Application Oriented Automatic Structuring of Time-Delay Neural Networks for High Performance Character and Speech Recognition

Ulrich Bodenhausen and Alex Waibel

Computer Science Department, University of Karlsruhe, Postfach 6980, 7500 Karlsruhe 1, FRG,

and

School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213

Abstract-Highly structured artificial neural networks have been shown to be superior to fully connected networks for real-world applications like speech recognition and handwritten character recognition. These structured networks can be optimized in many ways, and have to be optimized for optimal performance. This makes the manual optimization very time-consuming. A highly structured approach is the Multi State Time Delay Neural Network (MSTDNN) which uses shifted input windows and allows the recognition of sequences of ordered events that have to be observed jointly. In this paper we propose an Automatic Structure Optimization (ASO) algorithm and apply it to MSTDNN type networks. The ASO algorithm optimizes all relevant parameters of MSTDNNs automatically and was successfully tested with three different tasks and varying amounts of training data.

I. INTRODUCTION

Highly structured artificial neural networks have been shown to be superior to fully connected networks for real-world applications like speech recognition [1, 2] and handwritten character recognition [3]. The importance of the network structure has also recently been examined by Geman et al. [4]. They come to the conclusion that “dedicated machines are harder to build but easier to train” and suggest that important properties of the task have to be built into the architecture of the network. Similar conclusions have been made by Minsky and Papert [5]. But manual design of intermediate representations and the network structure can be very time-consuming for real-world applications. It may not even be possible to optimize the structure before the exact application of the user is known. For example, the user might try to train a speech recognizer on a language and/or an amount of training data that the structure was not optimized for[1].

1. See Table 3 for an example: A network that was well optimized for a training set of 1170 patterns also generalized reasonably for a training set of 1560 patterns, but failed to learn a reduced training set of 520 patterns with various learning rates and momentums.

One reason for the introduction of structure to the network is the relationship between the number of trainable parameters, amount of training data and generalization (see [6, 7, 8] and others). Networks with too many trainable parameters for the given amount of training data learn well, but do not generalize well. This phenomenon is usually called overfitting. With too few trainable parameters, the network fails to learn the training data and performs very poorly on the testing data. Imposing structure into the network can increase the generalization performance by reducing the number of trainable parameters [1, 2].

Unfortunately, highly structured networks can be optimized in many more ways than fully connected networks. In order to achieve optimal performance without time-consuming manual optimization, we propose an AUTOMATIC STRUCTURE OPTIMIZATION (ASO) algorithm that automatically optimizes the structure and the total number of parameters synergetically and also considers the current amount of training data. Rather than starting with a distributed internal representation, the structure of the network is constructed by adding units and connections in order to selectively improve certain parts of the network. At the beginning of the training run the internal representation is completely local and gets more and more distributed in the following optimization process. Only a concept for structuring the network has to be specified before training. The concept for structuring the network is derived from (simple) knowledge about the task, such as invariances. The algorithm combines a constructive and a pruning method. The constructive approach is used to find a network structure that is specifically tailored for the task and the current amount of training data. Weight decay and/or Optimal Brain Damage (see next paragraph) can be used to further refine this architecture to achieve optimal performance.

II. PREVIOUS WORK

Two types of approaches have been proposed for the automatic optimization of network architectures: Constructive algorithms that start with a minimal architecture and increase the resources of the network; and algorithms that avoid overfitting by limiting the computational power of fixed architectures.

Regularization techniques like weight-decay or weight elimination [9, 10, 11] belong to the second group. They add some complexity measure to the cost function that is optimized by the learning rule. This can reduce the risk of overfitting the data with too many trainable parameters in the network, but the decay or elimination parameter has to be well adjusted for optimal results. Optimal Brain Damage (OBD) [12] is an algorithm that selectively removes unimportant weights from a network. The basic idea is to use second-derivative information to make a trade-off between network complexity and training set error. Although these algorithms can improve existing (reasonable architecures) considerably, it is not clear whether a foolish network can be optimized to state-of-the art performance.

Constructive algorithms like Cascade Correlation [13] start with a small network and increase the resources during the training phase. For example, Cascade Correlation starts with no hidden units and tries to solve the classification problem without them. If this fails, hidden units are added one after the other.

