Getting the Point Across: The Effect of Recurrent Network Biases on the Evolution of a Simple Language

Bradley Tonkes* Alan Blair*

Janet Wiles*†

*Department of Computer Science and Electrical Engineering

†School of Psychology

University of Queensland, 4072

Queensland, Australia

May 19, 1999

Running Heading: "Getting the Point Across".

Keywords: Recurrent Neural Networks, Evolution of Communication

Correspondence should be addressed to:

Bradley Tonkes

Dept of Computer Science and Electrical Engineering

University of Queensland, 4072

Queensland, Australia

Phone: +61 7 3365 1157

email: btonkes@csee.uq.edu.au

Abstract

Although language is an ostensibly symbolic medium, it is an open question whether the underlying concepts communicated through language are also symbolic, or whether they are ultimately more continuous and sub-symbolic in nature. Does language translate, in a sense, between a continuous concept space and a discrete, symbolic message space? If so, how could such a system have evolved? The languages which ultimately emerge will likely be influenced by the constraints and biases of the communicating agents.

We present a framework for exploring these issues, in which two recurrent neural networks – playing the roles of sender and receiver – are able to develop their own language for communicating a set of concepts. Each concept, represented by a point in Euclidean space, is translated into a symbol sequence by an encoder network. This sequence is then serially transmitted to a decoder network which attempts to translate it back to the original concept.

A series of simulations demonstrates that, within this framework, the biases of the sender are in conflict with those of the receiver, and the resulting languages represent a compromise between these competing interests.

1 Introduction

Human languages provide a system for communicating sophisticated concepts that is unparalleled in the biological world. It is a puzzling question how these languages emerged and why they exhibit such a perplexing blend of regularities and exceptions. The types of languages that are likely to emerge among a group of communicating agents will conceivably depend upon, and reflect, the computational constraints and biases inherent in those agents. The emergence of human languages, for example, has presumably been influenced by the computational constraints and biases of the human brain. Furthermore, to remain viable a language must also be learnable by subsequent generations. Those languages that match the innate learning biases of their users will have a distinct advantage over those that do not (Kirby, 1998a).

The relationship between language classes and computational constraints was explored by Chomsky (1959) who proposed a hierarchy of formal languages and corresponding symbolic automata. While the inherently symbolic systems of the Chomsky hierarchy have proven useful for describing certain aspects of human languages, it has more recently been suggested that dynamical systems may provide a more complete model (Pollack, 1987; Elman, 1991; for a collection of articles on the issue, see Port and van Gelder, 1995). Results suggest that such systems may correspond to a different, though related, set of language classes, as compared to their symbolic counterparts (Moore, 1998). A better understanding of these relationships may provide important insights into the underlying mechanisms governing human languages.

One class of dynamical system that provides a promising model of language processing is the class of recurrent neural networks (RNNs). Part of their appeal is the ability to incorporate syntax and semantics into a single encompassing model (Elman, 1991). They have also demonstrated competence in learning a wide range of grammatical structures, for example from an introductory linguistics text (Lawrence et al., 1998). Furthermore, they reflect human performance on a number of language tasks (Weckerly and Elman, 1992; Christiansen and Chater, 1998), and can account for historical descriptions of language change (Hare and Elman, 1995). As well as processing constraints, RNNs have *learning* constraints. That is, they are constrained not only in what they can represent, but in what they can learn. Returning to our original question, we consider how the emergence of a language between communicating RNNs will be affected by their representational and learning biases.

We note that communication is essentially a shared task between sender and receiver, in which the kinds of languages favoured by the sender may not be convenient for the receiver and vice-versa. That is, the biases of the sender and receiver may be different. The language that ultimately emerges may arise as a compromise between these competing interests. We also note that whereas language seems to be temporal and highly symbolic, dynamical representations are typically spatial and continuous. An important task for recurrent networks is thus how to communicate a spatial representation over a symbolic channel. In our scenario, the sender is required to map a spatial concept into a symbolic time-series and the receiver is then required to map the resulting sequence back into a continuous space.

We consider a simple language task in which two recurrent neural networks try to communicate a semantic "concept" represented by a point in a subset $U \subset \mathbb{R}^n$ of Euclidean space. One network acts as an *encoder* and is presented with points $x_i \in U$ selected according to some probability distribution p. The encoder outputs a sequence s_i of symbols taken from some alphabet Σ which is serially transmitted across a channel to a *decoder* network. The decoder network receives the sequence as input, and outputs $y_i \in U$. If the communication is successful, then y_i should approximate x_i. The set of transmitted sequences $S = \bigcup_i \{s_i\} \subseteq \Sigma^*$ forms a *language* or *code* for communicating U (see Figure 1). We consider the special case where U is the unit interval $[0, 1] \subset \mathbb{R}$, and Σ is an alphabet of two symbols. For clarity, we take these symbols to be '0' and '1' although in principle they are arbitrary.

The framework considered here varies from other studies of the emergence of communication which have typically examined communication between symbolic agents over a symbolic channel (Steels, 1997a; **?**; Kirby, 1998b) or communication of a single concept with a single symbol, or set of symbols in parallel (Oliphant, 1998; Di Paolo, 1998; Cangelosi and Parisi, 1998; Oliphant and Batali, 1996).[1] An approach similar to ours has been taken by Batali (1998) who also used recurrent networks as communicative agents. However, whereas our *concepts* are represented as points *within* the unit hypercube (a continuous space), his *concepts* are points on the corners of the unit hypercube (a discrete space). Furthermore, Batali makes no distinction between sender and receiver, while in our simulations this distinction is critical.

Before beginning to train encoders and decoders together (sections 3 and

[1] For a more extensive review of approaches to the evolution of communication, see (Steels, 1997b).

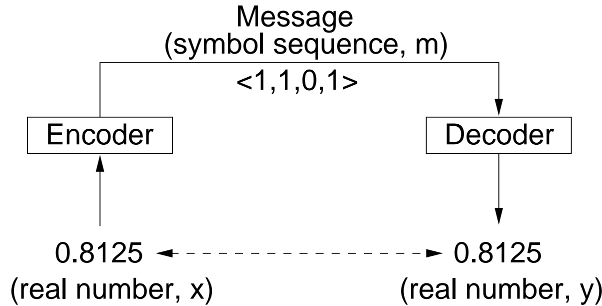

Figure 1: **Getting the point across.** Two recurrent networks are used as encoder and decoder for a communication channel. The encoder is presented with a real number, $x \in [0, 1] \subset \mathbb{R}$ and outputs a sequence of bits, $m \in \Sigma^*, \Sigma = \{‘0’, ‘1’\}$. This sequence of symbols is then used as input for the decoder, which outputs a value $y \in [0, 1] \subset \mathbb{R}$ after all symbols in the sequence have been processed. If the communication is successful, then y approximates x. If a numeric encoding is used then the input value 0.8125 would be mapped to the string $\langle 1, 1, 0, 1\rangle$, since $0.8125_{10} = 0.1101_2$. The decoder, upon receiving the sequence, would output 0.8125.

