A Paradox of Neural Encoders and Decoders or Why Don't We Talk Backwards?*

Bradley Tonkes[1], Alan Blair[1], and Janet Wiles[1,2]

[1] Department of Computer Science and Electrical Engineering
[2] School of Psychology
University of Queensland
QLD 4072 Australia
{btonkes, blair, janetw}@csee.uq.edu.au

Abstract. We develop a new framework for studying the biases that recurrent neural networks bring to language processing tasks. A semantic concept represented by a point in Euclidian space is translated into a symbol sequence by an encoder network. This sequence is then fed to a decoder network which attempts to translate it back to the original concept. We show how a pair of recurrent networks acting as encoder and decoder can develop their own symbolic language that is serially transmitted between them either forwards or backwards. The encoder and decoder bring different constraints to the task, and these early results indicate that the conflicting nature of these constraints may be reflected in the language that ultimately emerges, providing important clues to the structure of human languages.

1 Introduction

The study of automata and the languages they can process has a history dating back to Turing [9] and beyond. Entwined with this story is the study of natural languages and of the human mind. The issue is essentially one of constraints. The particular constraints on an automaton, such as time and space, place bounds on the types of tasks it can perform such as the types of languages it can process. Likewise, it is believed that the constraints of the human mind are reflected in the languages we use, so that by examining the features of language we may better understand the principles that guide human language and thought processes.

Perhaps the best known work relating automata and languages, which also seems highly relevant to natural languages, is Chomsky's hierarchy [1]. Chomsky's hierarchy is a family of language classes that can be recognised by a corresponding family of automata classes. With different restrictions on the automata, different language classes may be processed. However, this hierarchy was designed with symbolic systems in mind. It has been suggested that dynamical

* We thank Tony Plate and Elizabeth Sklar for helpful discussions. The research was supported by an APA to BT, a UQ Postdoctoral Fellowship to AB and an ARC grant to JW.

systems, including many connectionist models, may bring different biases to language processing tasks relative to their symbolic counterparts [8], necessitating a re-evaluation of the automata/language relationship.

As well as processing constraints, connectionist models also have *learning* constraints. That is, they are limited not only in what they can represent, but in what they can learn. The distinction between learning and representation is important when we consider how human languages have developed. For a natural language to be viable, it must not only be representable by its users, but also learnable by subsequent generations [6]. The learning and representational constraints of the human brain dictate the set of languages humans are able to understand and learn, and consequently the languages that have emerged.

Recurrent neural networks (RNNs) have shown significant promise as computational models of various aspects of the human language processing system. Part of their major appeal is the ability to incorporate syntax and semantics into a single encompassing model [3]. They have also demonstrated competence in learning a wide range of grammatical structures [7], and often reflect real-world data on natural language tasks [2, 10], and language change [5]. Although these results have been affected by it, the issue of constraints of RNNs has not been explicitly examined. It seems important then, to investigate the constraints of recurrent networks and the way that they influence the properties and emergence of language.

This paper is motivated by the observation that communication is essentially a shared task between sender and receiver, in which the kind of language favoured by the sender may not be convenient for the receiver and vice-versa. That is, the constraints of the sender and receiver may be different. The language that ultimately emerges may arise as a compromise between these competing interests.

We consider a simple language task in which two RNNs try to communicate a semantic "concept" represented by a point in a subset, $U \subset \mathbb{R}^n$ of Euclidean space . One network sends a *message* in the form of a sequence of bits, which the other network receives and decodes back into a point in the same Euclidean space (Fig. 1). In this paper we consider only the case where U is the unit interval $[0, 1] \subset \mathbb{R}$.

While the task is superficially simplistic, it has a number of interesting properties. The "concept" is specified in a continuous space with arbitrary precision, whereas the "language" is a sequence of symbols from a finite alphabet. Unlike studies that have looked at language emergence between symbolic agents over a symbolic channel, this task requires a transformation from a concept described with arbitrary precision in a continuous space to a symbolic language. A trade-off is required between the amount of precision in the concepts and the length of the symbol sequences in the language.

