4a: Recurrent Networks

Week 4: Overview

This week, we will explore the use of neural networks for sequence and language processing. Simple Recurrent Networks (SRN) can be trained to recognize or predict formal languages, and we can analyse their hidden unit dynamics. By the use of a gating mechanism, Long Short Term Memory (LSTM) and Gated Recurrent Networks (GRU) are able to learn longer range dependencies than SRN.

Then we will look at the statistics of language, and show how word vectors can be assigned in a systematic way using approximations to singular value decomposition such as word2vec and GloVe.

Weekly learning outcomes

By the end of this week, you will be able to:

- design and train neural networks for processing temporal sequences
- describe different architectures including sliding windows, simple recurrent networks, Long-Short Term Memory (LSTM), and Gated Recurrent Units (GRU)
- analyse hidden unit dynamics of recurrent networks
- describe word frequencies, n-gram model, co-occurrence matrix
- describe the components and properties of singular value decomposition
- describe the word2vec model, including architecture, cost function and efficiency issues

Recurrent Networks

Processing Temporal Sequences

There are many tasks for which the output depends on a sequence of inputs rather than a single input. For example:

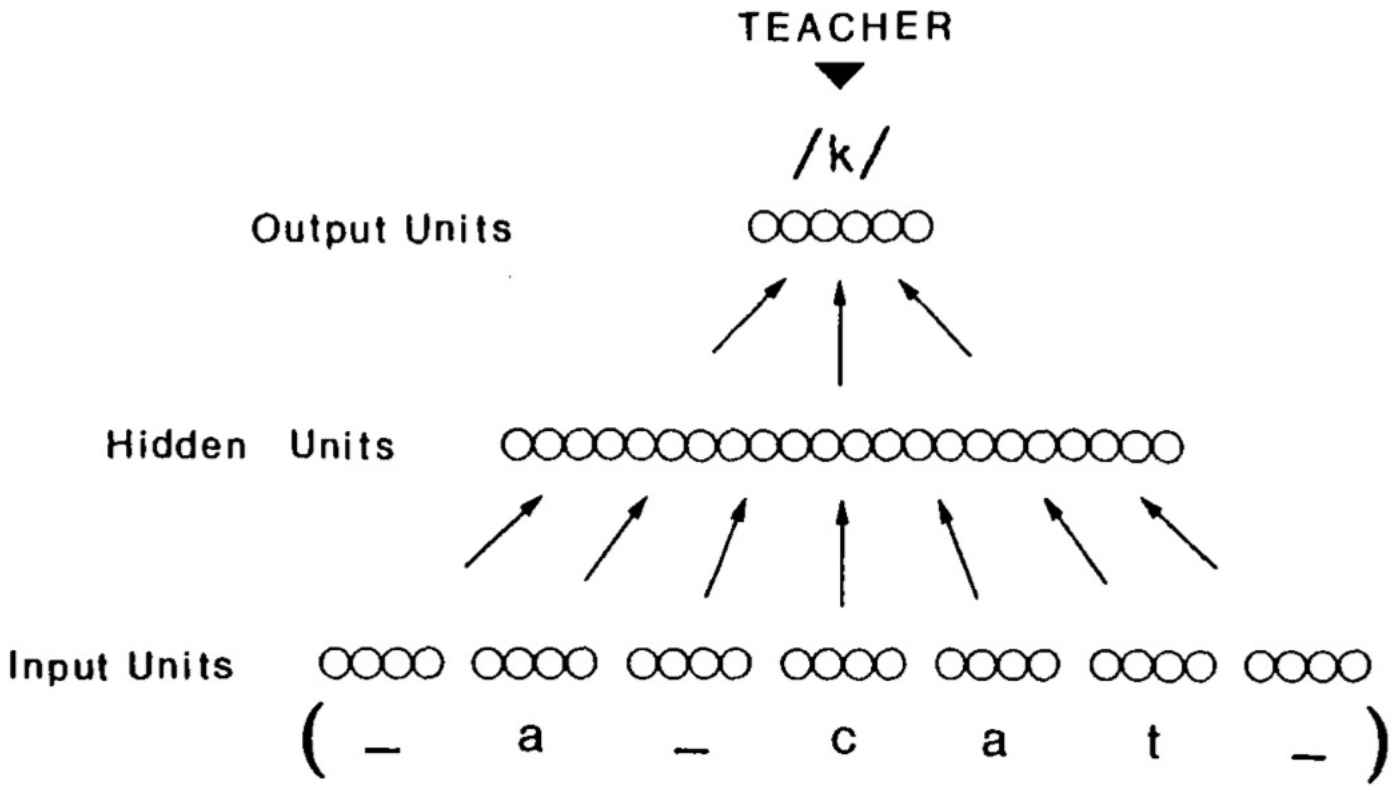
- speech recognition
- time series prediction
- machine translation
- handwriting recognition

How can neural network models be adapted for these tasks?

