TUNING BY DOING: FLEXIBILITY THROUGH AUTOMATIC STRUCTURE OPTIMIZATION

Ulrich Bodenhausen and Alex Waibel
University of Karlsruhe, Computer Science Department,
ILKD, 7500 Karlsruhe 1, Germany
and
School of Computer Science, Carnegie Mellon University,
Pittsburgh, PA 15213, USA

ABSTRACT

The successful application of speech recognition systems to new domains greatly depends on the tuning of the architecture to the new task, especially if the amount of training data is small. In this paper we present 1.) an improved version of our Automatic Structure Optimization (ASO) algorithm that does this tuning automatically and 2.) a new Automatic Validation Analyzing Control System (AVACS) that is designed to detect poorly generalizing models as early as possible and to selectively change their learning and automatic structuring process. ASO and AVACS were applied to a Multi State Time Delay Neural Network and could improve the generalization performance of an already handtuned architecture from 85% to 92.3% on an alphabet recognition task.

1. INTRODUCTION

Despite the aim to develop general purpose, speaker independent, very large vocabulary speech recognition systems, there is also a considerable number of applications that require the best possible recognition accuracy on a small, well defined, and customized domain. Achieving the best possible performance with HMMs, Neural Networks, or hybrid systems greatly depends on the tuning of the architecture to the particular task, especially if the amount of training data is small (which is often true for customized applications). In this paper we present

- an improved version of our *Automatic Structure Optimization* (ASO) algorithm [1], [2], [3] that does this tuning automatically for neural network speech recognition systems and
- a new *Automatic Validation Analyzing Control System* (AVACS) that is designed to detect poorly generalizing models on a class by class basis as early as possible and to selectively change their learning and automatic structuring process.

AVACS is an attempt towards better teaching methods for artificial neural networks. Instead of only presenting examples that have to be learned by the network, the system is frequently tested with a validation set to identify models that seem to learn well, but do not lead to good generalization. While a validation set is used in many other systems to determine the stopping criterion (by training until the maximum validation performance is reached), we extend the use of this set and propose a comparison of the confusion matrices on train and validation set to selectively detect the classes that generalize poorly. Because of the constructive method that is used by ASO it is then possible to identify all resources that contribute to these classes and restructure and/or retrain them.

In addition to the advantage of offering an automatic architecture optimization that is automatically validated and controlled, our approach also offers an attempt towards controlled *error equalization*. Consider a speech recognition task with 50 words only. Although a recognition performance of 94% does not sound that bad it is highly undesirable if all errors occur for three words only. AVACS detects these kinds of irregularities and tells the learning/structuring module that something is going wrong.

2. THE AUTOMATIC STRUCTURE OPTIMIZATION ALGORITHM (ASO)

For the application of neural networks to speech recognition all of the following architectural parameters have to be well adapted to the task and the given amount of training data (see Fig. 1):

- the number of hidden units,
- the size of input windows and
- the number of states that model an acoustic event.

The ASO algorithm *automatically adapts* all of these architectural parameters to the given task and amount of training data in a single training run. The algorithm offers the flexibility to apply neural net speech recognition systems to new domains without the need for manual tuning of the architecture.

The ASO algorithm tries to optimize the architecture of the system for best possible generalization performance. According to Moody [6], the expected error on the test set can be approximated as follows:

$$\langle E_{test}(\lambda) \rangle_{\xi\xi'} \approx \langle E_{train}(\lambda) \rangle_{\xi} + 2\sigma^2_{eff} \frac{p_{eff}}{n} \quad (1)$$

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

The idea of the ASO algorithm is to start with a small number of parameters for the given number of training exemplars (leading to a small second summand on the right side of the above equation) and increasing this number to decrease the