III. CONCEPT OF THE AUTOMATIC STRUCTURE OPTIMIZATION ALGORITHM

The proposed algorithm is based on four principles:

- built-in invariances
- task decomposition
- confusion matrix dependant construction of the network
- early constructive changes of the network architecture

A. Built-in Invariances:

If there is any knowledge about the task, it should be built into the structure of the network. For speech and handwritten character recognition, a classifier that is robust against temporal distortions is highly desirable. This can be achieved by using shifted input windows over time as in the Time-Delay Neural Network (TDNN) [1, 2]. Shifting the window reduces the number of weights and ensures that the hidden abstractions that are learned are invariant under translations in time.

B. Task Decomposition:

Instead of learning very complex decision surfaces for the classification of events, it may be better to decompose the classification into the recognition of subevents that have to be observed jointly. In many cases the decision surfaces for the recognition of these subevents are much easier to learn. This method is used in many speech recognition systems. For example, the recognition of words can be decomposed into the recognition of sequences of phonemes or phoneme like units. TDNN's have recently been extended to Multi State Time-Delay Neural Networks (MSTDNNs) [14, 15] that allow the recognition of sequences of ordered events that have to be observed jointly. Unfortunately, this also means another architectural parameter has to be optimized.

C. Confusion Matrix Dependant Construction of the Network Architecture:

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

D. Early Constructive Changes of the Network Architecture:

Waiting for a whole training run and then making decisions on the further optimization of the network is computationally very expensive. Our experience shows that it is possible to detect the most important mistakes very early in the training run and change the architecture early in the training run. Starting the training run again is not necessary.

IV. APPLICATION OF THE ASO ALGORITHM TO THE MSTDNN

As can be seen, MSTDNN type networks conform with the first two principles of paragraph 2 and are also very powerful classifiers [14, 15]. The architecture of these highly structured networks can be optimized in many ways. For best performance, the size of the input windows, the number of hidden units and the (word specific) state sequence topologies are of critical importance for optimal performance. This makes MSTDNNs a suitable candidate for the demonstration of the ASO algorithm.

The ASO algorithm optimizes all relevant parameters of MSTDNN structures for a given amount of training data. The minimal configuration of a MSTDNN consists of an input layer, a state layer and an output layer (see Fig. 1). Let us consider a word recognition task where each output unit represents a word. Each state unit represents a small piece of the utterance like phonemes or sub-phonemes. The network is initialized with a window size of one (one connection between an input unit and a unit of the following layer) and one state unit per output unit. The net input of the output units is computed by integrating the weighted activity of the single or multiple state unit(s) over time. The activation of the output units is given by the sigmoid of the net input. The state units can be regarded as a special type of hidden units because of their very constrained connectivity to the output units.

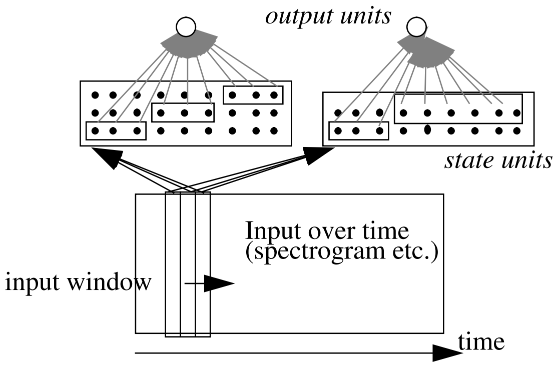

Fig. 1: An example of a simple MSTDNN with an input layer, a state layer and an output layer (consisting of two ouput units). In this example the first output unit is connected with three state units and the second output unit is connected with two state units.

