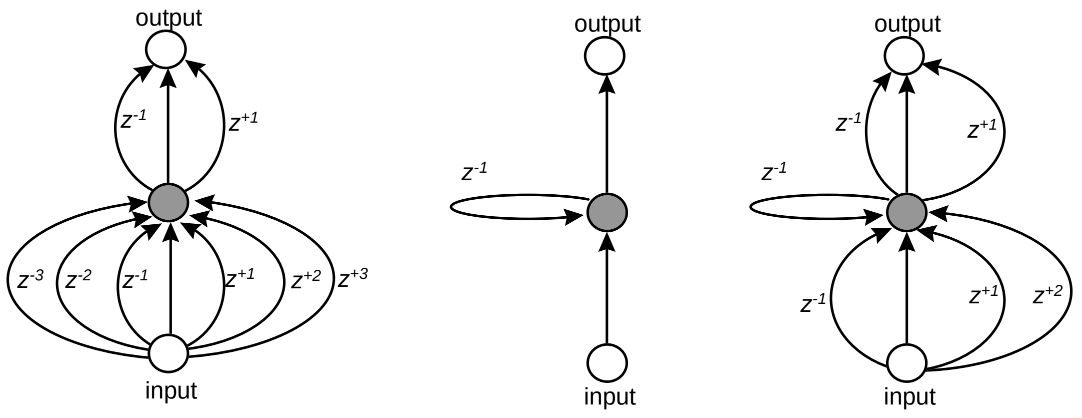


Figure 1. Different types of dynamic networks. The z^{-x} operator indicates that the connection is delayed x time-frames. For simplicity, the input, hidden, and output layers have only one unit each in this figure.

Left: time-delay network (TDNN). The units in higher layers have access to a time-delay window of the activities of units in lower layers. Note that both time-delay and look-ahead (z^{+x}) connections are used. A consequence of this is that computation of activities in units in higher layers must be delayed until the activities of the look-ahead connections are known.

Middle: Recurrent network (RNN). The hidden units are recurrently connected back to the hidden layer. In this architecture, the activities recursively depend on the activities of all previous time-frames.

Right: The combined architecture (RTDNN) with both time-delay windows, and recurrent connections.

It is not hard to show that any feed-forward network with look-ahead connections has an equivalent network with no look-ahead, but delayed targets instead. In the pseudo-code of Table 1 we show this by constructing such an equivalent network, and in the process we also get the order in which unit activities must be computed. In fact, the constructed equivalent network is the one used in the actual computer simulations.

The reason for introducing the extra complication of working with two separate network representations is that look-ahead connections are more intuitive and simplify the network design. For example, if one decides to use a wider time-delay window or change the dynamic structure in some other way, it is unnecessary to select or compute a new appropriate delay for the target values. This is instead handled automatically in the conversion process outlined in Table 1. In the simulations, the target values are delayed because that is the computationally most advantageous representation, but this is hidden from the network designer who can focus on selecting a dynamic structure suitable for the particular problem.

- Let all units of the new, equivalent network have a new property called delay and initialize it to zero for all units.

```
Let the delay of unit $i$ be $d'_i$.

Let $count = 0$
do {
    for each unit $i$ {
        for each connection $w_{ji,d}$ in the original network flowing to unit $i$ from unit $j$ with delay $d$ {
            if $d'_j - d > d'_i$ then let $d'_i = d'_j + d$
        }
    }
    Let $count = count + 1$
}
until no delay changes during a whole iteration or $count$ is greater than the total number of units
```

- If the loop was terminated because *count* grew larger than the number units, the network cannot be a feed-forward network and thus the design is in error. This check should be made each time new connections are added to a network.
- Sort the units in order of increasing delay. This is the order in which the units are computed in the simulations.
- Modify the delays of the equivalent network so that for each connection $w_{ji,d}$ in the original network, the corresponding connection is $w'_{ji,d'}$, where $d' = d'_i - d'_j + d$, i.e., the difference in the two unit's delays is taken into account. It is easy to see from the construction of the unit delays that no connections in the equivalent network can have negative delay (look-ahead).
- Target values for output units are delayed by the respective unit's delays.

Table 1 Algorithm for converting a feed-forward network with look-ahead connections to one with delayed targets. A check is also made that the network is indeed a feed-forward one; if the variable 'count' grows larger than the number of units in the network, there must be some loop in the network that allows a units' activity to depend on its own value, and the network is therefore not a feed-forward network.

The equations of section 2.1 must be generalized to take dynamic connections into account. In the RTDNN framework, (2), (5) and (6) are generalized to

$$\begin{cases} a_{i,t} = \tanh\left(net_{i,t}\right) & \text{if unit } i \text{ is a tanhyp unit} \\ a_{i,t} = net_{i,t} & \text{if unit } i \text{ is a linear unit} \\ a_{i,t} = 1.0 & \text{if unit } i \text{ is the bias unit} \\ a_{i,t} = clamped & \text{if unit } i \text{ is an input unit} \\ a_{i,t} = \exp\left(net_{i,t}\right) & \text{if unit } i \text{ is an exponential unit} \\ a_{i,t} = 1/net_{i,t} & \text{if unit } i \text{ is an inverter unit} \\ a_{i,t} = prod_{i,t} & \text{if unit } i \text{ is a multiplication unit} \end{cases} \tag{12}$$

$$net_{i,t} = \sum_j w_{jid} a_{j,(t-d)} \tag{13}$$

$$prod_{i,t} = \prod_{j,\ \text{unit } j \text{ feeds into unit } i} a_{j,(t-d)} \tag{14}$$

where w_{jid} is the connection weight for the connection from unit j to unit i with delay d (possibly negative for look-ahead) and $a_{i,t}$ is the activity of unit i at time t.

2.3 Back-propagation through time

In the previous section the forward equations were extended to allow for time-delayed connections in a rather straight-forward fashion. In this section we focus on the backward equations, i.e., the computation of the derivatives of the objective function, E, with respect to the connection weights. Following the formalism of the previous section, (9), (10) and (11) can be generalized as follows:

$$\delta_{i,t} = -\frac{\partial E}{\partial net_{i,t}} = \begin{cases} \left(1-a_{i,t}\right)\left(1+a_{i,t}\right)backnet_{i,t} & \text{if } i \text{ is a tanh unit} \\ -\frac{1}{a_{i,t}^{\ 2}}backnet_{i,t} & \text{if } i \text{ is a inverter unit} \\ \exp\left(a_{i,t}\right)backnet_{i,t} & \text{if } i \text{ is a exponential unit} \\ backnet_{i,t} & \text{if } i \text{ is an input, linear, multiplication or the bias unit} \end{cases} \tag{15}$$

where

$$backnet_{i,t} = -\frac{dE}{da_{i,t}} + \sum_{\substack{j,d \\ \text{unit } j \text{ is not a} \\ \text{multiplication unit}}} \delta_{j+d} w_{ij,d} + \sum_{\substack{j,d \\ \text{unit } j \text{ is a} \\ \text{multiplication unit}}} \delta_{j+d} \frac{a_{j,t+d}}{a_{i,t}} \tag{16}$$

where the two sums together have one term for each connection flowing out from unit i. Further, the derivatives with respect to the connection weights are generalized to:

$$\frac{\partial E}{\partial w_{ij,d}} = -\sum_{t=t_0}^{t_1} \delta_{j,t+d} a_{i,t} \tag{17}$$

where t is the time index of activities and t_0 and t_1 are the boundaries of the sequence. Recall that in the network used in the computer simulations, all delays d are non-negative. This has the consequence that (15) describes a recursive set of equations where the δ's are computed in reverse order of time, t, hence the name back-propagation through time, (Rumelhart, Hinton and Williams, 1986; Pearlmutter, 1990).

A way to visualize back-propagation through time is to draw the spatial dimension of the network in one dimension, e.g., line up all units in one column. Then *unfold* the network in the time dimension, i.e., draw one column of units for each time point. Figure 2 shows a very simple example of such an unfolded network. The unfolded version of the network is structurally similar to a network with no delays but as many layers as there are time points. The important difference is that the connection weights are shared by all connections that correspond to the same connection in the original network. Back-propagation through time is equivalent to normal back-propagation with this additional constraint on the weights.

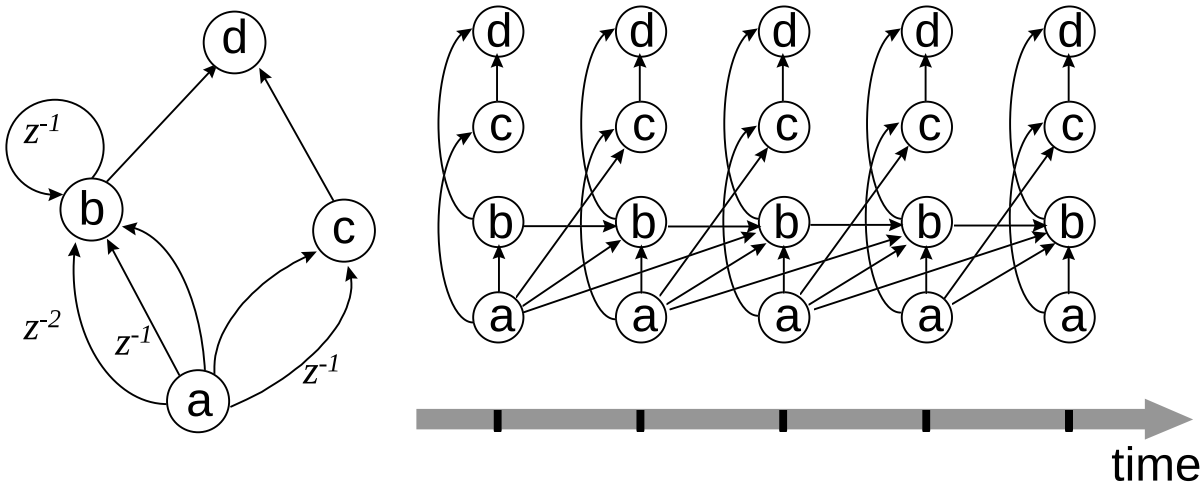


Figure 2. A simple dynamic network (left) and the same network unfolded in time (right) where the nodes a-d are duplicated for each time point. Arcs labeled z^{-x} indicates that the connection is delayed x time points. It is easy to see in the right figure that the network is feed-forward because all arcs flow from bottom left to upper right.

2.4 Weight updating scheme

Training an ANN using the back-propagation paradigm is an optimization problem, i.e., finding the set of connection weights that minimizes the objective function E. The backward equations (15) -(17), provide us with the derivatives of the objective function with respect to the connection weights which makes gradient descent methods feasible. However, gradient descent optimization is a very broad class of methods and it is the particular weight updating scheme (based on the gradient) that determines the success or failure of an implementation of the algorithm. In our experience, the classic stochastic weight updating scheme by Rumelhart, Hinton and Williams (1986), with some modifications, continues to be a good choice for problems with a large amount of training data. It can be written

$$\Delta w_{ij}{}^{(0)} = 0$$
$$\Delta w_{ij}{}^{(n)} = \eta\,\Delta w_{ij}{}^{(n-1)} + \gamma\frac{\partial E}{\partial w_{ij}}$$
$$w_{ij}{}^{(n+1)} = w_{ij}{}^{(n)} + \Delta w_{ij}{}^{(n)} \qquad \textbf{(18)}$$

where superscript *(n)* indicates a parameter after iteration number *n* and γ and η are the gain and momentum parameters respectively.

There are many well-known methods that utilize curvature information for the optimization. General optimization methods, e.g., Newton's method or conjugate gradient methods (see for example Luenberger, 1984) can be applied, as well as more or less specialized methods for ANN-training like QuickProp (Fahlman 1988), and application of Levenberg/Marquardt's method, Levenberg (1944), Marquardt (1963). However, it is non-trivial combine these methods with stochastic approximation algorithms, where weights are updated before the whole training data is processed (an epoch). In the simple updating scheme of (18), networks without delayed connections can be updated after every input/output pattern – so called pattern updating. Although the updating is based on an approximation of the gradient computed from only one pattern, the algorithm will still converge if the gain is small enough (and in many cases much faster than with epoch updating).

The picture becomes more complicated in the case of back-propagation through time because the derivatives depend not only on the current pattern, but on the whole sequence of patterns. We have adopted the approximate scheme to update the weights every *N* frames, i.e., approximate the derivative based on sub-sequences of the training data. This method has also been used by Robinson (1994) and is described in more detail in Table 2. The gradient computed in this manner is not only an approximation because it is based on a small number of time-points – it is also approximate because the δ's of (15) actually depend on the unit activities at *all* following points (not just the ones computed so far). The approximation is clearly worse if the weights are updated more frequently, but on the other hand it is desirable to update the weights as often as possible to speed up the process. In the simulations of this study, weights are updated every 20-30 time points. This choice was made after some preliminary experiments and is intuitively reasonable as it corresponds roughly to the length of a syllable. It is also a number similar to that used by Robinson (1994). The exact number of frames between each update is chosen randomly from a square distribution between 20 and 30, this has the effect that the points of update are different from epoch to epoch.

1) Let $t_0 = 1$
2) Let *step* be an integer randomly chosen from the square distribution [20, 30].
3) Let $t_1 = t_0 + step$
4) Compute unit activities using (12) and (13) with the given t_0 and t_1
5) Compute derivatives backwards from t_1 to t_0 using (15) - (17).
6) Update connection weights according to (18).
7) Let $t_0 = t_1 + 1$
8) Go to step 2

Table 2. Weight updating scheme for back-propagation through time. The random step-length in step 2 makes the update points differ from epoch to epoch.

