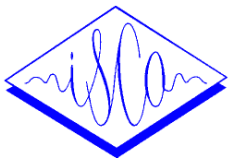

2nd European Conference on
Speech Communication and Technology
EUROSPEECH '91
Genova, Italy, September 24-26, 1991

LEXICAL ACCESS USING A RECURRENT ERROR PROPAGATION NETWORK

N.H. Russell, F. Fallside, A.J. Robinson, R.W. Prager
Cambridge University Engineering Department
Trumpington Street, Cambridge CB2 1PZ, UK

Abstract

The paper explores the use of a recurrent neural net to perform lexical access. It is trained to effect a mapping from the phonemic output of a neural net front end (reported in [5]), to lexical items. A cascaded multi-level approach is used which allows common phonetic variation in words to be captured by the front end. In order to keep the context duration manageable, non-linear time compression is performed on the input data to the lexical access network, such that transitional segments are retained, while steady state probability segments are much shortened, but duration information is not completely obscured.

A post-processor, based on a simple finite state automaton, is used to decode the raw NN output, which takes the form of a word pseudo-probability vector (and is synchronous with the input) to produce a symbolic (*i.e.* orthographic) interpretation of the input speech. Encouraging results using digit strings are presented.

1 The Problem of Lexical Access

This paper investigates the use of a recurrent network to perform lexical access. Lexical access involves searching for temporally contiguous word hypotheses at each position in an utterance, accounting for all possible durations of each word hypothesis. Classical methods such as dynamic programming [3] and HMM's compute a globally optimal solution for the identity of the unknown speech, and have explicit methods to account for the temporal ambiguity and non-linear time distortion associated with the location and time registration of tokens; the very idea of global optimality implies that utterance boundaries are determined by independent means. However, classical methods become computationally expensive (even intractable) as the recognition units become larger (*i.e.* words) and more numerous (larger vocabulary), and as syntax constraints are relaxed. Hence there is a combinatorial explosion associated with all explicit/symbolic search techniques which are globally optimal.

In contrast, neural net techniques employ a distributed representation, and in terms of the search paradigm, compute a locally optimal match and thus avoid some of the problems described above associated with symbolic, globally optimal search. However, NN historically have only provided a frame by frame phonetic classification, whereas lexical access requires symbol by symbol lexical classification. More recently, recurrent nets have been used for phoneme recognition [5], and give improved performance over static networks by virtue of their ability to model temporal context. Here their use will be extended to the lexical level, and it is shown how orthographic interpretation can be obtained in a straigthforward manner.

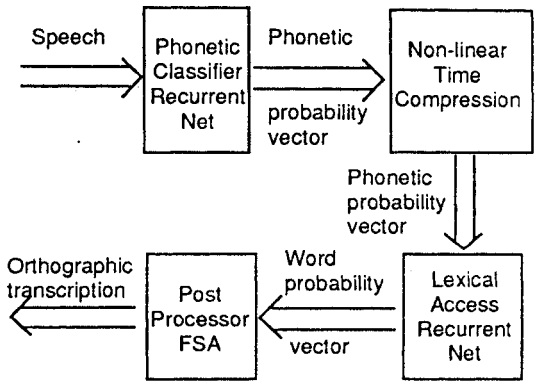

Figure 1: System Architecture

2 System Architecture

A cascaded two-level approach was adopted as shown in Figure 1. A recurrent net front end phonetically classifies the input waveform producing a sequence of phone probability vectors (one element per member of the symbol set). This is used as input to a second recurrent net which performs lexical access to produce a sequence of word probability vectors. A post-processor then produces orthographic output.

The most naive approach to using a (recurrent) neural net for word recognition would involve training a network to recognise words directly from a spectrally encoded version of the input waveform. In learning to recognise each word the network has to independently learn, in the context of each word, to recognise each of the constituent phonemes. This appears to be an inefficient use of the information present in the training data, and provides the motivation for the cascaded approach where the training data is not diluted over different contexts. The cascaded approach has the further advantage that much of the speaker variability present in the acoustic waveform will have been removed by the front end, and thus less training data should be required for the lexical access component. For example, assuming a vocabulary of 1000 words, and a phonetic symbol set size of 50, then a monolithic word recogniser would require approximately 20 times as much training data as its phonetic counterpart. The cascaded approach adopted here is in contrast to some other work in the area, *viz* the modified Kanerva model in [4].

