Stability of Learning in Classes of Recurrent and Feedforward Networks

William H. Wilson
billw@cse.unsw.edu.au

School of Computer Science and Engineering
University of New South Wales
Sydney 2052 Australia

Abstract

This paper concerns a class of recurrent neural networks related to Elman networks (simple recurrent networks) and Jordan networks and a class of feedforward networks architecturally similar to Waibel's TDNNs. The recurrent nets used herein, unlike standard Elman/Jordan networks, may have more than one state vector. It is known that such multi-state Elman networks have better learning performance on certain tasks than standard Elman networks of similar weight complexity. The task used involves learning the graphotactic structure of a sample of about 400 English words. Learning performance was tested using regimes in which the state vectors are, or are not, zeroed between words: the former results in larger minimum total error, but without the large oscillations in total error observed when the state vectors are not periodically zeroed. Learning performance comparisons of the three classes of network favour the feedforward nets.

1 Introduction

Jordan [3] studied a class of recurrent networks, sometimes called sequential or Jordan nets, which use a *state vector* which contains copies of the output layer activations in the previous time step; there are weighted connections from the state vector to the hidden layer. Elman [1,2], and Servan-Shreiber et al. [9] worked with an analogous class of nets, now termed simple recurrent nets, or Elman nets, which differed from Jordan nets in that the state vector is a copy of the *hidden* layer in the previous time step, as illustrated in Figure 1, left side. Such networks can learn sequential structures.

Elman used his network architecture, along with the backpropagation learning algorithm [6], to learn the grammatical structure of a set of sentences randomly generated from a limited vocabulary and grammar. His specific task was to predict the next word in the sentence from the current word and the representation of past words held in the state vector. Further details are in [1,2] and a summary from the point of view of the current work is in [13].

Waibel [10], and Lang and Waibel [4], devised non-recurrent time-delay neural networks (TDNNs) which learnt speech recognition tasks by considering past as well as present inputs.

The relationship to work by Mozer, [5], on induction of temporal structure, is briefly described in [13].

In the research reported here, the task is similar to Elman's: predicting the next letter in a word (or the end of the word) from the current letter and the representation of past letters held in the state vector. While the original motivation for this task was linguistic [12], the current paper focuses on the efficacy of a range of network architectures, and learning regimes, applied to the task.

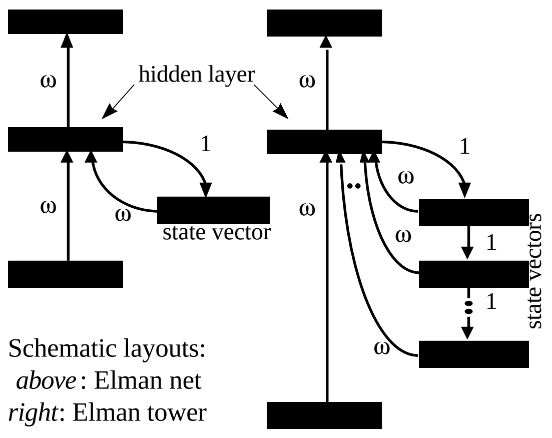

Figure 1: Architectures of Elman net and Elman tower network: ω signifies that layers are totally interconnected with trainable weights; 1 signifies that destination layer activations are direct copies of source layer activations *in the previous processing cycle*. The Elman tower can have two or more state vectors.

The state vector in Elman's networks provides the potential for such networks to store information about previous inputs.

If one state vector is useful for suitable tasks, it is possible that two or more state vectors (as illustrated in Figure 1, right side) will let a network perform even better at sequential tasks. We term such networks Elman tower networks. This possibility was confirmed in [13], where it was shown that among networks with (about) the same number of weights, but different numbers of state vectors, the networks with more state vectors learned faster and found lower weight configurations with lower total error. For example, a standard Elman network with 24 hidden units (1923 weights) found weight values which gave total error of around 920 units, whereas an Elman tower with 13 hidden units and 7 state vectors (1925 weights) found weight values which gave total error of around 815 units.

However, the learning performance reported in [13], was erratic, as illustrated in Figure 2.

