1997: BIDIRECTIONAL RNNs

Bidirectional Recurrent Neural Networks

Mike Schuster and Kuldip K. Paliwal, *Member, IEEE*

***Abstract*— In the first part of this paper, a regular recurrent neural network (RNN) is extended to a bidirectional recurrent neural network (BRNN). The BRNN can be trained without the limitation of using input information just up to a preset future frame. This is accomplished by training it simultaneously in positive and negative time direction. Structure and training procedure of the proposed network are explained. In regression and classification experiments on artificial data, the proposed structure gives better results than other approaches. For real data, classification experiments for phonemes from the TIMIT database show the same tendency.**

In the second part of this paper, it is shown how the proposed bidirectional structure can be easily modified to allow efficient estimation of the conditional posterior probability of complete symbol sequences without making any explicit assumption about the shape of the distribution. For this part, experiments on real data are reported.

***Index Terms*— Recurrent neural networks.**

TIMIT

I. Introduction

A. General

MANY classification and regression problems of engineering interest are currently solved with statistical approaches using the principle of "learning from examples." For a certain *model* with a given *structure* inferred from the prior knowledge about the problem and characterized by a number of *parameters*, the aim is to estimate these parameters accurately and reliably using a finite amount of training data. In general, the parameters of the model are determined by a supervised training process, whereas the structure of the model is defined in advance. Choosing a proper structure for the model is often the only way for the designer of the system to put in prior knowledge about the solution of the problem.

ANN's

Artificial neural networks (ANN's) (see [2] for an excellent introduction) are one group of models that take the principle "infer the knowledge from the data" to an extreme. In this paper, we are interested in studying ANN structures for one particular class of problems that are represented by temporal sequences of input–output data pairs. For these types of problems, which occur, for example, in speech recognition, time series prediction, dynamic control systems, etc., one of the challenges is to choose an appropriate network structure that, at least theoretically, is able to use all available input information to predict a point in the output space.

MLP

Many ANN structures have been proposed in the literature to deal with time varying patterns. Multilayer perceptrons (MLP's) have the limitation that they can only deal with static data patterns (i.e., input patterns of a predefined dimensionality), which requires definition of the size of the input window in advance. Waibel *et al.* [16] have pursued time delay neural networks (TDNN's), which have proven to be a useful improvement over regular MLP's in many applications. The basic idea of a TDNN is to tie certain parameters in a regular MLP structure without restricting the learning capability of the ANN too much. Recurrent neural networks (RNN's) [5], [8], [12], [13], [15] provide another alternative for incorporating temporal dynamics and are discussed in more detail in a later section.

TDNNs

RNNs

In this paper, we investigate different ANN structures for incorporating temporal dynamics. We conduct a number of experiments using both artificial and real-world data. We show the superiority of RNN's over the other structures. We then point out some of the limitations of RNN's and propose a modified version of an RNN called a bidirectional recurrent neural network, which overcomes these limitations.

B. Technical

Consider a (time) sequence of input data vectors

$$\boxed{\mathbf{x}_1^T = \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \ldots, \mathbf{x}_{T-1}, \mathbf{x}_T\}}$$

TIME SEQUENCE OF INPUT VECTORS

and a sequence of corresponding output data vectors

$$\boxed{\mathbf{y}_1^T = \{\mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3, \cdots, \mathbf{y}_{T-1}, \mathbf{y}_T\}}$$

OUTPUT VECTORS

with neighboring data-pairs (in time) being somehow statistically dependent. Given time sequences $\mathbf{x}_1^T$ and $\mathbf{y}_1^T$ as training data, the aim is to learn the rules to predict the output data given the input data. Inputs and outputs can, in general, be continuous and/or categorical variables. When outputs are continuous, the problem is known as a *regression problem*, and when they are categorical (class labels), the problem is known as a *classification problem*. In this paper, the term *prediction* is used as a general term that includes *regression* and *classification*.

1) Unimodal Regression: For unimodal regression or *function approximation*, the components of the output vectors are continuous variables. The ANN parameters are estimated to maximize some predefined objective criterion (e.g., maximize the likelihood of the output data). When the distribution of the errors between the desired and the estimated output vectors is assumed to be Gaussian with zero mean and a fixed global data-dependent variance, the likelihood criterion reduces to the

Manuscript received June 5, 1997. The associate editor coordinating the review of this paper and approving it for publication was Prof. Jenq-Neng Hwang.

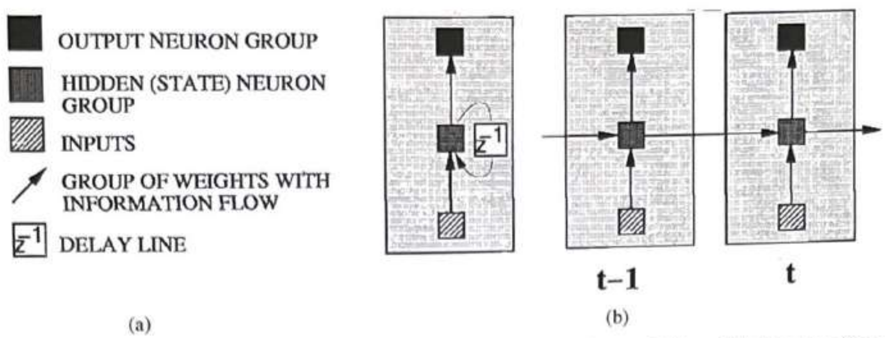
M. Schuster is with the ATR Interpreting Telecommunications Research Laboratory, Kyoto, Japan.

K. K. Paliwal is with the ATR Interpreting Telecommunications Research Laboratory, Kyoto, Japan, on leave from the School of Microelectronic Engineering, Griffith University, Brisbane, Australia.

Publisher Item Identifier S 1053-587X(97)08055-0.

CLASSIFICATION VS REGRESSION

(x_t, y_t)



INVOKE DSP PRINCIPLES

Fig. 1. General structure of a regular unidirectional RNN shown (a) with a delay line and (b) unfolded in time for two time steps.

MSE

convenient Euclidean distance measure between the desired and the estimated output vectors or the *mean-squared-error criterion*, which has to be minimized during training [2]. It has been shown by a number of researchers [2], [9] that neural networks can estimate the conditional average of the desired output (or target) vectors at their network outputs, i.e., $\hat{\mathbf{y}}_t = E[\mathbf{y}_t|\mathbf{x}_1^T]$, where $E[\cdot]$ is an expectation operator.