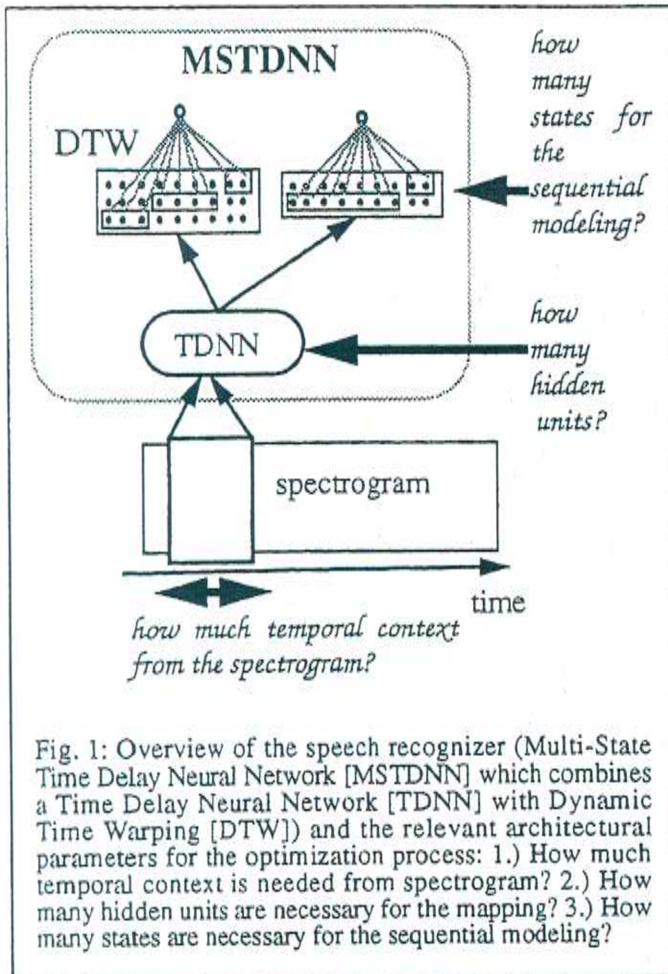

Fig. 1: Overview of the speech recognizer (Multi-State Time Delay Neural Network [MSTDNN] which combines a Time Delay Neural Network [TDNN] with Dynamic Time Warping [DTW]) and the relevant architectural parameters for the optimization process: 1.) How much temporal context is needed from spectrogram? 2.) How many hidden units are necessary for the mapping? 3.) How many states are necessary for the sequential modeling?

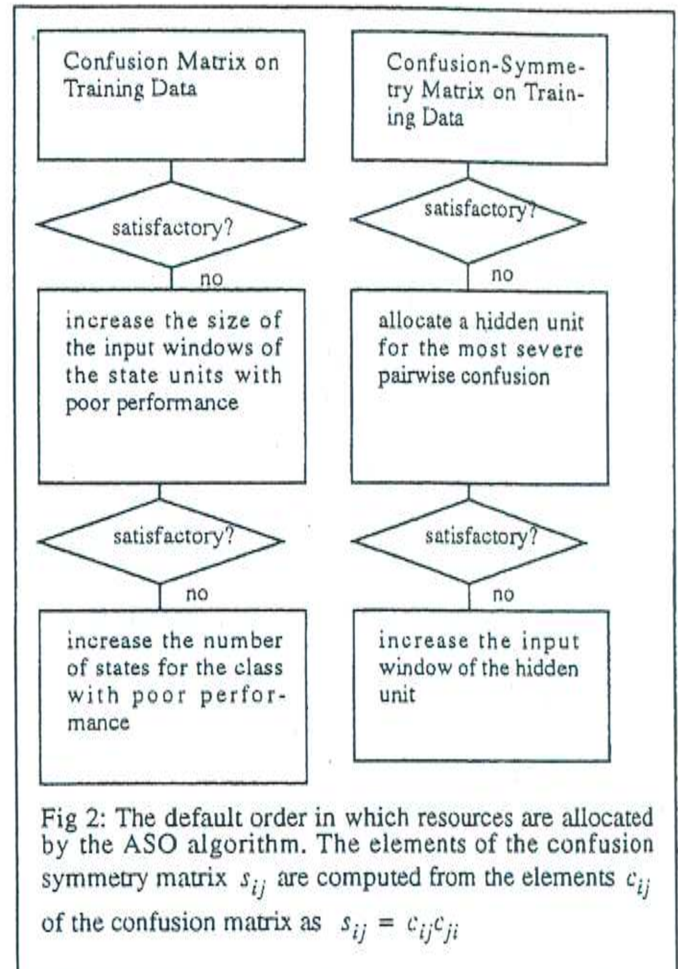

Fig 2: The default order in which resources are allocated by the ASO algorithm. The elements of the confusion symmetry matrix s_{ij} are computed from the elements c_{ij} of the confusion matrix as $s_{ij} = c_{ij}c_{ji}$

expected error on the training set (the first summand in the above equation). The goal is to increase the second summand and to decrease the first summand until the best possible compromise between a low training error and a high number of parameters is reached.