NetTalk sliding window approach

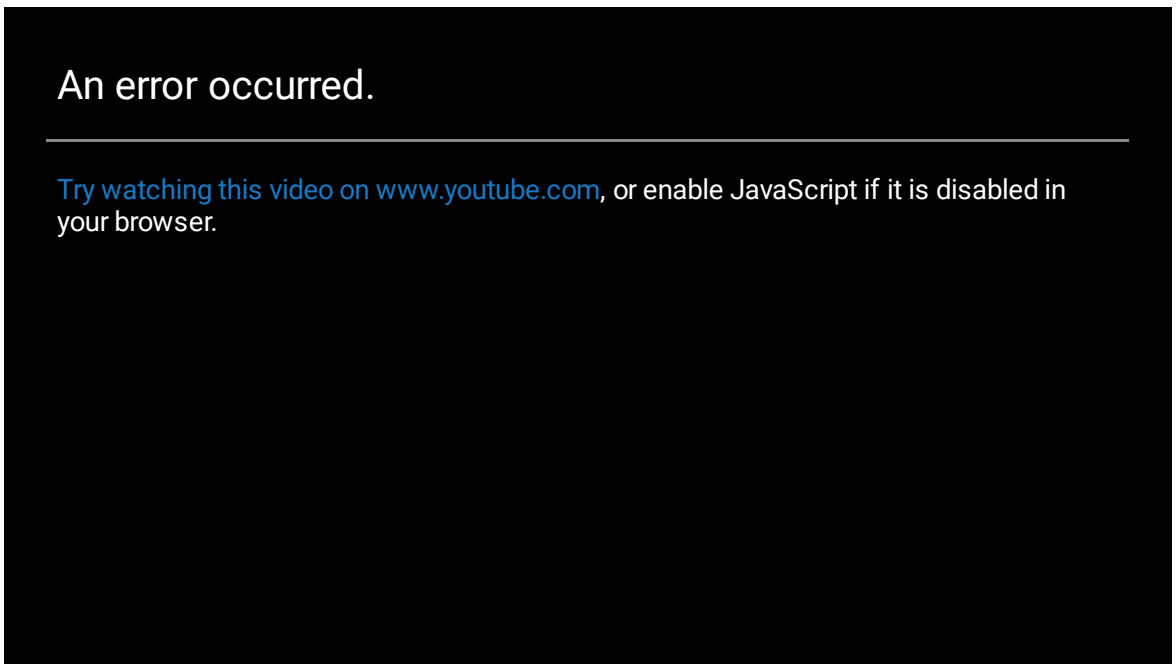
The simplest way to process temporal sequences using a neural network is the 'sliding window' approach, first used in the NetTalk system (Sejnowski & Rosenberg, 1987). English text is fed to NetTalk as a sequence of characters, and it outputs a sequence of phonemes indicating how that text should be pronounced. Specifically, its input consists of seven consecutive characters and it aims to output a set of phonetic attributes specifying the correct pronunciation for the middle character. In order to illustrate why it is necessary to know a few characters to the left and right of the middle character, consider how the first vowel would be pronounced in each of the following examples:

- pa, pat, pate, paternal
- mo, mod, mode, modern.



For each of the seven input characters, a one-hot encoding is used with 29 units (including the 26 letters, plus punctuation and word boundaries) making a total of 203 inputs. There were 80 hidden units with sigmoidal transfer function, and 26 outputs encoding 21 articulatory features plus stress and syllable boundaries.

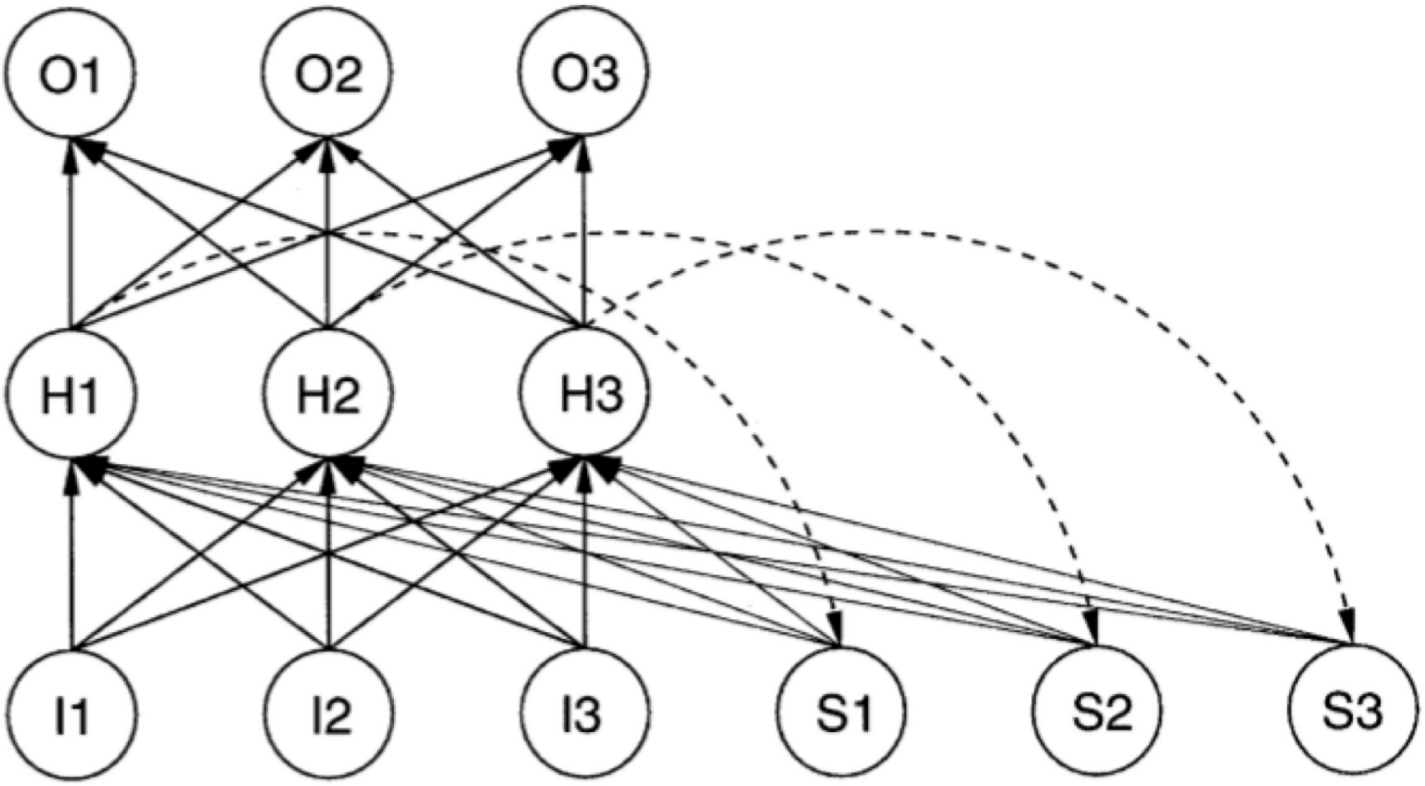
NetTalk gained a lot of media attention at the time, partly because the output was hooked up to a speech synthesiser. In the early stages of training, it sounds a bit like a babbling baby. When fully trained, it sounds reasonable, although somewhat robotic.