2) Classification: In the case of a classification problem, one seeks the most probable class out of a given pool of K classes for every time frame t, given an input vector sequence $\mathbf{x}_1^T$. To make this kind of problem suitable to be solved by an ANN, the categorical variables are usually coded as vectors as follows. Consider that i is the desired class label for the frame at time t. Then, construct an output vector $\mathbf{y}_t$ such that its ith component is one and other components are zero. The output vector sequence $\mathbf{y}_1^T$ constructed in this manner along with the input vector sequence $\mathbf{x}_1^T$ can be used to train the network under some optimality criterion, usually the cross-entropy criterion [2], [9], which results from a maximum likelihood estimation assuming a multinomial output distribution. It has been shown [3], [6], [9] that the ith network output at each time point t can be interpreted as an estimate of the conditional posterior probability of class membership $[\hat{y}_t^{(k)} = \Pr(C_t = k|\mathbf{x}_1^T)]$ for class i, with the quality of the estimate depending on the size of the training data and the complexity of the network.

ONE-HOT ENCODING

MLE

For some applications, it is not necessary to estimate the conditional posterior probability $\Pr(C_t = k|\mathbf{x}_1^T)$ of a *single* class given the sequence of input vectors but the conditional posterior probability $\Pr(c_1, c_2, \cdots, c_T|\mathbf{x}_1^T)$ of a *sequence* of classes given the sequence of input vectors.[1]

SINGLE CLASS vs. SEQUENCE OF CLASSES

C. Organization of the Paper

This paper is organized in two parts. Given a series of paired input/output vectors $\{(\mathbf{x}_t, \mathbf{y}_t), t = 1, 2, \cdots, T\}$, we want to train bidirectional recurrent neural networks to perform the following tasks.

- Unimodal regression (i.e., compute $\hat{\mathbf{y}}_t = E[\mathbf{y}_t|\mathbf{x}_1^T]$) or classification [i.e., compute $\hat{y}_t^{(k)} = \Pr(C_t = k|\mathbf{x}_1^T)$ for every output class k and decide the class using the maximum *a posteriori* decision rule]. In this case, the outputs are treated statistically independent. Experiments for this part are conducted for artificial toy data as well as for real data.
- Estimation of the conditional probability of a complete sequence of classes of length T using all available input information [i.e., compute $\Pr(c_1, c_2, \cdots, c_T|\mathbf{x}_1^T)$]. In this case, the outputs are treated as being statistically dependent, which makes the estimation more difficult and requires a slightly different network structure than the one used in the first part. For this part, results of experiments for real data are reported.

II. Prediction Assuming Independent Outputs

A. Recurrent Neural Networks

DELAY LINE

RNN's provide a very elegant way of dealing with (time) sequential data that embodies correlations between data points that are close in the sequence. Fig. 1 shows a basic RNN architecture with a delay line and unfolded in time for two time steps. In this structure, the input vectors $\mathbf{x}_t$ are fed one at a time into the RNN. Instead of using a fixed number of input vectors as done in the MLP and TDNN structures, this architecture can make use of all the available input information up to the current time frame t_c (i.e., $\{\mathbf{x}_t, t = 1, 2, \cdots, t_c\}$) to predict $\mathbf{y}_{t_c}$. How much of this information is captured by a particular RNN depends on its structure and the training algorithm. An illustration of the amount of input information used for prediction with different kinds of NN's is given in Fig. 2.

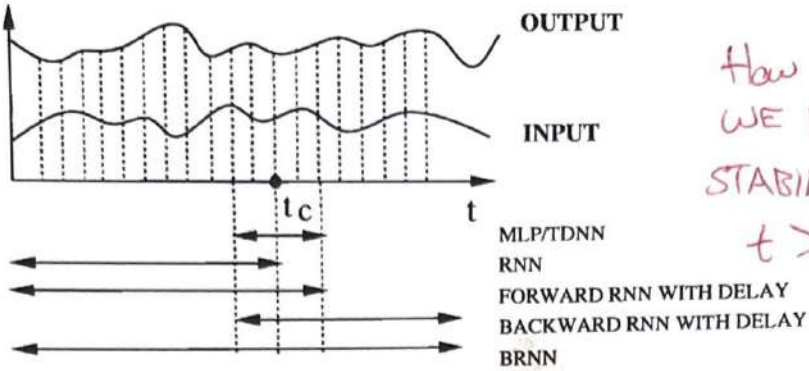
Future input information coming up later than t_c is usually also useful for prediction. With an RNN, this can be partially achieved by delaying the output by a certain number of M time frames to include future information up to $\mathbf{x}_{t_c+M}$ to predict $\mathbf{y}_{t_c}$ (Fig. 2). Theoretically, M could be made very large to capture all the available future information, but in practice, it is found that prediction results drop if M is too large. A possible explanation for this could be that with rising M, the modeling power of the RNN is increasingly concentrated on remembering the input information up to $\mathbf{x}_{t_c+M}$ for the prediction of $\mathbf{y}_{t_c}$, leaving less modeling power for combining the prediction knowledge from different input vectors.

OPTIMAL DELAY

While delaying the output by some frames has been used successfully to improve results in a practical speech recognition system [12], which was also confirmed by the experiments conducted here, the optimal delay is task dependent and has to

[1] Here, we want to make a distinction between C_t and c_t. C_t is a categorical random variable, and c_t is its value.

$M_a >> M_b \Rightarrow \frac{\partial C}{\partial w} \to 0$ VANISHING GRADIENT

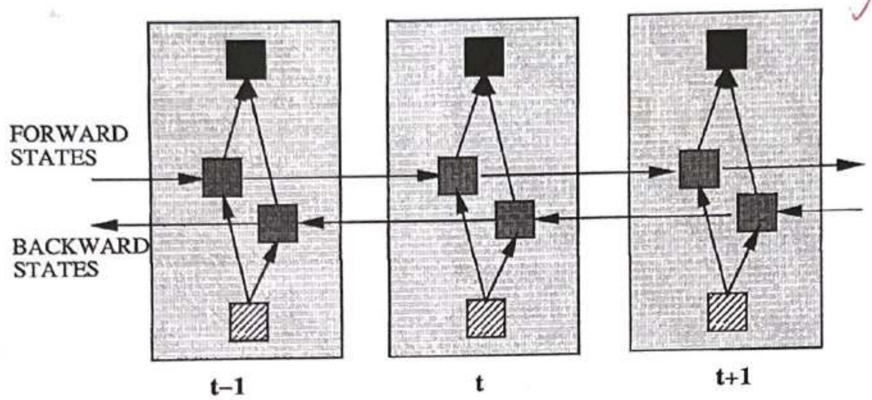


How CAN WE PROMOTE STABILITY S.T. $t >> t_c$

Fig. 2. Visualization of the amount of input information used for prediction by different network structures.

