Neural Nets and Symbolic Reasoning

Structure in Time: Recurrent Networks

Outline

- The role of time in cognition
- Finding structure in time: Elman's simple recurrent networks
- Learning to count without a counter
- General conclusions

1 The role of time in cognition

Why time is important

- Time is clearly important in cognition. It is inextricably bound up with many behaviors which express themselves as temporal sequences.
- How to deal with such basic problems as goal-directed behavior, planning, or causation without some way of representing time?
- The example of sentence processing

  (A) Sentences are processed sequentially in time

  (B) Sentences exhibit long-distance dependencies (Agreement phenomena; binding phenomena)

Why time is a problem

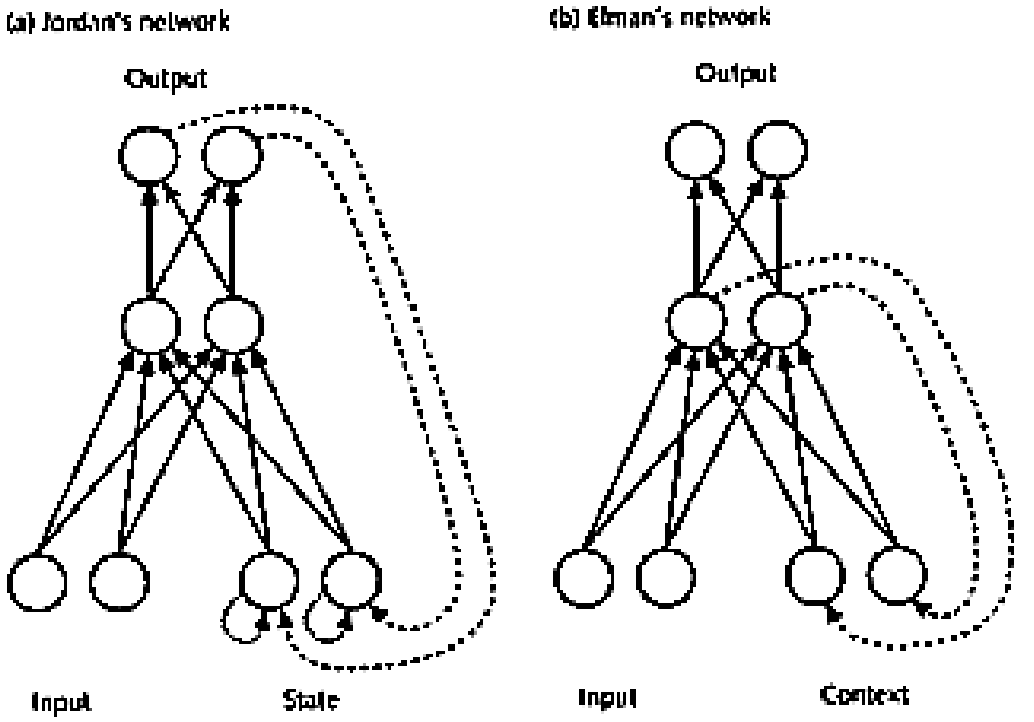
- The parallel nature of neural processing seems to be ad odds with the serial nature of time
- However, even within traditional (serial) frameworks, the representation of serial order presents challenges. For example, in models of motor activity an important issue is whether the action plan is a literal specification of the output sequence, or whether the plan represents serial order in a more abstract manner (e.g., Lashley, 1951)
- Research in natural language parsing suggests that the problem with long-distance dependencies is not trivially solved if language is processed sequentially (e.g., Frazier & Fodor; 1978; Marcus, 1980).

Approaching time in PDP

- Standard feedforward networks? At a given point in time such a network has only access to the LTM (weights) and to the patterns generated by its current input. No access to previous inputs.
- RAAM? They are able to retain the constituent structure of a sentence. The successive coding in terms of hidden unit patterns establishes a kind of STM for already processed parts of a sentence. However, the copying of pattern is in space, not really in time. (Parallels time: giving time a spatial representation)
- Representing time by the effect it has on processing and not as an additional dimension of the input (Recurrent networks)

Two architectures

Jordan (1986) added recurrent connections for copying the pattern on the output units to the state units. Elman (1990) modified this account and copied the content of the hidden units back to the context units. (state = context).



Sending the pattern of the hidden unit back to the network, makes the network's activity sensitive to its own construal of the immediately preceding input (Also in RAAM: the copy come from the hidden unit).