We have still not discussed the choice of the parameters γ and η. An oversight in the famous work by Rumelhart, Hinton and Williams (1986) is that they let γ and η be constants. It is well-known from statistics theory that back-propagation training with stochastic updating, converges to a local minimum of *E*, only if a few constraints on the decay of the gain parameter are fulfilled. An survey of results in the statistical analysis of ANN learning schemes is given by White (1989). In section 4.2 of his survey, the statistical properties of stochastic updating is discussed and necessary conditions for convergence is given. In another study, Juang and Katagiri (1992) give the following conditions for convergence:

$$\sum_{n=0}^{\infty} \gamma^{(n)} = \infty \qquad \textbf{(19)}$$

$$\sum_{n=0}^{\infty} \left[\gamma^{(n)}\right]^2 < \infty$$

where superscript *(n)* again indicates the parameter after iteration number *n* (e.g

Also in practice is it fruitful to let the gain parameter decrease during the optimization. We have combined this feature with cross-validation in a manner similar to Bourlard and Morgan (1993). The idea is to decrease the gain parameter every time the objective function fails to decrease on the validation set. To be more specific, the training data is partitioned into a training set and a smaller validation set. The training set is used for back-propagation training with weight updating according to (18), but after each epoch, the objective function is computed for the validation set too. This is done using only the forward equations, i.e., without updating the weights. The objective function for the validation set is recorded for each epoch and whenever it fails to decrease, the gain parameter γ is multiplied by a constant factor $\alpha < 1$. In this study α is always 0.5.

We appreciate that the decay of the gain parameter and the cross validation procedure are two separate concepts and that it would therefore be more elegant to control them independently. However, in practice the two are closely related, and the described strategy have resulted in fast accurate optimization for the phoneme probability estimation task.

The momentum parameter, $\eta \in [0, 1]$, controls the smoothing of the gradient estimates, and can have a considerable effect on the convergence rate. In the simulations presented below, η is always 0.7.

2.5 Weight initialization

It is clear that the initial values of the connection weights are important for the performance of the back-propagation training. The algorithm finds one particular local minimum of the objective function, and the particular minimum found depends heavily on the starting point in the search space, i.e., the initial connection weights. It is common practice to initialize the weights to small random numbers (e.g., Fahlman, 1988). This implies that sigmoid and tanhyp units operate in the linear region of the non-linearity. In our experiments, the weights are initialized to square distributed random numbers [-0.1; 0.1] (but see also section 3.2).

2.6 Interpretation of the output activation values

It seems intuitively clear that reducing the error function *E* improves the classification performance of the ANN. However, it is essential for the understanding of the ANN classifier to formalize this notion. For our needs, it seems sound to define the best possible classifier to be the Bayesian discriminant function. Any function that implements the classification procedure: “associate the input observation with the class that has the highest *a posteriori* probability”, constitutes the Bayesian discriminant function. One obvious implementation is to accurately estimate the *a posteriori* probabilities for each class and then select the most probable class. In this case, the degree to which a classifier succeeds in its task depends on the accuracy in the estimation of the *a posterioris*. Justifying interpreting the output activities as *a posteriori* probabilities is fundamental for the theoretical foundation of the hybrid ANN/HMM speech recognition paradigm discussed in section 3.5.

It was proved early on that training networks with the mean square error (MSE) objective function is equivalent to minimizing the MSE of the *a posterioris*. Duda and Hart (1973) formulated the proof for the simple perceptron and it was later extended to multi-layer perceptrons by a number of authors: Baum and Wilczek (1988), Bourlard and Wellekens (1988), Richard and Lippman (1991), and Gish (1990). The proofs are valid under the condition that the functional capacity of the network exceeds the functional complexity of the *a posterioris*.

In our simulations, the cross-entropy error function is used. An analogous relationship between the *a posteriori* probabilities and the output activities of networks trained with the cross-entropy objective function was given by Hampshire and Pearlmutter (1990). They show that the MSE objective function is just a special case of a class of "reasonable" error measures that yield networks with output units that converge to the *a posteriori* probabilities $P(c_i \mid o)$. Here we outline a simplified version of their proof, showing only that the cross entropy objective function is a member of the class of "reasonable" error measures.

An important concept of the proof is *prototypes*. The input vector space is partitioned into regions o_p, where $P(c_i \mid o \in o_p)$ is "essentially" constant. The regions are called prototypes. This modeling of the probability distributions is consistent with the limited resolution in the modeling of probability density functions due to finite amount of training data. To make the idea of prototypes more concrete, we note that the prototype concept is similar to that of vector quantization, often used in speech technology applications. In this analogy, prototypes correspond to entries in a quantization code-book.

If we consider one particular class c_i and let N be the total number of samples in the training data, we can write the contribution of the error from output unit i as:

$$E_i = \frac{1}{N}\sum_{t=1}^{N} e_{i,t} \qquad \textbf{(20)}$$

where $e_{i,t}$ is the contribution from unit i and sample t. Now, let us sum over prototypes instead of samples; let P be the number of prototypes, N_p the number of samples from prototype number p, and n_p the number of samples from prototype number p. belonging to class i. Recall that the target $\tau_{i,RWH}$ is 1 for the n_p samples belonging to class i, and 0 for the remaining $N_p - n_p$ samples. Inserting into (7), we get the following expression for the contribution of the class's unit to the error:

$$E_i = \sum_{p=1}^{P} \frac{N_p}{N} \left\{ \frac{n_p}{N_p} \log a_p + \frac{N_p - n_p}{N_p} \log\left(1 - a_p\right) \right\} \qquad \textbf{(21)}$$

where $a_p = a(o_p)$ is the output activity of the class's unit in the RHW domain of prototype p. The asymptotic behavior as $N \to \infty$ is

$$\lim_{N\to\infty} E_i = \sum_{p=1}^{P} P\left(o_p\right)\left\{ P\left(c_i \middle| o_p\right)\log a_p + \left(1 - P\left(c_i \middle| o_p\right)\right)\log\left(1 - a_p\right)\right\} \qquad \textbf{(22)}$$

where we have replaced the expressions for relative frequency in (21) with probabilities (law of large numbers). A necessary and sufficient condition for local optimization is that the gradient of E with respect to a_p is zero for all prototypes. For class i we get:

$$\left|\nabla_{a_p} E_i\right| = 0 \Rightarrow \frac{P\left(c_i \middle| o_p\right)}{a_p} - \frac{1 - P\left(c_i \middle| o_p\right)}{1 - a_p} = 0 \Rightarrow a_p = P\left(c_i \middle| o_p\right) \tag{23}$$

Thus, the output activities of the network asymptotically approximates the *a posteriori* probabilities.

The networks of our simulations are typically large with high functional capacity and the training database is fairly large, so properly trained networks should be able to estimate $p(c_i \,|\, o)$ with good accuracy. However, given the complexity of the training algorithm with several different approximations and *ad hoc* parameters, an empirical study is called for. In section 5 such an experiment is presented and it is shown that the output activities do indeed approximate $p(c_i \,|\, o)$ closely.

2.7 The “softmax” output activation function

The interpretation of the output activities as the *a posteriori* phoneme observation probabilities established in the previous section, leads to some concern about the appropriateness of the simple activation function of the output units. If the output units are the class probabilities and the classes cover the entire observation space, then their sum must clearly be exactly one. However, the simple local activation function used does not enforce this constraint, indicating that there is some redundancy in the model. This constraint can be enforced by normalizing the activities, yielding a more complex non-local activation function for the output values. A theoretically appealing normalization is the so called “softmax” activation function (Bridle 1989; Robinson, 1994; Bourlard and Morgan, 1993),

$$a_i = \frac{e^{net_i}}{\sum_{j=1}^{N} e^{net_j}} \tag{24}$$

where index j runs over the output units of all classes. Clearly the softmax function ensures that the output activities sum to one, but this is accomplished at the cost of some increase in complexity. In our framework, the softmax activation function is implemented using exponential units, multiplication units and an inverter unit as illustrated in Figure 3. The redundancy in the model can now be eliminated by removing all connections to one of the output units. It is easy to show that this yields an equivalent activation function to (24), by showing that adding a constant bias to all net_i has no effect:

$$\frac{e^{net_i + bias}}{\sum_{j=1}^{N} e^{net_j + bias}} = \frac{e^{bias} e^{net_i}}{e^{bias} \sum_{j=1}^{N} e^{net_j}} = \frac{e^{net_i}}{\sum_{j=1}^{N} e^{net_j}} \tag{25}$$

Because of its theoretical appeal, the softmax function is used in the simulations, but it should be noted that the performance increases only marginally. Networks trained with softmax tend to converge to globally more optimal local minima (lower E), probably because low probabilities are modeled better, but we have seen no improvement in phoneme recognition experiments and only a small decrease in word error-rate.

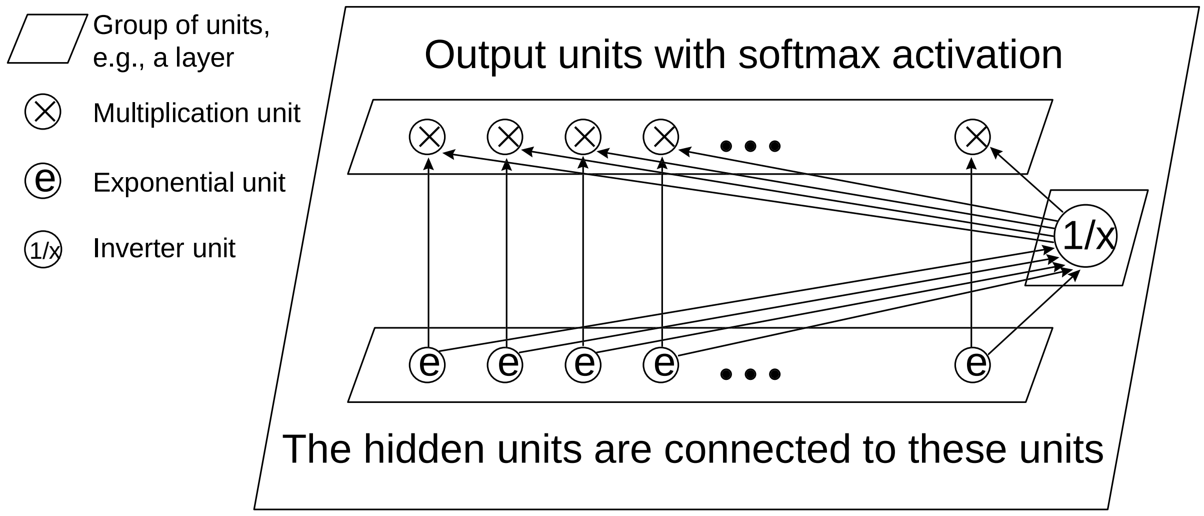

Group of units, e.g., a layer
Multiplication unit
Exponential unit
Inverter unit
Output units with softmax activation
1/x
The hidden units are connected to these units

Figure 3. Sub-network implementing the softmax activation function. The units in the top layer computes the activities of (24). This sub-network is regarded as one single layer to other layers of the ANN. Connections from hidden units to this output layer are connected to the lower layer only. The internal weights of the sub-network have the constant weight 1.0 and are not changed during training.

3 Phoneme probability estimation

In this section we show how the theory and algorithms of section 2 are applied to the particular task of estimating the phoneme probabilities given acoustic input.

3.1 Input feature representation

The raw speech waveform is not well suited for input to an ANN classifier. Therefore, it is common practice to transform the input speech signal to the short-time frequency domain before feeding it to the phonetic classifier. The transform used in our experiments computes a standard variation of the Mel cepstrum coefficients. Readers familiar with the HTK toolkit (Young *et al.*, 1995) will notice that the features used here are almost identical to those of its feature extraction tool. The procedure is discussed more elaborately in (Ström, 1996). Here we give only an outline of the procedure.

The speech signal is divided into short overlapping frames as shown in Figure 4. The frame-rate is 100 frames per second. The DC offset in each frame is removed and pre-emphasis is applied to the signal. The signal of each frame is then Hamming windowed and padded with zero-valued samples to the nearest power of two samples, and the FFT transform is applied to get the magnitude spectrum. In Figure 4, the different windows are shown together with an example of a speech signal.

The magnitude spectrum of the FFT is a frequency-domain representation of the speech in the frame as desired, but it is still not well suited as input to the ANN. The high frequency resolution of the spectrum gives input vectors of high dimension, which requires unnecessarily many weights to estimate in the ANN. This in turn leads to weak estimates and poor performance. For this reason, the spectrum of each frame is mapped to a more compact representation by a compressing transform. This is done in two steps: first a filter-bank is applied to the FFT spectrum and then the cosine transform is applied to the vector of filter-bank outputs. The filter-bank consists of a number of overlapping triangular, equidistantly spaced filters (see Figure 4) on the perceptually motivated *Mel* frequency scale (Schroeder, Atal and Hall, 1979). We use 24 Mel spaced filters covering the frequency range 0-8000 Hz. The cosine transform is applied to the filter-bank vector, yielding the cepstrum coefficients. The first twelve cepstrum coefficients are used, and together with the logarithm of the energy of the frame they constitute the feature representation that is fed to the ANN classifier as the input observation.