Lin. $\hat{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$

Log. $\hat{p} = \frac{\prod_{i=1}^{n} p_i^{w_i}}{\sum_{j=1}^{n}\prod_{i=1}^{n} p_i^{w_i}}$



ACYCLIC BACKPROPAGATION

Fig. 3. General structure of the bidirectional recurrent neural network (BRNN) shown unfolded in time for three time steps.

be found by the "trial and error" error method on a validation test set. Certainly, a more elegant approach would be desirable.

To use all available input information, it is possible to use two separate networks (one for each time direction) and then somehow merge the results. Both networks can then be called experts for the specific problem on which the networks are trained. One way of *merging the opinions of different experts* is to assume the opinions to be independent, which leads to arithmetic averaging for regression and to geometric averaging (or, alternatively, to an arithmetic averaging in the log domain) for classification. These merging procedures are referred to as *linear opinion pooling* and *logarithmic opinion pooling*, respectively [1], [7]. Although simple merging of network outputs has been applied successfully in practice [14], it is generally not clear how to merge network outputs in an optimal way since different networks trained on the same data can no longer be regarded as independent.

LINEAR OPINION POOLING

LOG OPINION POOLING

IMPORTANT DISTINCTION!

B. Bidirectional Recurrent Neural Networks

To overcome the limitations of a regular RNN outlined in the previous section, we propose a bidirectional recurrent neural network (BRNN) that can be trained using all available input information in the past and future of a specific time frame.

STRUCTURE

1) Structure: The idea is to split the state neurons of a regular RNN in a part that is responsible for the positive time direction (forward states) and a part for the negative time direction (backward states). Outputs from forward states are not connected to inputs of backward states, and vice versa. This leads to the general structure that can be seen in Fig. 3, where it is unfolded over three time steps. It is not possible to display the BRNN structure in a figure similar to Fig. 1 with the delay line since the delay would have to be positive and negative in time. Note that without the backward states, this structure simplifies to a regular unidirectional forward RNN, as shown in Fig. 1. If the forward states are taken out, a regular RNN with a reversed time axis results. With both time directions taken care of in the same network, input information in the past and the future of the currently evaluated time frame can directly be used to minimize the objective function without the need for delays to include future information, as for the regular unidirectional RNN discussed above.

TRAINING

2) Training: The BRNN can principally be trained with the same algorithms as a regular unidirectional RNN because there are no interactions between the two types of state neurons and, therefore, can be unfolded into a general feed-forward network. However, if, for example, any form of back-propagation through time (BPTT) is used, the forward and backward pass procedure is slightly more complicated because the update of state and output neurons can no longer be done one at a time. If BPTT is used, the forward and backward passes over the unfolded BRNN over time are done almost in the same way as for a regular MLP. Some special treatment is necessary only at the beginning and the end of the training data. The forward state inputs at $t = 1$ and the backward state inputs at $t = T$ are not known. Setting these could be made part of the learning process, but here, they are set arbitrarily to a fixed value (0.5). In addition, the local state derivatives at $t = T$ for the forward states and at $t = 1$ for the backward states are not known and are set here to zero, assuming that the information beyond that point is not important for the current update, which is, for the boundaries, certainly the case. The training procedure for the unfolded bidirectional network over time can be summarized as follows.

1) *FORWARD PASS*
   Run all input data for one time slice $1 \leq t \leq T$ through the BRNN and determine all predicted outputs.
   a) Do forward pass just for forward states (from $t = 1$ to $t = T$) and backward states (from $t = T$ to $t = 1$).
   b) Do forward pass for output neurons.
2) *BACKWARD PASS*
   Calculate the part of the objective function derivative for the time slice $1 \leq t \leq T$ used in the forward pass.
   a) Do backward pass for output neurons.
   b) Do backward pass just for forward states (from $t = T$ to $t = 1$) and backward states (from $t = 1$ to $t = T$).
3) *UPDATE WEIGHTS*

$1 \rightarrow T$ FS
$T \rightarrow 1$ BS
FORWARD PASS

$T \rightarrow 1$ FS $\Rightarrow$ $1 \rightarrow T$ BS
BACKWARD PASS

C. Experiments and Results

In this section, we describe a number of experiments with the goal of comparing the performance of the BRNN structure with that of other structures. In order to provide a fair comparison, we have used different structures with a comparable number of parameters as a rough complexity measure. The experiments are done for artificial data for both regression and classification tasks with small networks to allow extensive experiments and for real data for a phoneme classification task with larger networks.

WHAT ABT LARGER NETS?

1) Experiments with Artificial Data:

a) Description of Data: In these experiments, an artificial data set is used to conduct a set of regression and classification experiments. The artificial data is generated as follows. First, a stream of 10 000 random numbers between zero and one is created as the one-dimensional (1-D) input data to the ANN. The 1-D output data (the desired output) is obtained as the weighted sum of the inputs within a window of 10 frames to the left and 20 frames to the right with respect to the current frame. The weighting falls of linearly on both sides as

WEIGHTING PROCEDURE

$$\boxed{\begin{aligned} y(t) = \frac{1}{10} \sum_{\Delta t=-10}^{-1} x(t+\Delta t) \cdot \left(1 - \frac{|\Delta t|}{10}\right) \\ + \frac{1}{20} \sum_{\Delta t=0}^{19} x(t+\Delta t) \cdot \left(1 - \frac{|\Delta t|}{20}\right). \end{aligned}}$$

The weighting procedure introduces correlations between neighboring input/output data pairs that become less for data pairs further apart. Note that the correlations are not symmetrical, being on the right side of each frame, which is twice as "broad" as on the left side. For the classification experiments, the output data is mapped to two classes, with class 0 for all output values below (or equal to) 0.5 and class 1 for all output values above 0.5, giving approximately 59% of the data to class 0 and 41% to class 1.