A phonemic pseudo-probability vector is produced by the front end every 16 ms. This is too fine a granularity for the lexical access component which has to recognise units whose duration is typically 20–40 time slots on that scale; training becomes progressively more difficult as the duration of the context window is increased since the error signal is attenuated as it is back propagated through the net layers corresponding to each time slot. In order to keep the context duration manageable, time compression is performed on the input data. A vector is only accepted as input if it is sufficiently dissimilar from the last vector accepted, (*i.e.* the distance between the two vectors exceeds threshold D_t according to a Euclidean metric), or else one has not been accepted for a given number of time slots C, where C places an upper limit on the compression rate. Thus, transitional segments are retained, while steady state probability segments are much shortened, but duration information is not completely obscured.

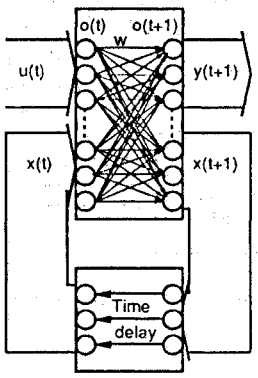

Figure 2: Recurrent Net with Internal State

3 Lexical Access Network

The output $\boldsymbol{Y}$ of a dynamic (or recurrent) net is determined both by its internal state $\boldsymbol{X}$ and the external input $\boldsymbol{U}$. The state $\boldsymbol{X}$ is itself of course a function of all past inputs by virtue of the recurrent connections, see Figure 2. The network is based on a single layer net with external inputs $U_{0\ldots L-1}$ (the phonemic probability vector), and external outputs $Y_{0\ldots M-1}$ (the word probability vector). The internal output (or state vector) $X_{0\ldots N-1}$ is fed through a unit time delay to the internal state input in the next time frame. All output units (Y_i, X_i) are connected to all input units (U_i, X_i) via adaptive link weights. The output of each unit is

$$o_i = f(net_i) \quad ; \quad net_i = \sum_j w_{ij} o_j$$

where net_i is the net input to node i, and f is the usual sigmoid function

$$F(x) = \frac{1}{1+e^{-x}}$$

The net is trained by a modified version of the back propagation algorithm [6]. The standard Euclidean error function E_p is used,

$$E = \sum_p E_p \quad ; \quad E_p = 1/2 \sum_i (t_{pi} - o_{pi})^2$$

The target values for the network represent pseudo-probabilities, and are set to 0.9 for the output denoting the current symbol in the reference transcription, and 0.1 for all other outputs. The error signal for the external outputs is calculated in the usual way as:

$$\delta_i^o = f'(net_i)(t_i - o_i) \tag{1}$$

where t_i is the target value (*i.e.* probability) for y_i. However the error signal for the state units at time t, $\delta_{t,i}^x$ is not known until time $t+1$ when it can be back-propagated through the recurrent connections of the basic network from its external output nodes according to the usual back-propagation recursion,

$$\delta_{t+1,i}^x = f'(net_{t,i}) \sum_k \delta_{t,k} w_{ki} \tag{2}$$

where k indexes over all output units in the succeeding network at time $t+1$, NET$(t+1)$. Thus in order to train the network to fully utilise context, one has to imagine it 'expanded' in time as a series of network instances shown in Figure 3.