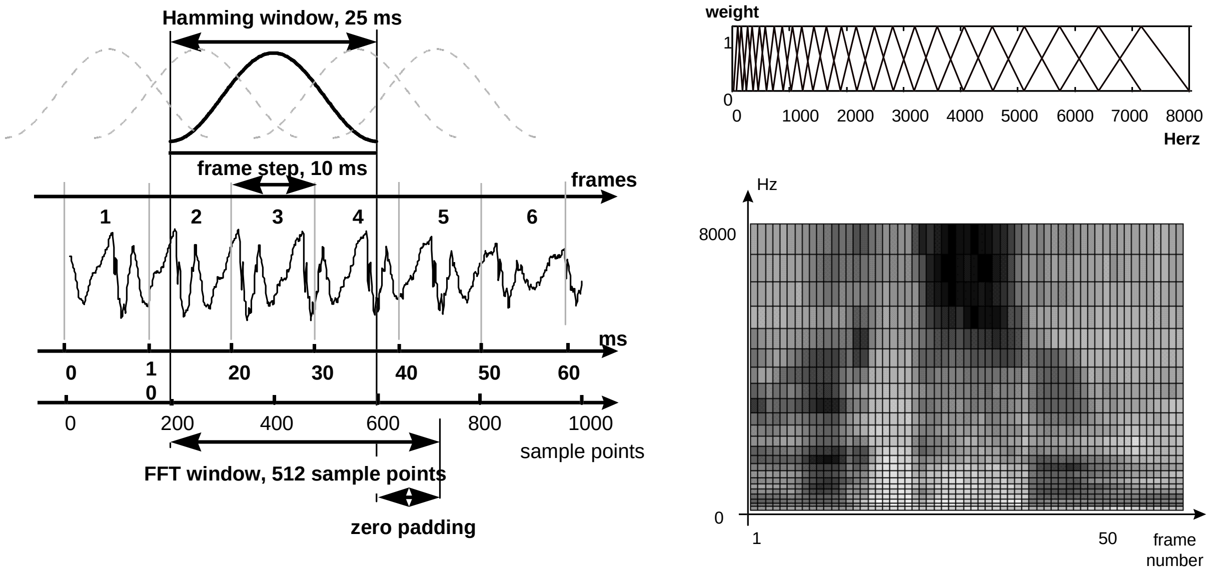


Figure 4. Left: Signal processing time constants and a sample speech signal. The three different time axes are 1) sample points, the sample frequency is 16k Hz, 2) ms and 3) frames. The features of a frame is computed from the signal convoluted with a Hamming window of 25 ms giving an effective window of about 10 ms.

Top right: Mel scaled filter-bank with triangular filters.

Bottom right: Illustration of the resulting time and frequency resolution in a spectrogram-like plot. The uttered (Swedish) word is "Waxholm" (vakshɔlm).

3.2 Delta coefficients

It is common practice to include the first and second time-derivatives (delta coefficients) of the cepstrum coefficients in the input vector. In the case of the standard HMM-model, this mainly serves the purpose of introducing some dynamic information to the classifier. For dynamic ANNs, the reason for including delta coefficients is less clear. In fact, a TDNN with a time-delay window of five frames can "learn" to extract delta coefficients from the input vector during training as they are simply linear combinations of the input. One could therefore argue that, if delta coefficients are productive for the classification performance, they will evolve in the network during training, and there is therefore no need for explicitly supplying them. However, this is true only in the ideal situation that there is an unlimited amount of training data and that the optimization algorithm is perfect, i.e., finds the global optimum regardless of the initial values of the parameters.

If we assume that the delta parameters are good features for representing dynamic information, they have two advantages over the brute-force method of widening the time-delay window to take dynamic features into account. First, the derivatives are a more compact representation of the dynamics than a window of input frames, leading to fewer parameters to estimate and therefore more robust estimates. Second, the delta coefficients can be seen as a particularly good initialization of connection weights, leading to a more well behaved optimization. The interpretation of delta coefficients as a choice of initial weights becomes more clear when we describe how delta and delta-delta coefficients are implemented in the networks. Because the delta coefficients are linear combinations of the existing input activities, they can be modeled by linear units in the network. Thus, the delta coefficients are linear units with four in-flowing connections with weights clamped to the values in the following formula

$$d_t = \frac{1}{10}\left(2c_{t+2} + c_{t+1} - c_{t-1} - 2c_{t-2}\right) \qquad \textbf{(26)}$$

where d_t is the activity of the delta coefficient unit and c_{t-x} are the activities of the original input unit delayed x frames. Equation (26) can be derived from linear regression and is equivalent to the default delta coefficients of the HTK toolkit (Young *et al.*, 1995). Second order time derivatives, so called delta-delta parameters, are computed in the same manner by applying (26) again to the delta coefficients. Figure 5 illustrates the network implementation of the delta and delta-delta units.

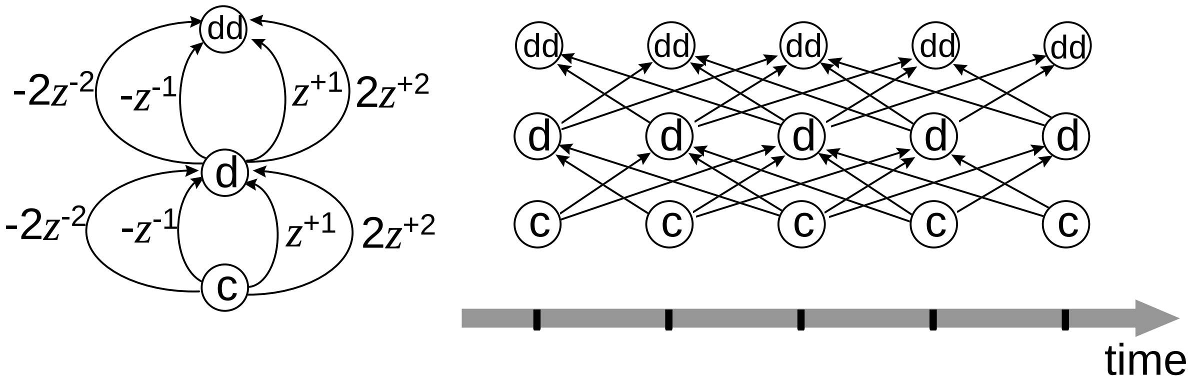


Figure 5. Implementation of delta and delta-delta coefficients in a network. The unit marked **c** *is a cepstrum coefficient input unit. The units marked* **d** *and* **dd** *are the corresponding delta and delta-delta units. The weights of the linear units implementing* **d** *and* **dd** *are clamped according to (26).*

3.3 Input normalization

In section 2.5 we saw that a well-chosen weight initialization scheme is helpful for finding good local minima. For the same reason it is also desirable that the activities of the input units are of similar magnitude as the other units of the network. This is achieved by applying a linear normalization to the input values. The coefficients of the linear transform can be determined in different ways, but we have chosen to base the normalization on the mean and standard deviation of the input values in the training database. After collecting second order statistics, it is easy to linearly transform each input to a variable with zero mean and a controlled variance. In the simulations, we enforced a standard deviation of 1.0 for all input units.

The normalization was applied to the input units using a special normalization step that is performed only for input/output units. The delta and delta-delta units that are not input units to the network (but play a similar role) are also normalized by adding a connection from the bias-unit for the constant offset, and scaling the other connections appropriately. The connection weights of connections flowing into the normalized delta and delta-delta units are then clamped and are not altered in the back-propagation training. After this normalization, the activities of the input units and the delta and delta-delta units all have mean zero and standard deviation one.

3.4 Network topology

The topology of the network, i.e., the number of hidden units and the manner in which they are connected, determines the functional capacity of the classifier. It has been proved (e.g., Hornik, Stinchcombe and White, 1989) that ANNs with one layer of hidden units can approximate with arbitrary precision any smooth function on a compact domain. However, this is a theoretical result that requires that the function is completely known and that the number of hidden units is unbounded. Neither of these conditions can be fully satisfied in our problem of phoneme probability estimation, but it still gives some guidance. Although it is possible that a network topology with more than one hidden layer could utilize the connection parameters in a more efficient manner to estimate the probability functions, we have limited the experiments to networks with one hidden layer and varied only the number of hidden units. The main motivation for this decision was to reduce the number of configurations to evaluate in the computationally rather costly computer simulations.

The input units, the delta and the delta-delta units are connected to the hidden units with a skewed time-delay window with dynamic connections ranging from five frames look-ahead to one frame delay. The motivation for the skewed window is that the hidden units additionally have recurrent connections between each other, with time-delay ranging from one to three frames. The recurrent connections provide the hidden units with additional information about the state of the network at past frames, and therefore the delayed side of the time-delay window can be smaller than the look-ahead side. Finally, the output units are connected to the hidden units with a symmetric time-delay window from plus one to minus one frame. The network topology is illustrated in Figure 6.

The total number of connections in the network, determining the computational effort needed for computer simulations, can now be computed. The number of connections implementing the delta and delta-delta parameters is small (104) compared to the connections to and from the hidden layer. There are 13 input units and therefore 3×13=39 units in the lowest layer (see Figure 6). Further, let there be *N* units in the hidden layer and 61 units in the output layer (the number of phonemes in the TIMIT database). This yields

$$104 + 39\times N\times 7 + N\times N\times 3 + N\times 61\times 3 = 3N^2 + 456N + 104 \qquad \textbf{(27)}$$

connections in total. As an example, the moderate number of 300 hidden units gives 406,904 connections – a respectable number that implies a substantial computational effort for computer simulations. This problem is addressed in section 4 where we show how it is possible to reduce the number of connections without decreasing the number of hidden units.

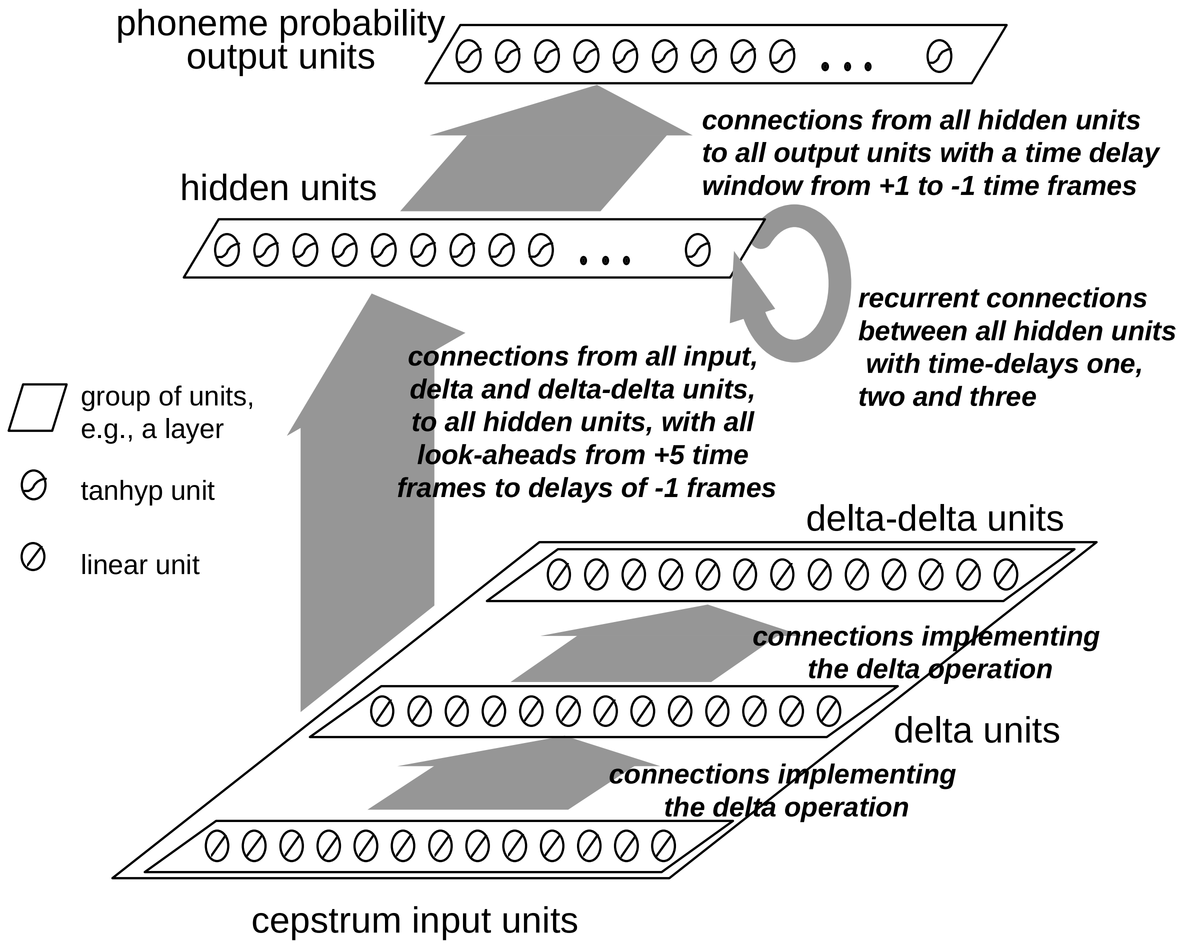

phoneme probability output units
connections from all hidden units to all output units with a time delay window from +1 to -1 time frames
hidden units
recurrent connections between all hidden units with time-delays one, two and three
connections from all input, delta and delta-delta units, to all hidden units, with all look-aheads from +5 time frames to delays of -1 frames
group of units, e.g., a layer
tanhyp unit
linear unit
delta-delta units
connections implementing the delta operation
delta units
connections implementing the delta operation
cepstrum input units

Figure 6. The network topology of the ANNs. The parallelograms indicate layers, and arrows between layers represent sets of connections from the units in one layer to units in another. Because of the dynamic nature of the networks, there are typically several connections with different time-delay or look-ahead between two connected units. The layer of output units is sometimes replaced by a group implementing the softmax activation function (see section 2.7).