The ASO algorithm uses the following tuning strategies:

- The time-shift invariance of the task are used to reduce the number of trainable parameters and to avoid learning of undesired features from the training data (like the length of phonemes).
- The confusion matrix on the training data is evaluated to selectively improve certain parts of the acoustic modeling.
- The number of states is increased if the acoustic modeling is too complex for the given number of states.
- Hidden units with sigmoid activation function are allocated to specifically solve pairwise confusions (class "A" is confused with class "B" and vice versa) which are caused by inadequate decision boundaries. The approach is similar to the *Boundary Hunting Radial Basis Function* classifier [4]. The hidden units are added additionally to the direct connections between the input and the state units .
- The criterion for the allocation of resources is modified depending on the quotient p_{total}/n, where p_{total} is the total number of parameters and n is the number of training patterns.

Unlike the human developer, the ASO algorithm starts making decisions about resource allocations very early in the training run, i.e. it is tuning the architecture while the network is learning the task ("tuning by doing"). This allows the algorithm to complete the optimization process in a single training run.

The default order in which resources are allocated is as follows: At first, the size of the input windows is incremented depending on the confusion matrix on the training data. If a certain class performs worse than the average class the width of the input windows is incremented by one frame. This procedure is repeated in the following epochs. If the size of the input windows gets close to the average duration of the sound that the corresponding state unit is modeling and the performance is still not satisfactory, then a new state unit is added. The size of the input window of the first state is halved (to avoid a dramatic increase in the number of parameters). The input window of the new state is gets the same size as the window of the first state, but with randomly initialized connections.

Hidden units are allocated between the input and the state units in additon to the direct connections from the input to the state units (similar to the *Cascade Correlation* Algorithm [5]). The allocation is dependent on the pairwise confusion (class "A" is confused with class "B" and vice versa). The size of the input window of the new hidden unit is increased in the following epochs if the pairwise confusion could not be eliminated.

The default schedule for the allocation of resources is shown by Fig. 2. It can be altered by the Automatic Validation Analyzing Control System (AVACS, see next paragraph) if it does not lead to good generalization.

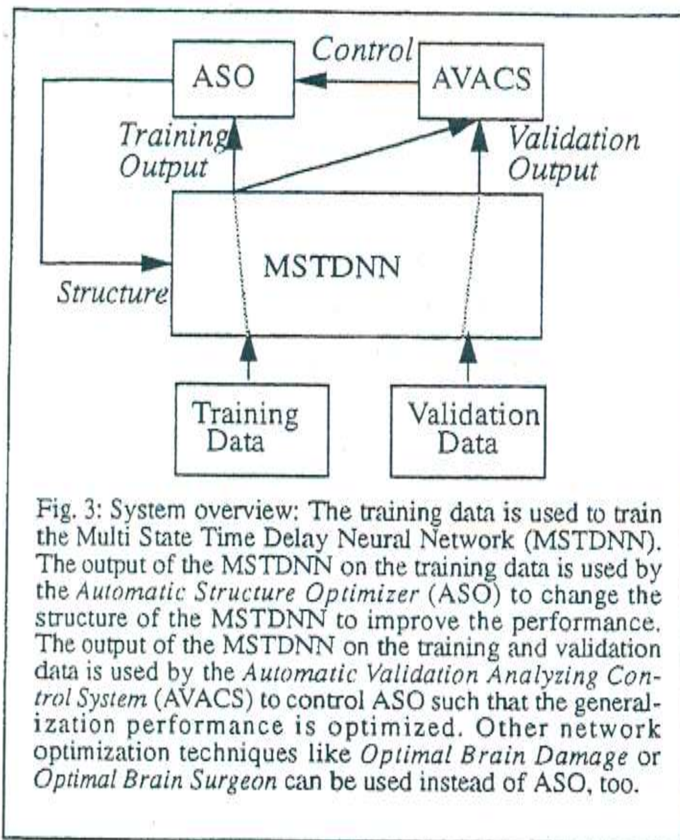

Fig. 3: System overview: The training data is used to train the Multi State Time Delay Neural Network (MSTDNN). The output of the MSTDNN on the training data is used by the *Automatic Structure Optimizer* (ASO) to change the structure of the MSTDNN to improve the performance. The output of the MSTDNN on the training and validation data is used by the *Automatic Validation Analyzing Control System* (AVACS) to control ASO such that the generalization performance is optimized. Other network optimization techniques like *Optimal Brain Damage* or *Optimal Brain Surgeon* can be used instead of ASO, too.