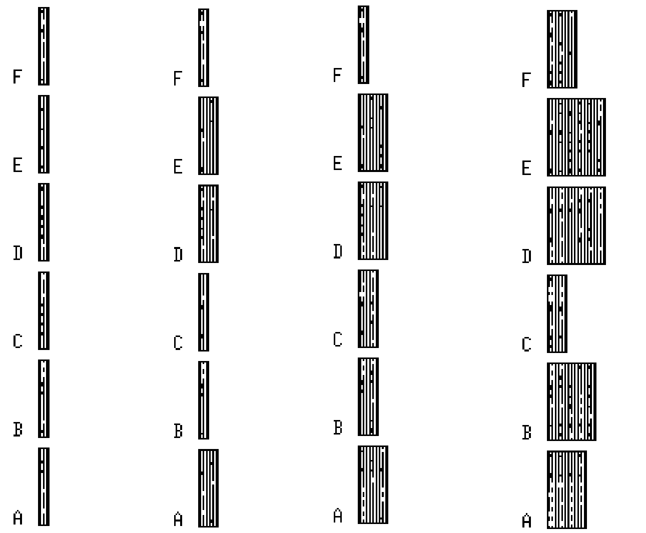

Fig. 2: The weights from the input units to the state units for the recognition of the capital letters A .. F. Negative weights are displayed by white blobs, positive weights by black blobs. Eight input units are used to represent the features recorded from the touch sensitive tablet (see [3]).

Left: The weights after initialization. All windows have a size of one.

Middle left: The weights after the first epoch. The input window for the characters “A”, “D” and “E” were increased to a size of two. The input windows for the characters “B”, “C” and “F” were not increased.

Middle right: The input window for the characters “A”, “D” and “E” were increased again. The windows for the characters “B” and “C” were increased to a size of two.

Right: The weights after the fifth epoch.

During training, the size of the input window of the state units as well as the number of state units increases depending on the performance of the corresponding output units. The criterium for the allocation of further resources is derived from the confusion matrix on the training set. At each epoch the mistakes of each output unit are counted. If the counter for output unit j is higher than the mean of all counters, then resources for this particular unit are added. At first, the size of the input window from the input layer to the corresponding state unit is increased by adding one set of random connections (see Fig. 2). In the next epoch, these new connections are trained together with the already existing connections and the above procedure is applied again.

If the size of the input window of a state unit converges and the corresponding output unit still makes more mistakes than the average unit, then a new state unit is added (see Fig. 3). The size of the input window of the ‘old’ state unit is halved to avoid a dramatic increase of the number of trainable parameters. The ‘new’ state unit receives input from an input window of the same size as the ‘old’ state unit, but with random connections. From now on, the output unit receives input from both state units.

The allocation of resources is controlled by a simple scheme:

- Adding more resources is easy if the number of connections is small compared to the number of training patterns and gets harder with an increasing number of connections. This avoids hard upper bounds for the network resources.
- All resources that are added to the network are initialized randomly. This reduces the risk that the new resources disturb the learning process. A side-effect is that noise is added which prevents the network from getting stuck in local minima. This noise is reduced afterwards because the new connections are trained together with the already existing connections.
- The maximal size of the input windows also depends on the number of states that model a word. If a word is modeled by many states the state units don’t need such a large input window as a state unit that models a whole word.

In case of more than one state unit per output unit the inputs of the output units can be computed in three different ways: The simplest way is to give each state unit an equal share of the time slice that the output unit represents. The second possibility is to use Dynamic Time Warping (DTW) to find the best path through the activation matrix of the state units [16]. The third possibility is to smooth the DTW path by Gaussian functions positioned according to the DTW segmentation (see Fig. 4). Smoothing of the DTW path allows the states to model the transitions between two states more accurately. If each state specializes on different parts of the spectrogram, then the transition between these parts may not be modeled by any of them. Smoothing allows both states to partially represent the transition.

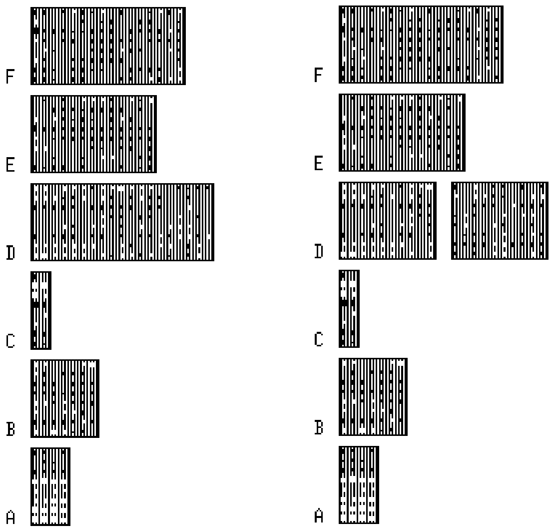

Fig. 3: The splitting of a state in the same training run as in Fig. 2. The modeling of the character “D” with one state and a big window is still insufficient (left side). A new state is added and the size of the old window is halfed (right side). The weights for the old (first) state remain the same and the weights for the new (second) state are initialized randomly.