Critics have sometimes claimed that a decision tree could produce equally good or better accuracy. In

any case, this kind of approach can only learn short-term dependencies, not the medium or long-term dependencies that are required for some tasks.

Simple Recurrent Network



Simple Recurrent Networks (SRN) were introduced in Elman (1990). A sequence of inputs are fed to the network one at a time. At each timestep, the current activations in the hidden layer are copied to a 'context' layer. The new activations for the hidden layer are then calculated from the current input and the context layer. Thus, the context layer is used to 'remember' whatever information the network requires in order to predict the correct output.

The basic SRN can be augmented with 'shortcut' connections directly from input to output, or connections from the output back to the hidden layer (sometimes called “Jordan Networks”).

Backpropagation through time

For any given input sequence, we can 'unroll' a recurrent architecture into an equivalent feedforward architecture, with shared weights. Applying backpropagation to the unrolled architecture is referred to as **backpropagation through time**. We can backpropagate just one timestep, or a fixed number

of timesteps, or all the way back to the beginning of the sequence.

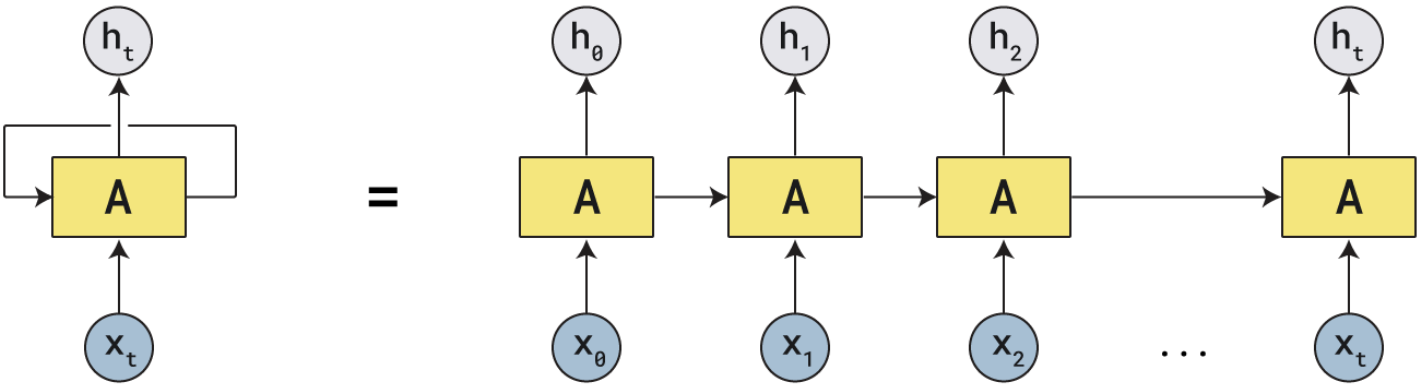


Image adapted from http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

Optional video

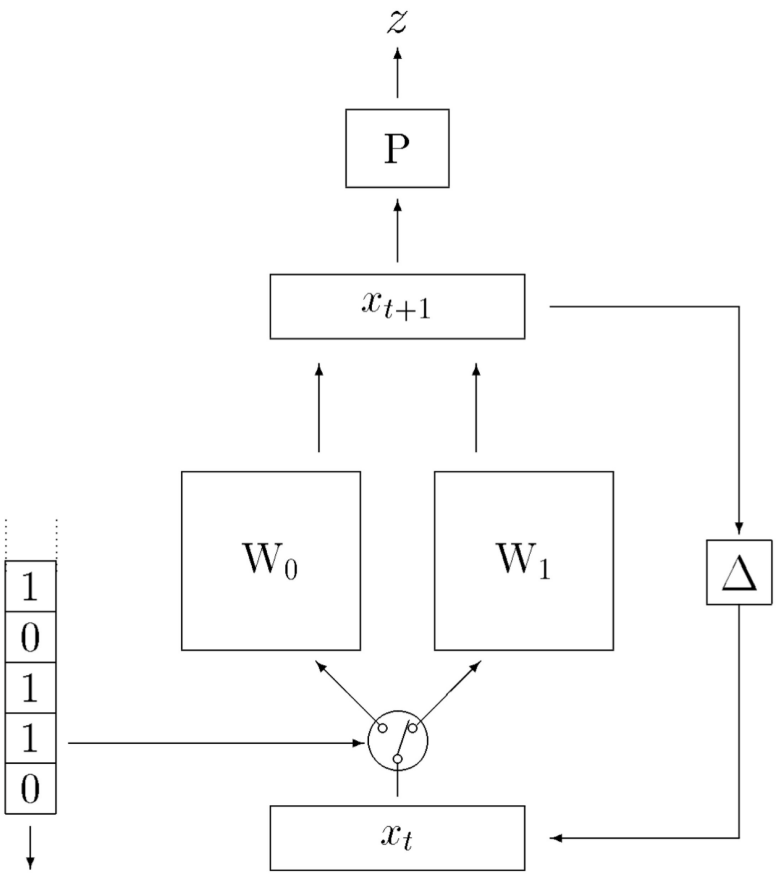
References

Sejnowski, T. J., & Rosenberg, C. R. (1987). Parallel networks that learn to pronounce English text. *Complex Systems*, *1*(1), 145–168.

Elman, J. L. (1990). Finding structure in time. *Cognitive Science*, *14*(2), 179–211.

Recognizers and Predictors