4), we conducted some preliminary experiments (described in section 2) to see whether encoders and decoders, trained separately, could learn their respective tasks in isolation. In order to do so, it is necessary to choose a benchmark code. It is possible to accomplish the task using a *numeric* encoding, typically recognised as the “standard” binary representation (where the sequence of transmitted symbols corresponds to its base-two representation). Within our framework there are two obvious ways in which such a sequence can be transmitted — either the most significant bits (MSB) of the message can be sent first, or the least significant bits (LSB) can be sent first.

In section 3, the encoder and decoder are *co-evolved* together and are at liberty to determine their own code, with the constraint that encoders can produce only codes of an externally prescribed, fixed length. In contrast, section 4 describes simulations with encoders that are free to control the number of symbols they send. We conclude with some remarks regarding the structure of human languages.

2 Encoders and Decoders

In the first series of simulations we investigate the ability of the individual encoders and decoders to perform their respective mappings in isolation. In total, four mappings are considered.

1. Encoding a real value as a MSB first numeric sequence.
2. Encoding a real value as a LSB first numeric sequence.
3. Decoding from an MSB first numeric sequence to a real value.
4. Decoding from an LSB first numeric sequence to a real value.

2.1 Encoders

For the encoder, the architecture we use is a simple recurrent network, but with additional connections from the output units to the hidden units.[2] A single binary-threshold output unit codes '0' when off and '1' when on. The networks are presented with the appropriate input value at the first time-step, and an input of 0 for subsequent time-steps. Input values are chosen in accordance with a numeric binary encoding: if the networks are given k time-steps in which to encode a value, then the inputs are chosen to be those values that can be encoded with k bits (i.e. $\{n.2^{-k}\}, 0 \leq n < 2^k$) which we will refer to as the k-bit values or *k-bit precision*. The sequence of outputs at each time-step from the network is taken to be the encoding.

Given this general architecture, it is relatively straightforward to hand-code a network with a single hidden unit to perform an encoding for a numeric MSB first sequence. A linear-threshold activation function, as in equation (1), is used for the hidden unit.

$$act(x) = \begin{cases} 1, \text{ if } x \geq 1 \\ -1, \text{ if } x \leq -1 \\ x, \text{ otherwise} \end{cases} \quad (1)$$

Such a network is shown in Figure 2. By contrast, a network that performs the LSB first encoding requires a large number of hidden units. For any value encoded with this scheme, the first output symbol is different to that of neighbouring values.[3] This encoding creates a fractal structure on the space

[2] Essentially a combination Jordan/Elman network.

[3] For example, with 4-bit precision the first symbol of the encoding for $\frac{3}{16}$ is '1', whereas for $\frac{2}{16}$ and $\frac{4}{16}$ the first symbol is '0'.

which is difficult to process. The fractal nature of the numeric encoding can be seen from a graphical depiction. Figure 3 shows a 5-bit numeric code, with the range of values varying along the x-axis, and the resulting code on the y-axis, with white areas representing '0' and black areas representing '1'.

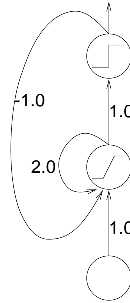

Figure 2: **MSB First Encoder.** A recurrent network that takes a real number in the unit interval $[0, 1]$ and encodes it to the numeric code, MSB first. The hidden unit uses a linear threshold activation function that saturates at -1 and 1, and the output unit is a binary (0.5) threshold unit. The input value is presented at the first time-step only, subsequent inputs are 0. The network can encode values of arbitrary precision if allowed to produce a sufficiently long output sequence.

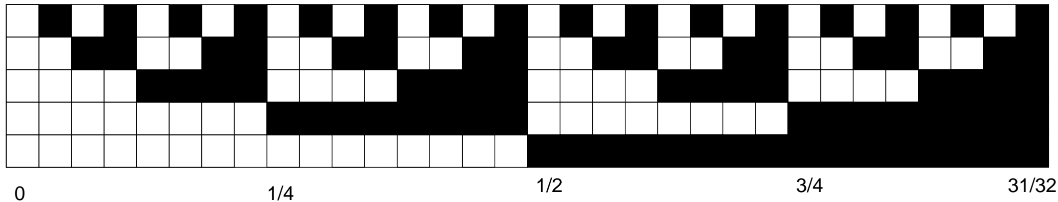

Figure 3: Representation of 5-bit numeric encoding. The MSB encoding may be read off bottom-to-top, the LSB encoding top-to-bottom. For example, $\frac{1}{2}$ is encoded as $\langle 1, 0, 0, 0, 0\rangle$ (MSB first) and $\langle 0, 0, 0, 0, 1\rangle$ (LSB first).

Although a solution could be hand-coded (at least in the MSB case), it was unknown whether a solution could be learned. A series of simulations was designed to test whether the MSB first or LSB first encoding could be found. Given the relative ease in hand-coding solutions for the two mappings, we expected that the LSB first encoding would be harder to evolve, if it could at all, than the MSB first encoding. Networks were evolved using

a simple hill-climbing algorithm to perform the LSB and MSB mappings.

A "champion" decoder was created with initially random weights distributed uniformly between -0.1 and 0.1. A single mutant was then spawned by randomly perturbing the weights of the champion according to a Gaussian distribution with 0 mean and initially 0.005 standard deviation. If the mutant was able to encode values as well as, or better than the champion, then the mutant became champion and a new mutant was spawned. To evaluate the accuracy with which values were encoded, a perfect numeric encoder was assumed, and the sum-squared error between encoder input and decoder output was calculated.