TABLE I
DETAILS OF REGRESSION AND CLASSIFICATION ARCHITECTURES EVALUATED IN OUR EXPERIMENTS

STRUCTURE	STATES (FOR/BACK)	SHIFT-RANGE
RNN-FOR	2/0	-5 to +20
RNN-BACK	0/2	+5 to -20
MERGE	2/2	-2/+2 to +10/-10
BRNN	2/2	none

b) Experiments: Separate experiments are conducted for regression and classification tasks. For each task, four different architectures are tested (Table I). Type "MERGE" refers to the merged results of type RNN-FOR and RNN-BACK because they are regular unidirectional recurrent neural networks trained in the forward and backward time directions, respectively. The first three architecture types are also evaluated over different shifts of the output data in the positive time direction, allowing the RNN to use future information, as discussed above.

100 x INIT

Every test (ANN training/evaluation) is run 100 times with different initializations of the ANN to get at least partially rid of random fluctuations of the results due to convergence to local minima of the objective function. All networks are trained with 200 cycles of a modified version of the resilient propagation (RPROP) technique [10] and extended to a RPROP through a time variant. All weights in the structure are initialized in the range $(-1, 1)$ drawn from the uniform distribution, except the output biases, which are set so that the corresponding output gives the prior average of the output data in case of zero input activation.

200 CYCLES

RPROP

$w \sim U(-1,1)$

For the regression experiments, the networks use the tanh ( ) activation function and are trained to minimize the mean-squared-error objective function. For type "MERGE," the arithmetic mean of the network outputs of "RNN-FOR" and "RNN-BACK" is taken, which assumes them to be independent, as discussed above for the *linear opinion pool.*

tanh()

MSE

LINEAR OPINION POOL

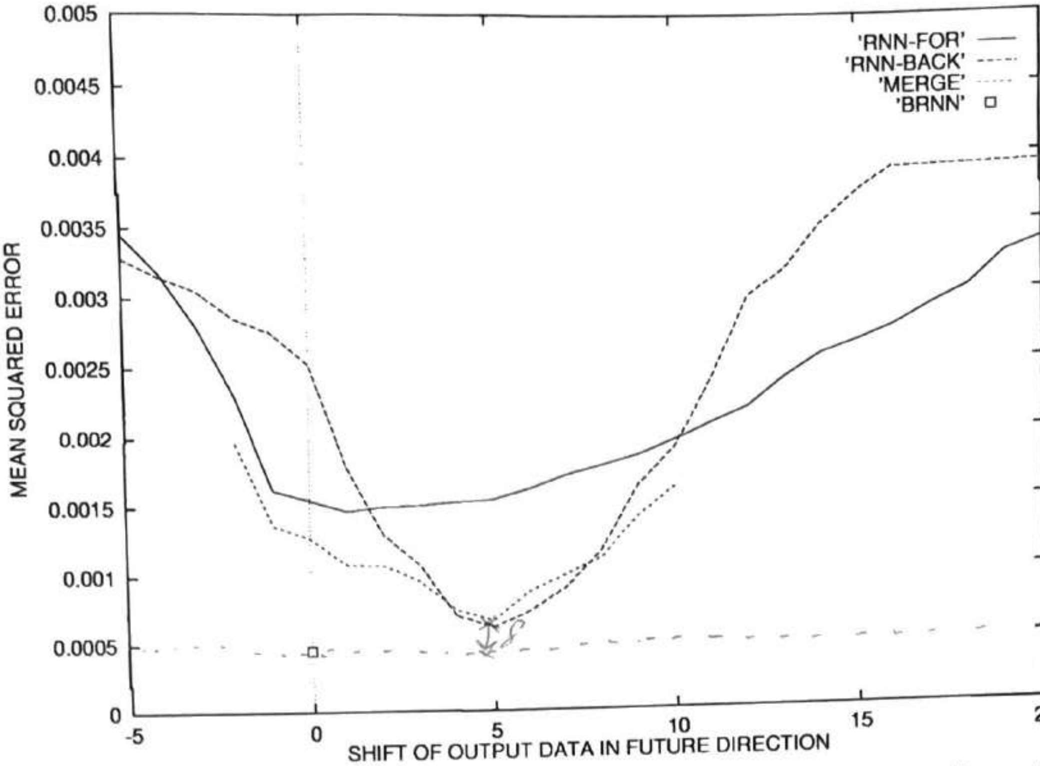
For the classification experiments, the output layer uses the "softmax" output function [4] so that outputs add up to one and can be interpreted as probabilities. As commonly used for ANN's to be trained as classifiers, the cross-entropy objective function is used as the optimization criterion. Because the outputs are probabilities assumed to be generated by independent events, for type "MERGE," the normalized geometric mean (*logarithmic opinion pool*) of the network outputs of "RNN-FOR" and "RNN-BACK" is taken.

Softmax

cross-entropy

LOG OPINION POOL

c) Results: The results for the regression and the classification experiments averaged over 100 training/evaluation runs can be seen in Figs. 4 and 5, respectively. For the regression task, the mean squared error depending on the shift of the output data in positive time direction seen from the time axis of the network is shown. For the classification task, the recognition rate, instead of the mean value of the objective function (which would be the mean cross-entropy), is shown



WOULD BE INTERESTING TO SEE THIS THEORY IN TERMS OF HYPERBOLIC ATTRACTORS

Fig. 4. Averaged results (100 runs) for the regression experiment on artificial data over different shifts of the output data with respect to the input data in future direction (viewed from the time axis of the corresponding network) for several structures.

because it is a more familiar measure to characterize results of classification experiments.

Several interesting properties of RNN's in general can be directly seen from these figures. The minimum (maximum) for the regression (classification) task should be at 20 frames delay for the forward RNN and at 10 frames delay for the backward RNN because at those points, all information for a perfect regression (classification) has been fed into the network. Neither is the case because the modeling power of the networks given by the structure and the number of free parameters is not sufficient for the optimal solution. Instead, the single time direction networks try to make a tradeoff between "remembering" the past input information, which is useful for regression (classification), and "knowledge combining" of currently available input information. This results in an optimal delay of one (two) frame for the forward RNN and five (six) frames for the backward RNN. The optimum delay is larger for the backward RNN because the artificially created correlations in the training data are not symmetrical with the important information for regression (classification) being twice as dense on the left side as on the right side of each frame. In the case of the backward RNN, the time series is evaluated from right to left with the denser information coming up later. Because the denser information can be evaluated easier (fewer parameters are necessary for a contribution to the objective function minimization), the optimal delay is larger for the backward RNN. If the delay is so large that almost no important information can be saved over time, the network converges to the best possible solution based only on prior information. This can be seen for the classification task with the backward RNN, which converges to 59% (prior of class 0) for more than 15 frames delay.

