*Proceedings of the Fourth Australian Conference on Neural Networks (ACNN'93)* 189-192

# A Comparison of Architectural Alternatives for Recurrent Networks

William H. Wilson

School of Computer Science and Engineering
University of New South Wales
Sydney NSW 2052 Australia
E-mail: `billw@cse.unsw.edu.au`

## Abstract

*This paper describes a class of recurrent neural networks related to Elman networks. The networks used herein differ from standard Elman networks in that they may have more than one state vector. Such networks have an explicit representation of the hidden unit activations from several steps back. In principle, a single-state-vector network is capable of learning any sequential task that a multi-state-vector network can learn. This paper describes experiments which show that, in practice, and for the learning task used, a multi-state-vector network can learn a task faster and better than a single-state-vector network. The task used involved learning the graphotactic structure of a sample of about 400 English words.*

## 1 Introduction

Elman [1,2] introduced a particular class of recurrent network in which the feedback connections are from the state vector to the hidden layer, as illustrated in Figure 1. Elman used this neural network architecture, along with the backpropagation learning algorithm [4], to learn the grammatical structure of a set of sentences randomly generated from a limited vocabulary and grammar. A major point of Elman's work was to study the hidden unit activation patterns in a trained network, produced in response to a sequence of inputs, and to use techniques such as cluster analysis to infer a structure for the data as represented in the hidden unit activation patterns. He was able to extract a cluster hierarchy corresponding to the syntactic rules from which the data had been constructed: nouns, verbs, animate and inanimate nouns, transitive and intransitive verbs, etc. This information was only implicitly present in the data presented to the network: in other words, the network had learned the structure of the linguistic data from the examples presented to it.

A significant aspect of the training data set used by Elman is that it included no negative examples. Backpropagation nets are frequently viewed as learning a classification mapping: $\mu$: X $\rightarrow$ T where X=$\{x_1,x_2,...\}$, and T=$\{t_1,t_2,...\}$. In such cases, the input training data consist of instances of each of the types $t_j$ to which an input $x_i$ can be classified. If a network is partially-trained on a set

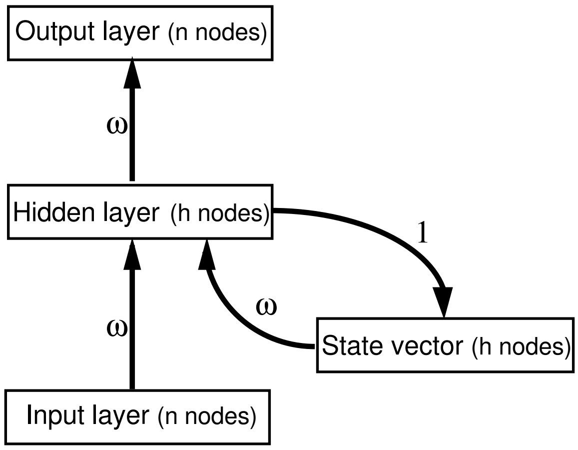


Figure 1: Architecture of Elman's recurrent network; ω signifies total interconnection with trainable weights; 1 signifies that the activations at the destination are a copy of the activations at the source *in the previous processing cycle.*

of inputs which includes no instances of type $t_k$, say, then the partially-trained network might map such an input to a random type. In particular, for a binary classification network, (i.e. T = {+,–}) a standard backpropagation network must be trained on inputs of type + and of type –.

There are two reasons why this model of learning - training on examples and non-examples - is inappropriate to learning *syntax*, as in Elman's task, or *graphotactics*, the task used in this paper:

- syntax and graphotactics[1]/phonotactics are learnt by *humans* essentially on the basis of examples, rather than examples and non-examples.
- words and sentences are of variable length and are perceived sequentially, or at least, the sequence of words or letters can affect the meaning; it is not practical or linguistically plausible to have an input pattern representing each possible sentence or each possible word. Instead, for example, a word might be represented as a sequence of input patterns: *cat* could be represented as a sequence of bit patterns corresponding to the letters *c*, *a* and *t*.