The values chosen to be encoded were selected by taking the "staged learning" approach that has proven successful in similar domains (Elman, 1993). Initially, only two values, 0 and $\frac{1}{2}$, were presented, requiring the single-symbol encodings of $\langle 0 \rangle$ and $\langle 1 \rangle$ respectively for both the MSB and LSB first encodings. Once a network was able to perform this mapping, $0, \frac{1}{4}, \frac{1}{2}$ and $\frac{3}{4}$ were encoded into 2-symbol sequences: $\langle 0,0 \rangle, \langle 0,1 \rangle, \langle 1,0 \rangle$ and $\langle 1,1 \rangle$ respectively in the MSB first case, and $\langle 0,0 \rangle, \langle 1,0 \rangle, \langle 0,1 \rangle$ and $\langle 1,1 \rangle$ respectively in the LSB first case. In general, once 2^k values could be successfully encoded into k-symbol sequences, networks were given 2^{k+1} values to encode into $(k+1)$-symbol sequences. The activations of the networks were reset to 0 before each value was encoded.

The standard deviation (σ) with which mutants were generated was modulated throughout the course of the simulations. Since the encoders use a binary threshold output, a small change in the weights may have no effect on the output of the networks. Consequently, whenever the mutant and champion encoded the input-set equally well, σ was increased by 0.1%.[4] Furthermore, whenever a mutant lost to the champion σ was mutated with 1% Gaussian noise, up to a maximum of 0.1. No change was made to σ when the mutant won.

Simulations were run for a maximum of 100K generations, or until all 32 5-bit values could be encoded. Simulations were performed using networks with 1, 2, 3 and 5 hidden units. Fifty networks were evolved in each condition for both LSB and MSB first encodings.

2.2 Decoders

Simple recurrent networks (Elman, 1990) were used for the decoders. The task for the decoders was the inverse of the encoders' task with minor vari-

[4]Note that in the case of equally-good encodings, the mutant is declared the winner so that the space around a particular solution is better explored.

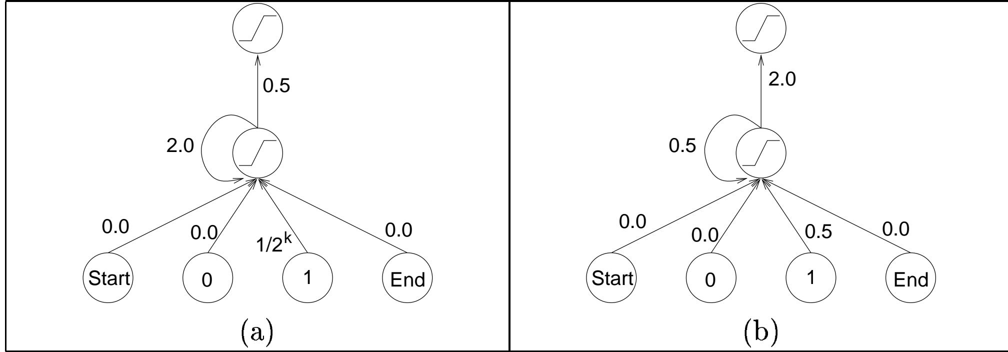

Figure 4: Simple recurrent networks that decode numeric sequences (a) MSB first and (b) LSB first. The input to the network is wrapped with start and end markers. After presentation of the end marker, the output unit activation corresponds to the appropriate value. Linear (0,1) threshold activation functions are used for hidden and output units on both networks. Whereas the LSB first decoder (b) is able to decode sequences of arbitrary length, the MSB first decoder (a) can only decode sequences of length k.

ations. Each sequence presented to a decoder was enclosed by start-of-sequence and end-of-sequence inputs. For example, an encoder output of $\langle 0, 1, 0, 1 \rangle$ would be received by the decoder as $\langle \$, 0, 1, 0, 1, \# \rangle$, where '$' and '#' are start and end symbols. These additional symbols become important later when we consider variable-length encodings (Section 4). The additional inputs did not appear to affect the simulations presented in this section.

Unlike the encoder, the decoder is capable of decoding either MSB or LSB first, though with some important differences. Figure 4 shows hand-coded simple recurrent networks that decode (a) MSB first and (b) LSB first. Although an LSB decoder is able to decode sequences of varying lengths with only a single hidden unit, an MSB decoder can only decode strings of a known length with a single hidden unit. Unlike the LSB decoder, the solution for k-bit precision does not generalise to $(k + 1)$-bit precision for the MSB decoder.

Simulations were performed with a hill-climbing algorithm similar to that used for the encoder. A perfect encoder was used to encode values to MSB or LSB first numeric sequences. Start and end markers were added to each of the resulting sequences and used as input to the decoder. The decoders were evolved to output the appropriate value after receiving the end-of-sequence marker, '#'. Activations of the decoder were reset to 0 before presentation

of each input sequence. Decoders were compared by the sum-squared error across all presented strings. In much the same manner as the encoders, σ was modulated during the course of a run. The same principle of staged learning was applied: whenever decoders could successfully decode k-bit values, precision was incremented to $k+1$ bits. Decoders were judged to be "correct" when the decoded value was closer to the relevant encoded value than any other value in the input set (i.e. within $\frac{1}{2^{k+1}}$ of the desired value for k-bit precision).

As for the encoding task, simulations were performed for networks with 1, 2, 3 and 5 hidden units using both LSB and MSB codes. Again, fifty networks were trialed in each condition for a maximum of 100K generations or until all 5-bit values could be decoded successfully.

2.3 Encoders and Decoders: Results

The results of these simulations (shown in Table I) indicate that there is a significant difference in the sequence order preferred by the encoders and decoders. The encoders were only able to learn the MSB first encoding, whereas the decoders were more successful at learning LSB first sequences. For the encoders in the LSB case, no network was ever able to encode more than 2-bit values, whereas in the MSB case, networks of all sizes were able to encode 5-bit values, the most successful of which were the networks with 2 hidden units.

Of the decoders presented with strings LSB first, 30 of 50 were able to perform the task for 5-symbol input sequences (3 hidden units). No MSB decoders were successful on 5-symbol sequences, although with 5 hidden units one network out of the 50 was able to process 4-symbol sequences and 27 could process 3-symbol sequences.

2.4 Backpropagation Through Time

Since hill-climbing is often a relatively slow process we performed additional simulations using backpropagation through time (BPTT) (Williams and Zipser, 1989) on the decoder. In the combined system there is no sensible way to train the encoder with this algorithm since the language is not fixed and there are consequently no target outputs. However, if we assume that the decoder "understands" what the encoder is communicating[5] then the decoder may be trained towards a target in a sensible fashion.

[5] A common assumption for many studies of language learning.