It is possible to accomplish the task by using a numeric encoding — interpreting the sequence of bits as its numerical (binary) value. For this numeric code, two possibilities are immediately obvious: either the most significant part of the message can be sent first, or it can be sent last. For example, $0.8125_{10} = 0.1101_2$ may be sent most-significant-bit (MSB) first as $<1, 1, 0, 1>$ or least-significant-

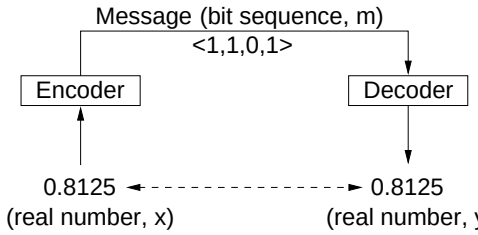

Fig. 1. Getting the point across. Two recurrent networks are used as encoders and decoders for a communication channel. The encoder is presented with a point from a subset of Euclidean space, $x \in U \subset \mathbb{R}^n$, and outputs a sequence of bits, $m \in \Sigma^*, \Sigma = \{0, 1\}$. This sequence of bits is then used as input for the decoder, which outputs a value $y \in \mathbb{R}^n$ after the last bit in the sequence has been processed. If the communication is successful, then y should approximate x. For example, if $U = [0, 1] \subset \mathbb{R}^1$, and a strict numeric encoding is used, the input value 0.8125 would be mapped to the string $<1, 1, 0, 1>$, since $0.8125_{10} = 0.1101_2$. The second network acts as a decoder, receiving a binary string and mapping it to a real value. For example, the input $< 1, 1, 0, 1 >$ would be mapped back to the output 0.8125.

bit (LSB) first as $< 1, 0, 1, 1 >$. This paper investigates the effect that encoder and decoder constraints have on the way that the concept space and message sequence can be related.

In Sect. 2, encoder and decoder networks are each trained separately, using a hill-climbing algorithm, to perform the task using the numeric encoding. In the following section, the encoder and decoder are *co-evolved* together and are at liberty to determine their own "language". We conclude with some remarks relating the results of the simulations to features of natural language.

2 Simulations 1 and 2: Encoders and Decoders

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

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

2.1 Encoders

The architecture for the encoder is a simple recurrent network (SRN) with additional connections from the output units to the hidden units.[1] Since a real value may require representation by an arbitrarily long binary string, we initially intended that the encoder would output an end-of-sequence symbol once

[1] Essentially a combination Jordan/Elman network.

the number had been encoded. Pilot simulations suggested that this encoding was difficult to evolve so the length of sequences that can be sent was artificially limited.

Given this general architecture, it is relatively straightforward to hand-code a network with a single hidden unit to perform the encoding for an MSB first sequence. Such a network is shown in Fig. 2(a). However, it is not possible to perform the LSB first encoding without a large number of hidden units due to the fractal nature of such an encoding. (For messages of n bits, 2^n values $(\frac{0}{2^n}, \frac{1}{2^n}, \ldots, \frac{2^n-1}{2^n})$ may be encoded. For any value, $\frac{k}{2^n}$, the first bit of output is the opposite to that of its neighbours.)

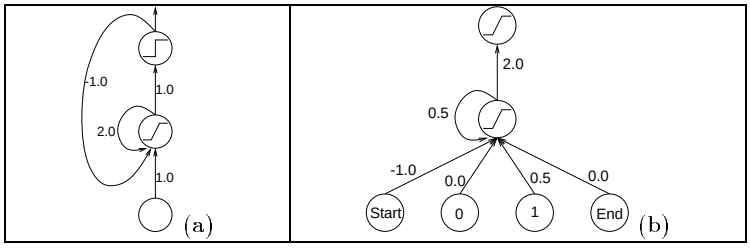

Fig. 2. (a) **MSB encoder:** A RNN that takes a real number between 0 and 1 and encodes it as a numeric string, most significant bit first. The hidden unit uses a linear threshold activation function that saturates at -1 and 1, whereas the output units use binary (0,5) threshold units. The input value is presented at the first time-step only. (b) **LSB decoder:** A SRN that decodes numeric sequences LSB first. The input 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 activations are used on all units.

Although a solution could be hand-coded, it was unknown whether it was learnable, so a series of simulations was designed to address this question. Networks were evolved using a simple hill-climbing algorithm to perform both the LSB and MSB mappings. A "champion" decoder was created with initially random weights. A single mutant was then spawned by randomly perturbing the weights of the champion according to a Gaussian distribution with 0 mean and initially 0.1 variance. 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, the strings were decoded with a perfect numeric decoder, and the sum squared error between encoder input and decoder output was calculated.

The values chosen to be encoded were selected by taking a staged learning approach [4]. Initially, only two values, 0 and 0.5, were encoded, and the number of bits that could be sent was accordingly set to 1. Once a network was able to perform this mapping, 2 bits could be sent, encoding 0, 0.25, 0.5 and 0.75. In general, after 2^k numbers could be successfully encoded into k bits, the networks were given 2^{k+1} values to encode into $(k+1)-$bit sequences. The variance was modulated throughout the course of the simulations. Simulations were run for

a maximum of 100K generations, or until all 5-bit values could be encoded. Networks with 1, 2, 3 and 5 hidden units were evolved.