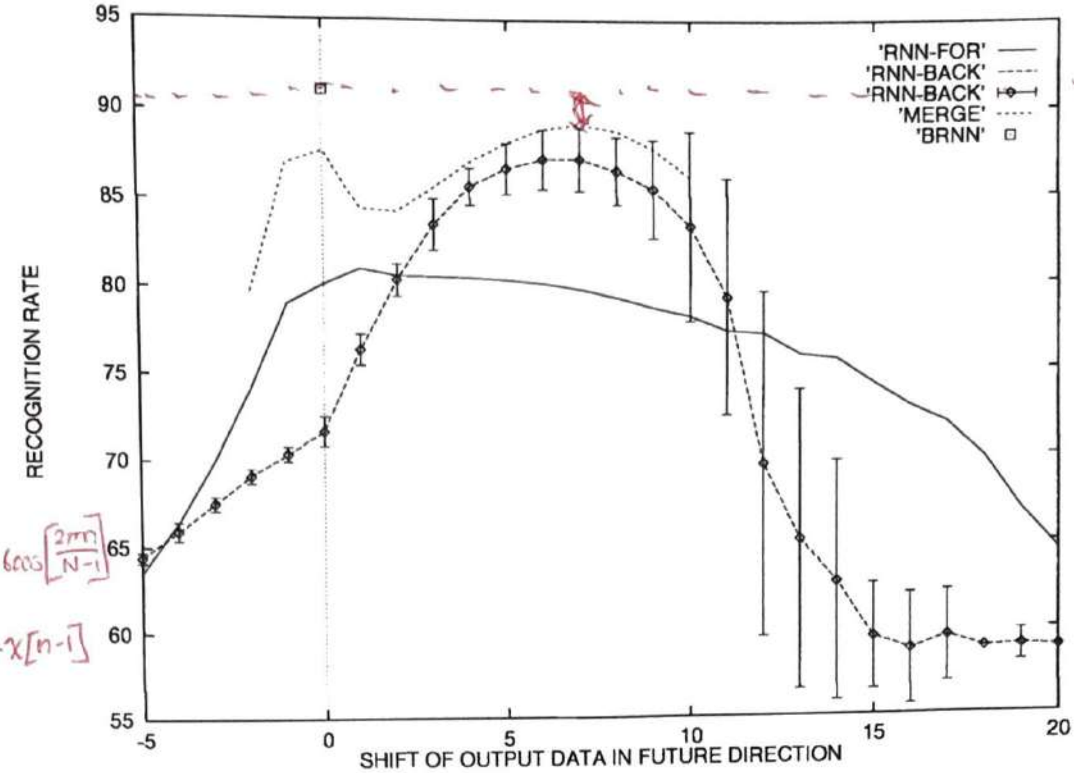
Another sign for the tradeoff between "remembering" and "knowledge combining" is the variation in the standard deviation of the results, which is only shown for the backward RNN in the classification task. In areas where both mechanisms could be useful (a 3 to 17 frame shift), different local minima of the objective function correspond to a certain amount to either one of these mechanisms, which results in larger fluctuations of the results than in areas where "remembering" is not very useful (-5 to 3 frame shift) or not possible (17 to 20 frame shift).

If the outputs of forward and backward RNN's are merged so that all available past and future information for regression (classification) is present, the results for the delays tested here (-2 to 10) are, in almost all cases, better than with only one network. This is no surprise because besides the use of more useful input information, the number of free parameters for the model doubled.

For the BRNN, it does not make sense to delay the output data because the structure is already designed to cope with all available input information on both sides of the currently evaluated time point. Therefore, the experiments for the BRNN are only run for SHIFT = 0. For the regression and classification tasks tested here, the BRNN clearly performs better than the network "MERGE" built out of the single time-direction networks "RNN-FOR" and "RNN-BACK," with a comparable number of total free parameters.

2) *Experiments with Real Data:* The goal of the experiments with real data is to compare different ANN structures

HAMMING:
$w(n) = 0.54 - 0.46\cos\left[\frac{2\pi n}{N-1}\right]$
Pre-emphasis:
$y[n] = x[n] - \alpha \cdot x[n-1]$



HAMMING WINDOW

Fig. 5. Averaged results for the classification experiment on artificial data.

$\alpha = 0.97$

TIMIT

for the classification of phonemes from the TIMIT speech database. Several regular MLP's and recurrent neural network architectures, which make use of different amounts of acoustic context, are tested here.

6.3K SENTENCES

a) Description of Data: The TIMIT phoneme database is a well-established database consisting of 6300 sentences spoken by 630 speakers (ten sentences per speaker). Following official TIMIT recommendations, two of the sentences (which are the same for every speaker) are not included in our experiments, and the remaining data set is divided into two sets: 1) the training data set consisting of 3696 sentences from 462 speakers and 2) the test data set consisting of 1344 sentences from 168 speakers. The TIMIT database provides hand segmentation of each sentence in terms of phonemes and a phonemic label for every segment out of a pool of 61 phonemes. This gives 142 910 phoneme segments for training and 51 681 for testing.

FEATURE

BOUNDARY LOCATIONS

TRANSFORMATION

In our experiments, every sentence is transformed into a vector sequence using three levels of feature extraction. First, features are extracted every frame to represent the raw waveform in a compressed form. Then, with the knowledge of the boundary locations from the corresponding label files, segment features are extracted to map the information from an arbitrary length segment to a fixed-dimensional vector. A third transformation is applied to the segment feature vectors to make them suitable as inputs to a neural net. These three steps are briefly described below.

12 MFCC's

1) **Frame Feature Extraction:** As frame features, 12 regular MFCC's (from 24 mel-space frequency bands) plus the log-energy are extracted every 10 ms with a 25.6-ms Hamming window and a preemphasis of 0.97. This is a commonly used feature extraction procedure for speech signals at the frame level [17].
2) **Segment Feature Extraction:** From the frame features, the segment features are extracted by dividing the segment in time into five equally spaced regions and computing the area under the curve in each region, with the function values between the data points linearly interpolated. This is done separately for each of the 13 frame features. The duration of the segment is used as an additional segment feature. This results in a 66-dimensional segment feature vector.
3) **Neural Network Preprocessing:** Although ANN's can principally handle any form of input distributions, we have found in our experiments that the best results are achieved with Gaussian input distributions, which matches the experiences from [12]. To generate an "almost-Gaussian distribution," the inputs are first normalized to zero mean and unit variance on a sentence basis, and then, every feature of a given channel[2] is quantized using a scalar quantizer having 256 reconstruction levels (1 byte). The scalar quantizer is designed to maximize the entropy of the channel for the whole training data. The maximum entropy scalar quantizer can be easily designed for each channel by arranging the channel points in ascending order according to their feature values and putting (almost) an equal number of

$\mu = 0$
$\sigma^2 = 1$

[2]Here, each vector has a dimensionality of 66. Temporal sequence of each component (or feature) of this vector defines one channel. Thus, we have here 66 channels.