	Hidden Units	Number at 5-bit Precision		Average Precision	
		MSB First	LSB First	MSB First	LSB First
Encoders	1	11	0	2.00	0.98
	2	18	0	2.84	1.14
	3	11	0	3.00	1.28
	5	7	0	3.20	1.56
Decoders	1	0	22	1.22	2.42
	2	0	26	1.62	3.28
	3	0	30	2.02	4.14
	5	0	22	2.39	3.82

Table I: Performance of encoders and decoders on their respective tasks both MSB and LSB first and with varying numbers of hidden units. Shown are the number of networks, out of the fifty trialed, that were able to encode or decode 5-bit values within 100K generations, and the average precision finally obtained.

Backpropagation works best with an activation function with non-zero gradient everywhere, so we replaced the linear threshold activation functions used for the hill-climbers with equation (2) which is linear in the interval $[-1, 1]$ and has non-zero gradient everywhere.

$$act(x) = \begin{cases} 2.0 - \frac{1}{x}, \text{ if } x > \quad 1.0 \\ -2.0 - \frac{1}{x}, \text{ if } x < -1.0 \\ x, \text{ otherwise} \end{cases} \tag{2}$$

For training, networks were unfolded for up to 5 time-steps, but no further than the start of a message. Networks were only given a target value on presentation of the end-of-sequence symbol, '#'; no errors were propagated as a result of intermediate outputs.

Simulations were performed for both MSB and LSB first encodings in much the same manner as the simulations in section 2.2. The networks' performance was tested after each epoch and precision was incremented accordingly. Decoders were trained for a maximum of 10K epochs with a learning rate of 0.01 and a momentum value of 0.9.

2.4.1 BPTT: Results

A similar set of results were obtained for the decoders trained with BPTT as were obtained using the hill-climbing algorithm. Table II documents the

success of the decoders in each condition. Decoders of all sizes learned to decode values of 5-bit precision when presented with LSB first sequences. No decoders attained this level of precision when presented with MSB first sequences, though networks with more than a single hidden unit could decode 4-bit values.

Hidden Units	Number at 5-bit Precision		Average Precision	
	MSB First	LSB First	MSB First	LSB First
1	0	24	1.08	2.92
2	0	35	1.80	3.88
3	0	20	2.06	2.64
5	0	7	0.86	1.36

Table II: Performance of decoders trained with BPTT on both MSB and LSB first tasks and with varying numbers of hidden units. Shown are the number of networks, out of the fifty trialed, that were able to decode 5-bit values within 10K epochs, and the average precision finally obtained (*cf* Table I).

Again, the performance of the MSB first decoders exceeded that of the LSB first decoders. The performance of the larger networks was disappointing, with many failing to decode values of even 1-bit precision and many others failing at 2-bit precision. The BPTT-trained decoders were often able to find good solutions more quickly than the hill-climbers, in terms of both the number of weight updates and simulation time[6]. Indeed, in the LSB first condition, many networks were able to decode 5-bit values after as few as 200 epochs.

3 Combined System: Forwards and Reversed

Having established that both encoders and decoders are capable of learning a pre-determined code is isolation, we turn to the question of whether they are able to develop their own code within a co-evolutionary framework (Figure 1). The results of section 2 show that the encoder and decoder have quite different biases in term of the codes that are easier to learn. More specifically, we are led to the conjecture that, if a code is among those preferred by the encoder, then the *reverse* of that code is likely to be preferred by the decoder. In order to test this conjecture we conducted experiments under two different

[6]BPTT has far greater computational complexity per weight update than hill-climbing.

conditions: the *forwards* condition, in which the symbols produced by the encoder are passed on in the same order to the decoder, and the *reversed* condition, in which these symbols are effectively held on a stack, and then fed to the decoder in reverse order.

Preliminary experiments in which the hill-climbing algorithms of sections 2.1 and 2.2 were used for both the encoder and decoder proved to be unsuccessful. Consequently, we evolved the combined system by using the hill-climbing algorithm on the encoder and BPTT on the decoder. Weights of the encoder and decoder were not changed synchronously. Rather, a mutant encoder was spawned and a decoder was then trained for many epochs on the output of the encoder. This approach encourages the output of the encoder towards a code that is learnable by the decoder.

Encoders were also screened on the basis of their *variability* (the number of different encodings they produce for their set of inputs). If the mutant does not demonstrate greater variability than the champion, it is discarded without training a decoder. That is, there is an artificially introduced selective pressure for variability in the encoders (they are required to "babble").[7] Note however, that encoders with more variable codes will still only survive if the code they produce is learnable by the decoder. The complete algorithm is outlined in Figure 5.

One further aspect of the simulations warrants attention. For the individual encoders and decoders, only the minimum number of symbols were sent: when 2^k values were being encoded, k symbols were sent. For the combined system this condition was relaxed to permit more than the strict minimum number of symbols to be sent, thus allowing codes that achieve less than optimal efficiency. For the reversed systems, 2^k values were encoded into $k+2$ symbols, and for the forwards systems, 2^k values were encoded into $2k$ symbols. Greater bandwidth was given to the forwards system since, unlike the reversed system, we do not expect it to develop a code as compact as the numeric code.

Fifty systems were evolved in each condition for a maximum of 100 generations[8] or until all 5-bit values could be successfully communicated. Decoders were trained for $750k$ epochs, k being the precision of the communicated values. Decoders were trained using BPTT with a learning rate of 0.01 and a momentum of 0.9. Both encoder and decoder had 2 hidden units.

[7] When this condition was weakened to allow mutants to be of equal variability but producing a different encoding, evolution was typically unsuccessful.

[8] One generation being the selection of a mutant encoder and the subsequent training of a decoder.

1. Create a champion encoder with random weights distributed uniformly between -0.1 and 0.1 and a decoder with random weights distributed uniformly between -1.0 and 1.0.
2. Set σ to 0.01.
3. Create a mutant encoder by perturbing the weights of the champion with Gaussian noise of standard deviation σ. If σ exceeds 1.0, reset it to 0.01.
4. If the variability of the mutant is less than that of the champion, increase σ by 1%, and return to step 3.
5. Create a mutant decoder with weights initialised randomly between -1.0 and 1.0.
6. For n iterations, present all inputs of the current precision to the encoder. Train the decoder on the output of the encoder.
7. If the final sum-squared error of the mutant encoder and decoder across all strings is lower than that of the champions, make the mutants the champions. Furthermore, if the mutants correctly communicate all values then increase the precision. Return to step 2.

Figure 5: Evolutionary algorithm for combined encoder/decoder system.

3.1 Forwards and Reversed: Results

Simulations produced systems capable of communicating values of varying levels of precision (see Table III). Clearly, the evolution of the system is more successful when the communicated sequence is reversed. This result is not unexpected since the natural biases of the encoders and decoders push the system towards a solution in the *reversed* case, whereas the path to a successful solution is less clear in the *forwards* case.