3. THE AUTOMATIC VALIDATION ANALYZING CONTROL SYSTEM (AVACS)

AVACS monitors the learning and tuning process and is designed to detect poorly generalizing models on a class by class basis as early as possible (see Fig. 3). A validation set is used to test the generalization ability of the system frequently in the training run. The confusion matrices are computed for both the training and the validation data. From these matrices a new confusion-difference matrix with the elements d_{ij} is computed as follows:

$$d_{ij} = \bar{c}_{ij}(train) - \bar{c}_{ij}(validation) \qquad (2)$$

where $\bar{c}_{ij}$ are the elements of the confusion matrices normalized by the number of appearances of a particular class in the data. The interpretation of the difference matrix is straightforward:

- Small numbers or positive numbers indicate that the network generalizes well on the validation data. This means that the network should also generalize well on the final test set if the validation set is representative for the task. In this case there is no need to limit the allocation of further resources to further increase the performance on the training data. See Fig. 4 for examples. Small numbers are usually not visible
- Negative numbers indicate that the performance on the validation data is worse than the performance on the training data, which is quite normal depending on the number of effective parameters, the number of training patterns and the noise variance of the data [6], [7]. However, it is possible to detect those classes that generalize worse than other classes. This could indicate four possible problems:

1.) The ASO algorithm accidentally allocated too many parameters.

2.) The particular model does not fit because of initial conditions.

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

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

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

More sophisticated methods for 'poor generalization recovery' were tried, too. For example, it is possible to change the default order in which resources are allocated (see Fig. 2). Another option is to completely reinitialize the poorly generalizing parts of the network and to retrain them. In a limited number of experiments non of these methods performed better than the contamination with noise. In a real application it is probably best to try a certain number of these options and, if none of these helped, tell the user to collect more training data or to accept the current generalization capability.

4. SIMULATIONS

The ASO algorithm with AVACS was applied to Multi State Time-Delay Neural Networks (MSTDNNs, [8], [9]), an extension of the TDNN [10]. The results are summarized in Table 1. The ASO algorithm alone could improve the generalization performance of an already optimized architecture from 85% to 91.7% for an alphabet recognition task with 2200 training patterns. While constructive and pruning methods tend to be very successful in optimizing tasks with medium-sized training databases (50 - 500 examples per class), it is much harder to optimize an architecture for extremly small databases (10 - 30 examples per class). The ASO algorithm could still achieve 81.5% with only one quarter of the training data (20 examples per class = 520 training patterns) because a smaller network was constructed. AVACS further improved the results on this extremely small training set (520 training patterns) from 81.5% to 83.5% in preliminary experiments.

5. DISCUSSION AND CONCLUSIONS

The results with the ASO algorithm alone suggested that the algorithm can construct efficient architectures in a single training run that achieve comparable or better recognition accuracies than manually tuned architectures. ASO offers the flexibility to use a given amount of available training data without the need to manually adapt the architecture to this amount. The good generalization ability on extremely small training sets [2], [3] can be explained by the unequal amount of training that the weights of the final system have received. Many connections are added late in the training run when the error is already very low. These connections are never trained by large error derivatives and their weights remain very close to their random initialization [3]. The ASO algorithm performs similarly on on-line handwritten character recognition tasks we have tested [1], [2], [3].