The basic SRN can be augmented with 'shortcut' connections directly from input to output, or connections from the output back to the hidden layer (sometimes called “Jordan Networks”). Another architecture which has been explored for binary sequences is the Second Order Network or Gated Network, where the input symbol (0 or 1) is used to choose between two sets of weights W_0 and W_1. The weights W_0, W_1 and P and the initial state x_0 can all be trained by backpropagation through time.



$$x_t^j = \tanh\left(\mathrm{W}_{\sigma_t}^{j0} + \sum_{k=1}^{d} \mathrm{W}_{\sigma_t}^{jk} x_{t-1}^k\right)$$

$$z_t = g\left(\mathrm{P}_0 + \sum_{j=1}^{d} \mathrm{P}_j x_t^j\right)$$

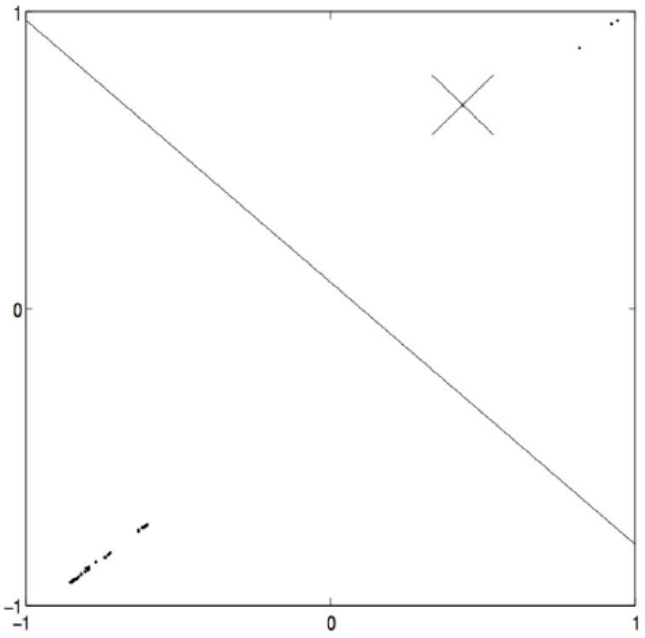
Dynamical Recognizers

One common sequence processing task is Formal Language Recognition. The network scans a sequence of characters one at a time, and must then output either 1 or 0, in order to classify the

sequence as Accept or Reject, respectively. Consider, for example, this set of training data:

Accept	Reject
1	0
11	10
111	01
1111	00
11111	011
111111	110
1111111	11111110
11111111	10111111

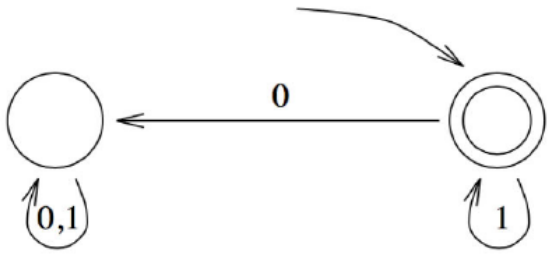
The natural generalisation from these data is to classify strings consisting entirely of 1's as Accept and any string containing 0 as Reject. This diagram shows the result of training a Second Order Network on these data. The cross indicates the initial state while the dots indicate all the hidden states which occur while the network is processing a set of randomly generated test strings. The dividing line shows the boundary between final states for which the string is classified as Accept, and those for which it is classified as Reject.



$$W_0 = \begin{bmatrix} -0.89 & -0.09 & -0.14 \\ -1.13 & -0.09 & -0.14 \end{bmatrix}$$