24 MEL-SPACE BANDS

MFCC's = MEL FREQUENCY CEPSTRAL COEFFICIENTS ⇒ SPECTRAL ENVELOPE

STATISTICALLY SIGNIFICANCE +2.7%

TABLE II
TIMIT PHONEME CLASSIFICATION RESULTS FOR FULL TRAINING AND TEST DATA SETS WITH ≈ 13 000 PARAMETERS

Structure	Rec-Rate % TRAIN 61 (39)	Rec-Rate % TEST 61 (39)
MLP-1 (1 segment)	61.32 (70.20)	59.67 (68.95)
MLP-3 (3 segments)	68.37 (75.74)	65.69 (73.48)
MLP-5 (5 segments)	66.97 (74.60)	64.32 (72.35)
FOR-RNN	65.42 (74.27)	63.20 (72.51)
BACK-RNN	64.57 (72.83)	61.91 (70.94)
FOR-RNN (1 delay)	68.45 (75.37)	65.83 (73.00)
FOR-RNN (2 delay)	65.97 (73.03)	63.27 (70.77)
MERGE (FOR+BACK)	66.94 (75.01)	65.28 (73.73)
BRNN	70.73 (77.33)	68.53 (75.48)

erf^{-1}

channel points in each quantization cell. For presentation to the network, the byte-coded value is remapped with value $= \mathrm{erf}^{-1}[2 \cdot (\mathrm{byte} + 1/2)/256 - 1]$, where erf^{-1} is the inverse error function [erf( ) is part of math.h library in C]. This mapping produces on average a distribution that is similar to a Gaussian distribution.

The feature extraction procedure described above transforms every sentence into a sequence of fixed dimensional vectors representing acoustic phoneme segments. The sequence of these segment vectors (along with their phoneme class labels) are used to train and test different ANN structures for classification experiments, as described below.

b) Experiments: Experiments are performed here with different ANN structures (e.g., MLP, RNN, and BRNN), which allow the use of different amounts of acoustic context. The MLP structure is evaluated for three different amounts of acoustic context as input.

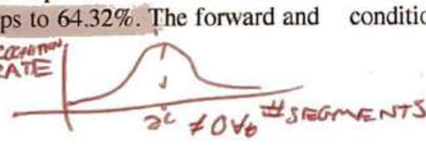
1) one segment;
2) three segments (middle, left, and right);
3) five segments (middle, two left, and two right).

The evaluated RNN structures are unidirectional forward and backward RNN's that use all acoustic context on one side, two forward RNN's with one and two segment delays to incorporate right-hand information, the merged network built out of the unidirectional forward and backward RNN's, and the BRNN. The structures of all networks are adjusted so that each of them has about the same number of free parameters (approximately 13 000 here).

c) Results: Table II shows the phoneme classification results for the full training and test set. Although the database is labeled to 61 symbols, a number of researchers have chosen to map them to a subset of 39 symbols. Here, results are given for both versions, with the results for 39 symbols being simply a mapping from the results obtained for 61 symbols. Details of this standard mapping can be found in [11].

The baseline performance assuming neighboring segments to be independent gives 59.67% recognition rate (MLP-1) on the test data. If three consecutive segments are taken as the inputs (MLP-3), loosening the independence assumption to three segments, the recognition rate goes up to 65.69%. Using five segments (MLP-5), the structure is not flexible enough to make use of the additional input information, and as a result, the recognition rate drops to 64.32%. The forward and backward RNN's (FOR-RNN, BACK-RNN), making use of input information only on one side of the current segment, give lower recognition rates (63.2 and 61.91%) than the forward RNN with one segment delay (65.83%). With a two segment delay, too much information has to be saved over time, and the result drops to 63.27% (FOR-RNN, two delay), although theoretically, more input information than for the previous network is present. The merging of the outputs of two separate networks (MERGE) trained in each time direction gives a recognition rate of 65.28% and is worse than the forward RNN structure using one segment delay. The bidirectional recurrent neural network (BRNN) structure results in the best performance (68.53%).

BELL CURVE



III. PREDICTION ASSUMING DEPENDENT OUTPUTS

ESTIMATE COND. POSTERIOR PROBABILITY OF SEQUENCE OF CLASSES

In the preceding section, we have estimated the conditional posterior probability $\Pr(C_t = k|\mathbf{x}_1^T)$ of a *single* class k at a certain time point t, given the sequence of input vectors $\mathbf{x}_1^T$. For some applications, it is necessary to estimate the conditional posterior probability $\Pr(c_1, c_2, \cdots, c_T|\mathbf{x}_1^T)$ of a *sequence* of all classes from $t = 1$ to $t = T$ instead of $\Pr(C_t = k|\mathbf{x}_1^T)$, given the sequence of input vectors. This is a difficult problem, and no general practical solution is known, although this type of estimation is essential for many pattern recognition applications where sequences are involved.

A. Approach

Bidirectional recurrent neural networks can provide an approach to estimate $\Pr(c_1, c_2, \cdots, c_T|\mathbf{x}_1^T)$. Using the rule $p(x, y) = p(x|y)p(y)$, we decompose the sequence posterior probability as

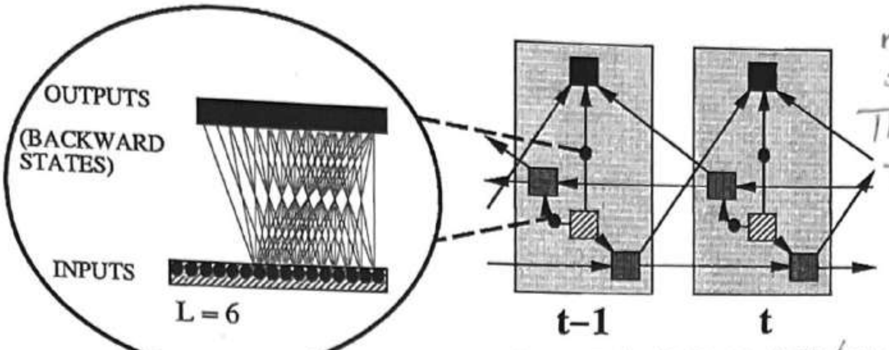
$$\boxed{\begin{aligned}\Pr(c_1, c_2, \cdots, c_T|\mathbf{x}_1^T) &= \underbrace{\prod_{t=1}^{T} \Pr(c_t|c_{t+1}, c_{t+2}, \cdots, c_T, \mathbf{x}_1^T)}_{\text{backward posterior probability}} \\ &= \underbrace{\prod_{t=1}^{T} \Pr(c_t|c_1, c_2, \cdots, c_{t-1}, \mathbf{x}_1^T)}_{\text{forward posterior probability}}.\end{aligned}}$$