3.5 Dynamic decoding

The ANNs described so far classify 10 ms frames of audio-input into one of the phoneme classes. In the dynamic decoding step of the ASR, this frame-based output from the network is used to find the (in some sense) optimal sequence of phonemes or words. The dynamic decoding of the system used in this study is described in more detail in (Ström, 1996). To summarize this study, the well-known hybrid HMM/ANN paradigm (Bourlard and Wellekens, 1990) is adopted, where the output activities are interpreted as the *a posteriori* phoneme probabilities, $p(c_i \mid o)$ (see section 2.6). The observation probabilities, $p(o \mid c_i)$, are derived from the *a posteriori* phoneme probabilities using Bayes's rule. In the case of tanhyp output units, it is necessary to normalize to the RHW range (see section 2.1, Equation (3). Thus, the observation probabilities can be written:

$$p(o|c_i) = \frac{p(c_i|o)}{p(c_i)} p(o) \approx \frac{a_{c_i}+1}{2} \frac{p(o)}{p(c_i)} \quad \text{for tanhyp output units} \tag{28}$$

$$p(o|c_i) = \frac{p(c_i|o)}{p(c_i)} p(o) \approx a_{c_i} \frac{p(o)}{p(c_i)} \quad \text{for the softmax activation function}$$

where $p(c_i)$ are the *a priori* class frequencies that are estimated off-line from the training data and a_{c_i} is the activation of the output unit for phoneme i (with range [-1, 1]). The unconditioned observation probability, $p(o)$, is constant for all classes and is therefore dropped in the computations.

The Markov model for a phoneme is shown in Figure 7. The observation probability is constant for all transitions of the phoneme model, and the transition probabilities are maximum likelihood (ML) estimates from the durations of the training database. In addition to the coarse duration model imposed by the transition probabilities, a phoneme-dependent minimum duration constraint is used. It is implemented by adding extra nodes to the HMM and putting a self-loop on the last node only, as shown in Figure 9. The minimum duration, m frames, for each phoneme is selected such that about 5% of the phones in the training data are shorter than m frames. The fraction 5% was chosen, after some experimenting, to optimize the recognition performance. However, the improvement is very small for phoneme recognition.

A probabilistic word-class bi-gram grammar is used in the word-level evaluation and a phoneme bi-gram grammar is used in the phoneme recognition evaluations.

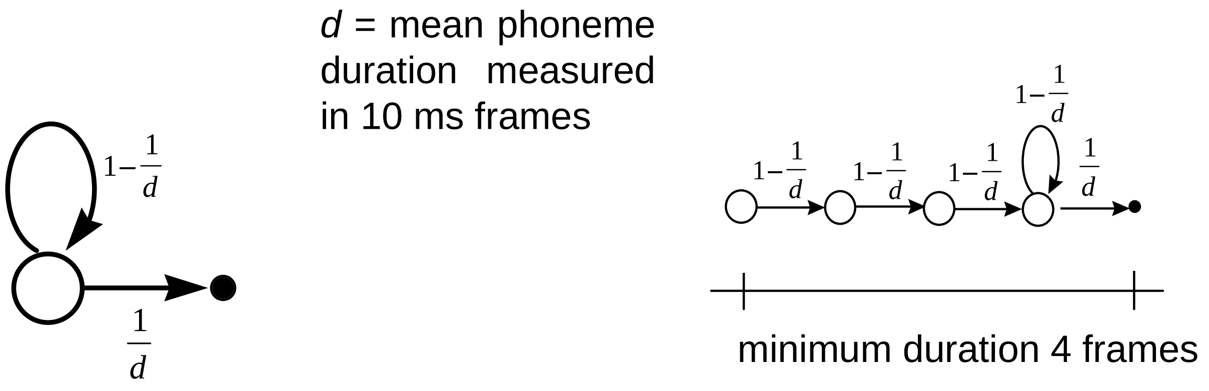


Figure 7. Phoneme HMM. Left: One-state HMM with transition probabilities expressed in mean duration. It is easy to show that the indicated probabilities are the ML estimates of the model parameters. The mean duration is computed from the phones of the training data. Right: Phoneme HMM with minimum duration constraint. See the main text for details.

4 Pruning and sparse connection

The number of hidden units determines to a large extent a network's ability to estimate the *a posteriori* probabilities accurately. It is therefore desirable to experiment with networks with a large hidden layer. Unfortunately, as was illustrated by (27), the number of connections grows rapidly with the number of hidden units, and that number is practically bounded by the available computational resources. One could argue that it is the number of free trainable parameters in the network that is the important factor. In this view, it is not the number of units, but the number of connections that is important. However, from experience we know that not only the number of parameters is important, but also how they are put to use.

In an attempt to evaluate the efficiency of the fully connected structure that leads to (27), we studied the statistical properties of the magnitudes of the connection weights. The idea is that the weight is an indication of the salience of an individual connection in a trained network (this is elaborated in section 4.1). Figure 8 is a histogram of the connection weights in a typical ANN after back-propagation training. The notable concentration of weights close to zero inspired us to experiment with sparsely connected networks and connection pruning schemes for trained networks.

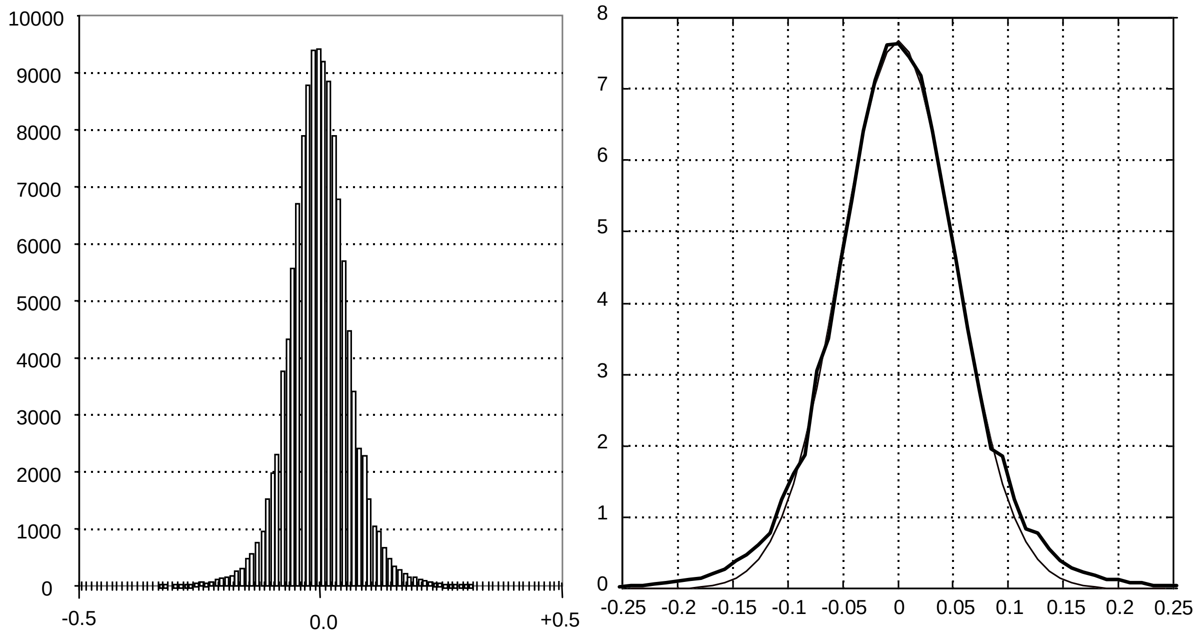


Figure 8. Left: histogram of the connection weights in a typical network with about 123,000 connections. The x-axis shows the connection weight and the y-axis is the number of connections in each bin. Right: the histogram converted to a probability distribution plot. The thin line is a Normal distribution with mean zero and standard deviation 0.052. The close agreement clearly indicates that the weight distribution is approximately normal for small weights

4.1 Connection Pruning

If the assumption that weights of small amplitude are suspendable is correct, it can be seen in Figure 8 that many connections can potentially be removed. This would greatly reduce the computational effort for running trained networks. Since pruning reduces the number of free parameters, it can also improve the network's generalization ability (e.g. Le Cun, Denker, and Solla, 1990a; Sietsma and Dow, 1991), but because we use a truncated training scheme with a validation set (recall section 2.4), this potential benefit is less important in this study. Several different criteria for selecting connections for deletion have been suggested in the literature. An overview of the methods can be found in (Thimm and Fiesler, 1995). The most well known are smallest variance (Sietsma and Dow, 1991) and optimal brain damage (OBD) (Le Cun, Denker, and Solla, 1990a). The OBD method is based on a local approximation of the contribution of individual weights to the objective function. If the Hessian is assumed to be diagonal, i.e., the mixed derivatives between weights are neglected, the Mc Laurin series of E with respect to a weight w can be written

$$E = E_o + w\frac{\partial E}{\partial w} + \frac{1}{2}\frac{\partial^2 E}{\partial w^2}w^2 + O(w^3) \tag{29}$$

where E_0 is a constant independent of w. Close to minima, the gradient is small so the second term of (29) is approximately zero. Thus, if $|w|$<<1, the contribution of w to E is dominated by the third term. Le Cun, Denker and Solla show how the diagonal Hessian can be computed with the same algorithmic complexity as computing the gradient in a recursive procedure similar to the back-propagation algorithm. The connection pruning is then governed by the "salience", $0.5 \times \partial^2 E/\partial w^2 \times w^2$, of each weight. An overview of methods for computing second derivatives in feed-forward ANNs can be found in (Bundine and Wiegend, 1994).

There are three approximations active in the OBD method: the off-diagonal terms of the Hessian, the higher-than-quadratic terms and the gradient's deviation from zero. The influence of the gradient is easily handled because the gradient is known from the back-propagation, and consequently the second term of (29) does not need to be approximated. The other two approximations however, restricts the algorithm to operating on small weights.

In the simulations of this study, we base the pruning on a more simplistic salience-measure using only the information gathered in the back-propagation computations, i.e., the weights and the gradient. The pruning implemented in the toolkit of the Appendix, is controlled by the two parameters α and β in the following rule

$$\text{Remove the connection with weight } w \text{ iff } \begin{cases} |w| < \alpha \quad \alpha << 1 \\ \text{and} \\ \left| w\frac{\partial E}{\partial w} \right| < \beta \end{cases} \tag{30}$$

where sensible values for α are lower than 0.10, and β is dependent on the network and the training data, but can be tuned to get the desired pruning aggressiveness. However, experiments have indicated that the second constraint is much weaker than the first. The same amount of pruning and almost identical network performance is achieved by just considering the absolute value of the weight. Therefore, we use only the first criterion in the evaluations of section 5 (or equivalently, we set $\beta = \infty$). Thus, weights are simply removed if $|w| < \alpha$. In this case, it is α that is tuned to get the preferred amount of pruning.

The probabilistic descent method used for weight updating produces weights that are the sum of many small updating steps. Let us consider the possibility that small weights are merely the result of the weight updating procedure. If we assume that for small weights, individual updating steps are independent random variables with identical distribution, the resulting weights are Normal distributed (Central Limit theorem). Studying the weight distribution of Figure 8, it is seen that close to zero, the empirically found distribution can indeed be well approximated by a Normal distribution, but for weights with a magnitude greater than about 0.075, this is no longer true. We conclude that at least not all weights greater than 0.075 are mere artifacts of the training procedure – a useful observation for selection of the parameter α.

After pruning, the network is retrained to find the optimal weights of the new, more constrained network. The retraining converges much faster than the original training – the network is smaller, and typically a few epochs is enough. Although the main effect of the retraining is to correct for the disturbance from the deleted connections, a possible side-effect is that the ANN escapes a local minimum of the error function.

A weakness in the case made for OBD by Le Cun, Denker, and Solla is that the OBD approach is compared to, and found superior to, the simple magnitude-based method *before* retraining, but no comparison is reported after retraining. This is unsatisfactory since the effect of retraining is far larger than the reported difference between the two methods. The simulations in section 5 of this study show that, although the simple pruning criterion is used, the network performance after retraining can be equal to or even better than that of the original network.

4.2 Sparse connection

Pruning the connections of an already trained network has no impact on the computational effort for training. To be able to apply the pruning criterion to the connections, the weights of the fully connected network must first be trained. Therefore, a natural next step in our efforts to reduce the computational cost is to start the training with already sparsely connected networks. However, before training, there is no available information about which connections are salient, so a random set of connections must be selected. Of course, this is in general not the optimal set, but as will be illustrated in section 5, the resulting classifier may still be competitive.

A straight-forward random connection scheme is to consider all connections in a hypothetical, fully connected network and let each be a connection in the actual sparse network with probability ϕ (connectivity). The expected number of connections in the sparse network is then ϕN, where N is the number of connections in the fully connected network. A useful extension is to allow ϕ to be different for different sets of connections, e.g., the connections from the input units can be treated differently than the recurrent connections and the connections to the output units etc. Ghosh and Tumer (1994) point out that the sparse connectivity has the effect of decoupling the output units, i.e., all output units are not connected to the same hidden units. They report results from several comparative studies where sparsely connected networks perform as well as, and in some cases better than both OBD and fully connected networks. Because the decoupling effect is larger if connections in higher layers are more sparsely connected than lower layers, it is suggested that this is an advantageous scheme.