3.2 Analysis of Codes

In all cases, systems trained with the *reversed* channel produced codes similar in nature to the MSB first numeric code of section 2. One code, for 5-bit precision, is shown graphically in Figure 6 in the same manner in which

Final Precision Reached	Forwards ($2k$)	Reversed ($k+2$)
3	41	1
4	9	30
5	0	19
Total	50	50

Table III: The level of precision obtained by systems in both the forwards and reversed conditions. The forwards systems, sending $2k$ symbols, attained an average precision of 3.18 bits, whereas the reversed systems, sending $k+2$ symbols, averaged 4.36-bit precision. Note that whereas tables I and II show how many networks attained 5-bit precision for varying numbers of hidden units, this table and table 5 document the final precision attained by each system, having 2 hidden units for both encoder and decoder.

the numeric code is shown in Figure 3. Shown graphically, there are some similarities apparent between the evolved code and the numeric code. The nature of the evolved code is also highlighted by the frequency with which symbols alternate in various positions. Over the range of inputs, the first symbol (shown as the bottom row of Figure 6) alternates only once: the first symbol for encoding the smallest seven inputs is a '0', and for the largest nine inputs a '1'. For the second symbol four such sequences are apparent, with an increasing number for subsequent symbols. The code shows a clear significance from first bit to last, as would be expected from a numeric code.

Many of the evolved codes did not share such obvious similarities with the numeric code as the one in Figure 6. Table IV shows a typical code communicated by a reversed system for 4-bit values. At first glance, this code appears to be unrelated to a numeric code. However, if we negate the first, third and fifth symbols in the sequence then the numeric ordering of the messages is the same as the inputs. This type of transformation was common to many codes but was by no means universal, others already being in numeric order (or reverse numeric order). The phenomenon may be attributed to negative recurrent weights that reverse the "meaning" of alternate symbols.

To provide a more direct comparison between the codes from the forwards and reversed systems, an additional 50 reversed systems were evolved allowing $2k$ symbols to be sent by the encoder, as used in the forwards case. Of these 50 systems 46 attained at least precision 4, the maximum attained by the forwards systems. Additional systems were also trained in the forwards condition, to bring the total number of forwards systems at

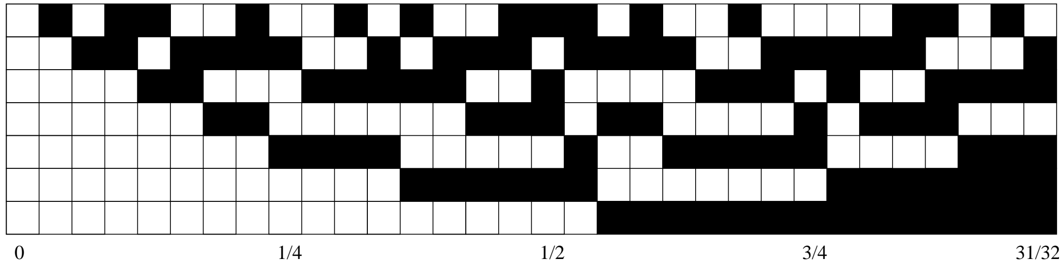

Figure 6: Graphical depiction of a code for communicating 5-bit values produced by a system from the *reversed* condition. The first output symbol from the encoder (and consequently the last input symbol to the decoder) is shown as the bottom row. Note the similarities to the numeric binary code shown in Figure 3.

4-bit precision to 14. The 14 forwards and 46 reversed 4-bit codes form the basis for comparing how the different biases affect the code.

A cursory inspection of the two sets of codes did not suggest any obvious structural differences, both resembling the numeric code to an extent. What we expected to observe in the forwards codes was a bias to balance the information more evenly throughout a sequence, to cater for the biases of both the encoder and decoder. To find how information was distributed throughout the codes we considered the optimal squared-error (OSE) that a decoder could possibly obtain if it only saw the first n bits of the code, for $0 \leq n < 2^k$. If the first n bits of each message are sufficient to identify it uniquely, then each point can be precisely determined and the OSE of the code is 0. If two or more messages share the same initial n bits, then the optimal decoder outputs their average value, and the OSE can be calculated accordingly.

For an MSB first numeric code, OSE drops very quickly as n increases since the most significant bits are at the beginning of the sequence. Conversely, for an LSB first numeric encoding, OSE decreases slowly for the first values, then very rapidly for later ones. Figure 7 shows these curves for compact 4-bit MSB first and LSB first numeric encodings.

For the evolved codes, two scenarios are considered: the OSE of the code when read from the first bit sent by the encoder to the last bit sent (i.e. from the perspective of the encoder); and from the last bit sent by the encoder to the first bit sent (as would be seen by a decoder in the reversed condition). The resulting pair of curves is effectively the same as for the graph in Figure 7 since an LSB code is simply an MSB code viewed in reverse. Note that while

Input	Message	Alternately Negated	Output
0.0000	100111	001101	0.0029
0.0625	100100	001110	0.0420
0.1250	111001	010011	0.1211
0.1875	111111	010101	0.1943
0.2500	110010	011000	0.2738
0.3125	110011	011001	0.3136
0.3750	110000	011010	0.3608
0.4375	001001	100011	0.4424
0.5000	001111	100101	0.4945
0.5625	001100	100110	0.5412
0.6250	000011	101001	0.6105
0.6875	000000	101010	0.6577
0.7500	000001	101011	0.7272
0.8125	000111	101101	0.8002
0.8750	011000	110010	0.8488
0.9375	011001	110011	0.9184

Table IV: Language for a *reversed* system at 4-bit precision. Negating alternate symbols of the message (symbols 1, 3 and 5) shown in the third column transforms the code into a numeric-order code. The symbol-swapping behaviour is a consequence of having negative recurrent weights that oscillate the interpretation of successive symbols.

the numeric code curves are symmetric, this observation does not necessarily follow for the evolved codes which are longer than optimal. (Consider a code formed by the concatenation of an MSB first code and an LSB first code, resulting in perfect information in the first half of the code when viewed in either direction.) Figure 8 shows the average OSE for both the forwards and reversed systems, when looking at the codes first-to-last and last-to-first.