The state vector in Elman's networks provides the potential for such networks to store information aboutprevious inputs. An ordinary backpropagation network without some form of feedback loop would be unable to perform tasks which require it to know what the previous input was: such networks could not recognize a tune or any other temporal structure.

If one state vector is useful for suitable tasks, is it possible that two or more state vectors (as illustrated in Figures 2 & 3) will let a network perform even better at sequential tasks? This is the focus of the research reported here. We compare networks with 1, 2, 4, and 7 state vectors. The 4- and 7- state-vector networks are similar to 2-state vector networks, but with more state vectors.

1 graphotactics studies which letters can be adjacent in the words of a language: e.g., English words cannot begin with the letter-sequence *xp* or contain the sequence *fff*.

Of course, it would seem unsurprising if the addition of an extra state-vector to a basic recurrent network like the one shown in Figure 1 should increase its performance. After all, with the extra state-vector, we add weighted connections back to the hidden layer, and thus increase the learning potential of the system. In making comparisons, we have to allow for such effects.

Section 2 describes the design of recurrent networks with different numbers of state vectors, but otherwise similar computational power, and outlines simulation experiments done with such networks. Section 3 reports and analyses the results of these simulations. The results indicate that networks with more state vectors do indeed perform better, although network learning behaviour became erratic with the largest number of state vectors tried (7). Section 4 gives some pointers to other approaches to elaboration and analysis of the performance of recurrent networks.

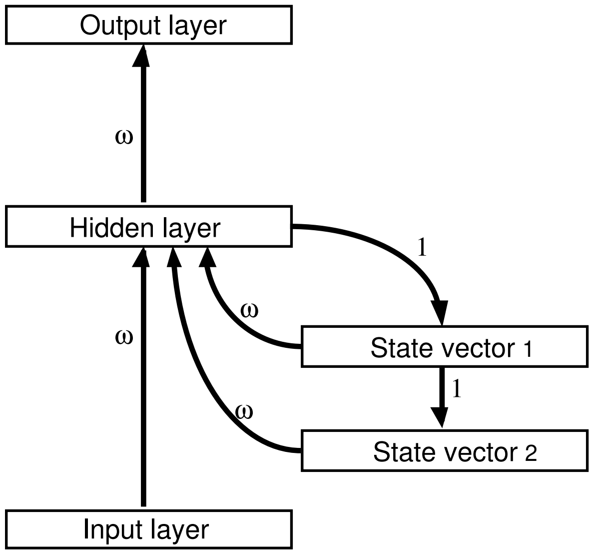


Figure 2: Architecture of Elman-style recurrent network with 2 state vectors

## 2 Experiments with Multi-State Recurrent Networks

The aim of the experiments was to compare recurrent networks having different numbers of state vectors while holding as many other factors as possible constant. The most likely repository of computational information in a neural network is the set of weights. Thus particular attention was paid to equalizing the numbers of weights between the examples of the different architectures being tested. It might be suggested that hidden units, rather than weights, are a critical resource in a network. This hypothesis was not tested directly in the experiments, but it did turn out that network performance was best for networks with less hidden units and more state vectors.

The number of weights and biases in a recurrent network with $n$ outputs, $h$ hidden units and $s$ state vectors is:

$$\begin{aligned} nweights &= \text{weights(inputs}\rightarrow\text{hidden layer)} \\ &\quad + \text{weights(hidden layer}\rightarrow\text{outputs)} \\ &\quad + \text{biases(hidden layer)} + \text{biases(outputs)} \\ &\quad + s\times\text{weights(one state vector}\rightarrow\text{hidden layer)} \\ &= nh + h + hn + n + sh^2 \end{aligned}$$

In these experiments, $n = 27$ (26 letters in the alphabet + 1 symbol for end of a word), so the formula for the weights is $w(s,h) = 54h + 27 + h + sh^2$. What we need is a number of pairs of values of $h$ and $s$ for which $w(s,h)$ evaluates to the same, or nearly the same, number. It turns out that with ($s$=1, $h$=24), ($s$=2, $h$=20), ($s$=4, $h$=16), and ($s$=7, $h$=13), $w(s,h)$ evaluates to numbers in the relatively narrow range from 1923 to 1931 weights. Details are shown in Table 1. Figure 3 shows a recurrent net with 4 state vectors; a recurrent network with 7 state vectors is analogous.