Our motivation for sparse connection is different from that of Ghosh and Tumer in that we focus on the reduced computational demands instead of the potential improved generalization ability. Because the number of connections is much larger than the number of units, the computational cost for both training and running the ANNs is proportional to the number of connections. From (27) we see that the number of connections between a fixed-size layer and the hidden layer, is proportional to the connectivity and the number of hidden units. Thus, it is possible to vary the size of the hidden layer, while keeping the number of connections fixed by imposing $\phi H = constant$, where H is the number of hidden units. The recurrent units are more problematic. By inspecting (27) again, it is seen that the number of recurrent connections is proportional to the square of the size of the hidden layer. Therefore, for large networks, the recurrent connections dominate the total number heavily. Imposing $\phi H^2 = constant$ would yield ANNs with varying hidden layer size and equal number of connections, but for large H, the connectivity would drop too fast. Initial experiments showed that, in this case the reduction of the network's functional capacity due to the very sparsely connected recurrent connections can not be compensated for by the increased number of hidden units.

To overcome the quadratic relationship between the layer size and the number of hidden units, localized connectivity is introduced. Recurrent connections are added with probability

$\mu \exp(-\mathbf{d}[u_1,u_2] / \sigma)$, where $\mathbf{d}[u_1,u_2]$ is the distance between unit u_1 and unit u_2, μ is the overall connectivity constant, and σ is a constant of "spread". Distance is simply defined as the difference in ordering number within the layer, e.g., the distance between the 5th and the 17th unit is 12. Thus, the self-loop connectivity is μ, and σ controls how fast the connectivity decays as a function of distance. In the simulations, we use $\mu = 1.0$ and σ is varied to control the number of connections. Figure 9 illustrates the connectivity for the case $\sigma = 20$.

Local connectivity has previously been used primarily for sets of units with an intuitively clear metric, e.g., in character recognition (e.g., Le Cun *et al*., 1990b) where the units correspond to positions on a two-dimensional surface, and other tasks related to vision. In these tasks the local connectivity is often combined with weight-sharing to impose plausible task-dependent symmetry conditions (Le Cun *et al*., 1990b). In the speech field, a metric can be defined by some scale on the frequency dimension in filter-bank representations of the input features. An experiment with local connectivity based on this metric is reported by Basu and Svendsen (1993), but the recognition accuracy is too low both for the baseline and the locally connected network for any definite conclusions. In contrast to the mentioned experiments, the hidden layer of our networks has no metric that reflects some structure of the problem. The intuition behind the local connectivity scheme in this case, is that sub-groups of units with many connections connecting the group, may promote the development of unit clusters that solves sub-tasks efficiently. The underlying assumption is that breaking up the classification problem into sub-tasks leads to better local minima. The assumption is not self-evident but, empirically we have found that the local connection scheme outperforms simple sparse connection when the total connectivity is low (see section 5).

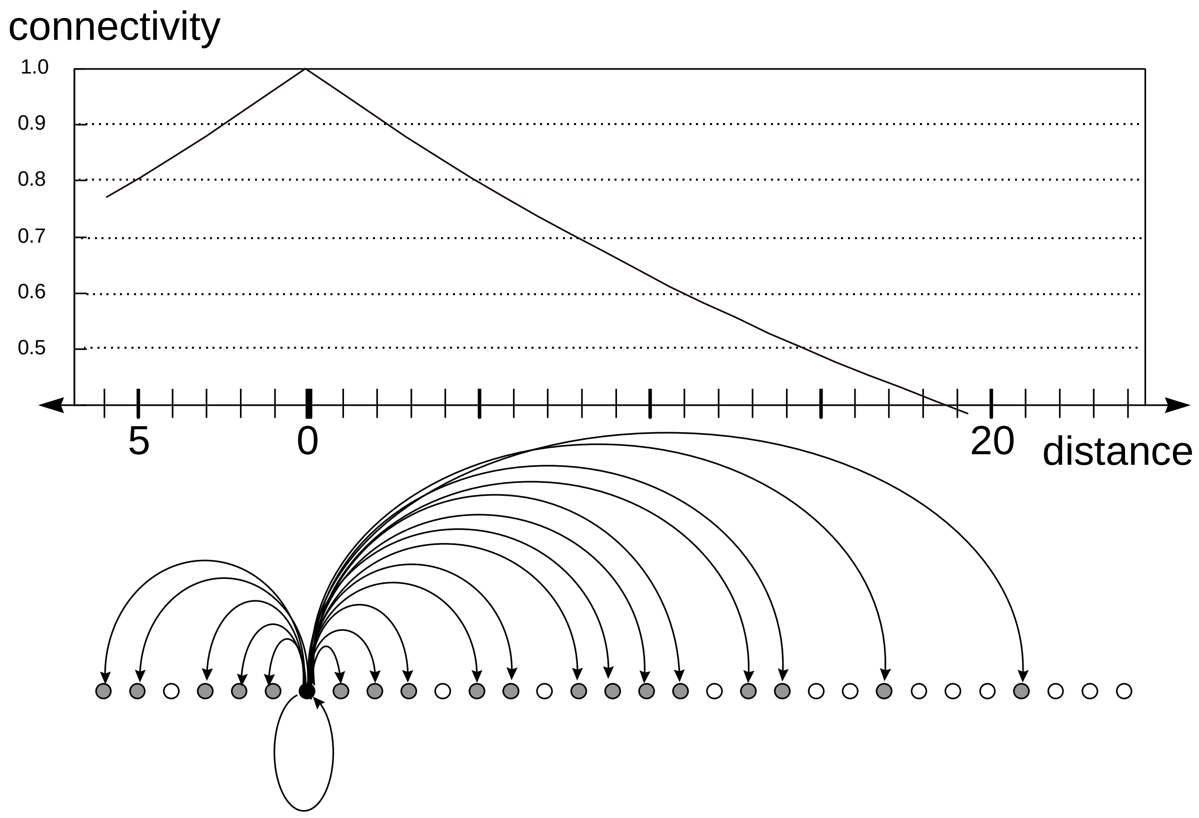


Figure 9. Connectivity in the hidden layer with localized connection rule. Connections are added with probability $\mu \exp(-\mathbf{d}[u_1,u_2] / \sigma)$. *Top: the connection probability with* $\mu = 1.0$ *and* $\sigma = 20$. *Bottom: sample layer with recurrent connections from one unit indicated.*

5 Recognition results

The phoneme probability ANNs have been evaluated in two different experimental environments. The system used in the evaluations was developed for speech recognition in the WAXHOLM human/machine dialog system (Blomberg *et al.*, 1993). The Swedish database collected in the development of that system (Bertenstam *et al.*, 1995a,b) is used for evaluation on the word level. Many aspects of the word recognition used in the WAXHOLM system are not covered in this paper, but the recognition module of the system is accounted for in depth by Ström (1996). In addition to the word-level evaluation, phoneme recognition on the well-known American English TIMIT database (Zue, Seneff and Glass, 1991) has been performed to be able to calibrate the performance against other systems. The decoder from the word recognizer of the WAXHOLM system was used also for the phoneme recognition experiments by simply assigning a word to each phoneme.

Word/phone error-rate is computed in the standard dynamic time warping (DTW) alignment fashion, i.e., the recognized string of symbols is optimally aligned with the correct string by minimizing an objective function that penalizes substitutions with the value 10 and deletions and insertions with 7. The error-rate is defined as the total number of substitutions (S), deletions (D) and insertions (I) divided by the number of symbols (N) in the correct string, i.e., (S + D + I) / N.

5.1 Phoneme recognition results on the TIMIT database

No other databases have been so thoroughly illuminated by different research sites using a multitude of methods for phoneme recognition, as the TIMIT database. Table 3 shows a few of the numerous reported recognition results for different methods.

The training and test partition of the database as well as the smaller core test-set is defined in the documentation distributed with the database. All training utterances except the so called "sa-sentences" were used for training. For evaluation, we chose to focus on the core test-set because it is better dialect and gender balanced than the full set.

The phoneme set to use for evaluation is not equally standardized. The 61 symbols of the database are sometimes considered a too narrow transcription for practical use, and are therefore collapsed into a smaller number of classes. In this study we perform evaluations using the 39 phoneme set defined in (Lee and Hon, 1989) that have evolved into an unofficial standard for phoneme recognition experiments. However, in conformity with the study of Robinson (1994), the full 61 symbol set is represented in the output layer of the ANN as well as in the phoneme bigram grammar of the dynamic decoding. After recognition, the recognized symbol string is mapped to the 39 phoneme set for evaluation.

The back-propagation training, described in detail in section 2, was terminated after 30 epochs for all networks. In all cases, the learning curve had leveled out after this amount of training (see Figure 13).

Author(s)	**Method**	**Full test set**	**Core set**
Lee and Hon, 1989	Continuous density HMM	33.9 %	
Digalakis, Ostendorf and Rohlicek, 1992	Stochastic segment model	36.0 %	
Lamel and Gauvain, 1993	Continuous density HMM		30.9 %
Goldenthal, 1994	Trajectory model		30.5 %
Robinson, 1994	Recurrent ANN	**25.0 %**	**26.1 %**
Mari, Fohr, and Junqua, 1996	2nd order HMM		31.2 %
Glass, Chang, and McCandless, 1996	Feature based recognition		30.5 %
Ström, 1997 (this paper)	**Sparse, Recurrent, Time-delay ANN**	**27.0 %**	**27.8 %**

Table 3. A few reported phoneme recognition results for the TIMIT database. Phone error-rates are given for the full test set and the core test separately. The core test set is slightly harder to recognize because of its more balanced dialect and gender distribution. The error-rate of the best network of this paper is lower than all other reported results except the recurrent network of Robinson (1994).

5.1.1 Experiments with varying connectivity

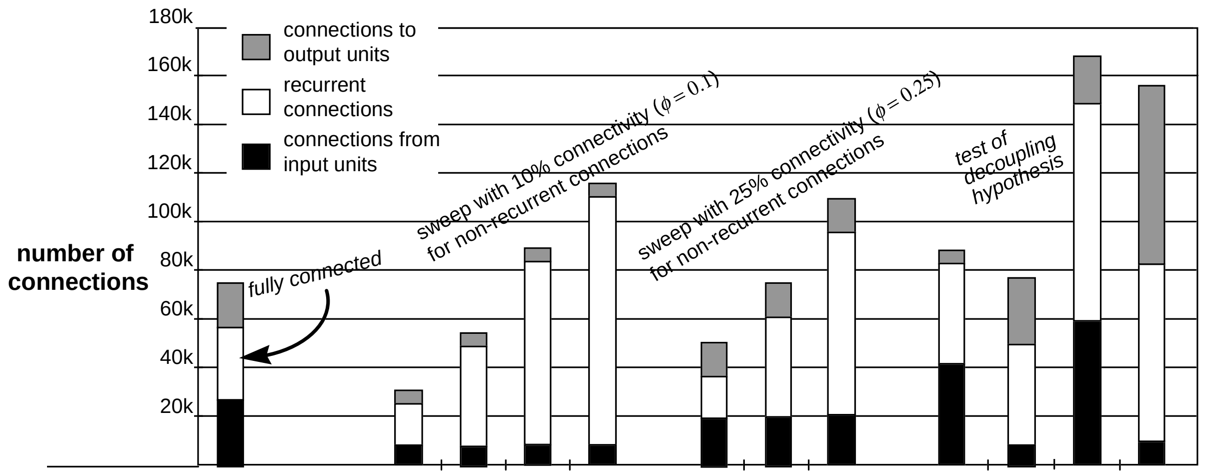
An ensemble of networks with the topology described in section 3.4, and varying connectivity and hidden layer size have been trained and evaluated. The connections were divided into three sets: *A*) the connections from the cepstrum feature units to the hidden units, *B*) the recurrent connections of the hidden layer and *C*) the connections from the hidden layer to the phoneme output units. Sets *A* and *C* were sparsely connected by randomly selecting a portion of the connections of a fully connected network (controlled by the ϕ parameter), and set *B* was connected using the localized connection scheme defined in section 4.2 (controlled by the σ parameter). Motivated by preliminary experiments, a hidden layer of 300 units was selected for a study of the effect of varying the connectivity. More hidden units give higher recognition results, but the price is longer training times, and it was assumed that 300 units is enough for making experiments that yield insights that scale up to larger networks. In Figure 10 the phoneme recognition results with 300 hidden units are shown.

A fully connected network with 100 hidden units is included in the figures for comparison, but it is clearly outperformed by the sparsely connected ANNs. Compare for example, the underlined error-rates in Figure 10, for two networks with the same number of connections.

The first two sweeps in Figure 10 show how the classification performance is affected by increasing the connectivity in the recurrent connections. It can be seen that the sweep with 10% connectivity (ϕ = 0.10) for the non-recurrent connections gives a better trade-off between performance and number of connections, indicating that the number of recurrent connections should optimally be relatively large compared to the other types of connections.

5.1.2 Decoupling the output units

The four rightmost columns of Figure 10 was designed to test the hypothesis of Ghosh and Tumer (1994), that the connections to the output (phoneme) layer should be more sparse than the lower layer, to decouple the output units. As can be seen, this is supported by our results – a fact that will be utilized in future experiments to design more efficient networks. In summary, the experiments with 300 units and varying connectivity indicates that high connectivity should be assigned to the connections from the input to the hidden units, low connectivity should be assigned to connections to the output units, but the number of recurrent connections should be relatively large.



hidden units	**100**	300	300	300	300	300	300	300	300	300	300	300
connectivityϕ to output units		10%	10%	10%	10%	25%	25%	25%	10%	50%	25%	100%
σ for recurrent connections		10	25	35	50	10	25	35	25	25	50	50
connectivityϕ from input units		10%	10%	10%	10%	25%	25%	25%	50%	10%	100%	25%
error-rate	36.1%	33.1%	31.0%	29.1%	28.9%	32.3%	30.5%	29.1%	28.7%	29.5%	**28.3%**	30.5%

Figure 10. Phone error-rate for an ensemble of networks with 300 hidden units and different connectivity. Error-rate is defined as the total number of insertions, deletions and substitutions per phone. A fully connected ANN with 100 hidden units is also included for comparison. By comparing the underlined results, it is clearly seen that the fully connected network is outperformed by its sparsely connected counterparts with an equal number of connections.