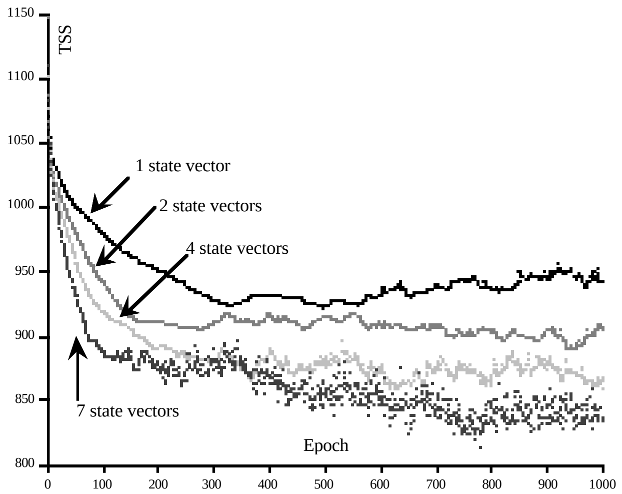

Figure 2: Typical Error Plots for a range of Elman, and Elman tower architectures, from [13].

This paper describes three types of advance on the work reported in [13]: (i) a learning regime which largely removes the erratic learning performance; (ii) results with *Jordan* networks and towers, and with TDNNs - see Figure 3; and (iii) results with larger numbers of weights.

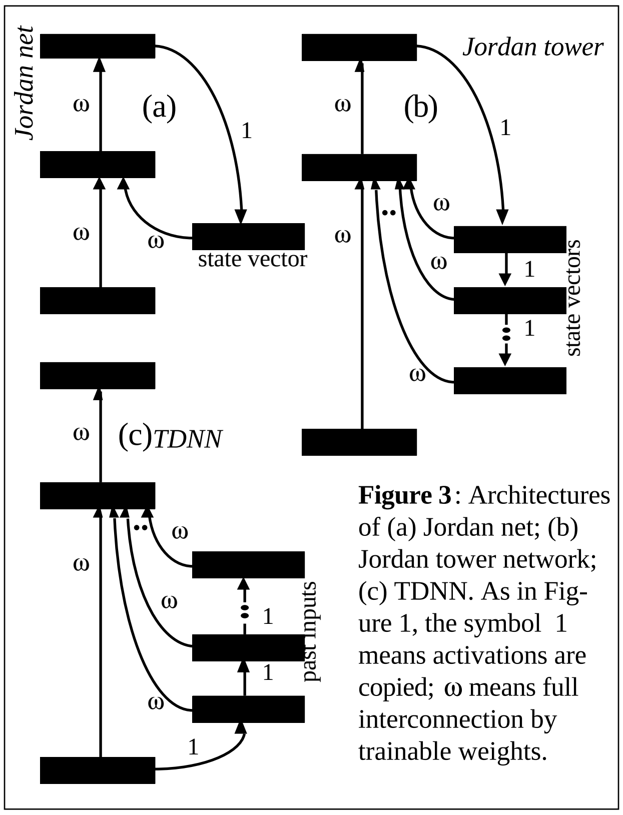

Figure 3: Architectures of (a) Jordan net; (b) Jordan tower network; (c) TDNN. As in Figure 1, the symbol 1 means activations are copied; ω means full interconnection by trainable weights.

Section 2 describes the design of networks with different numbers of state vectors, but otherwise similar computational power, and outlines simulation experiments done with such networks, and with the new training regimes. Section 3 presents the results of these simulations.

The results indicate that, for this task, the new training regimes do provide more *stable* learning, but at the cost of greater *error*.

2 Experiments with Tower-Recurrent Networks

Let us refer to Elman tower networks and Jordan tower networks together as *tower-recurrent networks*. We use the term *time-delay neural net* (TDNNs) to refer to feedforward nets whose architecture is shown in Figure 3(c). (The TDNNs of Waibel et al. [10,4] sometimes include an extra hidden layer, and are trained by a backpropagation-through-time type algorithm ([11]).) Such nets are analogous to Elman and Jordan nets and towers, with a “feedback” connection from the *input* layer to the “state vectors,” which are thus better termed *past inputs*. (These nets seem to remind many people of the NETtalk architecture [8]. In NETtalk, however, the output was a phoneme to pronounce, and the inputs were a window of letters *on either side* of the current letter.)

The aim of the experiments was to compare networks differing in numbers of state vectors, Jordan-type vs Elman-type vs TDNNs, and training regime, while holding network complexity otherwise constant. Many architectural parameters were used as factors in the experiments, but it was still possible to hold constant the number of weights, one factor in network learning capacity. Thus the nets used were chosen to have equal numbers of weights, as calculated below.