2 Finding structure in time:

Elman's simple recurrent networks

COGNITIVE SCIENCE, **14**, 179-211 (1990).

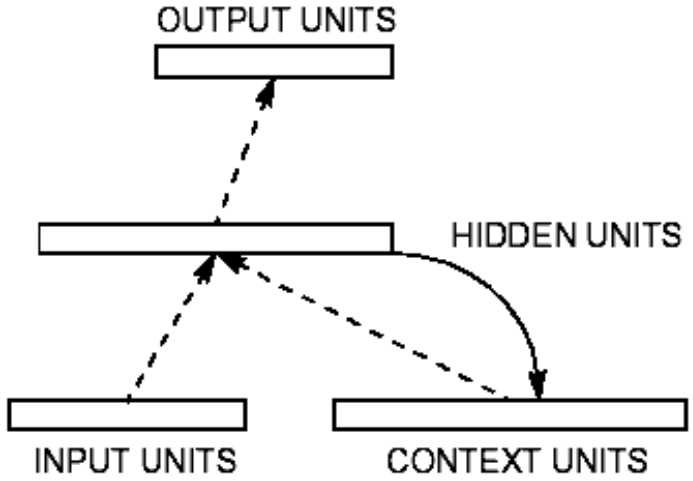
Finding Structure in Time

JEFFREY L. ELMAN
University of California, San Diego

The approach described here employs a simple architecture but is surprisingly powerful. The are several points worth highlighting:

- Some problems change their nature when expressed as temporal events: a sequential version of the XOR
- The time-varying error signal as a clue to temporal structure
- There is no separate representation of time
- Memory is neither passive nor a separate subsystem.

1. The input units receive the first input
2. Both the input units and context units activate the hidden units
3. - The hidden units feed forward to activate the output units
   - The hidden units also feed back to activate the context units (copying the content of the hidden units)



In this time cycle, there is a learning phase: The output is compared with a teacher input and backpropagation of error is used to incrementally adjust connection strength. Recurrent connections are fixed at 1.0 and are not subject to adjustment.

Internal representation of time

- The hidden units develop internal representations for the input patterns
- The context units remember the previous internal state
- The hidden units: Mapping external input and previous internal state to some desired output
- The hidden units must accomplish this mapping. At the same time they develop representations which are useful encodings of the temporal properties of the sequential input
- Thus, the internal representations that develop are sensitive to temporal context; the effect of time is implicit in these internal states.

Standard XOR:

input vector: (00, 11, 01, 10)

output vector: (0, 0, 1, 1)

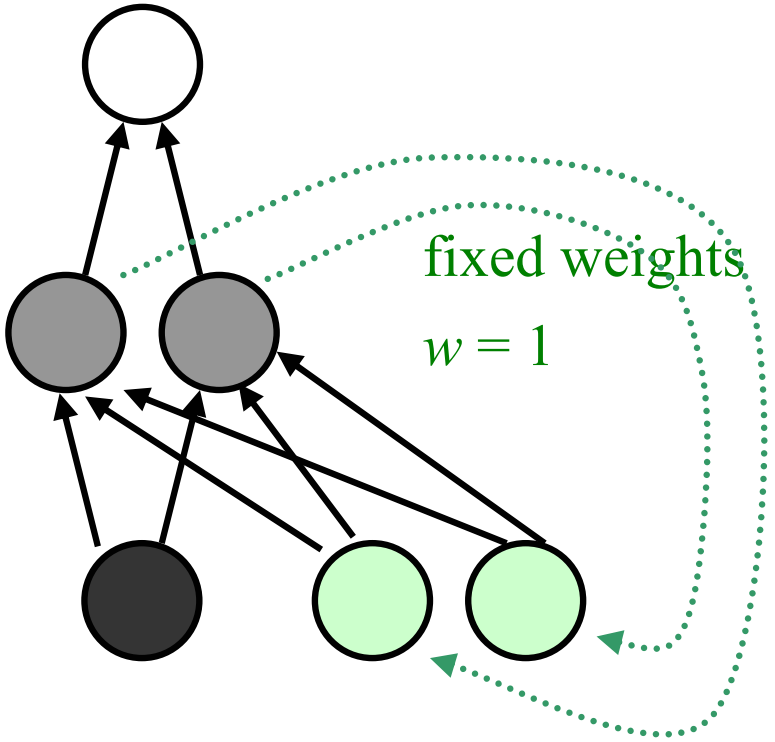
Serial version:

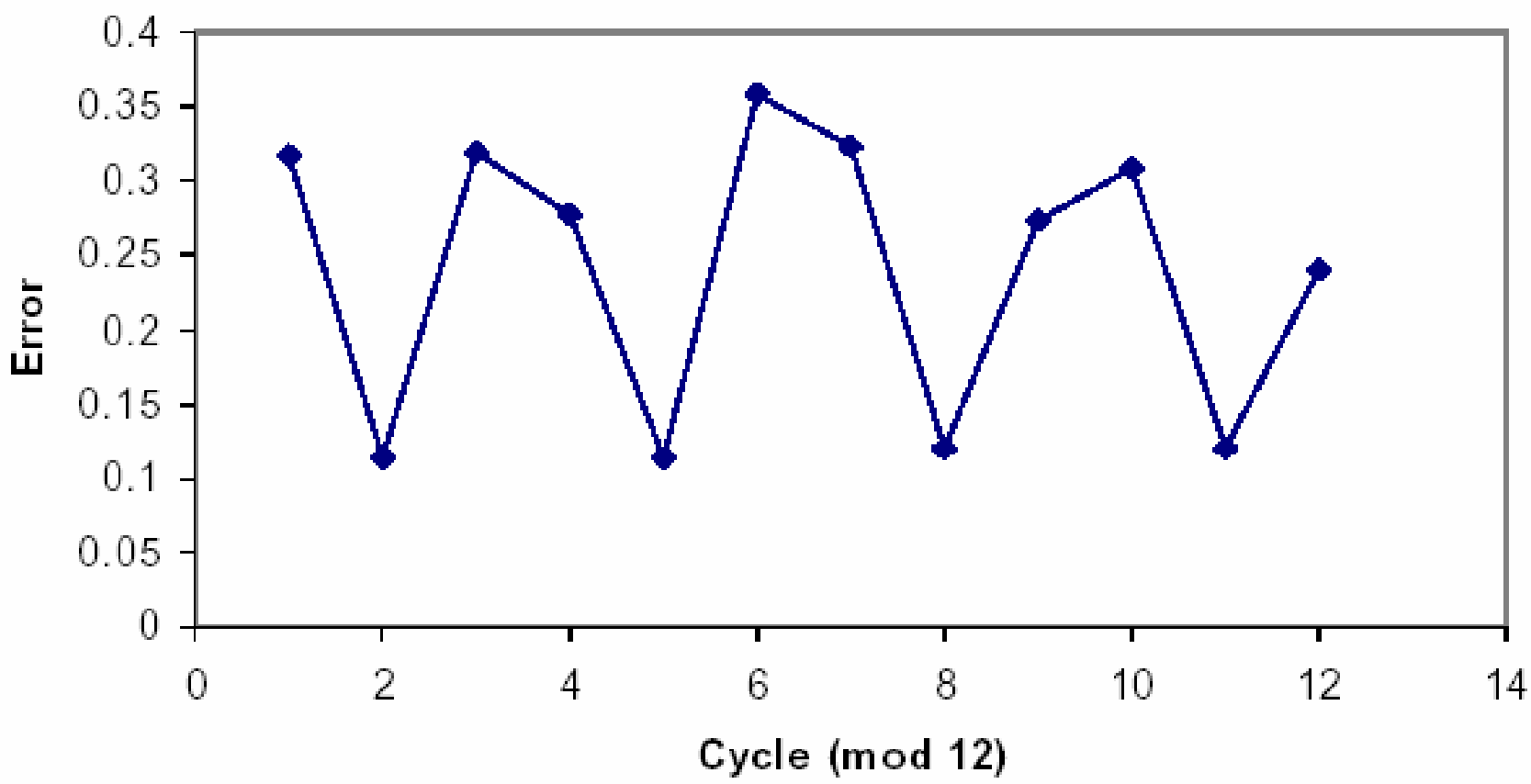
input: 1 0 1 0 0 0 0 1 1 1 1 0 1 0 1 . . .

output: 0 1 0 0 0 0 1 1 1 1 0 1 0 1 ? . . .

chance: ♦ ◆ ♦ ♦ ◆ ♦ ♦ ◆ ♦ ♦ ◆ ♦ ♦ ◆

1 2 3 4 5 6 7 8 9 101112 1314





Graph of root mean squared error over 12 consecutive inputs in sequential XOR task. Data points are averaged over 1200 trials

Some problems change their nature when expressed as temporal events

- simultaneous version: the two hidden nodes are 0 if the two input elements are the same, otherwise the two hidden nodes are different
- sequential version: one of the two hidden units is highly activated when the input sequence is a series of identical elements (all 1s or 0s), whereas the other unit is highly activated when the input elements alternate.
- Hence, the solution to the sequential version of the XOR problem involved detection of state changes (frequency-sensitive hidden units)
- Casting the XOR problem in temporal terms led to a different solution than is typically obtained in feed-forward networks.