| *Name* | *State Vectors* | *Hidden Units* | *Weights + Biases* |
|---|---|---|---|
| S1H24 | 1 | 24 | 1923 |
| S2H20 | 2 | 20 | 1927 |
| S4H16 | 4 | 16 | 1931 |
| S7H13 | 7 | 13 | 1925 |

Table 1: Parameters of the Four Architectural Variants

These four architectures were simulated for 10 runs (2 runs on each of 5 different Apollo workstations) each using random starting states. The task used was the graphotactic task of predicting the next letter in an English word given an initial string of letters in the word. This task is related to the problem discussed by Wilson [5]. Its motivation comes from the problem of classifying strings of letters which are not present in an on-line lexicon as either (a) likely typographical/spelling errors, or (b) valid words which happen to be absent from the lexicon - valid words are likely to be graphotactically acceptable, whereas many errors will not be. Note that this paper does not set out to report success or failure in applying recurrent networks to this task, but rather to report the relative performance of different recurrent architectures on this task.

The networks were trained on the same set of 373 3-to-7-letter English words which gave rise to 1880 input patterns. For a word like *bush*, the net gets four patterns, corresponding to $b \rightarrow u$ (if the input is $b$, then the output should be $u$), $u \rightarrow s$, $s \rightarrow h$, and $h \rightarrow$*end-of-word.*

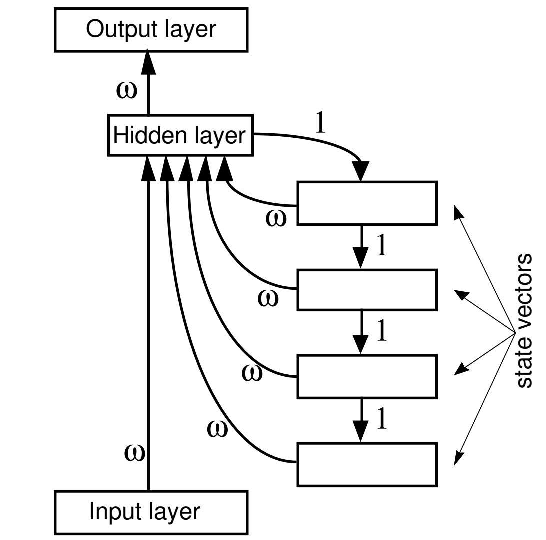


Figure 3: Recurrent network with 4 state vectors; note that the connections from the state vectors to the hidden layer feedback hidden layer activations from the preceding four processing cycles

In each case, TSS (Total Sum of Squares) starts at around 1150, descends rapidly for the first 100-300 epochs and then levels out to a high plateau. 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.

For example, if the letter *f* is presented as the first letter of a word, then the net could learn to predict that the next letter might be *a*, *e*, *i*, *l*, *o*, *r*, *u*, (or with low probability *j*, as in *fjord*), but no other. Absolute correctness is not possible. Late in a word, certainty is sometimes possible (particularly with the limited data set of 373 words). Thus, if these nets have seen the sequence *s h i f t*, then they could safely predict *y*, since *shifty* happens to be the only word in the training data that begins with *s h i f t*. Given *s h i f t y*, they could safely predict *end-of-word*.

So TSS descends to a plateau, and we would like the altitude of the plateau to be as low as possible, as this means, in a general way, that next-letter prediction is as good as possible.

## 3 The Experimental Results and Their Interpretation

Figure 4 shows typical plots of the TSS (Total Sum of Squares of errors) against epoch number for one (typical) run of each of the four architectures. In a general way, we can see that for these particular runs, the architecture performances can be ranked as S1H24 < S2H20 < S4H16 < S7H13. (However, a more systematic comparison is desirable: see below). Figure 4 also illustrates the fact that for all four architectures, TSS descends from an initial high value and then "wanders" on a plateau. For the S1H24 and S2H20 architectures, the oscillations of the plateau are small from one epoch to the next. For S4H16, and particularly for S7H13, the oscillations on the plateau are less stable.The architectures could be compared in various ways, including:

(a) the slope of the initial descent (measured as the epoch number of the first local minimum);
(b) the TSS-value at the global minimum over the 1000 epochs of the simulations.