The number of weights and biases in an Elman tower net with *n* inputs and outputs, *h* hidden units and *s* state vectors was computed in [13] to be

$$w_{Elman}(n, h, s) = 2nh + h + n + sh^2.$$

For Jordan towers and TDNNs, the formulae are:

$$w_{Jordan}(n, h, s) = 2nh + h + n + shn$$
$$w_{TDNN}(n, h, s) = w_{Jordan}(n, h, s).$$

So, in comparing a 2-state Elman tower with a 2-state Jordan tower, for example, we sought h_1, h_2 such that $w_{Elman}(n,h_1,2) \approx w_{Jordan}(n,h_2,2)$. In our task, n=27. There turned out to be many such nets with about 5859 weights, and these were used in the experiments: see Table 1. For instance, Elman and Jordan towers and TDNNs with 6 state vectors and 27 hidden units all have 5859 weights.

Architecture type	*Name*	*State Vectors*	*Hidden Units*
Elman	ES1H54	1	54
Elman	ES3H36	3	36
Elman	ES6H27	6	27
Jordan	JS1H72	1	72
Jordan	JS4H36	4	36
Jordan	JS6H27	6	27
TDNN	TS2H54	2	54
TDNN	TS6H27	6	27

Table 1: Some nets with 27 inputs and outputs, and 5859 weights

Each architecture considered was simulated for 10 runs of the PDP package's `bp` program ([7]) over 1000 epochs, with `lrate=0.05`, from random starting states. The task used was that of predicting the next letter in an English word given an initial string of letters in the word [13].

Two training regimes were used (cf. [7]): in training a net to predict next letters in, say, the English word *cat*, one presents the pattern for c as input and the pattern for a as output ($c \to a$), next, ($a \to t$) and finally ($t \to$ <*end-of-word*>). During this process, activations are accumulating in the state vectors. After training on *cat* one can either zero the state vectors or leave them as they are. There are arguments for both policies: briefly, the left-over activations are irrelevant to next-letter prediction in the next word, but it is hard to postulate a biological process which inactivates selected neurons which happen to have recurrent connections to them. In [13], state vectors were not zeroed at the end of words: in some of the simulations reported here, they were.

In each simulation, TSS (Total Error Sum of Squares) starts at around 1100, descends rapidly for the first 100-300 epochs and then levels out. With the task chosen, there is no hope of TSS approaching zero, as input patterns (letters) early in a word can only be used to predict the next pattern/letter in a probabilistic way.

However, we would like the altitude of the plateau to be as low as possible, as this means that, on average, next-letter prediction is as good as possible. We would also like the learning to be fast, and stable - TSS should not oscillate as it does in Figure 2.

3 Simulation Results and Interpretation

We have a number of comparisons to make: training regimes, effect of more states, Elman vs Jordan vs TDNN, and effect of more weights. The primary comparison of a pair of simulations must be in terms of the (best) error minimum found, but it is also interesting to consider speeds of learning. A measure of speed of learning is the initial slope of the error *vs* epoch curve. Indeed, a large part of the learning has occurred after the first 150 or so epochs.[1] Space constraints preclude detailed consideration of learning speed, but interested readers can glance at the slopes of the curves in Figures 2 and 6.

Figure 4 shows plots of the TSS against epoch number for one (typical) run of each training regime, for a 6-state Elman tower. In [13] and Figure 2, greatest oscillation occurred with large numbers of states, hence the choice of 6-state-vector architectures: similar effects, perhaps less pronounced, occur for fewer state vectors and for Jordan towers. It can be seen that zeroing the state vectors at the end of each word essentially removes the oscillation effect, but at the cost of slower and less effective learning. The minimum error averaged over 10 runs with zeroed state vectors was 702, whereas, without zeroed state vectors, the average minimum was 549 (153 less).

1 the measure of learning speed used in [12], namely epoch number of the first local minimum, is of course inappropriate with smoothly descending error curves.

Figure 5 shows the effect of the number of state vectors on the minimum error found in the first 1000 epochs using zeroed state vectors for Elman towers, and using zeroed past inputs for TDNNs. Compare Figure 7.