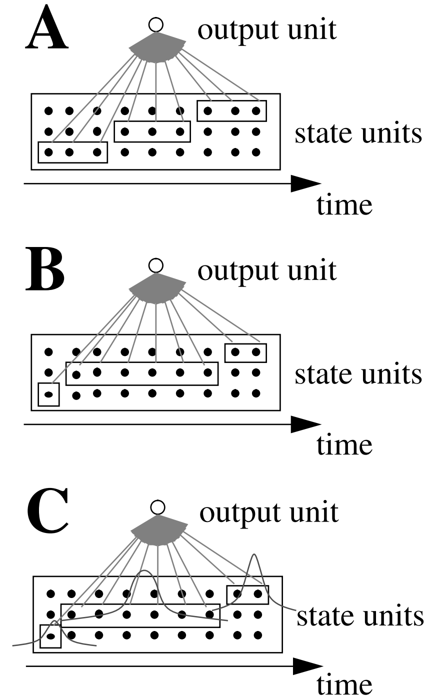

Fig. 4: Three choices for the connectivity between the state units and the output units. A: Each state gets an equal share of the time slice. B: Using DTW to find the best path through the activation matrix. C: Smoothing the DTW path by gaussian functions positioned according to the DTW segmentation.

V. SIMULATIONS

The ASO algorithm was tested with a speech recognition task and two handwritten character recognition tasks.The chosen tasks are small enough to allow a reasonable number of experiments with the algorithm and are large enough to be relevant for an application oriented algorithm. The tasks are:

- Recognition of the English alphabet (aprox. 3000 words spoken by a single male speaker (DBS) taken from the same database that was used in [14]). The letters of the english alphabet were recorded in a sound-proof booth with a sampling rate of 12kHz. The speech was Hamming windowed and a 256-point FFT computed every 5 ms. 16 normalized melscale coefficients were computed as described in [1]. [1]
- Recognition of the digits 0, 1, 2, ..., 9 written on a touch sensitive tablet (aprox. 1000 digits written by aprox. 70 writers, recorded as described in [3][2])
- Recognition of the capital letters A, B, ..., Z written on a touch sensitive tablet (aprox. 2500 capital letters written by aprox. 50 writers, recorded as described in [3])

The databases were cut into training data, validation data and testing data. The validation data is used to determine the stopping criterium for the training phase. For the current simulations the hidden layer of the original MSTDNN was left out. Methods that add hidden units between the input units and the state units are currently tested. The results for both manually optimized architectures and automatically optimized architectures are summarized below (see Tables 1-3). Results with different manually optimized architectures (single state TDNNs with hidden layer) are added for the handwritten character recognition task for comparison.

Fig.5 shows the connections from the input layer to the state layer after constructing the network with the ASO algorithm for the alphabet recognition task. 13 characters are modeled by three states, 8 characters are modeled by two states and 5 characters are modeled by one state only. Fig.5 also shows that the ASO algorithm constructs a rather inhomogenous architecture that would be hard to find manually. It should also be noted that the ASO algorithm constructs different architectures for the same data depending on the initialization of the network [17]. Fig.5 also shows that the

1. Melscale coefficients were computed from the power spectrogram by computing log energies in each melscale energy band, where adjacent coefficients in frequency overlap by one spectral sample and are smoothed by reducing the shared sample by 50%. Adjacent coefficients in time were collapsed for further data reduction resulting in an overall 10 ms frame rate. All coefficients were then normailzed.

2. During writing, the position and the pressure of the pen are recorded from the tablett. Resampling is used to reduce the temporal variations of the digits. From these data points, the directions and the curvatures of the pen strokes are computed and are added to the data.