5.1.3 Varying the hidden layer size

To also probe into the hidden layer size dimension, a series of networks with identical connectivity, but varying number of hidden units, was trained and evaluated. The results of this sweep of simulations are shown in Figure 11. It is seen that the error-rate is steadily decreasing with increasing number of hidden units. This figure illustrates the most important advantage of sparsely connected ANNs – sparse connection schemes allow us to work with larger hidden layers, with accompanying reduction in error-rate, than otherwise possible. For comparison, a fully connected network with 600 units would have 1,353,704 connections (Equation (27)).

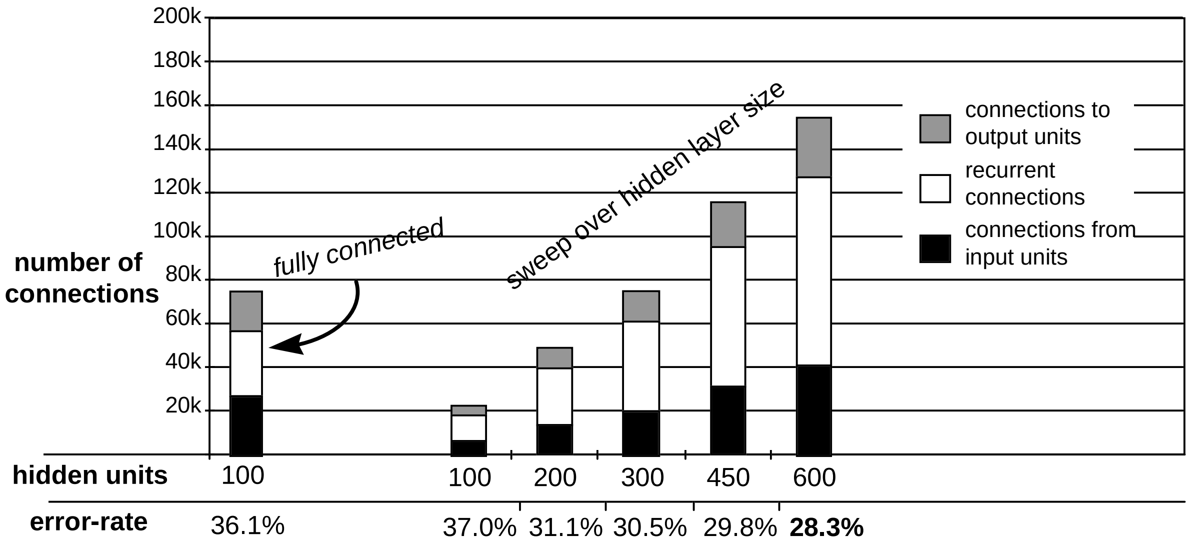


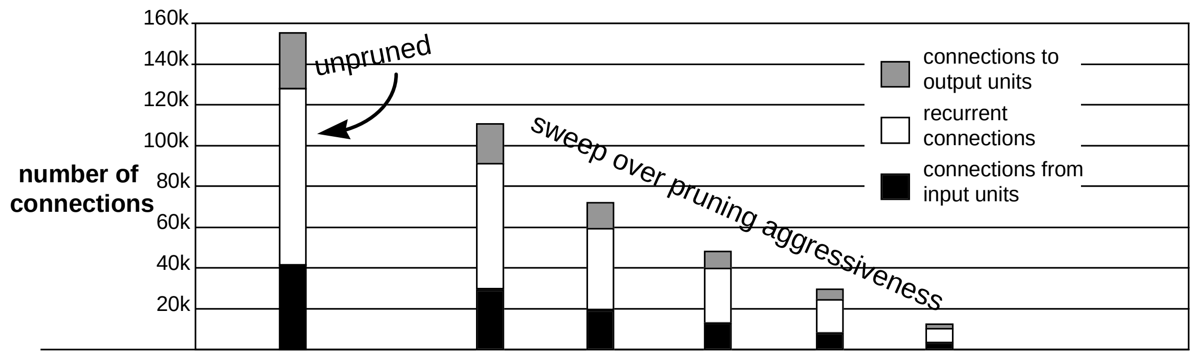
Figure 11. Phone error-rate for an ensemble of networks with varying hidden layer size. The connection probability is 25% ($\phi = 0.25$) for all non-recurrent connections and the connectivity parameter σ is 25 for all layer sizes. The fully connected network with 100 hidden units is included for comparison.

5.1.4 Connection pruning in trained networks

Connection pruning, as described in section 4.1, has no impact on the computational effort for training. Instead it adds an additional training phase for retraining the connection weights after pruning. However, it was found in our experiments that this training is much faster than the first phase – a few epochs is usually enough for convergence. Figure 12 shows the phone error-rate as a function of the aggressiveness of the pruning on the network with 600 units. It is interesting to compare the error-rate for different pruning thresholds with the probability distribution of Figure 8. Note that the distribution deviates from the Normal distribution for weights greater than about 0.075. It is also for pruning thresholds greater than 0.075 that the performance starts to drop significantly.

Pruning can in some cases increase classification performance because the reduced number of parameters can improve the network's generalization ability. The simulations show an improved performance after moderate pruning and retraining, but the cause is not clear; the classification performance increased on both the training and test data. It is possible that the cause of the improvement is that the disturbance due to the pruned connections caused the ANN to escape from a local minimum of the error function. The phone error rate of this pruned network is the best performing ANN of this study, therefore an evaluation on the whole TIMIT test set was performed with this network. The result, 27.0% error rate is included in Table 3.

An attractive feature of pruning is the reduced computational effort for running trained networks. In Figure 12 it can be seen that pruning with the weight magnitude criterion of section 4.1 and the parameter α set to 0.05 yields a network reduction of about 50% without reduction in performance. Because the number of connections is proportional to the computational cost of running the network, the corresponding speed-up can be very important in real-time recognition systems. It is notable that the most aggressively pruned network, with only about 13,000 connections, performs the phoneme recognition task more successfully than the fully connected network with about 75,000 connections.



hidden units	600	600	600	600	600	600
pruning threshold (α)		0.025	0.050	0.075	0.100	0.150
error-rate	28.3%	**27.8%**	28.2%	29.1%	31.4%	35.0%

Figure 12. Phoneme recognition result for the original ANN with 600 hidden units (also shown in Figure 12) of Figure 11, and for pruned versions after pruning and retraining.

5.1.5 Computational considerations

The original motivation for the experiments with sparse connection and connection pruning was to reduce the time for training and speed up the evaluation of the networks. The computational effort for performing the forward as well as the backward phase of the back-propagation algorithm is dominated by a term proportional to the number of connections in the network. Thus, the training time is proportional to the number of connections times the number of epochs needed for convergence, and the evaluation speed is proportional to the number of connections in the network.

The amount of computation for training different ANNs can only be compared in the context of some termination criterion. In this study, the training was always terminated after 30 epochs of training with the TIMIT training. In this case, the amount of computation is only dependent on the number of connections. A more sophisticated termination criterion may consider the rate of improvement per epoch. By inspecting Figure 13, we see that the training error function levels out after between fifteen and twenty-five epochs for all networks. Thus, comparison based on only the number of connections may be slightly too simplistic. However, the difference due to the number of connections is by far the most important factor. Compare for example the sparsely connected network, with 100 hidden units, with its fully connected counterpart. The training progress curves in Figure 13 are similar, but the fully connected network has more than three times as many connections.

The total training time for the ANN with 600 units and 155,000 connections was about 11 days on an HP C180 workstation. Training times for the other ANNs are in proportion to their number of connections.

The evaluation of the trained networks can be performed in real time or faster for all networks with less than 110,000 connections. For example, the best performing network of the study, the moderately pruned network in Figure 12 with 600 units and 109,000 connections, runs in 99% of real time on the HP C180 workstation.

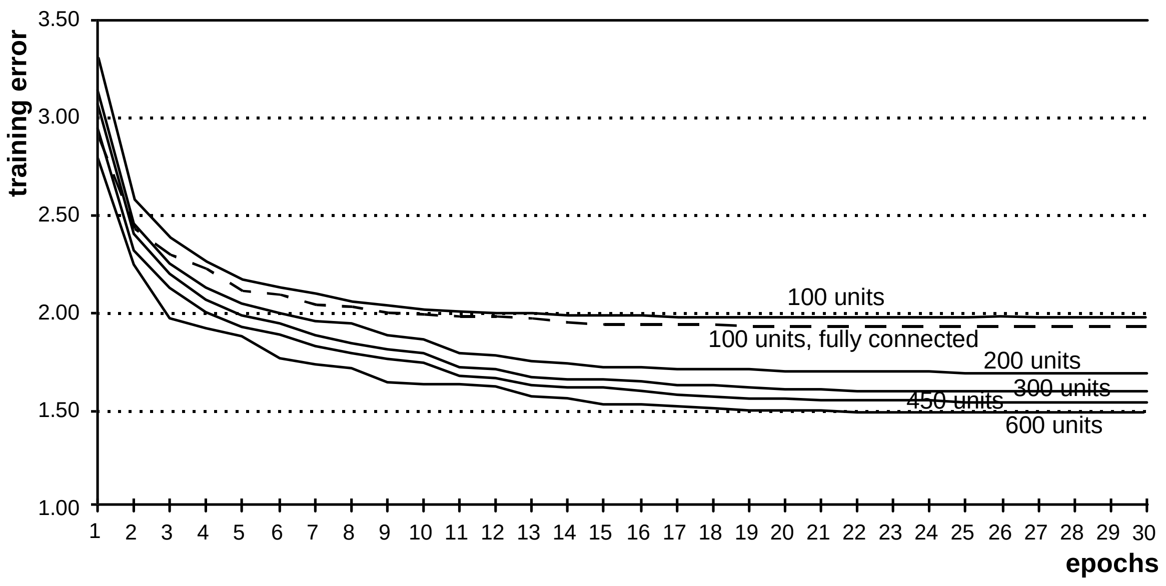


Figure 13. Training error as a function of number training epochs for six different networks. The dashed line indicates the training progress of the fully connected network with 100 hidden units, and the other five lines are the progress for the other ANNs with varying number of hidden units (the networks are the same as in Figure 11).

5.1.6 Test of the interpretation of activities as a posteriori probabilities

In section 2.6 it was shown that in theory, the output activities are estimators of the *a posteriori* phoneme observation probability. To test this empirically, statistics were collected for the activities of a typical ANN of this study. The *a posteriori* probability was estimated from the target values for bins of activity ranges. Thus, for each bin, the relative frequency was computed for each phoneme. Then the probability estimated by the ANN was plotted versus the relative frequency. Figure 14 shows the resulting histogram. The proximity to the Bayes classifier function supports the assumption that the activities can be interpreted as *a posteriori* probabilities. However, the ANN tends to assign too high probabilities when the actual *a posteriori* probability is above 0.3. This is a good illustration of a slight over-adaptation to the training data. The deviation for both sets at the lowest bin is a reflection of the large number of activities that have values much less than 0.05; the resulting observed relative frequencies of about 0.01 are artifacts caused by uneven distribution of the data within this bin.

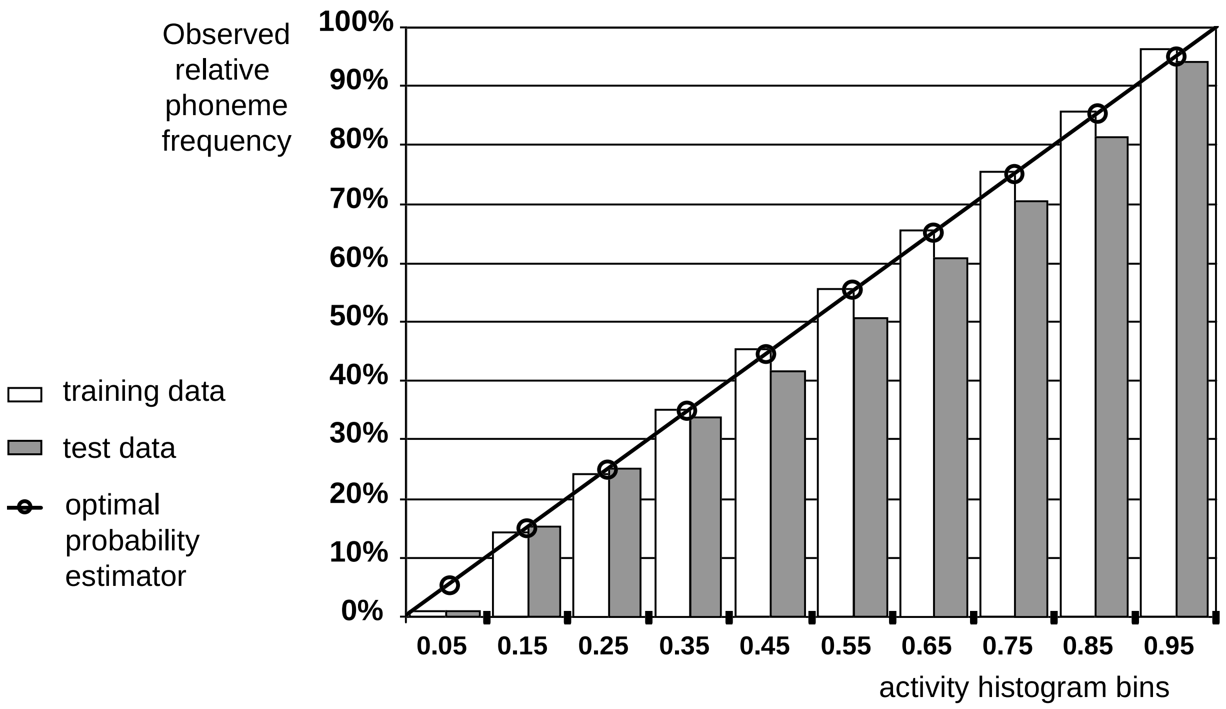


Figure 14. Estimated phoneme probability versus relative phoneme frequency. The frames of the training and test data respectively have been partitioned into bins with different output activity (in the RWH, [0;1] domain). The y-axis is the relative phoneme frequency computed from the targets in each bin. The shown relative frequency is the averaged statistics over all phonemes. The diagonal line is the optimal Bayesian classifier function. We see that the ANN is very close to a Bayessian classifier on the training data but deviates slightly from the ideal line for the test data.