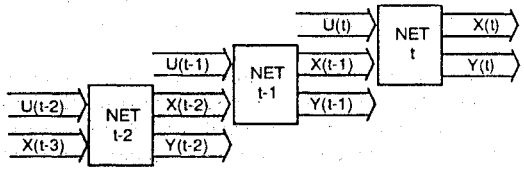

Figure 3: Expanded Dynamic Network (P=3)

For a formal presentation of the modified BP algorithm, the reader is referred to [5]. Here an informal mechanistic view of the training process will be given. For each time t an input vector $\boldsymbol{U}(t)$ is applied to the external inputs, and a target vector $\boldsymbol{T}(t)$ to the external outputs. The network output $\boldsymbol{Y}(t)$ and the state output $\boldsymbol{X}(t)$ are computed by a forward pass on the current instance NET(t) representing the state of the equivalent recurrent net at time t. Assuming the network has been running for time $t \geq P$, where P is the size of the context window, then a backward pass is made right to left over each instance of the network for times $t \ldots t-P+1$ as follows. For NET(t), the error signal is computed for the external outputs using equation (1), and the error signal for the state outputs is zeroed since it is unknown at time t. The error signal is back-propagated using equation (2) to the internal input nodes. For all prior instances NET$(q < t)$ the state unit error signal vector is copied from the state inputs of NET$(q+1)$ to the state outputs of NET(q). The error signal for the external outputs is computed as before. The error signals for *all* the output units are then back-propagated to the state input units. This is repeated for all prior instances. On completion of the backward pass the error signals are used to compute a local contribution to the gradient vector; these local contributions are then summed across the P net instances in the context window to give the total gradient contribution for the current time t. The context window is then advanced one frame to time $t+1$; the instance NET$(t-P)$ is now removed, and a new instance NET$(t+1)$ appended to NET(t) and the process is repeated. Note that weights and accumulated gradients are shared by all net instances.

In summary, the finite input duration architecture may be viewed as a feed forward net, where layers are connected

through the state units, where for each time frame the input layer is removed, and a new output layer is appended, and has the current input as its external input. The activation is computed for the new output layer but the activation (and target values) of all preceding layers is frozen. However, the error signal must be back-propagated afresh through the entire network, since the error signal vector for the state output X current instance NET(t) is zero, but becomes non-zero (in general) for time $t+1$ when it inherits an error signal vector from its successor. Thus by induction from the recursion defining back propagation, equation (2), all prior instances have a different error signal pattern for the new time $t+1$. In this manner the network learns to use contextual information over a window $U(t-P+1)\ldots U(t)$.

Weights are updated according to steepest descent using an adaptive learning rate and momentum term [1],

$$\Delta W(k) = \eta_k \nabla E + \lambda_0 |\nabla E| \Delta W(k-1)/|\Delta W(k-1)| \quad (3)$$

$$\eta_k = \eta_{k-1}(1+\cos(\theta)/2) \quad (4)$$

where $0 \leq \lambda_0 < 1$ and is constant. Training in epoch mode, where weights are updated once per epoch (or complete pass through the training data) is rather slow and impractical for large training sets. Faster training can be achieved using block update mode, where the data is divided into blocks, and weights are updated using the gradient estimated over each block. Two problems arise from block update mode. As a result of variation in the error surface estimates, the error measure $E(W)$ can vary by significantly more between successive blocks, than the mean reduction per block averaged over an epoch. Thus the binary backtracking procedure which can prevent overshoot instability in epoch mode (which is triggered if the fractional increase in the error exceeds some threshold) cannot be used in block mode.

Secondly, and perhaps more seriously, the direction of the gradient vector estimated over each block at a given point in weight space can vary. The learning rate is adapted according to equation (4). Thus if $\cos\theta$, the angle between the current gradient ∇E and the previous weight update vector ΔW has a negative average value over the epoch, then η_k will decay to zero, and learning will cease. This problem can be alleviated by computing a smoothed gradient vector ∇E^s by passing successive gradient estimates through a first order filter:

$$\nabla E^s(k) = S_c \nabla E^s(k-1) + (1-S_c)\nabla E(k)$$

where S_c is the smoothing coefficient. The smoothed gradient is used to compute $\cos\theta^s$ which replaces $\cos\theta$ in equation (4). Care must be taken in choosing S_c — too large a value will smooth out changes in ∇E resulting from genuine changes in the gradient direction between points on the true error surface and result in an exponentially increasing η_k.