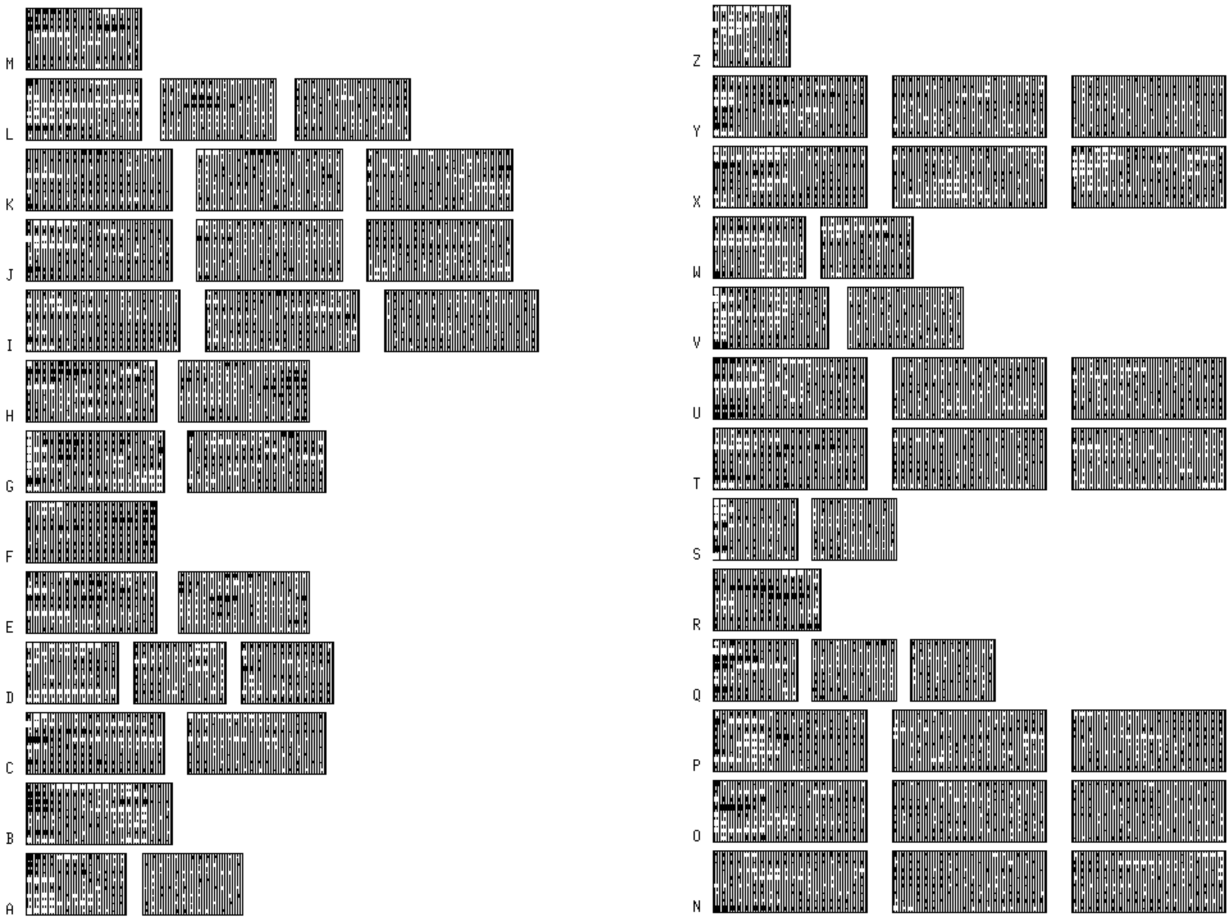

Fig. 5: The weights from the input units to the state units for the recognition of the capital letters after constructing the architecture (see Fig. Fig. 2 for further explanations). The character “A” is modeled by two state units. Each of these state units gets input from input windows with size 13. The character “B” is modeled by one state etc.. 13 characters are modeled by three states, 8 characters are modeled by two states and 5 characters are modeled by one state only. It can be seen that the new states generally have smaller weights (displayed by smaller blobs) than the older states.

new states generally have smaller weights than the older states. This can be explained by the amount of training that the weights get. Weights that are installed early in the training run are trained when the output error is still very high. Weights that are installed later when the output error is already lower do not get the same amount of training. This explains the good generalization abilities of some architectures with an unusual high number of trainable parameters that we sometimes observed.