Comparing the pairs of first-to-last/last-to-first curves in Figure 8 with those of Figure 7 reveals some of the differences between the codes of the forwards and reversed systems. The MSB first numeric code has the strongest possible ordering of significance from first bit to last, and the area between the MSB first and LSB first curves is consequently large. In Figure 8 the area between the curves of the reversed systems is larger than that for the forwards systems, suggesting that the reversed systems tend to develop codes with stronger ordering of significance than the forwards systems, as predicted. Calculating the area between the curves for each code in both

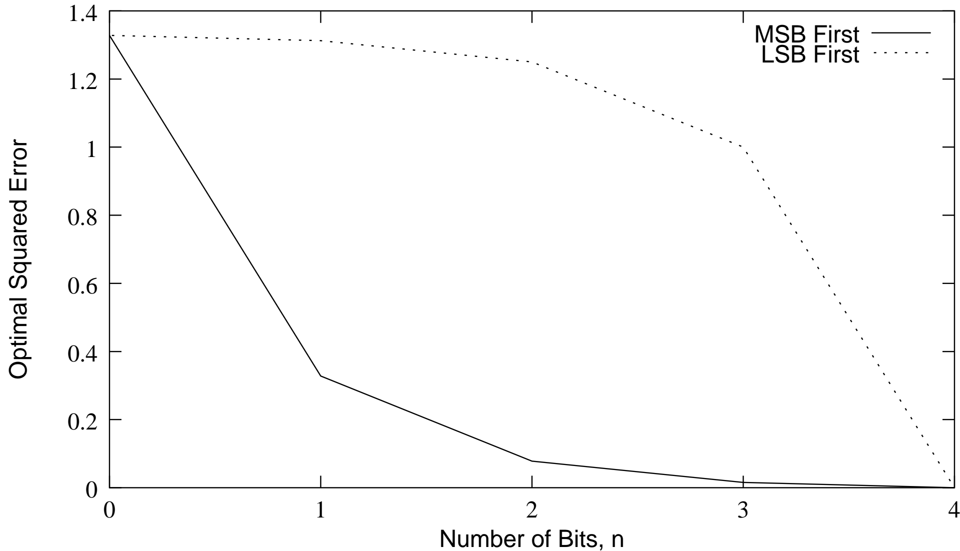

Figure 7: Least squared error obtainable by an optimal decoder after seeing only the first n bits of MSB first and LSB first numeric codes.

the forwards and reversed systems, and comparing the two groups using a Mann-Whitney U test[9], reveals a statistically significant difference in the two populations ($p < 0.05$). The graph demonstrates that the effect of the opposing biases in the forwards system is to distribute information more evenly throughout the sequence.

4 Variable Length

In the simulations presented thus far, the encoders have generated an externally specified number of symbols. In the following series, encoder networks have an additional output unit that can be used to vary the length of sequence produced. So long as this additional output remains on, symbols are sent in the same manner as the previous simulations. When this output unit turns off, or some maximum number of symbols is reached, an "end-of-sequence" (EOS) symbol is added to the message and communication ceases. The additional output unit may be considered as the "push-to-talk" unit.[10]

[9] The Mann-Whitney U test is a non-parametric version of the t test.

[10] Whether or not this unit feeds back to the hidden layer is largely inconsequential since its output remains constant throughout the encoding of a message, thus acting as an additional bias input.

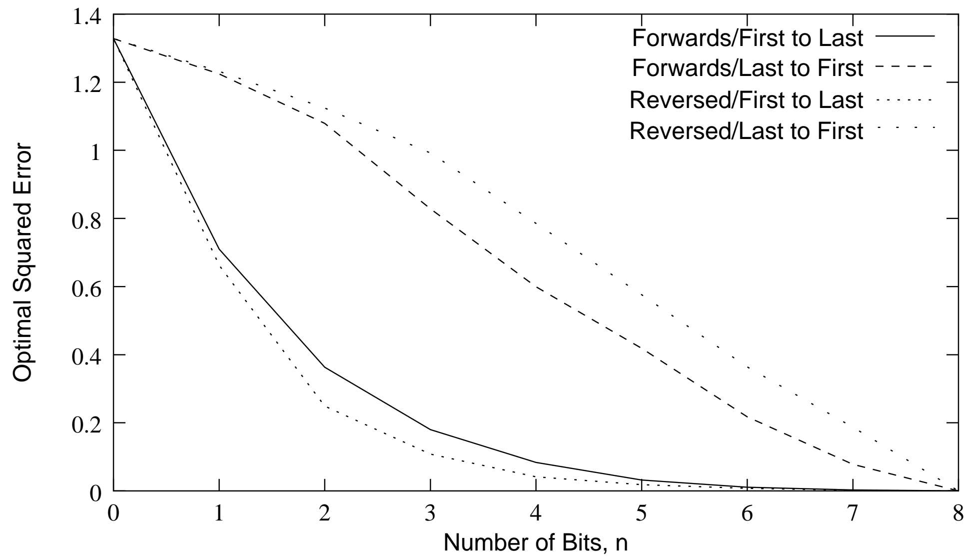

Figure 8: Least squared error obtainable by an optimal decoder after seeing the first n bits of codes developed by the forwards and reversed systems when read both first-to-last and last-to-first. The two lower curves are when the codes are read first-to-last, and the inner curves belong to the forwards system.

This change allows encoders to send the same sequences when the number of values to be communicated increases. It also makes success less likely for the type of MSB first decoder in Figure 4(a) which can only process sequences of known length.

Simulations were performed in much the same manner as Figure 5, with some minor changes regarding EOS. In both the previous and following series of simulations, encoders were generated to increase the number of different codes produced. In the following simulations, once an encoder attains maximum variability, subsequent encoders are generated to maintain variability in the codes, but also to increase the number of EOS symbols produced (i.e. to increase the number of strings that are properly terminated). Additionally, a small error ($2^{-2(k+1)}$ at k-bit precision) is added to the entire system for each string that is successfully communicated but improperly terminated, to differentiate between systems with different numbers of properly terminated sequences, yet similar error.

As before, 50 forwards and reversed systems were evolved to a maximum of 5-bit precision. Systems in the forwards condition were allowed to send a maximum of $2k$ symbols (plus the EOS symbol), and in the reversed system, the maximum was set to $k+2$ symbols. As for the fixed-length simulations, an additional 50 reversed systems were evolved using a maximum of $2k$ symbols.

4.1 Results

The performance of the systems in this series of simulations was significantly worse than in the fixed-length case. None of the reversed systems with a maximum message size of $k+2$ symbols attained 5-bit precision.[11] Results are summarised in Table V. Again, the systems that reversed the encoders' output outperformed those in which the messages were transmitted normally.