Measure (b) is of particular interest because it indicates best possible performance of this class of network (over the period of the simulation). Measure (a) is of interest because it is an indicator of learning *speed*.

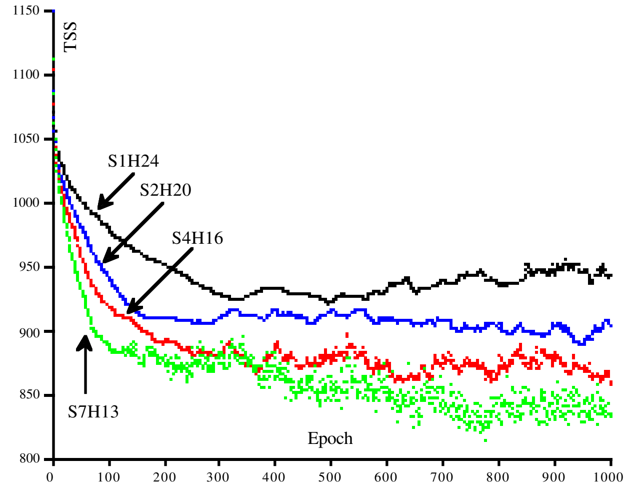


Figure 4: Typical TSS Plots for the 4 Architectures

The mean epoch numbers of the first local minimum over each of the four architectures were calculated (see Table 2) and all pairs of means were found to be significantly different. That is, our subjective judgment, from Figures 4 and 5, that the initial descents of the curves are at different rates, is borne out by statistical analysis of the results of all 40 runs. There is overlap of performance, however, particularly between the 2- and 4- state vector architectures.

| | S1H24 | S2H20 | S4H16 | S7H13 |
|---|---|---|---|---|
| Epoch Numbers | 260 | 150 | 104 | 75 |
| | 274 | 171 | 129 | 83 |
| | 277 | 184 | 131 | 89 |
| | 299 | 185 | 140 | 91 |
| | 314 | 190 | 153 | 99 |
| | 319 | 209 | 160 | 100 |
| | 330 | 230 | 176 | 107 |
| | 339 | 230 | 181 | 110 |
| | 340 | 235 | 199 | 112 |
| | 379 | 239 | 203 | 113 |
| Mean | 313.1 | 203.1 | 157.6 | 97.9 |

Table 2: Epoch Number of First Local Minimum

The global minima (over 1000 epochs) are shown in Figure 6. The analysis of the global minima for the 40 runs is complicated by the fact that the minima sets for the four architectures are heteroscedastic (do not have the same variances), so that non-parametric analysis is necessary. For this reason, the global minima were ranked from 1 to 40 (40 means largest global minimum). Table 3 shows ranked global minima tabulated by architecture.