Structure in letter sequences: Discovering the notion word

- What is a word?
  - Defined as a *unit* on a certain level of representation
  - The commitment to such distinct levels is often problematic
  - Languages differ dramatically in what they treat as words
  - Even in English no consistently definable distinction between words (*apple*), compounds (*apple pie*), phrases (*Library of Congress*)
- Computational mechanism to detect the boundaries between words?
  - Phonetic and prosodic structure is not enough
  - Manyyearsagoaboyandgirl …

Cues for word boundaries

Manyyearsagoaboyandgirl…

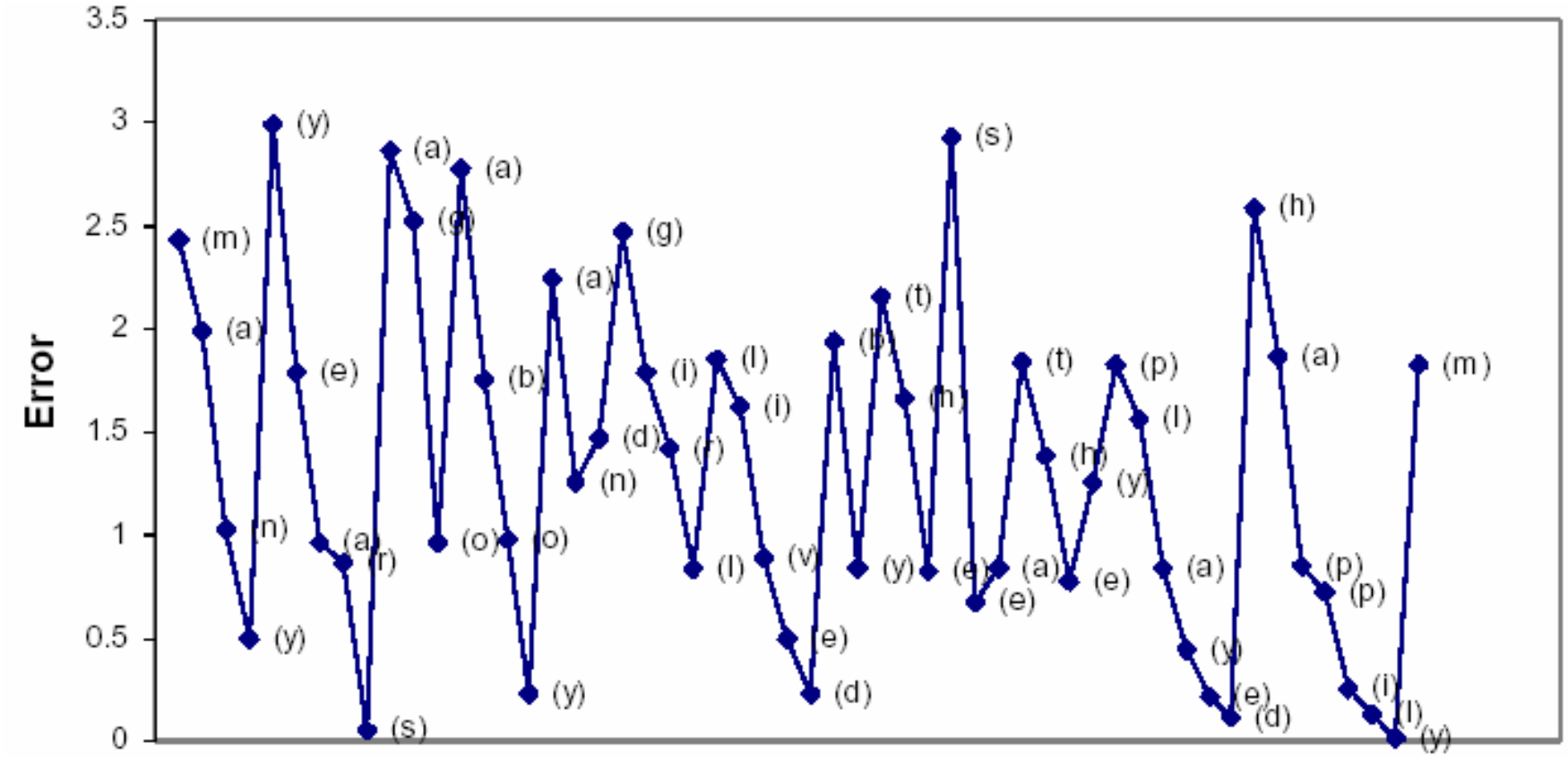
Many years ago a boy and girl …

- One can ask whether the notion "word" (or something which maps on to this concept) could emerge as a consequence of learning the sequential structure of letter sequences which form words and sentences (but in which word boundaries are not marked).
- Is there information in the signal which could serve as a cue as to the boundaries of linguistic units which must be learned?
- Simulation that shows that a simple recurrent network can extract relevant probabilistic information (statistics of co-occurrence) that correlates with words. No semantics is required at this point.

Input		Output	
0110	*m*	0000	*a*
0000	*a*	0111	*n*
0111	*n*	1100	*y*
1100	*y*	1100	*y*
1100	*y*	0010	*e*
0010	*e*	0000	*a*
0000	*a*	1001	*r*
1001	*r*	1001	*s*
1001	*s*	0000	*a*
0000	*a*	0011	*g*
0011	*g*	0111	*o*
0111	*o*	0000	*a*
0000	*a*	0001	*b*
0001	*b*	0111	*o*
0111	*o*	1100	*y*
1100	*y*	0000	*a*
0000	*a*	0111	*n*
0111	*n*	0010	*d*
0010	*d*	0011	*g*
0011	*g*	0100	*i*
0100	*i*	1001	*r*
1001	*r*	0110	*l*
0110	*l*	1100	*y*
1100	*y*		

Simulation study