4.2 Analysis of Codes

As in the fixed-length case, both the forwards and reversed systems produced codes that could be placed in numeric order. The introduction of the EOS symbol created one novel difference: to be placed in numeric order the EOS often had to be assigned a value, indicating that it served as more than a syntax marker. A typical code found by a reversed $(k+2)$ system is shown in Table VI. This table also shows the code from the only forwards system that

[11] Although one of two preliminary simulations did reach 5-bit precision.

Final Precision Reached	Forwards ($2k$)	Reversed ($k+2$)	Reversed ($2k$)
2	6	4	1
3	43	27	27
4	1	19	21
5	0	0	1
Total	50	50	50

Table V: The level of precision obtained by systems in both the forwards and reversed conditions. The forwards systems, sending $2k$ symbols, attained an average precision of 2.90 bits. The reversed systems attained, an average precision of 3.30 bits when transmitting $k+2$ symbols and 3.44 bits when transmitting $2k$ symbols.

attained 4-bit precision, which is clearly of a different nature to the reversed system's code. While some reversed systems produced codes of a similar nature to this forwards system's code, none were as regular. Codes from a reversed ($2k$) system and the best forwards system are depicted graphically in figure 9.

Calculating the average OSE for the two sets of codes as we did in the fixed length case, shows a difference between the codes of the two conditions. Figure 10 shows the average OSE for the reversed systems and for the single forwards system. Again, the two curves of the forwards system are closest together, indicating that information is distributed evenly throughout the code (this result is not surprising considering the code, shown in Table VI). With only one forwards code it is difficult to draw conclusions, though there does appear to be a significant difference between the two sets of codes.[12]

5 Discussion

The first series of simulations (section 2) demonstrated the differing biases of the encoders and decoders on the numeric encoding task. Whereas the encoder was only able to encode values MSB first, the decoder had a strong preference for decoding values LSB first. The second series of simulations (section 3) showed how these different biases could influence the evolution of a simple language. Systems were evolved in two conditions: the *forwards* condition where messages were sent serially between the encoder and

[12] The two sets of codes from systems at 3-bit precision were also compared using this test, but showed no significant difference.

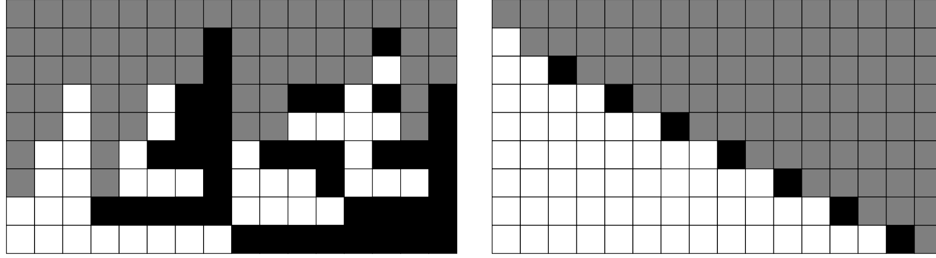

Figure 9: Graphical depiction of variable length codes for communicating values of 4-bit precision from reversed (left) and forwards (right) systems. In this figure, '1' is represented by black regions, '0' by white regions, with the grey areas showing where the encoder stopped producing output. Both codes were evolved using $2k$ symbol channel length, in constrast to the reversed code of Table VI which was evolved with a maximum of $k + 2$ symbols in the communication channel.

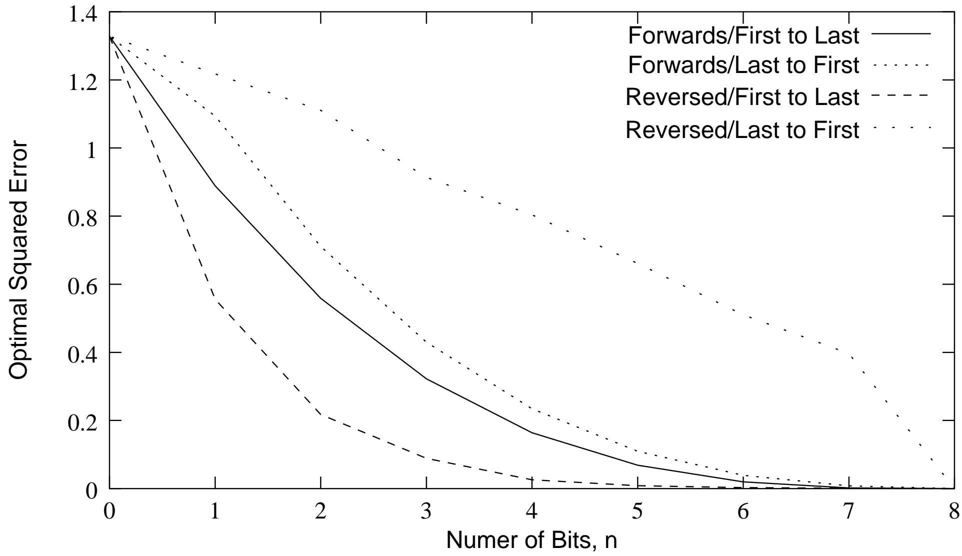

Figure 10: Optimal squared error obtainable from the one forwards system code, and the average obtainable from the reversed system codes at 4-bit precision. As in Figure 8 the inner curves correspond to the forwards system and the lower curves result from calculating OSE first bit to last.

	Reversed ($k+2$)			Forwards ($2k$)	
Input	Message	Numeric	Output	Message	Output
0.0000	000000#	0000002	-0.0037	00000000#	-0.0019
0.0625	0000#	00002	0.0631	0000000#	0.0660
0.1250	000#	0002	0.1251	0000001#	0.1177
0.1875	00#	002	0.2132	000000#	0.1867
0.2500	010000#	0100002	0.2384	000001#	0.2506
0.3125	0#	02	0.3309	00000#	0.3126
0.3750	10000#	100002	0.3829	00001#	0.3790
0.4375	1000#	10002	0.4279	0000#	0.4382
0.5000	100#	1002	0.4909	0001#	0.5047
0.5625	10#	102	0.5790	000#	0.5629
0.6250	11000#	110002	0.6366	001#	0.6294
0.6875	1100#	11002	0.6822	00#	0.6868
0.7500	110#	1102	0.7452	01#	0.7532
0.8125	11100#	111002	0.8219	0#	0.8097
0.8750	1110#	11102	0.8671	1#	0.8646
0.9375	11110#	111102	0.9371	#	0.9380

Table VI: **Left:** Language for a variable length, reversed system at 4-bit precision, evolved with $k+2$ symbol channel length. Interpreting the EOS symbol (#) as a value greater than 1, places the messages in numeric order. **Right:** Language for a variable length forwards system at 4-bit precision, evolved with channel length $2k$.

decoder, and the *reversed* condition where the ordering of symbols within each message was reversed before transmission. In the reversed condition, the biases of the encoder matched those of the decoder, and a code was evolved that resembled the numeric code used to train the individual encoders and decoders in section 2. The codes that emerged were, by design, more sparse than the numeric code and often alternated the "meaning" of successive symbols.