POSTERIOR PROBABILITY DECOMPOSITION

The probability term within the product is the conditional probability of an output class given all the input to the right- and left-hand side plus the class sequence on one side of the currently evaluated input vector. The two ways of decomposing $\Pr(c_1, c_2, \cdots, c_T|\mathbf{x}_1^T)$ (many more are possible) are here referred to as the *forward* and the *backward* posterior probabilities. Note that these decompositions are only a simple application of probability rules, i.e., no assumptions concerning the shape of the distributions is made.

In the present approach, the goal is to train a network to estimate conditional probabilities of the kind $\Pr(c_t|c_1, c_2, \cdots, c_{t-1}, \mathbf{x}_1^T)$ (which are the probability terms in the products). The estimates for these probabilities can then be combined by using the formulas above to estimate the full conditional probability of the sequence. It should be noted

EXPLAIN!



min $\frac{\partial C}{\partial w}$
subject to
$\prod Pr(c_t|c_1, c_2, \ldots, c_{t-1}, x_1^T) - \prod Pr(c_t|c_{t+1}, c_{t+2}, \ldots, c_T, x_1^T) = 0$

SET EQUALITY OF FORWARD/BACKWARD POSTEROR DISTRIBUTION AS OPTIMIZATION CONSTRAINT

Fig. 6. Modified bidirectional recurrent neural network structure shown here with extensions for the forward posterior probability estimation.

that the forward and the backward posterior probabilities are exactly equal, provided the probability estimator is perfect. However, if neural networks are used as probability estimators, this will rarely be the case because different architectures or different local minima of the objective function to be minimized correspond to estimators of different performance. It might therefore be useful to combine several estimators to get a better estimate of the quantity of interest using the methods of the previous section. Two candidates that could be merged here are $\Pr(c_t|c_1, c_2, \cdots, c_{t-1}, \mathbf{x}_1^T)$ and $\Pr(c_t|c_{t+1}, c_{t+2}, \cdots, c_T, \mathbf{x}_1^T)$ at each time point t.

B. Modified Bidirectional Recurrent Neural Networks

A slightly modified BRNN structure can efficiently be used to estimate the conditional probabilities of the kind $\Pr(c_t|c_1, c_2, \cdots, c_{t-1}, \mathbf{x}_1^T)$, which is conditioned on continuous $(\mathbf{x}_1^T)$ and discrete inputs $(c_1, c_2, \cdots, c_{t-1})$. Assume that the input for a specific time t_c is coded as one long vector containing the target output class c_t and the original input vector $\mathbf{x}_t$ with, for example, the discrete input c_t coded in the first L dimensions of the whole input vector. To make the BRNN suitable to estimate $\Pr(c_t|c_1, c_2, \cdots, c_{t-1}, \mathbf{x}_1^T)$, two changes are necessary. First, instead of connecting the forward and backward states to the current output states, they are connected to the next and previous output states, respectively, and the inputs are directly connected to the outputs. Second, if in the resulting structure the first L weight connections from the inputs to the backward states and the inputs to the outputs are cut, then only discrete input information from $t < t_c$ can be used to make predictions. This is exactly what is required to estimate the forward posterior probability $\Pr(c_t|c_1, c_2, \cdots, c_{t-1}, \mathbf{x}_1^T)$. Fig. 6 illustrates this change of the original BRNN architecture. Cutting the input connections to the forward states instead of the backward states gives the architecture for estimating the backward posterior probability. Theoretically, all discrete and continuous inputs $c_1, c_2, \cdots, c_{t-1}, \mathbf{x}_1^T$ that are necessary to estimate the probability are still accessible for a contribution to the prediction. During training, the bidirectional structure can adapt to the best possible use of the input information, as opposed to structures that do not provide part of the input information because of the limited size of the input windows (e.g., in MLP and TDNN) or one-sided windows (unidirectional RNN).

TABLE III
CLASSIFICATION RESULTS FOR FULL TIMIT TRAINING AND TEST DATA WITH 61 (39) SYMBOLS

Structure	Rec-Rate % TRAIN 61 (39)	Rec-Rate % TEST 61 (39)
forward. mod. BRNN	79.11 (84.42)	72.70 (79.08)
backward. mod. BRNN	79.38 (83.27)	72.74 (77.44)
both merged, lin.	83.57 (87.17)	77.53 (82.11)
both merged, log.	83.89 (87.45)	77.75 (82.38)

C. Experiments and Results

CROSS ENTROPY

1) Experiments: Experiments are performed using the full TIMIT data set. To include the output (target) class information, the original 66-dimensional feature vectors are extended to 72 dimensions. In the first six dimensions, the corresponding output class is coded in a binary format (binary $[0, 1] \rightarrow$ network input $[-1, 1]$). Two different structures of the modified BRNN (one for the forward and the other for the backward posterior probability) are trained separately as classifiers using the cross-entropy objective function. The output neurons have the softmax activation function and the remaining ones the $\tanh()$ activation function. The forward (backward) modified BRNN has 64 (32) forward and 32 (64) backward states. Additionally, 64 hidden neurons are implemented before the output layer. This results in a forward (backward) modified BRNN structure with 26 333 weights. These two structures, as well as their combination—merged as a linear and a logarithmic opinion pool—are evaluated for phoneme classification on the test data.

2) Results: The results for the phoneme classification task are shown in Table III. It can be seen that the combination of the forward and backward modified BRNN structures results in much better performance than the individual structures. This shows that the two structures, even though they are trained on the same training data set to compute the same probability $\Pr(c_1, c_2, \cdots, c_T|\mathbf{x}_1^T)$, are providing different estimates of this probability, and as a result, the combination of the two networks is giving better results. The slightly better results for the logarithmic opinion pool with respect to the linear opinion pool suggest that it is reasonable to assume the two estimates for the probability $\Pr(c_1, c_2, \cdots, c_T|\mathbf{x}_1^T)$ as independent, although the two structures are trained on the same data set.