1. Using a lexicon of 15 words, 200 sentences were generated of varying length (4-9 words)
2. A big string with 4963 letters were generated from that. Each letter was converted into a 5 bit random vector
3. A SRN with 5 input units, 20 hidden units, 5 output units, and 20 context units was trained on 10 complete presentations of the sequence.
4. Errors for letter prediction were calculated.



Graph of root mean squared error in letter-in-word prediction task. The sequences bounded by high error correlate with words.

Conclusions

- The time-varying error signal can be used as a clue to temporal structure. Temporal sequences are not always uniformly structured, nor uniformly predictable
- The error signal is a good metric of where structure exists; it thus provides a potentially very useful form of feedback to the system
- The co-occurrence of sounds is only part of what identifies a word
- A similar procedure can used for learning grammatical categories. This time, sequences of words are teached instead of sequences of sounds/letters. Instead of analysing prediction errors, this time a cluster analysis of the patterns formed on hidden units is useful.

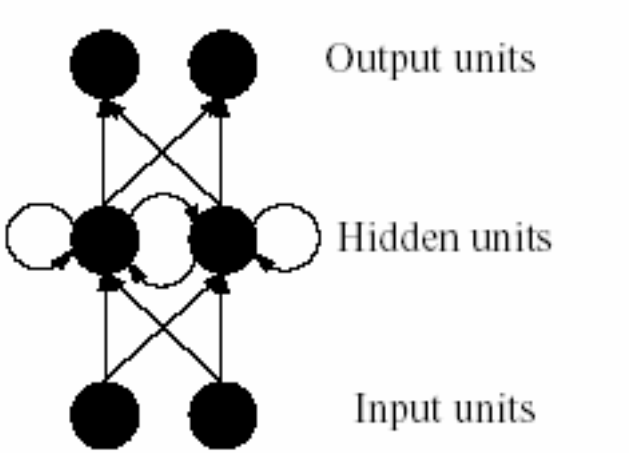
3 Learning to count without a counter

The computational power of (recurrent) neural networks

- Hava Siegelmann & Eduardo Sontag: On the computational power of neural nets
- Jiri Sima & Pekka Orponen: A computational taxonomy and survey of neural network models
- Peter Tino, Bill Horne, Lee Giles, and Pete Collingwood: Finite state machines and recurrent neural networks - automata and dynamical systems approaches
- Janet Wiles & Jeff Elman: Learning to count without a counter: A case study of dynamics and activation landscapes in recurrent networks.