$$W_1 = \begin{bmatrix} 0.20 & 0.68 & 0.96 \\ 0.20 & 0.81 & 1.19 \end{bmatrix}$$

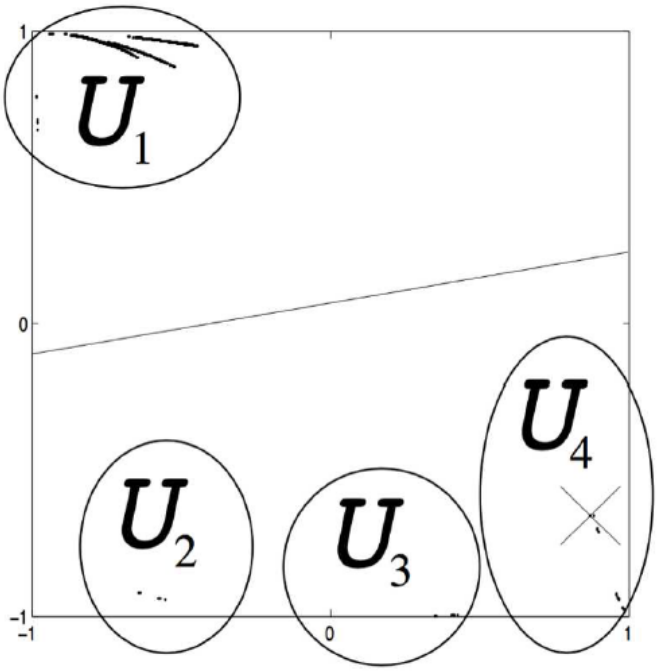
$$P = \begin{bmatrix} -0.07 & 0.66 & 0.75 \end{bmatrix}$$

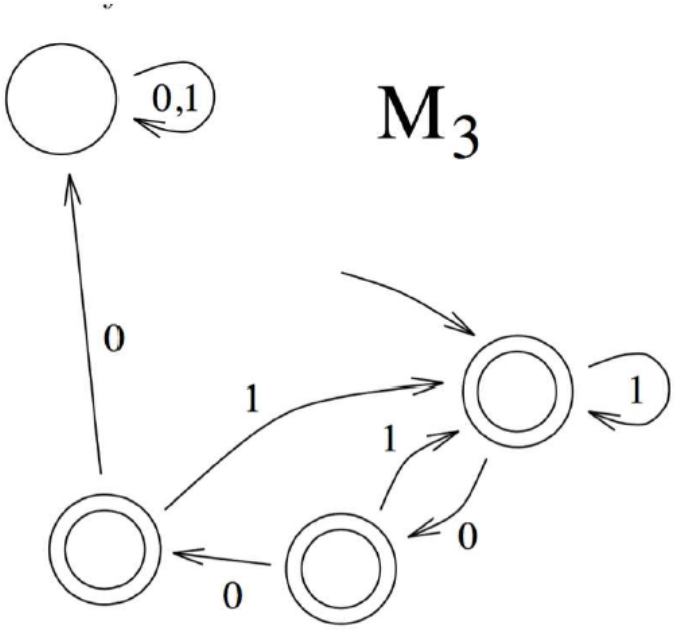


In some cases, we can use analytical tools to extract a finite state machine which exactly mimics the behaviour of the network, as shown on the right for this example (Blair & Pollack, 1997). Each cluster of hidden unit states is converted into a single state, the transitions between clusters are converted into arrows, and the initial state is indicated by an arrow with no origin. The state(s) on the Accept side of the dividing line are indicated by a double circle; those on the Reject side by a single circle.

Here is another example, where the network learns to Reject any string with three consecutive 0's and Accept everything else.

Accept	Reject
1	000
0	11000
10	0001
01	000000000
00	11111000011
100100	1101010000010111
001111110100	1010010001
0100100100	0000
11100	00000