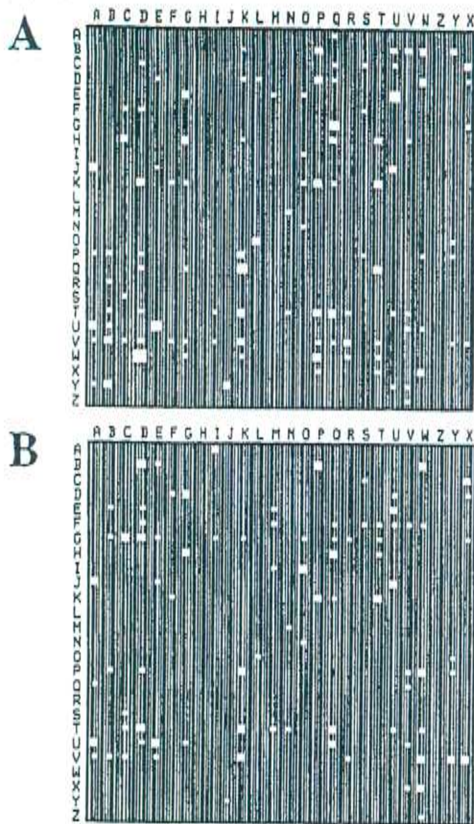

Fig. 4: A comparison of typical confusion-difference matrices for the alphabet recognition task with an extremely small training set (520 training patterns). The desired output is shown on the horizontal axis and the output of the network is shown on the vertical axis. White blobs indicate a poor generalization with the size of the blob proportional to the value of d_{ij}, the confusion-difference (see Eq. 2). Black blobs indicate a better performance on the validation data than on the training data. **A)** A handtuned architecture with standard weight decay reached the best performance on the validation set after 385 epochs because weight decay tends to slow down learning. Test performance was 75.7%. **B)** The architecture optimized by ASO and AVACS reached the best validation performance after 175 epochs. Test performance was 83.5%. A comparison of both matrices shows that 1.) ASO plus AVACS result in better generalization performance and 2.) ASO plus AVACS lead to fewer 'serious' generalization errors which are indicated by large white blobs in the matrices. A fixed architecture without weight decay performs even worse (more serious generalization errors and lower test performance.

AVACS is an attempt to improve the learning process by better analysis of the generalization errors of the system. It has the following advantages:

- A validation set and confusion matrices are used in many systems anyway, so it is easy to implement AVACS.
- AVACS can also work with pruning methods (like *Optimal Brain Damage* (OBD)[7] or *Optimal Brain Surgeon* (OBS) [11]). AVACS can propose poorly generalizing parts of the network that would benefit most from OBD or OBS. Thus the time-consuming computation of second derivatives that is required by these methods is not necessary for all parameters.
- It can be used to change the weight decay parameter which can be very useful.
- AVACS can also be used to selectively detect when more training data is needed for certain classes.
- AVACS allows a prediction of likely and less likely generalization errors.

Preliminary results on a difficult task (spelled alphabet recognition with only 20 training examples for each spelled letter) have been promising. Both proposed methods together (ASO + AVACS) allow the flexible use of neural networks for customized speech applications that require best possible performance for the given amount of training data (usually small). Further experiments on other tasks will be made for further evaluation.

TABLE 1. **Alphabet Recognition Results Depending On Training Set Size (Preliminary)**

	test performance (520 training patterns)	test performance (2200 training patterns)
handtuned MSTDNN	75.7%	85.0%
MSTDNN with ASO	81.5%	91.7%
MSTDNN with ASO + AVACS	83.5%	92.3%

ACKNOWLEDGEMENTS

The authors gratefully acknowledge the support of the McDonnel-Pew Foundation and would like to thank S. Fahlman, S. Manke, H. Hild and M.T. Vo.

REFERENCES

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

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

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

[4] E.C. Chen and R.P. Lippmann. A Boundary Hunting Radial Basis Function Classifier Which Allocates Centers Constructively. In: *Advances in Neural Information Processing Systems 5*, 1993

[5] S. E. Fahlman and C. Lebiere. The Cascade-Correlation Learning Architecture. In: *Advances in Neural Information Processing Systems 2*, 1989.

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

[7] Y. Le Cun, J. S. Denker, and S. A. Solla. Optimal Brain Damage.. In: *Advances in Neural Information Processing Systems 2*, 1989.

[8] P. Haffner, M. Franzini, and A. Waibel. Integrating Time Alignment and Neural Networks for High Performance Continuous Speech Recognition. In *Proceedings of the ICASSP-91*.

[9] P. Haffner and A. Waibel. Time-Delay Neural Networks Embedding Time Alignment: A Performance Analysis.In: *Proceedings Eurospeech* 91.

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

[11] B. Hassibi and D. G. Stork. Second Order Derivatives for Network Pruning: Optimal Brain Surgeon. In: *Advances in Neural Information Processing Systems 5*, 1993