5.2 Word recognition results on the WAXHOLM human/machine dialog task

The WAXHOLM demonstrator (Blomberg *et al*., 1993) is a human/machine spoken dialogue system for tourist information about the Stockholm archipelago. More specifically, the domain is boat traffic timetables and information about hotels, camping grounds and restaurants. A description of the ASR module of the WAXHOLM system is given in Ström (1996). In this paper, the WAXHOLM database (Bertenstam *et al*., 1995a,b), collected in wizard-of-oz simulations of the system, is used in recognition experiments to evaluate the hybrid HMM/ANN system on the word-level as well as the phoneme-level. The utterances are continuously spoken and the bigram perplexity is 28 for the test data.

The experience with the TIMIT database led us to train a network with the following topology, 300 hidden units, 50% connectivity for connections from the input units, $\sigma = 25$ for the recurrent connections and 10% connectivity for the connections to the output units. The phone error-rate achieved is 25.2% and the word error-rate is 23.2%. For comparison, a fully connected network with 125 hidden units was trained. As can be seen in Table 4, the fully connected ANN performs worse than the sparsely connected ANN, in spite of the benefit of 12% more connections.

Our results can be compared with a continuous density HMM system (Högberg and Sjölander, 1996), trained and evaluated on the same database as used in this study, yielding 25% phone error-rate, and later in (Sjölander and Högberg, 1996) 22.6% phone error-rate, and only 14.7% word error-rate. All results are summarized in Table 4.

In contrast to the TIMIT database, the phoneme recognition is slightly worse for the ANN-based system than the HMM. The reason is not clear, but the WAXHOLM database is smaller and has less contextual variation because of its domain dependent origin. However, it is the difference in word recognition rate that is the most noticeable. Clearly, the recognition results for the ANN system on the word-level are not equally competitive as on the phoneme-level.

	Baseline, fully connected network, 125 hidden units. 98,175 connections.	Sparsely connected network, 300 hidden units. 86,903 connections.	Tri-phone CDHMM recognizer.. (Sjölander and Högberg, 1996).
phone error rate	27.5%	25.2%	22.6%
word error rate	24.1%	23.2%	14.7%

Table 4. Recognition results on the WAXHOLM database.

One difference between our system and state-of-the-art HMM systems is that we do not use context-dependent models that are likely to be more helpful on the word level. Context-dependent models have been used in hybrid ANN systems with improved word recognition rate as a result (Bourlard and Morgan, 1993; Kershaw, Hochberg, and Robinson, 1996). The word error rate improvement reported by Kershaw, Hochberg, and Robinson, (1996) is about 19% (from 16% to 13%). Let us speculate that we would get the same reduction if context independent models were introduced in our ANN system. In this case the error rate would still be as high as 18.9% – significantly higher than for the HMM system. This indicates that there is also some other difference that manifests itself in the discrepancy between the word and the phoneme level.

The main difference in the application of the phoneme probabilities, between the word and the phoneme level, is that on the word level small probabilities are frequently employed in order to meet the word constraints from the lexicon. This happens much more seldom in the phoneme recognition case, because of the weaker top-down constraints. If this is the critical difference between the two levels, one can suspect that the estimation of small probabilities is worse in the ANN than the standard HMM. This appears to be a plausible explanation, but it is noteworthy that there is no evidence for that hypothesis in Figure 14, where instead it seems as if small probabilities are estimated more accurately.

6 Conclusions

Pruning and sparse connection were utilized to be able to train and evaluate large ANNs for phoneme probability estimation. Sparse connection made training of networks with large hidden layers and an elaborate dynamic connection scheme possible. Wide time-delay windows in combination with multiple time-delays in recurrent connections was used. A localized sparse connection scheme for recurrent connections removed the quadratic relation between the size of the hidden layer and the number of connections. This opened the possibility to work with recurrent ANNs with large hidden layers. Without sparse connection, the training would have required more computational resources than practically realistic.

The networks were evaluated on two different tasks: phoneme recognition on the TIMIT database, and word recognition on the WAXHOLM database. The lowest error-rate achieved on the core test set of the TIMIT database, 27.8%, compares favorably with systems using other methods. The only lower error-rate reported is from another ANN based system (Robinson, 1994). The word-level results on the WAXHOLM database are not as good as the phoneme results. A continuous density HMM trained and evaluated on the same data as the

ANN system performed significantly better. One reason for the contrast is that context-independent models are used, but it is not clear if this accounts for the whole difference.

Pruning reduces the computational demands of trained networks, which is of great importance in real-time recognition systems. A reduction by 50% can be accomplished without degraded performance. Also, moderate pruning improves the performance, either due to an increased generalization ability or because the network is allowed to escape from a local minimum of the error function.

Sparsely connected networks were shown to utilize the free parameters more efficiently than fully connected ANNs. In the phoneme recognition experiments, fully connected networks were clearly outperformed by their sparsely connected counterparts with equal or fewer connections. For example, in the TIMIT experiments, a network with only 13,000 connections was more successful than the fully connected network with 75,000 connections, and a sparsely connected network with less than 75,000 connections achieved as much as 21% lower phone error-rate than the fully connected ANN.

The toolkit used for all ANN simulations, including source-code and documentation, is freely available by anonymous ftp on the Internet. By making the software freely available, the results of the experiments can be validated by other researchers, and by giving easy access to an ANN toolkit for speech recognition, we wish to promote further development in the field.

7 Acknowledgments

I would like to thank Tan Lee, Rolf Carlson and Kjell Elenius for their valuable comments on the early versions of the manuscript.

8 References

Basu A. & Svendsen T. (1993): “A time-frequency segmental neural network for phoneme recognition,” *Proc. ICASSP ’93*, pp I-509 - I-512.

Baum E. B. & Wilczek F. (1988): “Supervised lerning of probabilty distributions by neural networks,” *Neural Information Processing Systems*, Ed. D. Z Anderson, American Institute of Physics.

Bertenstam, J. Blomberg, M., Carlson, R., Elenius, K, Granström, B., Gustafson, J., Hunnicutt, S., Högberg, J., Lindell, R., Neovius, L., de Serpa-Leitao, A. and Ström, N. (1995a): “The Waxholm Application Database,” *Proc. EUROSPEECH ' 95*Madrid, pp. 833-836.

Bertenstam, J. Blomberg, M., Carlson, R., Elenius, K, Granström, B., Gustafson, J., Hunnicutt, S., Högberg, J., Lindell, R., Neovius, L., de Serpa-Leitao, A., Nord, L. and Ström, N. (1995b): “ Spoken dialogue data collection in the Waxholm project,” *STL-QPSR 1/1995*, pp. 50-73.

Bishop C. M. (1995): *Neural Networks for Pattern Recognition*, Oxford University Press, Oxford.

Blomberg M., Carlson R., Elenius K., Granström B., Gustafson J., Hunnicut S., Lindell R., & Neovius L. (1993): “An Experimental Dialogue System: Waxholm,” *Proc EUROSPEECH '93*, pp. 1867-1870.

Bourlard & Wellekens (1988): “Links between Markov Models and Multilayer Perceptrons,” IEEE Trans on PAMI, **12**(12), pp. 1167-1178.

Bourlard H. & Morgan N. (1993): “Continuous Speech Recognition by Connectionist Statistical Methods,” *IEEE trans. on Neural Networks,* **4**(6), pp. 893-909.

Bridle J. S. (1989): “Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition,” *in Neuro-computing: Algorithms, Architectures and Applications*, Eds: Fougelman-Soulie and Hérault, pp. 227-236, Springer Verlag.

Bundine W. L. & Wiegend A. S. (1994): “Computing Second Derivatives in Feed-Forward Networks: A Review,” *IEEE Trans on Neural Networks*, **5**(3), pp. 1-9.

Cohen M., Franco H., Morgan N., Rumelhart D. & Abrash V. (1992): “Hybrid neural network/Hidden Markov Model continuous-speech recognition,” *Proc ICSLP '92*, pp. 915-918.

Digalakis V. V., Ostendorf M. & Rohlicek J. R. (1992): “Fast algorithms for phone classification and recognition using segment-based models,” IEEE Trans. on Signal Processing, Vol **40**, pp. 2885-2896.

Duda R. O. & Hart P. E. (1973): *Pattern Classification and Scene Analysis*, John Wiley and Sons, New York.

English, T. M. & Boggess, L. C. (1992): "Back-propagation training of a neural network for word spotting”, *Proc. IEEE ICASSP ´92*, **Vol 2**, pp. 357-360.

Fahlman S. E. (1988): “An empirical study of learning speed in back-propagation networks,” *Technical Report CMU-CS-88-162*, Carnegie-Mellon University, Computer Science Dept., Pittsburgh, PA.

Fant G. (1969): Acoustic Theory of Speech Perception, Mouton, The Hague, The Netherlands.

Ghosh G. & Tumer K. (1994): “Structural adaptation and generalization in supervised feed-forward networks,” Journal of Artificial Neural Networks, **1**(4), pp. 430-458.

Gish, H. (1990): “A Probabilistic Approach to the Understanding and Training of Neural Network Classifiers,” *Proc IEEE ICASSP '90*, pp1361-1364.

Glass J., Chang J., & McCandless M. (1996): “A Probabilistic Framework for Feature-Based Speech Recognition,” Proc ICSLP ’96 pp. 2277-2280.

Goldenthal W. (1994): “Statistical trajectory models for phonetic recognition,” *Technical report MIT/LCS/TR-642*, MIT Lab. for Computer Science.

Hampshire J. B. (1990): “Equivalence Proofs for Multi-Layer Perceptron Classifiers and the Bayesian Discriminant Function,” *Proc. of the 1990 Connectionist Models Summer School*, Eds: Touretsky, Sejnowski and Hinton, Morgan Kaufmann, San Mateo CA.

Högberg J. & Sjölander K. (1996): “Cross Phone State Clustering Using Lexical Stress and Context,” *Proc ICSLP ’96*, pp. 474-477.

Hornik K., Stinchcombe M. & White H. (1989): “Multilayer feed-forward networks are universal approximators,” *Neural Networks*, **2**, pp. 359-366.

Juang B.-H. & Katagiri S. (1992): “Discriminative Learning for Minimum Error Classification*,” IEEE trans. On Signal Processing*, **40**(12), pp. 3043-3054.

Kershaw D. J., Hochberg M. M. & Robinson A. J. (1996*): “Context-dependent classes in a hybrid recurrent network-HMM speech recognition system*,” In Advances in Neural Information Processing Systems **8**, eds: Touretsky D. S., Mozer M. C, and Hasselmo M. E., Morgan Kaufmann.

Lamel L. & Gauvain J. L. (1993): “High performance speaker-independent phone recognition using CDHMM,” *Proc. EUROSPEECH*, pp. 121– 124.

Le Cun Y., Boser B., Denker J. S., Henderson J. S., Howard R. E., Hubbard W. & Jackel L. D. (1990b): “Handwritten Digit Recognition with a Back-propagation Network,” *In*

Advances in Neural Information Processing Systems vol. **II**, ed: Touretsky D. S., pp. 396-404, San Mateo, California IEEE, Morgan Kaufmann.

Le Cun Y., Denker J. S. & Solla S. A. (1990a): “Optimal brain damage,” *In Advances in Neural Information Processing Systems* vol. **II**, ed: Touretsky D. S., pp. 589-605, San Mateo, California IEEE, Morgan Kaufmann.

Lee, K. F. (1989): *Automatic Speech Recognition; The Development of the SPHINX System*, Kluwer Academic Publishers, Dordrecht

Lee K-F & Hon H-W (1989): “Speaker-independent Phone Recognition using Hidden Markov Models,” *IEEE Trans. On Acoustics, Speech, and Signal Processing*, **37**(11), pp. 1641-1648.

Levenberg K. (1944): “A method for the solution of certain problems in least squares,” *Quart. Appl. Math.*, **2**, pp. 164-168.

Levin, E. (1990): "Word recognition using hidden control neural architecture”, *Proc IEEE ICASSP '90*, **Vol 1**, pp. 433-436.

Li, K. P., Naylor, J. A. & Rossen, M. L. (1992): "A whole word recurrent neural network for keyword spotting”, *Proc. IEEE ICASSP ´92,* **Vol 2**, pp. 81-84.

Luenberger G. L. (1984): Linear and Nonlinear Programming, Addison-Wesley Publishing Company, Inc.

Mari J. F., Fohr D. & Junqua J.C. (1996) “A second-order HMM for high per-formance word and phoneme-based continuous speech recognition,” *Proc. ICASSP*, pp. 435–438.

Marquardt D. (1963): “An algorithm for least-squares estimation of nonlinear parameters,” *SIAM Jl. Appl. Math.*, **11**, pp. 431-441.