The counting task

- Can a recurrent network be trained to predict the deterministic elements in sequences of the form $a^n b^n$ where n =1 to ...?
- $a^n b^n$ is one of the simplest CF languages
- The study shows that recurrent network are able to emulate certain aspects of a pushdown automaton
- It is suggested to use the proposed solution as a platform for developing a more general understanding of recurrent networks as computational mechanisms

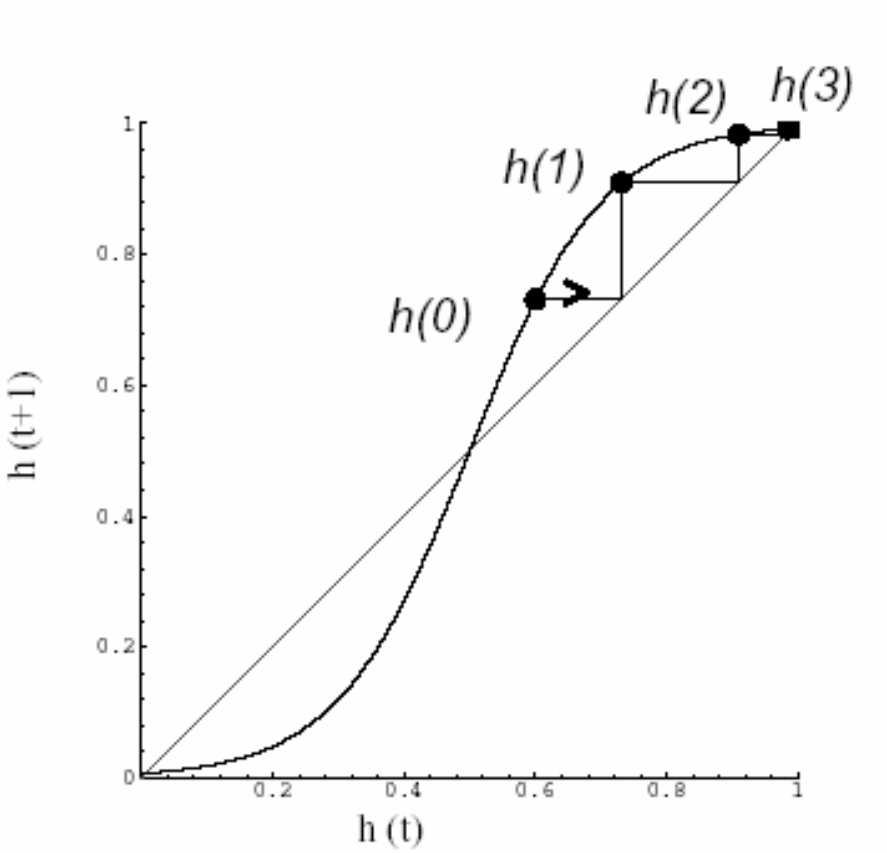


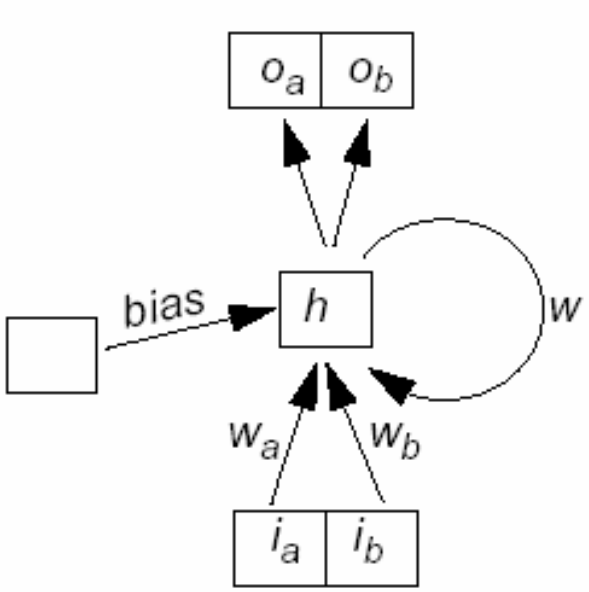
- A training set consisting of 356 strings, containing a total of 2298 tokens of *a* and *b* (coded as 10 and 01, respectively).
- These strings conformed to the form $a^n b^n$, with n ranging from 1 to 11
- Networks was trained using back propagation through time (for 8 time steps). Training was carried out for a total of 3 million inputs.
- A separate set of test stimuli were generated which consisted of all possible strings with n ranging from 1 to 30 so it was possible to test generalization to depths greater than that encountered during training.