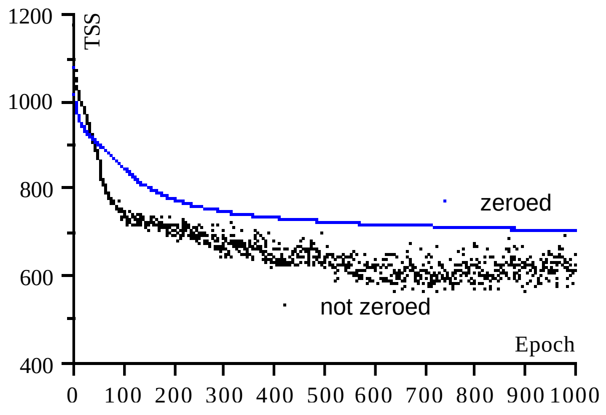

Figure 4: Error plot of a run of each training regime, for a 6-state Elman tower (27 hidden units).

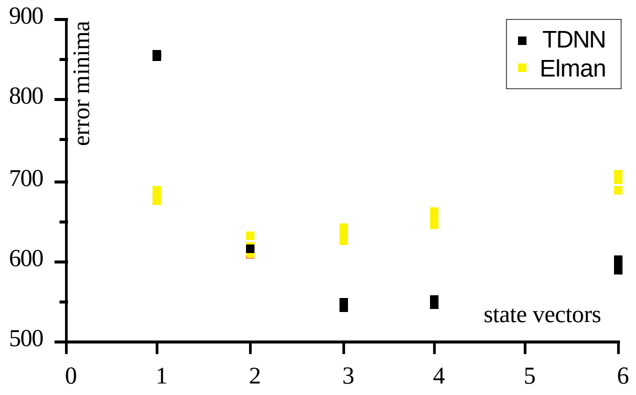

Figure 5: Error minima for Elman net and towers, and for TDNNs, with different numbers of state vectors, but similar weight counts. State vectors were zeroed between words. Similar effects occur for Jordan nets and towers.

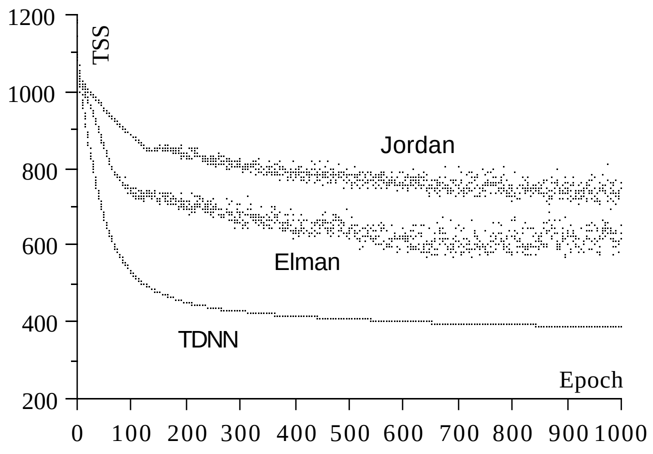

Figure 6: Error plots of one run of each architecture type, with 6 state vectors/past inputs, and non-zeroed state vectors.

It can be seen that for non-zeroed state vectors, the error steadily decreases as more state vectors are added (even

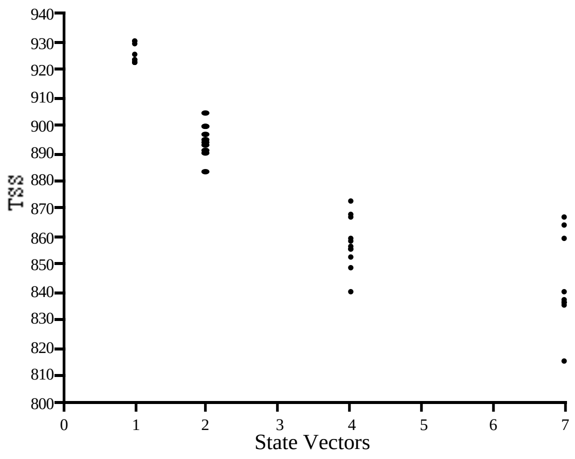

Figure 7: Global Minima for 10 runs of a range of Elman, and Elman tower architectures, from [13]. State vectors were not zeroed between words.

though the number of weights is held constant), while for zeroed state vectors/past inputs, the absolute minimum error occurs for 2 or 3 states/past inputs.

Figure 5 shows that TDNNs found somewhat better minima than Elman towers (in fact, they also did better than Jordan towers).

Figures 5 and 7 also permit the observation that while the not-zeroed nets produced minima within a fairly wide range (and the spread increases with the number of state vectors), the zeroed nets seem to have a fairly tight distribution of minima for a given architecture, and not much change in the spread with the number of state vectors/past inputs.