Mitchel C. D., Harper M. P. & Jamieson L. H. (1996): “Stochastic Observation Hidden Markov Models,” *Proc IEEE ICASSP '96*, pp. 617-620.

Pearlmutter B A (1990): “Dynamic Recurrent Neural Networks,” Technical Report CMU-CS-88-191, Carnegie-Mellon University, Computer Science Dept. Pittsburg, PA.

Rabiner L. and Juang B-H (1993): Fundamentals of Speech Recognition, Englewood Cliffs NJ, Prentice Hall.

Richard M. D. & Lippman R. P. (1991): “Neural network classifiers estimate Bayesian *a posteriori* probabilities ,” *Neural Computation*, vol **3**, pp. 461-483.

Ripley B. D. (1996): *Pattern Recognition and Neural Networks*, Cambridge University Press, Cambridge.

Robinson A.J. (1994): “An application of Recurrent Nets to Phone Probability Estimation,” *IEEE trans. on Neural Networks* Vol **5**(2), pp. 298-305.

Robinson T. & Fallside F. (1991): “A Recurrent Error Propagation Network Speech Recognition System”, *Computer Speech & Language* 5:3, pp. 259-274.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986): “Learning internal representations by error propagation,” in Rumelhart, D. E., G. E. Hinton, (eds.), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Vol. 1 Foundations.*, chapter 8. Bradford Books/MIT Press, Cambridge, MA, ISBN 0-262-18120-7.

Schroeder M. R., Atal B. S. & Hall J. L. (1979): “Objective Measure of Certain Speech Signal Degradations Based on Masking Properties of the Human Auditory Perception,” in *Frontiers of Speech Communication Research*, B. Lindblom and S. Öhman, eds., Academic Press, pp. 217-229.

Sietsma J.& Dow R. J. F. (1991): “Creating artificial neural networks that generalize,” *Neural Networks*, **4**(1) pp. 67-69.

Sjölander & Högberg (1996): “Trying to improve phone and word recognition using finely tuned phone-like units,” *Proc. Swedish Phonetics Conference ’96*, *PHONUM 4:1996*, pp. 125-128, Umeå universitets tryckeri, Umeå, Sweden.

Steeneken H. J. M. & van Leeuwen D. A. (1995): “Multi-lingual Assessment of Speaker Independent Large Vocabulary Speech-recognition Systems: the SQALE-project,” *Proc. EUROSPEECH '95*, pp. 1271-1274.

Solla S. A., Levin E. & Fleisher M. (1988): “Accelerated Learning in Layered Neural Networks,” Complex Systems, **2**, pp. 625-640.

Ström N. (1992): “Development of a Recurrent Time-Delay Neural Net Speech Recognition System,” *STL-QPSR* 2-3/1992, pp. 1-44, KTH (Royal Institute of Technology), Dept. of Speech, Music and Hearing, Sweden.

Ström N. (1996): “Continuous speech recognition in the WAXHOLM dialogue system,” *STL-QPSR* 4/1996, pp. , KTH (Royal Institute of Technology), Dept. of Speech, Music and Hearing, Sweden.

Tebelskis, J. & Waibel, A. (1990): "Large vocabulary recognition using linked predictive neural networks”, *Proc. IEEE ICASSP '90*, **Vol 1**, pp. 437-440.

Thimm G. & Fiesler E. (1995): “Evaluationg pruning methods,” *In 1995 International Symposium on Artificial Neural Networks*, *Proc. ISANN ‘95*, pp. A2 20-25, National Chiao-Tung University, Hsinchu, Taiwan.

Waibel A., Hanazawa T., Hinton G., Shikano K. & Lang K. (1987) : “Phoneme Recognition Using Time-Delay Neural Networks,” *ATR Technical Report TR-006,* ATR, Japan.

White H. (1989): “Learning in Artificial Neural Networks: A Statistical Perspective,” *Neural Computation* **1**(4), pp. 425-464.

Young S., Jansen J., Odell J., Ollason D. & Woodland P. (1995): HTK – Hidden Markov Toolkit, Entropic Cambridge Research Laboratory.

Zue V., Seneff S. & Glass J. (1991): “Speech Database Development: TIMIT and beyond,” *Speech Communication*, **9**(4), pp. 351-356.

Appendix: The NICO neural network toolkit

All ANN simulations of this study are performed using the NICO (Neural Inference COmputation) toolkit. It is a software only solution, written in portable ANSI C code and the complete toolkit, including source code, can be downloaded from the World Wide Web home-page http://www.speech.kth.se/NICO/index.html. The home page also contains the latest updated on-line documentation.

In this appendix, the commands for building and training the ANN for the TIMIT experiments are presented. Frame level classification performance can be evaluated with the *CResult* tool. The dynamic decoding however, is not included in the NICO toolkit, and therefore additional software is necessary to perform evaluation on the phoneme-segment and word levels. In this case, the *Excite* tool can be used to compute and store the output activities in one of several supported data formats.

A.1 Structuring the database

The training tools, feature extraction and target generation tools operate either on the data-files of one utterance or on a set of utterances specified in a script-file. All files corresponding to the same utterance should have the same base-name. For example, if the wave-form sample file is called homer.wav then the names of the feature file and the phoneme target files can be homer.mfcc and homer.targ. Thus, specifying the base-name and the appropriate file-extensions and directories for the different types of data, is enough to give the tools knowledge of where to find data. Typically, a command-line argument to a tool specifies the base-name and the various extensions and directories are given as command-line options to the tool. A script-file for processing a set of base-names is invoked by the option -S. In this case, the argument that was otherwise the base-name is interpreted as the name of the script. The script-file itself is a list with one base-name per line.

The described conventions for data-files imply that all files of each type, e.g., all wave-form files, must be in the same directory. Although this flat structure is often convenient, it causes problems when the toolkit is used with a database with a different inherent structure, such as the TIMIT database. A solution is to create new directories for the interesting file-types and make symbolic links to the physical files residing in the non-flat database. An example of a c-shell script file that creates the necessary links are shown in Table 5.

```
#!csh

# set this variable to the directory of the TIMIT database
set TIMITDIR = CDROM/TIMIT

# set these variables to where you want to store links to
# wave-form files and phoneme label files
set WAVEDIR = ~/waveform
set PHONEDIR = ~/phmlab

foreach  CATEGORY  ( ${TIMITDIR}/train  ${TIMITDIR}/test )
   foreach  DIALECT  ( ${CATEGORY}/dr? )
      pushd  ${DIALECT}
      foreach  SPEAKER  ( * )
         pushd  ${SPEAKER}
         foreach  FILE  ( *.wav )
            ln -s ${DIALECT}/${SPEAKER}/${FILE} $WAVEDIR/${ SPEAKER}_${FILE}
         end
```

```
        foreach  FILE  ( *.phn )
           ln  -s  ${DIALECT}/${SPEAKER}/${FILE}  $PHONEDIR/${ SPEAKER}_${FILE}
        end
        popd
     end
     popd
  end
end
```

Table 5. This script is used to create symbolic links to the wave-form and phonetic label files of the TIMIT database.

A.2 Feature extraction

The tool *Melfib* implements a standard feature extraction, based on a Mel-scaled filter-bank computed from the FTT spectrum. The -c option selects cepstrum features instead of filter-bank. See section 3.1, or (Ström, 1996) for a more extensive discussion of this procedure. In the following example it is assumed that the directory ~/waveform contains links to waveform files of the TIMIT database, all_utterances is a script-file with the base-names of the utterances to process, and ~/mfcc is the directory to for storing the feature files.

```
>Melfib -S -Fnist -n24 -c12 -E -e0.97 -d ~/mfcc -xmfcc -p ~/waveform -qwav all_utterances
```

A.3 Generation of phoneme targets

The phoneme target for each 10 ms frame of the feature files can be extracted from the phonetic transcription files of the database. The tool *Lab2Targ* does this by checking the length of the parameter files and using the time-marks of the transcription files to compute the targets for each phoneme output unit at each frame (see also section 2.1). In this example, it is assumed that the directory ~/mfcc contains the parameter files created in the previous section, ~/phmlab contains links to the phonetic transcription files of the TIMIT database, all_utterances is a script-file with the base-names of the utterances to process, and ~/phmtarg is the directory for storing the target files. The names of the 61 phonemes should be supplied in the file phoneme_set, one phoneme per line.

```
>Lab2Targ -S -p ~/phmlab -qphm -d ~/phmtarg -x targ -P ~/mfcc -Qmfcc -Ihtk -Ltimit -Fcodebook phoneme_set 13 all_utterances
```

A.4 Specifying the network structure

The toolkit has many commands for building the structure of the networks. Here we give the commands with a minimum of commentary. Detailed information about the commands is found in the toolkit documentation. The commands for building a network with 300 hidden units, connectivity 25% in connection sets *A* and *B*, and connection parameter σ=50 for the recurrent connections (see section 5.1) are as follows.

```
>CreateNet timit300.rtdnn timit300.rtdnn
>AddStream -x mfcc -d  ~/mfcc -F htk 13 r CEP timit300.rtdnn
>AddGroup cep timit300.rtdnn
>AddUnit -l -u 13 cep timit300.rtdnn
>LinkGroup CEP cep timit300.rtdnn
>MakeDiff cep diff1sttimit300.rtdnn
>MakeDiff diff1st diff2nd timit300.rtdnn
>AddStream -x targ -d ~/phmtarg -Fcodebook timit300.rtdnn -S phoneme_set 61 t PHONEME timit300.rtdnn
```

```
>AddGroup phoneme timit300.rtdnn
>AddUnit -o S phoneme_set phoneme timit300.rtdnn
>LinkGroup PHONEME phoneme timit300.rtdnn
>SetType -O cross phoneme timit300.rtdnn
>AddGroup param timit300.rtdnn
>Move cep network param timit300.rtdnn
>Move diff1st network param timit300.rtdnn
>Move diff2nd network param timit300.rtdnn
>AddGroup features timit300.rtdnn
>AddUnit -u 300 features timit300.rtdnn
>Connect -D -1 5 -s25 param features timit300.rtdnn
>Connect -D -1 1 -s25 features phoneme timit300.rtdnn
>Metricnct -s25 -D 1 3 features features timit300.rtdnn
```

These commands creates and modifies a binary file timit300.rtdnn holding all parameters of the network. The created network can now be examined with the *Display* command. Display has many options and can display all properties of a network. Details are found in the toolkit documentation. Enter for example the command

```
> Display timit300.rtdnn
```

to see an overview of the network (number of connections and units etc.).

So far we have created an ANN with simple uncoupled output units. The following commands alter the output layer to a softmax activation function (see section 2.7).

```
>AddGroup softmax timit300.rtdnn
>SetType -x phoneme timit300.rtdnn
>SetType -n phoneme timit300.rtdnn
>Move phoneme network softmax timit300.rtdnn
>Rename phoneme softin timit300.rtdnn
>AddGroup phoneme timit300.rtdnn
>AddUnit -r0.0 -o -S phoneme_set phoneme timit300.rtdnn
>Remove -c phoneme bias timit300.rtdnn
>SetType -m phoneme timit300.rtdnn
> SetType -O cross phoneme timit300.rtdnn
>Move phoneme network softmax timit300.rtdnn
>Pipe -w1.0 softmax.softin softmax.phoneme timit300.rtdnn
>AddGroup internal timit300.rtdnn
>AddUnit -r0.0 softsum timit300.rtdnn
>Move softsum network internal timit300.rtdnn
>Move internal network softmax timit300.rtdnn
>SetType -d softmax.internal.softsum timit300.rtdnn
>Connect -w1.0 softmax.softin softmax.internal.softsum timit300.rtdnn
>Connect -w1.0 softmax.internal.softsum softmax.phoneme timit300.rtdnn
>NormStream -c 1 0 -s PHONEME timit300.rtdnn
>LinkGroup PHONEME phoneme timit300.rtdnn
>Protect -R softmax.softin timit300.rtdnn
>Protect -S softmax.phoneme timit300.rtdnn
>Protect -P softmax.internal timit300.rtdnn
```

A.5 Training

The command for normalizing the input units is

```
>NormStream -S -s CEP -d1.0 timit300.rtdnn train_utterances
```

and the normalization of the delta and delta-delta features is performed by

```
>NormGroup -S -g diff1st -d1.0 timit300.rtdnn train_utterances
>NormGroup -S -g diff2nd -d1.0 timit300.rtdnn train_utterances
```

where train_utterances is a script holding the basenames of all training utterances. Note that these commands collect statistics from the whole training database, and therefore take some time to run.

The most important training tool of the toolkit is *BackProp*. It implements the training scheme described in section 2. In this example we assume that train_utterances is again a script with the training utterances and validation_utterances is a script with the validation utterances.

```
>BackProp -S -m0.7 -g1e-5 -i30 -F 20 30 -p logfile.log -B 0.5 10 -V validation_utterances PHONEME timit300.rtdnn train_utterances
```

A.6 Frame-level evaluation

Althogh the toolkit does not have tools for evaluation on the segment level, it is possible to evaluate the frame classification performance, i.e., how frequently the correct phoneme has the highest output activation. The tool to use is *CResult*. This example is an evaluation of the classification of the 61 phonemes. It is assumed that test_utterances is a script with the test utterances. The CResult tool has many options, e.g., confusion matrix, top-N evaluation, etc. Details can be found in the toolkit documentation.

```
>CResult -S timit300.rtdnn test_utterances
```

A.7 Connection pruning

The connection pruning of trained networks that is described in 4.1 is performed easily with the *Prune* tool. For example, pruning the network timit300.rtdnn with $\alpha = 0.1$, and $\beta = \infty$, the following command is used.

```
>Prune -w 0.1 timit300.rtdnn
```