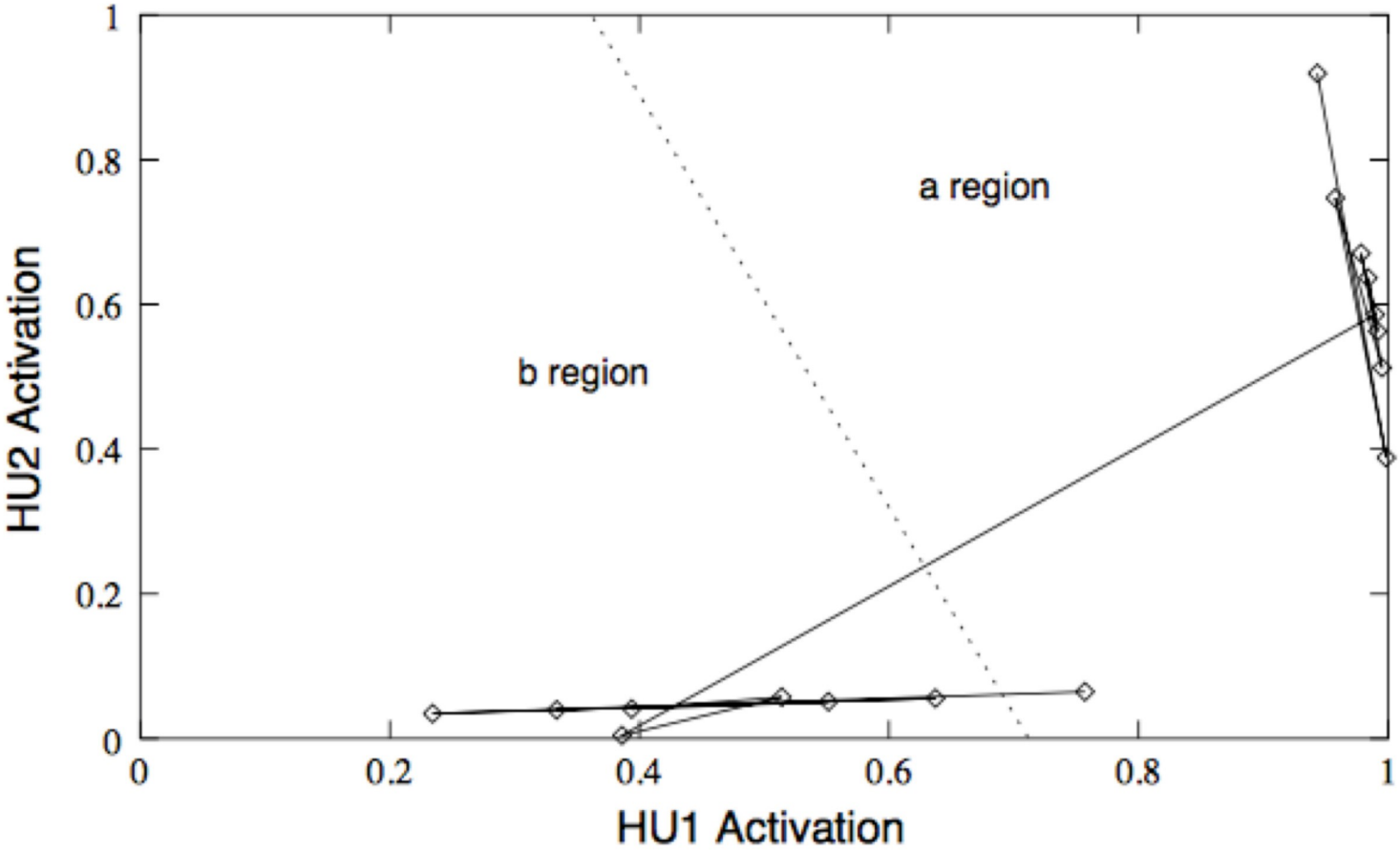


Non-Regular languages

Languages which can be characterized by a finite state machine are called **Regular** languages. One simple example of a non-Regular language is $a^n b^n$ which means that every sequence of consecutive a's must be followed by an equal number of consecutive b's — for example:

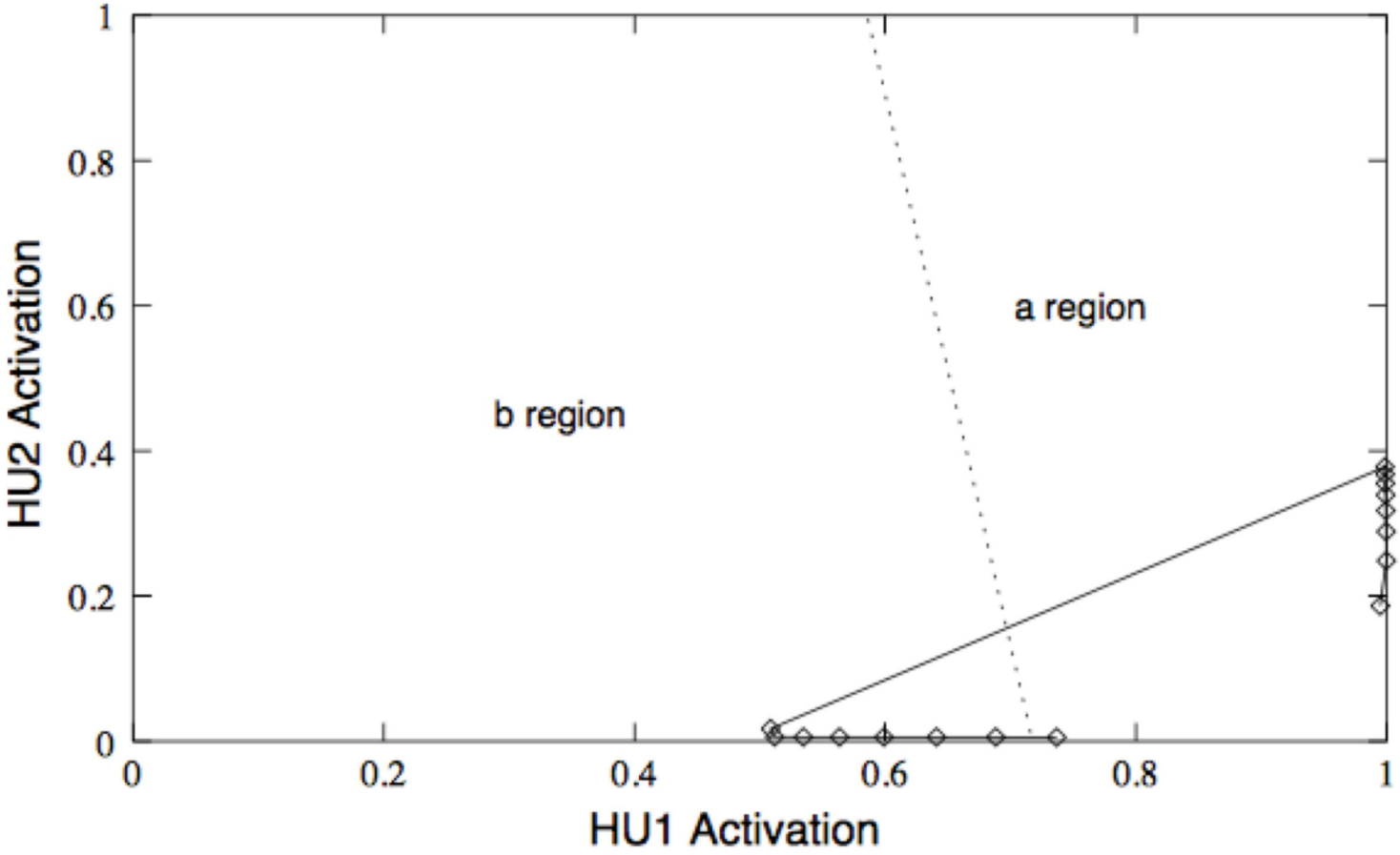
abaabbabaaabbbaaaabbbbabaabbaaaaabbbbb . . .