2.2 Decoders

SRNs were used as decoders. The task for the these networks was the inverse of the encoders' task with minor variations. Each string presented to a decoder was enclosed with start-of-sequence and end-of-sequence inputs, a legacy of the task originally considered for the encoder. The additional inputs did not appear to have a considerable impact on the simulations.

Unlike the encoder, the decoder is capable of decoding either MSB or LSB first, though with some significant differences. Figure 2(b) shows a SRN that decodes LSB first. Although an LSB decoder is able to decode strings of varying lengths with only a single hidden unit, an MSB decoder (not shown) can only decode strings of a fixed length with a single hidden unit. Simulations were carried out in a similar manner to the encoder. A perfect encoder was used to encode values to numeric binary sequences. Decoders were compared by the sum squared error across all presented strings. The same principle of staged learning was applied.

2.3 Results: Encoders and Decoders

Networks of all sizes were able to encode MSB first sequences and decode LSB first sequences of up to 5-bits. No networks could encode more than 2-bit values, LSB first. No networks were able to decode 5-bit sequences MSB first, although one network with 5 hidden units was able to decode 4-bit sequences. The results are broadly summarised in Table 1.

	MSB First				LSB First			
Hidden units:	1	2	3	5	1	2	3	5
Encoders	11	18	11	7	0	0	0	0
Decoders	0	0	0	0	22	26	30	22

Table 1. Number of networks (of 50 trialed) attaining 5-bit precision in each condition.

3 Evolving a Language

There is clearly a significant difference between the encoders and decoders. The encoders were only able to learn the MSB first encoding, whereas the decoders preferred learning LSB first sequences. This presents a serious dilemma when we consider the complete system of encoding and decoding (Fig. 1). If the system is to successfully communicate values, then the encoder and decoder must compromise on the nature of the code. An MSB or LSB code will not suffice for the combined system.

Simulations of the complete system were performed under two conditions. In the first, the communication channel reversed the message: whatever was sent

first by the encoder was received last by the decoder. This condition allows an MSB code with the encoder encoding MSB first and the decoder decoding LSB first. With the second condition, the order of the message on the communication channel was preserved. In this scenario an MSB code is more difficult, and the encoder and decoder must develop a code which can be effectively learned and processed by both networks.

Pilot simulations showed that using a hill-climber for both encoder and decoder was intractable. Tests of backpropagation through time (BPTT) on the decoder showed that it was qualitatively similar with respect to the learning task of Sect. 2.2, but faster. Hence, a hill-climber was used for the encoder and BPTT for the decoder. The basic algorithm for the co-evolution of the system is described in Fig. 3.

1. Create a champion encoder and decoder.
2. Create a mutant encoder by perturbing the weights of the champion.
3. If the encoding created by the mutant uses a greater variety of strings than the champion, select that mutant.
4. Create a mutant decoder with weights initialised between -1.0 and 1.0.
5. For k iterations, present all inputs of the current precision to the encoder. Train the decoder on the output of the encoder.
6. 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 got all strings correct, increase the precision. Return to step (2).

Fig. 3. Evolutionary algorithm for combined encoder/decoder system.

3.1 Forwards and Reversed

Both the encoder and decoder used two hidden units, with the decoder trained for 1000 epochs. The system was give $n + 2$ bits when communicating $n-$bit values in the reversed case, and $2n$ bits in the forwards case. The extra bits were found to be necessary for a successful code to develop and have the effect of increasing the proportion of codes that uniquely identify each value.

In the reversed condition, the system was able to create successful codes for 5-bit values. A typical code is shown in Table 2. The code is effectively a sparse numeric code. Although not all binary sequences are used, those that are used are ordered by their numeric values.

The simulations performed with the forwards channel were not nearly as successful as the reversed case. The best observed code, shown in table 2, encoded all 3-bit values. It is apparent that it is neither strictly MSB nor LSB first, since there is no clear ordering of the significance of each bit (less significant bits should tend to show greater sensitivity to changes in the input.)

4 Discussion and Conclusions

The first series of simulations demonstrated the different constraints of the encoder and decoder on the numeric encoding task. Whereas the encoder is only

	Reversed			Forwards		
Input	Message	Flipped	Output	Message	Flipped	Output
0.000000	100111	001101	0.002858	010111	000010	0.000000
0.062500	100100	001110	0.041986			
0.125000	111001	010011	0.121091	010101	000000	0.119414
0.187500	111111	010101	0.194269			
0.250000	110010	011000	0.273809	011101	001000	0.242485
0.312500	110011	011001	0.313552			
0.375000	110000	011010	0.360761	111101	101000	0.353821
0.437500	001001	100011	0.442445			
0.500000	001111	100101	0.494467	111111	101010	0.497711
0.562500	001100	100110	0.541167			
0.625000	000011	101001	0.610452	111110	101011	0.616155
0.687500	000000	101010	0.657662			
0.750000	000001	101011	0.727246	101110	111011	0.741359
0.812500	000111	101101	0.800160			
0.875000	011000	110010	0.848808	101010	111111	0.876082
0.937500	011001	110011	0.918393			