Figure 6 shows illustrates that for 6 non-zeroed state vectors or past inputs, a TDNN clearly outperforms an Elman tower which in turn outperforms a Jordan tower. Figure 8 shows the comparable situation with zeroed state vectors: there is still a little instability for Jordan towers, but not for Elman towers. However, the learning is poorer than with non-zeroed state vectors/past inputs.

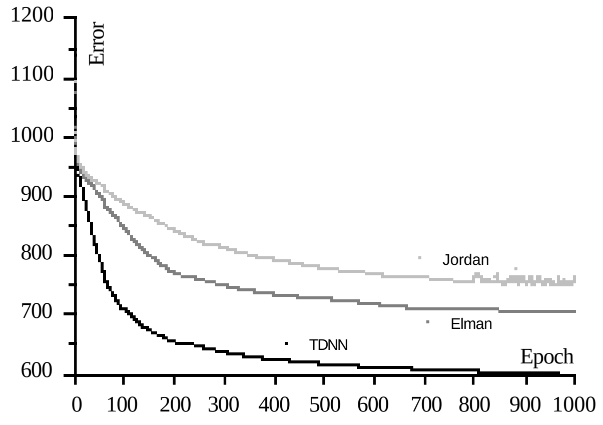

Figure 8: error plots of one run of each architecture type, with 6 state vectors/past inputs, zeroed state vectors.

5 Conclusions

The results obtained might be different with a different task. However, on this graphotactic prediction task, in which only the previous few letters may be useful predictors, it seems clear that the TDNN does best, then Elman towers. If state-vector activations / past inputs are zeroed at the end of each word, then only two or three state vectors / past inputs are helpful. If they are not zeroed, then the more state vectors / past inputs the better. Zeroing state vectors / past inputs increases error but increases stability of learning. We emphasize that our TDNNs are trained by a different method to that of Waibel et al. [10,4] (see Section 2).

References

[1] Elman, Jeffrey L., Representation and structure in connectionist models, *TRL Technical Report 8903*, Centre for Research in Language, Univ. of California, San Diego, La Jolla, CA 92093 (1989) 26 pages.

[2] Elman, Jeffrey L., Finding structure in time, *Cognitive Science* **14** (1990) 179-211.

[3] Jordan, M.I. (1986) Attractor dynamics and parallelism in a connectionist sequential machine, *Proceedings of the Eighth Annual Meeting of the Cognitive Science Society*, Hillsdale, NJ: Erlbaum.

[4] Lang, K.J., Waibel, A.H., and Hinton, G.E., A time-delay neural network architecture for isolated word recognition, *Neural Networks* **3** (1990) 23-43.

[5] Mozer, Michael C., Induction of multiscale temporal structure, in *Advances in Neural Information Processing Systems* **4**, J.E. Moody, S.J. Hanson, and R.P. Lippmann, (eds), Morgan Kaufmann, 1992.

[6] Rumelhart, David E., Hinton, G.E. and Williams, R.J., Learning internal representation by error propagation, pages 318-362 in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Vol. 1: Foundations*, edited by D.E. Rumelhart and J.L. McClelland, Cambridge, MA: MIT Press, 1986.

[7] Rumelhart, D.E.& McClelland, J.L *Explorations in Parallel Distributed Processing*, book with software, Cambridge, MA: MIT Press, 1989.

[8] Sejnowski, T.J. and Rosenberg, C.R., Parallel networks that learn to pronounce English text, *Complex Systems* **1** (1987) 145-168.

[9] Servan-Shreiber, D., Cleeremans, A. & McClelland, J.L., Learning sequential structure in simple recurrent networks, in *Advances in Neural Information Processing 1*, edited by D.S. Touretzky, San Mateo, CA: Morgan Kaufmann, 1989

[10] Waibel, Alex, Modular construction of time-delay neural networks for speech recognition, *Neural Computation* **1** (1989) 39-46.

[11] Werbos, P.J., Backpropagation through time: What it is and how to do it, IEEE Proceedings **78** (1990) 1550-1560.

[12] Wilson, William H., Dealing with unknown words: classifying unknown letter-strings using trigram analysis, *Australian Computer Science Communications* **14**(1) (1992) 981-988.

[13] Wilson, William H., A Comparison of Architectural Alternatives for Recurrent Networks, *Proceedings of the Fourth Australian Conference on Neural Networks (ACNN'93)* (1993) 189-192.