4 Post-Processor

A post-processor, based on a simple finite state automaton, is used to decode the raw NN output, which takes the form of a word pseudo-probability vector (and is synchronous with the network input) to produce a symbolic (*i.e.* orthographic) interpretation of the input speech. There is one state in the FSA for each member of the output symbol set. Associated with each state j is a probability $P_j(t)$ that the current section of input corresponds to the associated symbol. A state transition causes a word boundary to be appended to the orthographic output, and occurs if the state with the maximum probability is different to that at the previous time.

In order that ripple in the raw NN outputs does not cause spurious transitions (and potentially chronic insertion errors) in the vicinity of word boundaries, two measures can be taken. Firstly the raw outputs can be smoothed by a first order filter with coefficient C_o,

$$P_j(t) = (1-C_o)o_j(t) + C_o P_j(t-1)$$

and secondly a transition probability σ can provide an hysteresis for transitions, where the new state $S(t)$ is given by,

$$S(t) = \underset{k}{\text{argmax}} \left(\begin{cases} P_k(t) & k = S(t-1) \\ \sigma P_k(t) & k \neq S(t-1) \end{cases} \right)$$

5 Experimental Method

For preliminary work, a small multiple speaker data base was used. It comprises a total of 400 utterances divided equally between four English speakers from different backgrounds. For each speaker the data was divided into 60 training utterances and 40 test utterances. The utterances were randomly generated digit strings of length five, spoken naturally by the speakers. All recordings were made under quiet conditions and digitised at 10 kHz to an accuracy of 12 bits. In the original database only the 40 test utterances had been labelled manually using a speech editor at the word level. A DP algorithm using templates excised from the test data was used to automatically label the training data at the word level using a forced recognition rule which permitted optional silences between words.

The experiments are divided into three parts. Firstly, a series of speaker dependent networks was trained with various values of P to investigate the effect of context window duration on performance. Secondly, for a particular value of P the effects of variation of σ, C_o on post-processor performance was studied to find good values. Finally, a multiple speaker network was trained and tested on all four speakers data.

The same front end [5] was used in all experiments. Since none of the data was phonetically labelled, a network pretrained on the American English TIMIT database [2] was retrained on the digit database by using the network to first segment the data, and using this boundary information to retrain the network. This procedure was repeated until convergence was achieved. Excluding silences, the frame classification rate was 85.4% on the training data and 75.5% on the test data with time compression enabled; without compression the rates were 92.2% and 84.6% respectively.

The lexical access NN used in the experiments had 61 input nodes, 16 or 32 state units, and 11 output nodes. Values for the phonetic temporal compression C, D_t were chosen manually by inspecting the compressed output sequence. The values $C = 4, D_t = 0.08$ were used in all experiments. Time precluded a formal experiment to determine optimal values by monitoring network performance. The values chosen gave a mean temporal compression of 2.9 over the data. The lexical access networks were trained in block mode with gradient smoothing factor $S_c = 0.3$.

6 Results

The effect of variation of the post-processor parameters for a network with $P = 4$ is shown in Table 1. All recognition rates (symbolic and frame by frame) exclude silence segments. With smoothing off ($C_o = 0$), accuracy gradually increases with decreasing σ to a maximum around $\sigma = 0.1$, and suddenly falls for $\sigma = 0.05$. For $\sigma = 1$, varying the degree of smoothing C_o from zero results in a steady increase in accuracy (and decrease in number of correct symbols) up to $C_o = 0.9$; a further increase to 0.95 results in a decrease in the accuracy. This trend is a result of increasing deletion errors but more rapidly decreasing insertion errors with increasing C_o. Comparing these two factors in isolation, it seems smoothing is a more effective and reliable means to controlling insertion errors. Some further values around the optimal region are also shown — by inspection good values are $C_o = 0.9$, $\sigma = 0.8$.