Table 2. Left: Language for the reversed system, 4 bit precision. The code employed is not immediately apparent. Flipping alternate bits of the message (bits 1, 3 and 5) in the third column shows that the messages are, in fact, in numeric order. The bit-flipping behaviour is a consequence of having negative recurrent weights that oscillate the significance of successive bits. **Right:** The code from a forwards system for 3 bit precision. Flipping the bits of the message results in a code which is almost in numeric order both left-to-right and right-to-left, 0.0 proving the exception in both cases.

able to encode values MSB first, the decoder has a preference for decoding values LSB first. Interestingly, the encoder was unable to learn when it was required to output an end-of-sequence. For an MSB encoding this may not be a major issue. If the encoder stops earlier than it should, only the least significant bits are not encoded. However, for LSB or other encodings this issue is significant since the most important parts of the message may not be sent. This phenomenon warrants further attention in future simulations.

The second series of simulations are pilots and show how the different constraints of the networks may affect the evolved code. In both cases, co-evolution of the encoder and decoder was difficult. The primary cause of this appeared to be the lack of quality information given to direct the encoder's search through a combinatorically large space (functions from values to strings). Encouraging variability in the encoder proved a useful heuristic, since a necessary condition for a successful code is that every value has a unique encoding.

In the reversed condition, the biases of the networks were consistent and produced a numeric code. The system produced the type of encoding expected, given the results of the earlier component simulations. The codes developed in this condition were more sparse than strict numeric codes. Attempting to force a compact encoding on the encoder failed due to the small proportion of appropriate codes within the large search space.

When the message sent by the encoder was not reversed (the forwards case) the networks compromised on the code since neither was able to learn the encoding preferred by the other. Although the simulations did not develop codings to cope with significant levels of precision, they did give indications that the system employed neither MSB nor LSB codes, but instead those that could be read either backwards or forwards.

This is preliminary work and further simulations will be needed to substantiate the combined encoder/decoder study. We have presented a framework for studying the effects of constraints on the processing and emergence of language. The simulations presented here have been abstracted away from real languages, so an important goal of future work is to tie the framework more closely to natural language. A number of extensions for this purpose are immediately obvious including the use of multi-dimensional inputs, more symbols in the language, a non-uniform distribution of inputs and a population of communicators.

However, comparing the results of these initial simulations with human languages shows some interesting parallels. In the unrealistic reversed case, a code develops which resembles a numeric code. In the forwards case, the networks create a code that can be read either forwards or backwards, which is less efficient but meets the constraints of both the encoder and decoder. This is reminiscent of the tendency in human languages towards palindrome-like structure (e.g. N1 N2 N3 V3 V2 V1) which can be parsed in either direction. In further studies we hope to explore how certain features of human languages might have arisen as a compromise between the conflicting constraints of sender and receiver.

References

1. N. Chomsky. *Aspects of the Theory of Syntax*. MIT Press, 1965.
2. M. H. Christiansen and N. Chater. Toward a connectionist model of recursion in human linguistic performance. *Cognitive Science*, In press., 1998.
3. J. Elman. Distributed representations, simple recurrent networks and grammatical structure. *Machine Learning*, 7:195–224, 1991.
4. J. Elman. Learning and development in neural networks: The importance of starting small. *Cognition*, 48:71–99, 1993.
5. M. Hare and J. Elman. Learning and morphological change. *Cognition*, 56:61–98, 1995.
6. S. Kirby. Fitness and the selective adaptation of language. In James Hurford, Chris Knight, and Michael Studdert-Kennedy, editors, *Evolution of Language: Social and Cognitive Bases for the Emergence of Phonology and Syntax*. in press, 1998.
7. S. Lawrence, C. L. Giles, and S. Fong. Natural language grammatical inference with recurrent neural networks. To appear: IEEE Transactions on Knowledge and Data Engineering, 1998.
8. C. Moore. Dynamical recognizers: Real-time language recognition by analog computers. *Theoretical Computer Science*, 201(1–2):99–136, July 1998.
9. A. M. Turing. On computable numbers, with an application to entscheidungsproblem. *Proceedings of the London Mathematical Society, Series 2*, 42:230–265, 1936.
10. J. Weckerly and J. Elman. A PDP approach to processing center-embedded sentences. In *Proceedings of the Fourteenth Annual Conference of the Cognitive Science Society*, Hillsdale, NJ, 1992. Erlbaum.