Table 3 shows that the MSTDNN network optimized by ASO can adapt to different amounts of training data. The handtuned architecture performed equally well for the amount of data that it was optimized for, but did not generalize as well for more data and failed to learn a small subset completely for various learning rates and momentums.

VI. CONCLUSIONS

The results on three different tasks show that the ASO algorithm can achieve equal or better results than handtuned architectures without any tuning to the particular task. The results suggest that the ASO algorithm is able to optimize MSTDNN type networks for real world applications with varying amounts of training data effectively. Preliminary experiments with weight decay, weight elimination and OBD together with the ASO algorithm have been encouraging. In future, the algorithm will be applied to continous speech recognition and continous (cursive) handwritten character recognition tasks.

The design principles of the ASO algorithm (as described in paragraph III) should also allow the design of automatic structuring algorithms for other tasks.

TABLE 1.

Speech Recognition Performances (Alphabet Recognition)

	training	testing
manually optimized MSTDNN architecture with DTW	94.3%	85.0%
manually optimized MSTDNN with gaussian smoothing of the DTW path	98.9%	88.0%
automatically optimized MSTDNN architecture with standard DTW	97.1%	85.0%
automatically optimized MSTDNN with gaussian smoothing of the DTW path	99.5%	91.7%

TABLE 2.

Handwritten Character Recognition Performances (Digit Recognition)

	training	testing
manually optimized MSTDNN architecture without hidden units	98.3%	96.5%
automatic optimization of the window size, 1 state unit per output unit	97.2%	97.0%
automatic optimization of the window size and the number of state units	99.6%	98.0%
automatically optimized architecture with gaussian smoothing of the DTW path	100%	99.5%
TDNN architecture proposed by [Guyon, 1991] on the same data	100%	95.5%
TDNN architecture manually optimized for the same data	100%	98.5%

TABLE 3.

Handwritten Character Recognition Performances on Test Data (Capital Letters) depending on training set size

number of training patterns	TDNN architecture manually optimized for 1170 training patterns	automatically optimized MSTDNN architecture
520	no convergence	81.5%
1170	88.5%	88.5%
1560	90.5%	91.3%

ACKNOWLEDGEMENT

The authors gratefully acknowledge the support of the McDonnel-Pew Foundation (Cognitive Neuroscience Program) and would like to thank Patrick Haffner and Joe Tebelskis for lots of helpful discussions. Thanks to Stefan Manke for providing results from manually tuned networks.

REFERENCES

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

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

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

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

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

[6] Baum, E.B., and Haussler, D., “What Size Net Gives Valid Generalization”, *Neural Computation*, 1: 151-160, 1989

[7] Solla, S. , Schwartz, D.B., Tishby, N., and Levin, E., “Supervised Learning: a Theoretical Framework”. in: *Advances in Neural Information Processing Systems 2*, 1989

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

[9] Rumelhart, D.E. Learning and Generalization Plenary Address, IEEE International Conference on Neural Networks,San Diego, 1988

[10] Weigend, A.S., Hubermann, B.A., and Rumelhart, D.E., Predicting the Future: A Connectionist Approach, *TR SSL-90, Xerox Science Laboratory, Palo Alto, CA, 1990*

[11] Chauvin, Y. A Back-Propagation Algorithm with Optimal Use of Hidden Units In; Touretzky, D. editor, Neural Information Processing Systems 1, Denver, 1988, Morgan Kaufmann, 1989

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

[13] Fahlmann, S.E. The Cascade-Correlation Learning Architecture In; Touretzky, D. editor, Neural Information Processing Systems 2, Denver, 1989, Morgan Kaufmann, 1990

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

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

[16] Sakoe,H., and Chiba, S., Dynamic Programming Algorithm Optimization for Spoken Word Recognition, *IEEE Transactions on Acoustics, Speech and Signal Processing*, (26): 43-49, 1978

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