σ	C_o 0.0	0.5	0.7	0.8	0.9	0.95
1	95.0	90.0	87.5	87.5	86.5	80.5
	47.0	60.5	74.0.	82.0	85.5	78.5
0.8	93.0	–	–	87.0	86.0	–
	56.0			82.5	85.5	
0.7	–	–	–	86.5	85.0	–
				83.5	84.5	
0.5	91.5	–	–	–	–	–
	62.5					
0.25	89.0	–	–	–	–	–
	70.5					
0.1	78.5	–	–	–	4.5	–
	74.0				4.5	
0.05	27.5	–	–	–	–	–
	25.5					

Table 1: Variation of Post-processor parameters for network context duration $P = 4$ (%correct upper, %accuracy lower).

P, N_x	N	Cor%	Acc%	S	I	D	Ftr%	Ftst%
1,16	200	84.0	79.5	1	9	31	61.4	63.2
2,16	200	85.0	84.5	1	1	29	83.4	87.9
4,16	200	86.0	85.5	2	1	26	84.0	88.2
6,16	200	86.5	85.5	1	2	26	83.8	88.3
8,16	200	86.5	85.5	1	2	26	83.3	87.4
8,32	200	85.5	84.5	1	2	28	63.3	64.8
8,16	800	83.1	81.9	19	10	116	84.6	82.4

Table 2: Recognition results for variation of P. Last row is multiple speaker test, remaining rows are speaker dependent test. Correct = H/N, Accuracy = $(H - I)/N$, and the numbers of symbols are H = correct, N= total, S=substitutions, I=insertions, D=deletions. Ftr = frame rate for training, Ftst = frame rate for test data.

Inspection of Table 2 shows that for the speaker dependent tests the NN frame classification rate increases significantly from $P = 1$ (no context) to $P = 2$, but remains almost constant for $P = 4 \ldots 8$. Surprisingly performance is better on the test data than on the training data. This suggests that possibly the training labels (DP aligned) were not aligned sufficiently accurately for context to be fully utilised for larger window durations $P > 4$. For $P = 8$, increasing the number of state units N_x from 16 to 32 leads to a marked degradation, suggesting that the network is trapped in a local minimum. The multiple speaker test (last row) shows slightly inferior performance to the single speaker test ($P = 8$), with higher substitution and deletion rates.

Overall, substitution errors are rare, and the main source of error is symbol deletion. On inspecting the orthographic output it became clear that the main source of deletions is from duplicated symbols; *e.g.* the string 06447 would probably be recognised as 0647. For random digit strings of length 5, 8% of symbols are preceded by the same symbol and thus susceptible to deletion.

7 Conclusion

The results for the baseline system demonstrate that recurrent networks are potentially suited to lexical access, assuming a cascaded architecture, with non-linear temporal compression used at the interface of the two networks. It has been shown how the raw NN output can be decoded in a straightforward manner without recourse to relatively expensive DP algorithms. The main source of error is deletion of symbols which are duplicated in the unknown speech, which it should be possible to overcome with the adoption of a basic durational model.

8 Acknowledgements

The research reported here was carried out as part of the UK SERC funded LEARN project. The authors are grateful to Steve Austin for the construction of the speech database.

References

[1] L. W. Chan and F. Fallside. An adaptive training algorithm for back propagation networks. *Computer Speech and Language*, 2(3/4):205–218, 1987.

[2] J. S. Garofolo. *Getting Started with the DARPA TIMIT CD-ROM: An Acoustic Phonetic Continuous Speech Database*. National Institute of Standards and Technology (NIST), Gaithersburgh, MD, 1988.

[3] H. Ney. The use of a one-stage dynamic programming algorithm for connected word recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 32(2):263–271, Apr. 1984.

[4] R. W. Prager and F. Fallside. The modified Kanerva model for automatic speech recognition. *Computer Speech and Language*, 3:61–81, 1988.

[5] T. Robinson and F. Fallside. A recurrent error propagation network speech recognition system. *To appear in Computer Speech and Language*, 1991.

[6] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Vol. I: Foundations.*, chapter 8. Bradford Books/MIT Press, Cambridge, MA, 1986.