A SRN with 2 inputs, 2 hidden nodes and 2 outputs can be trained to predict the $a^n b^n$ language (Wiles & Elman, 1995). In this case, the network must scan a sequence of input characters one at a time, and try at each step to predict the next character in the sequence. In some cases, the prediction is probabilistic. For the $a^n b^n$ task, the first b is not predictable, but subsequent b's and the initial a in the next subsequence are predictable.



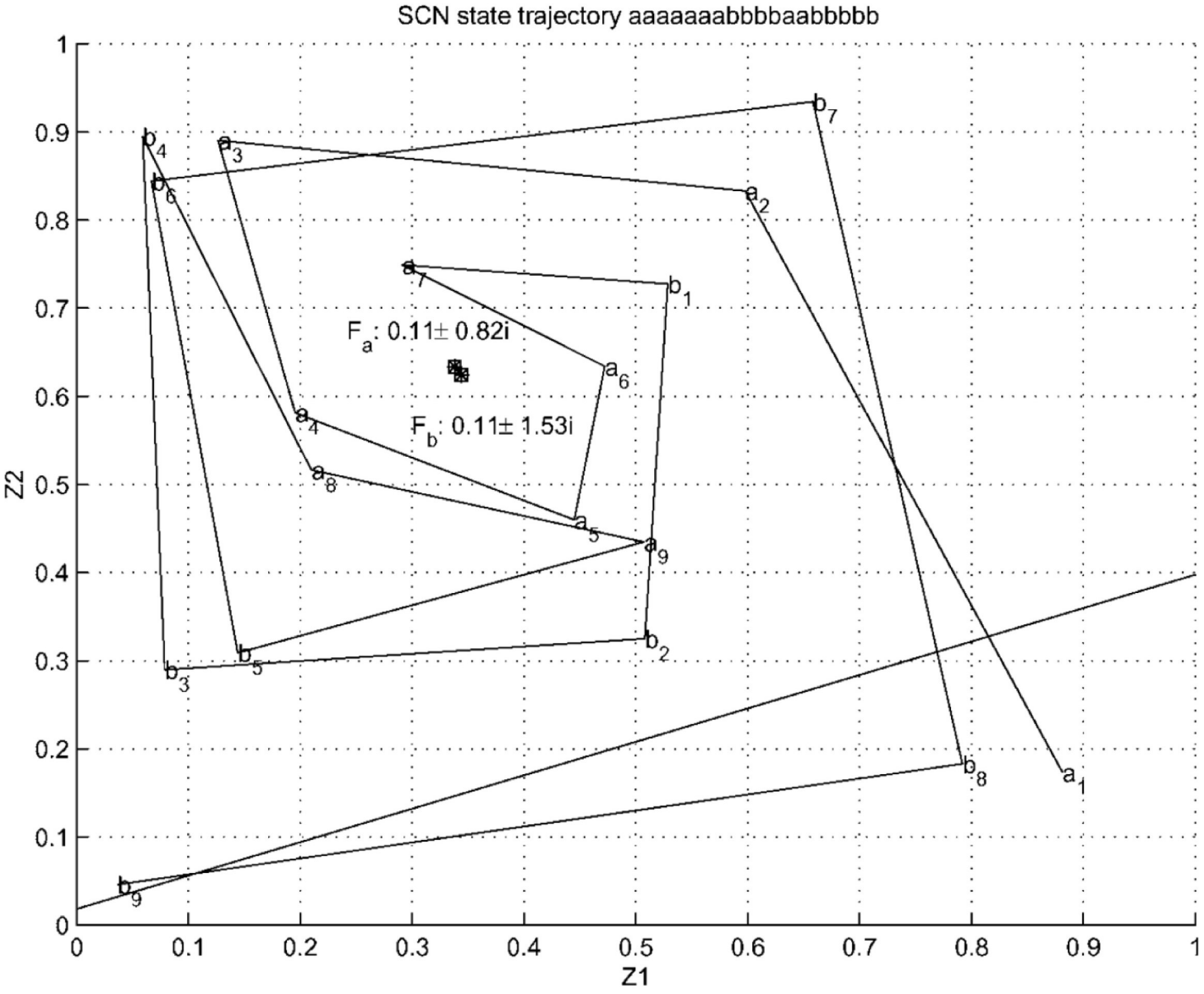
The network does not implement a Finite State Machine but instead makes use of two fixed points in activation space – one attracting, the other repelling (Wiles & Elman, 1995). The network effectively 'counts up' the a's as it oscillates towards the attractive fixed point and then counts down the same number of b's as it oscillates away from the repelling fixed point. When the recurrent mapping is linearised at the two fixed points we find that the two major eigenvalues are nearly reciprocal to each other. Interestingly, networks trained only up to $a^{10}b^{10}$ can often generalise up to $a^{12}b^{12}$.

Training the weights by evolution is more stable than by backpropagation. Networks trained by evolution sometimes exhibit monotonic rather than oscillating trajectories.