- After 1 million training cycles 9 of the 20 identical networks learned the language for n ≤ 7. One network generalized to n≤11. The other networks learned the language $a^* b^*$. This is the language consisting of any number of *a*s followed by any number of *b*s
- After 2 million training cycles 4 of the 20 identical networks generalized the correct language for to n ≤ 12. One network generalized to n≤18. The remaining networks had learned $a^* b^*$.
- Subsequent replications showed a similar statistics, with at least one network that generalized to approximately a depth of 18. We focus on that network for analysis.

$h(t) = \sigma(bias + i_a \cdot w_a + i_b \cdot w_b + h(t-1) \cdot w)$

$h(t) = \sigma(b + h(t-1) \cdot w)$



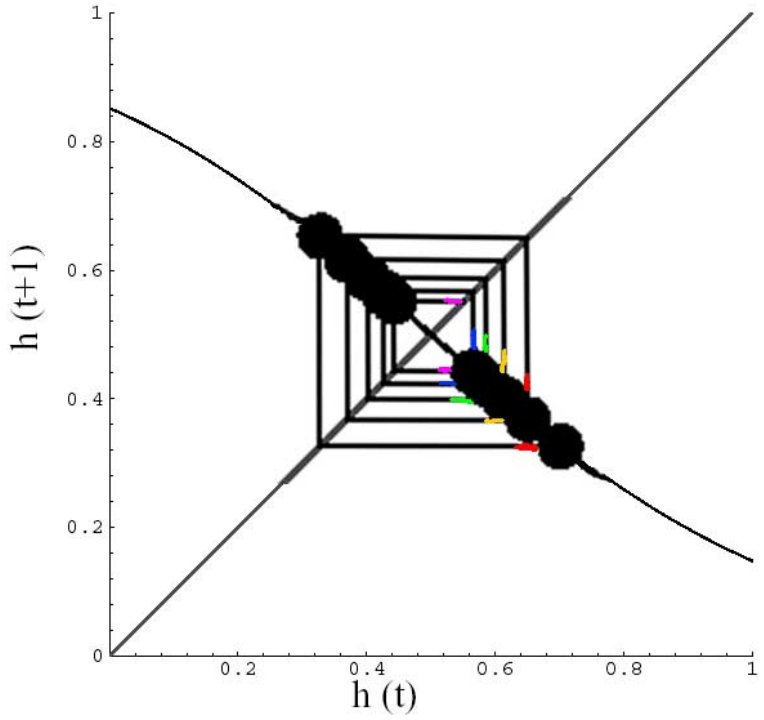


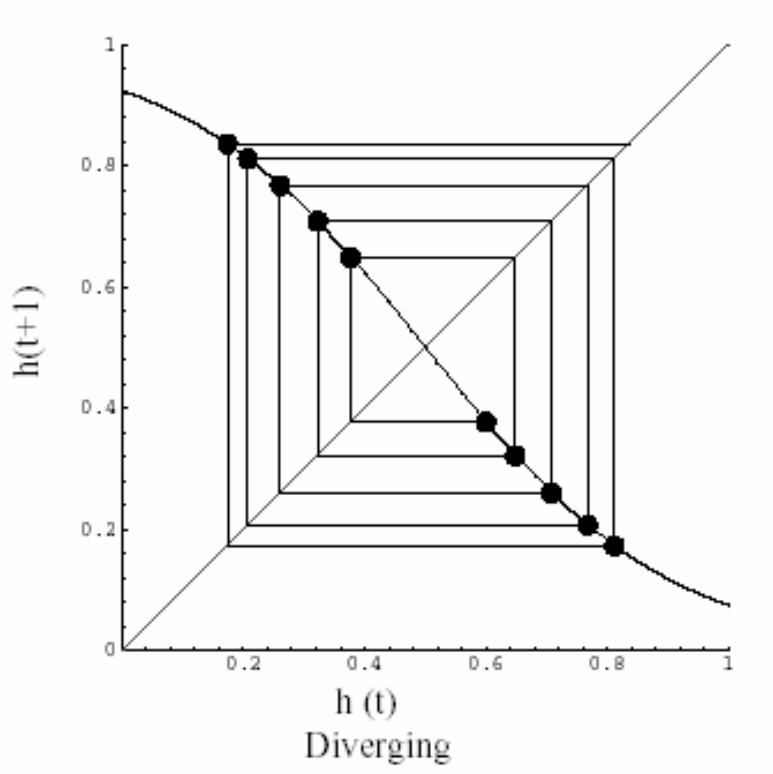
Dynamic properties of the network, with w=10 and b=–5.