The codes that emerged from systems in which messages sent by the encoder were not reversed (the *forwards* case) differed from their reversed counterparts in subtle ways. Calculating the OSE obtainable after seeing only a limited number of bits showed a significant difference in the way that information was distributed throughout a sequence with the forwards systems showing a preference for a more even distribution of information

throughout each message.

The final series of simulations (section 4) allowed encoders to vary the lengths of sequences they transmitted. This task proved to be considerably harder to learn than the case where the length of each sequence was externally specified, but had the advantage of discouraging the kind of approach taken by the MSB decoder of Figure 4. Consequently, in the forwards case, the increased bias of the decoder appeared to result in a greater compromise between encoder and decoder, yielding the code of Figure 9. Again, comparing the OSE of the codes that emerged from the final systems suggested that there was a stronger pressure to distributed information evenly through each message in the forwards case than in the reversed case.

6 Conclusions

We have introduced a framework for studying language emergence that appears relevant to connectionist language processing. Within the general framework proposed we were able to study the emergence of a language with a considerable degree of success. Although only a restricted case of the general framework was considered, it was sufficient to highlight the observation that communication is a shared task between sender and receiver. The simulations showed a clear distinction between the codes preferred by the encoder (MSB first) and those preferred by the decoder (LSB first). Furthermore, simulations of the combined system showed how these biases can interact to yield a system of communication that is satisfactory for both sender and receiver.

The simulations suggest an intriguing hypothesis on why human languages take their particular elaborate forms. One of the key questions for human languages concerns what class they form. Attempts to characterise this class have revealed that human languages form a class that is not easily described with any of our current formalisms, either symbolic or dynamical. The simulations support the idea that there may be a well-defined class for sending, and a different well-defined class for receiving so that "real" language, which is a compromise lying in the intersection of the two, may end up less easily characterised, and looking considerably stranger. Given this possibility, it is less surprising that attempts to locate language within the Chomsky hierarchy have met with little success. It is not a goal of the Chomsky hierarchy to distinguish transmission from reception. An alternative suggestion is that systems which need to process both as encoders and decoders share resources between the two sub-systems and compromise on

the preprocessing of language, rather than the language itself.

While it is intuitively plausible that the task of transmission may differ from reception, it has not been demonstrated to date the types of constraints that are imposed by each subpart of the task. Consequently, the final observation we make from the simulations is with regard to the biases of the recurrent networks themselves. We expect that a good model of human language processing will have constraints and biases that resemble those of humans. Current results suggest that RNNs reflect human performance on some aspects of language processing. Understanding how the biases of RNNs affect their performance on language tasks should give an indication of why RNNs do not explain the entire range of language phenomena and provide clues as to additional biases that may be necessary for RNNs to provide a more complete story. To this end, the simulations have demonstrated the biases of RNNs on a language task which, despite its simplicity, may underpin more significant aspects of language processing.

In future work we intend to apply this framework to more realistic tasks, requiring more of the syntactic structure of language, but retaining the essential element of encoding from spatial representations to sequential codes and back again.

Acknowledgements

Thanks to Jeff Elman and Tony Plate for helpful discussions. This work was supported by an APA to Brad Tonkes, a UQ Postdoctoral Fellowship to Alan Blair and an ARC grant to Janet Wiles.

References

Batali, J. (1998). Computational simulations of the emergence of grammar. In Hurford, J., Knight, C., and Studdert-Kennedy, M., editors, *Approaches to the Evolution of Language*, pages 405–426. Cambridge University Press, Cambridge.

Cangelosi, A. and Parisi, D. (1998). The emergence of a language in an evolving population of neural networks. *Connection Science*, 10(2):83 – 98.

Chomsky, N. (1959). On certain formal properties of grammars. *Information and Control*, 2(2):113–124.

Christiansen, M. H. and Chater, N. (1998). Toward a connectionist model of recursion in human linguistic performance. *To appear: Cognitive Science.*

Di Paolo, E. A. (1998). An investigation into the evolution of communication. *Adaptive Behavior*, 6(2):285–324.

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

Elman, J. (1991). Distributed representations, simple recurrent networks and grammatical structure. *Machine Learning*, 7:195–224.

Elman, J. (1993). Learning and development in neural networks: The importance of starting small. *Cognition*, 48:71–99.

Hare, M. and Elman, J. (1995). Learning and morphological change. *Cognition*, 56:61–98.

Kirby, S. (1998a). Fitness and the selective adaptation of language. In Hurford, J., Knight, C., and Studdert-Kennedy, M., editors, *Approaches to the Evolution of Language*. Cambridge University Press.

Kirby, S. (1998b). Syntax without natural selection: How compositionality emerges from vocabulary in a population of learners. Under review.

Lawrence, S., Giles, C. L., and Fong, S. (1998). Natural language grammatical inference with recurrent neural networks. To appear: IEEE Transactions on Knowledge and Data Engineering.

Moore, C. (1998). Dynamical recognizers: Real-time language recognition by analog computers. *Theoretical Computer Science*, 201(1–2):99–136.

Oliphant, M. (1998). The learning barrier: Moving from innate to learned systems of communication. To appear: Adaptive Behavior.

Oliphant, M. and Batali, J. (1996). Learning and the emergence of coordinated communication. Submitted.

Pollack, J. B. (1987). *On connectionist models of natural language processing*. PhD thesis, Computer Science Department, University of Illinois, Urbana, IL.

Port, R. and van Gelder, T. (1995). *Mind as Motion: Explorations in the Dynamics of Cognition*. MIT Press.

Steels, L. (1997a). The origins of syntax in visually grounded robotic agents. In Pollack, M., editor, *Proceedings of IJCAI 97*, pages 1632–1641, Los Angeles. Morgan Kauffman.

Steels, L. (1997b). The synthetic modeling of language origins. *Evolution of Communication*, 1(1):1–34.

Weckerly, J. and Elman, J. (1992). A PDP approach to processing center-embedded sentences. In *Proceedings of the Fourteenth Annual Conference of the Cognitive Science Society*, Hillsdale, NJ. Erlbaum.

Williams, R. J. and Zipser, D. (1989). A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1(2):270–280.