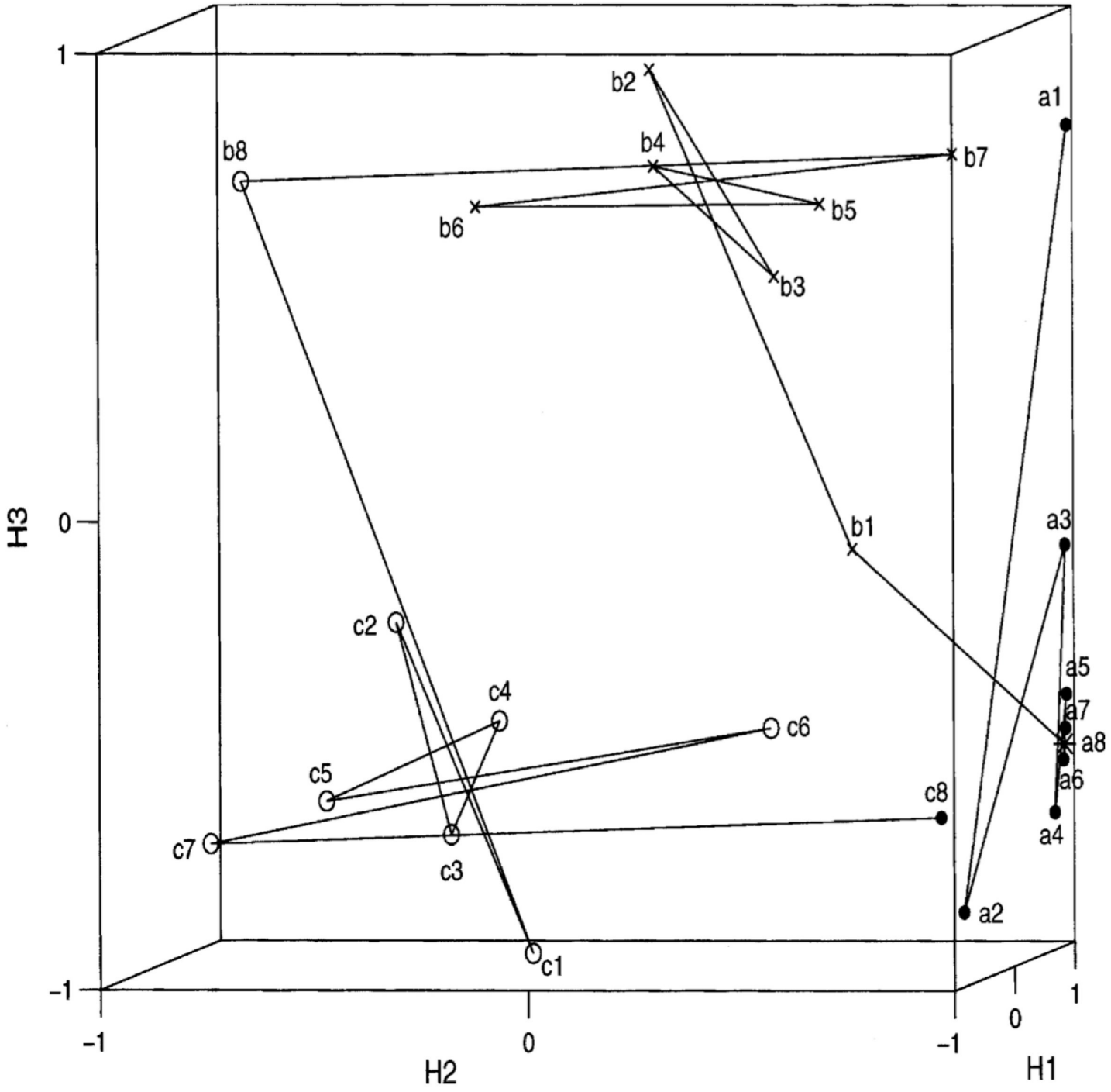
Counting by spiralling

SRN's can also be trained to recognise the balanced bracket language, treating a as open bracket and b as close bracket (Boden, 2003). In this case, the eigenvalues of the linearized mapping at the fixed points are complex numbers, and the network counts up the a's by spiralling inwards towards a fixed point and counts down the b's by spiralling outwards.



Hidden unit dynamics for $a^n b^n c^n$

An SRN with three inputs, three hidden units and three outputs can be used to learn the language $a^n b^n c^n$ which is classified as Mildly Context Sensitive (Chalup, 2003). The network counts down in one direction while simultaneously counting up in another direction, thus producing a star-shaped pattern.



This analysis of network dynamics with simple languages is illuminating. But, in the end, SRNs are limited in the range of dependencies they are able to learn, for reasons analogous to the vanishing gradient problem which we discussed in Week 2 in the context of feedforward networks. In the next section we will introduce Long Short Term Memory, which is able to learn longer range dependencies and — in combination with word vectors, attention and other enhancements — can perform serious real world tasks such as multi-lingual translation.

Optional video

References

Wiles, J., & Elman, J. (1995). Learning to count without a counter: A case study of dynamics and activation landscapes in recurrent networks. In *Proceedings of the Seventeenth Annual Conference of the Cognitive Science Society,* 482-487.

Blair, A.D., & Pollack, J. B. (1997). Analysis of dynamical recognizers. *Neural Computation*, *9*(5), 1127–1142.

Bodén, M., & Blair, A. (2003). Learning the dynamics of embedded clauses. *Applied Intelligence*, *19*(1), 51–63.

Chalup, S.K., & Blair, A.D. (2003). Incremental training of first order recurrent neural networks to predict a context-sensitive language. *Neural Networks*, *16*(7), 955–972.