If we begin with h(0) greater than 0.5, we see the movement in activation space shown in the figure.

Converging and diverging regimes

With negative weights w, two different regimes are found:
converge inward (left) and diverge out (right).

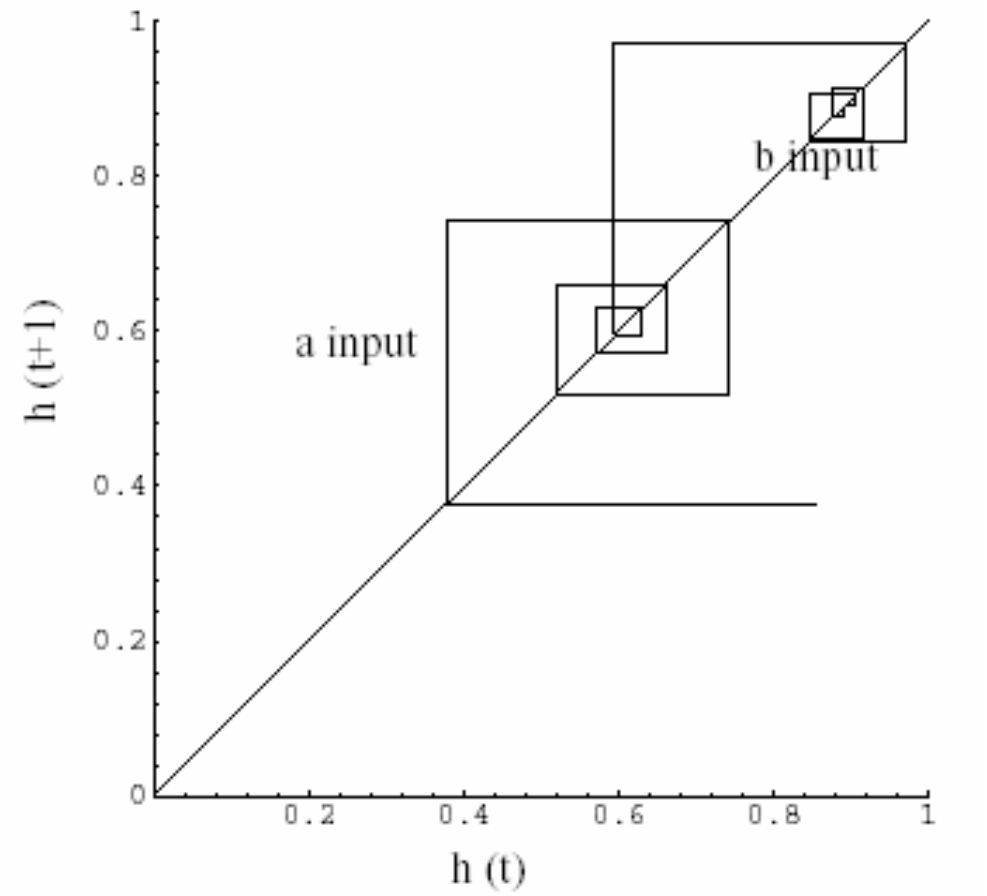




Network with two hidden units

Hidden unit oscillations in trained network, 7 *a*'s (spiral on lower left, representing hidden unit 1), followed by 7 *b*s (spiral on upper right, representing hidden unit 2).

First regime: winding up a spring. The presentation of the first b moves the network in the second regime: unwinding the spring.



Conclusion: how to count with a sand glass

aaaaaaaaaaaaaaaaaaa

bbbbbbbbbbbbbbbbb

4 General Conclusions

- The sort of memory that simple recurrent networks provide differs dramatically from the traditional cognitive models
- They process strings of items item by item but use their context units to incorporate information about previous items
- In such networks, the recursive activity can extend through many cycles, although the further back the cycle, the more degraded is the information in the context pattern and the less it contributes to current activity
- Prior notions of how recurrent networks might be expected to solve familiar computational problems are to be regarded as open hypotheses only. We should be prepared for surprises.