It should be noted that the modified BRNN structure is only a tool to estimate the conditional probability of a *given* class sequence and that it does not provide a class sequence with the highest probability. For this, all possible class sequences have to be searched to get the most probable class sequence (which is a procedure that has to be followed if one is interested in a problem like continuous speech recognition). In the experiments reported in this section, we have used the class sequence provided by the TIMIT data base. Therefore, the context on the (right or left) output side is known and is *correct*.

IV. Discussion and Conclusion

In the first part of this paper, a simple extension to a regular recurrent neural network structure has been presented, which makes it possible to train the network in both time directions simultaneously. Because the network concentrates on minimizing the objective function for both time directions simultaneously, there is no need to worry about how to merge outputs from two separate networks. There is also no need to search for an "optimal delay" to minimize the objective function in a given data/network structure combination because all future and past information around the currently evaluated time point is theoretically available and does not depend on a predefined delay parameter. Through a series of extensive experiments, it has been shown that the BRNN structure leads to better results than the other ANN structures. In all these comparisons, the number of free parameters has been kept to be approximately the same. The training time for the BRNN is therefore about the same as for the other RNN's. Since the search for an optimal delay (an additional search parameter during development) is not necessary, the BRNN's can provide, in comparison to other RNN's investigated in this paper, faster development of real applications with better results.

In the second part of this paper, we have shown how to use slightly modified bidirectional recurrent neural nets for the estimation of the conditional probability of symbol sequences without making any explicit assumption about the shape of the output probability distribution. It should be noted that the modified BRNN structure is only a tool to estimate the conditional probability of a *given* class sequence; it does not provide the class sequence with the highest probability. For this, all possible class sequences have to be searched to get the most probable class sequence. We are currently working on designing an efficient search engine, which will use only ANN's to find the most probable class sequence.

References

[1] J. O. Berger, *Statistical Decision Theory and Bayesian Analysis*. Berlin, Germany: Springer-Verlag, 1985.

[2] C. M. Bishop, *Neural Networks for Pattern Recognition*. Oxford, U.K.: Clarendon, 1995.

[3] H. Bourlard and C. Wellekens, "Links between Markov models and multilayer perceptrons," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 12, pp. 1167–1178, Dec. 1990.

[4] J. S. Bridle, "Probabilistic interpretation of feed-forward classification network outputs, with relationships to statistical pattern recognition," in *Neurocomputing: Algorithms, Architectures and Applications*, F. Fougelman-Soulie and J. Herault, Eds. Berlin, Germany: Springer-Verlag, 1989, NATO ASI Series, vol. F68, pp. 227–236.

[5] C. L. Giles, G. M. Kuhn, and R. J. Williams, "Dynamic recurrent neural networks: Theory and applications," *IEEE Trans. Neural Networks*, vol. 5, pp. 153–156, Apr. 1994.

[6] H. Gish, "A probabilistic approach to the understanding and training of neural network classifiers," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, 1990, pp. 1361–1364.

[7] R. A. Jacobs, "Methods for combining experts' probability assessments," *Neural Comput.*, vol. 7, no. 5, pp. 867–888, 1995.

[8] B. A. Pearlmutter, "Learning state space trajectories in recurrent neural networks," *Neural Comput.*, vol. 1, pp. 263–269, 1989.

[9] M. D. Richard and R. P. Lippman, "Neural network classifiers estimate Bayesian *a posteriori* probabilities," *Neural Comput.*, vol. 3, no. 4, pp. 461–483, 1991.

[10] M. Riedmiller and H. Braun, "A direct adaptive method for faster backpropagation learning: The RPROP algorithm," in *Proc. IEEE Int. Conf. Neural Networks*, 1993, pp. 586–591.

[11] T. Robinson, "Several improvements to a recurrent error propagation network phone recognition system," Cambridge Univ. Eng. Dept. Tech. Rep. CUED/F-INFENG/TR82, Sept. 1991.

[12] A. J. Robinson, "An application of recurrent neural nets to phone probability estimation," *IEEE Trans. Neural Networks*, vol. 5, pp. 298–305, Apr. 1994.

[13] T. Robinson, M. Hochberg, and S. Renals, "The use of recurrent neural networks in continuous speech recognition," in *Automatic Speech Recognition: Advanced Topics*, C. H. Lee, F. K. Soong, and K. K. Paliwal, Eds. Boston, MA: Kluwer, 1996, pp. 233–258.

[14] ——, "Improved phone modeling with recurrent neural networks," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, vol. 1, 1994, pp. 37–40.

[15] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error backpropagation," in *Parallel Distributed Processing, vol. 1*, D. E. Rumelhart and J. L. McClelland, Eds. Cambridge, MA: MIT Press, 1986, pp. 318–362.

[16] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. J. Lang, "Phoneme recognition using time-delay neural networks," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 37, pp. 328–339, Mar. 1989.

[17] S. Young, "A review of large vocabulary speech recognition," *IEEE Signal Processing Mag.*, vol. 15, pp. 45–57, May 1996.

Mike Schuster received the M.Sc. degree in electronic engineering in 1993 from the Gerhard Mercator University, Duisburg, Germany. Currently, he is also working toward the Ph.D. degree at the Nara Institute of Technology, Nara, Japan.

After doing some research in fiber optics at the University of Tokyo, Tokyo, Japan, and some research in gesture recognition in Duisburg, he started at Advanced Telecommunication Research (ATR), Kyoto, Japan, to work on speech recognition. His research interests include neural networks and stochastic modeling in general, Bayesian approaches, information theory, and coding.

Kuldip K. Paliwal (M'89) is a Professor and Chair of Communication/Information Engineering at Griffith University, Brisbane, Australia. He has worked at a number organizations, including the Tata Institute of Fundamental Research, Bombay, India, the Norwegian Institute of Technology, Trondheim, Norway, the University of Keele, U.K., AT&T Bell Laboratories, Murray Hill, NJ, and Advanced Telecommunication Research (ATR) Laboratories, Kyoto, Japan. He has co-edited two books: *Speech Coding and Synthesis* (New York: Elsevier, 1995) and *Speech and Speaker Recognition: Advanced Topics* (Boston, MA: Kluwer, 1996). His current research interests include speech processing, image coding, and neural networks.

Dr. Paliwal received the 1995 IEEE Signal Processing Society Senior Award. He is an Associate Editor of the IEEE Transactions on Speech and Audio Processing.