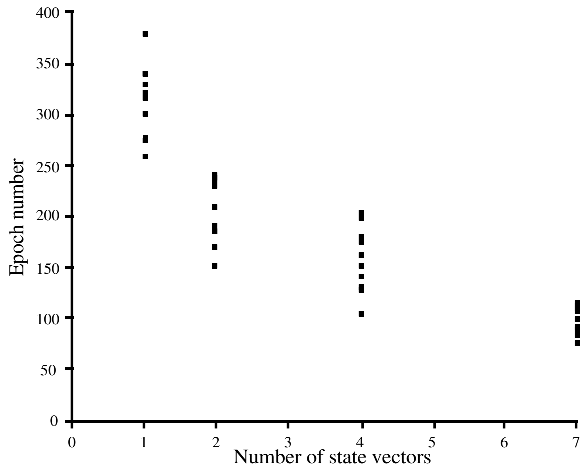


Figure 5: First Local Minima with Different Architectures

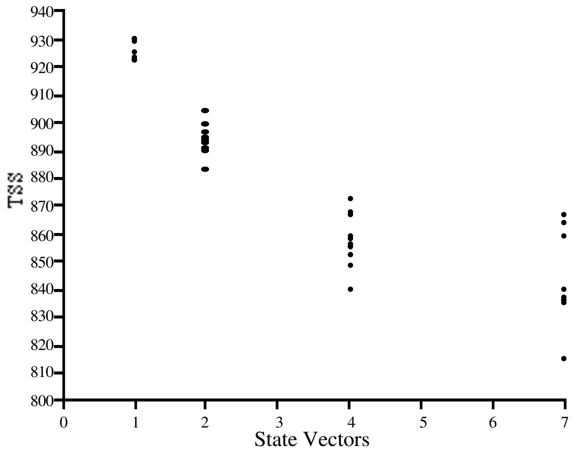


Figure 6: Global Minima with Different Architectures

All S1H24 global minima are larger than any S2H20 global minimum, which are in turn larger than the global minima for S4H16 and S7H13. The latter two architectures have some "crossed-over" runs, but statistical analysis showed that, overall, S7H13 performs significantly better than S4H16, when the pairs of means for these architectures are compared. Statistical analysis also confirmed the differences between all the other pairs of architectures.

## 4 Related Work

Another approach to improving the performance of recurrent networks is the work of Mozer (1992) on induction of temporal structure across longer temporal intervals by using hidden units that operate with different time constants. This approach is most relevant to tasks that involve recognizing the reappearance of a pattern presented in the relatively distant past. The architecture described in this paper presumably provides the hidden layer with more detailed information about the inputs seen in the relatively recent past, and might be expected not to provide a dramatic improvement on Mozer's task.

## 5 Conclusions

The best average global minimum and best average epoch number of first local minimum was for the S7H13 architecture. The best global minimum TSS for the 10 runs of this architecture was 88.35% of the best performance for the S1H24 architecture. The TSS values from epoch to epoch for instances of the S7H13 architecture were erratic, but ultimately this is unimportant: the absolute minimum TSS is what matters most in a sequential prediction task. Thus these experiments show that use of several state vectors, linked as exemplified in Figures 2 and 3, can improve the performance of a recurrent network with a given number of weights.

| | S1H24 | | S2H20 | | S4H16 | | S7H13 | |
|---|---|---|---|---|---|---|---|---|
| | *Rank* | *Min.* | *Rank* | *Min.* | *Rank* | *Min.* | *Rank* | *Min.* |
| Ranked global minima | 31 | 921.9 | 21 | 883.8 | 7 | 839.7 | 1 | 814.5 |
| | 32 | 922.3 | 22 | 890.0 | 9 | 848.3 | 2 | 834.9 |
| | 33 | 922.5 | 23 | 891.1 | 10 | 855.4 | 3 | 835.7 |
| | 34 | 922.9 | 24 | 892.6 | 11 | 855.5 | 4 | 835.8 |
| | 35 | 923.1 | 25 | 894.1 | 12 | 856.2 | 5 | 836.0 |
| | 36 | 923.1 | 26 | 894.3 | 13 | 858.5 | 6 | 837.1 |
| | 37 | 925.5 | 27 | 896.3 | 14 | 859.0 | 8 | 840.0 |
| | 38 | 929.0 | 28 | 899.4 | 18 | 867.1 | 15 | 859.1 |
| | 39 | 930.2 | 29 | 899.4 | 19 | 868.1 | 16 | 863.3 |
| | 40 | 930.2 | 30 | 903.9 | 20 | 872.8 | 17 | 866.5 |
| Mean | 35.5 | | 25.5 | | 13.3 | | 7.7 | |

Table 3: Global Minima and Their Ranks

## Acknowledgements

This work was supported by a grant from the Australian Telecommunications and Electronics Research Board. The author would like to thank Janet Wiles for helpful discussions about recurrent networks. Statistical analysis of the experimental results was performed by Kelly Reynolds under the supervision of Deborah Street and David Byron.

## 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] 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) San Mateo, CA: Morgan Kaufmann, 1992.

[4] 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.

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