Evaluating Systematicity in Neural Networks through Transformation Combination

Esteban Buz (ebuz@jhu.edu)
Department of Cognitive Science, Johns Hopkins University, 3400 N. Charles Street
Baltimore, MD 21218

Robert Frank (rfrank@jhu.edu)
Department of Cognitive Science, Johns Hopkins University, 3400 N. Charles Street
Baltimore, MD 21218

Abstract

The question of whether connectionist models can exhibit systematic rule-like behavior has engendered a great deal of debate. This paper suggests a new way of attacking this issue by looking at the ability of a neural network to combine the effects of independently learned transformational mapping operations. We explore this question in the context of Simple Recurrent Networks that are trained to map input strings to transformed output correspondents. We find that there is evidence for a capacity, even if imperfect, to combine operations as seen in symbolic rule systems. However, we find that the networks show a sensitivity to aspects of the input string that that would be unexpected if the transformations that the network induced were truly abstract. This is suggestive of a bias against abstraction in neural networks.

Keywords: rule learning, neural networks, systematicity, transformations

Introduction

Proponents of connectionist and symbolic models have fiercely debated the role and nature of abstract rules in cognition. On one hand, symbolists have argued that human cognition includes the capacity to induce abstract rules and that connectionist networks are incapable of inducing such rules on the basis of plausibly available data (Marcus, 2001). Others (e.g., Shultz & Bale, 2001; Altmann, 2002) have responded with demonstrations of connectionist models that are purported to succeed in inducing precisely the kind of generalizations that Marcus asserted to be impossible. It is probably fair to say that neither side has succeeded in convincing the other (Vilcu & Hadley, 2005; Shultz & Bale, 2006). Our goal is to suggest a new arena for investigating the question of the representation and induction of rules in cognition, and to present some preliminary investigations in this line.

Rather than focusing on rules that determine the well-formedness of certain forms, such as those studied by Marcus (2001), we would like to shift attention to a class of rules that have played an important role in the linguistic domain, namely *transformations*. Since the work of Chomsky (1957), it has been widely assumed that knowledge of language consists in part of a set of transformational operations that map one representation of linguistic structure to another.[1] Such transformations are abstract in the same way as Marcus' rules, applying uniformly to classes of syntactic, morphological, or phonological constitutents. Learning a syntactic transformation like passivization, then, entails identifying the abstract operations and units that are involved, without reference to specific words.

An interesting property of transformational rules is that they need not apply in isolation, but may apply in groupings. To take a grammatical example, the derivation of the question "Will the piggy be eaten by the wolf?" involves the application of both the passive transformation as well as a transformation involved in forming questions (subject-aux inversion).[2] We assume that alongside the capacity to induce and apply individual transformational mapping operations, human cognition provides us with the capacity of combining these operations, in potentially novel ways. Therefore, if connectionist models are to be considered as viable alternatives to their symbolic counterparts, it is important that they be shown to allow for this combinatorics in rule application.

In this paper, we study the combinatorics of transformational mapping operations as a way of assessing the ability of neural networks to induce abstract generalizations. Specifically, we focus on a domain involving strings that are related by the following pair of transformations:

$$\text{FRONT}(\langle \sigma_1, \sigma_2, \ldots, \sigma_k \rangle) = \langle \sigma_2, \sigma_1, \ldots, \sigma_k \rangle$$
$$\text{BACK}(\langle \sigma_1, \ldots \sigma_{k-1}, \sigma_k \rangle) = \langle \sigma_1, \ldots \sigma_k, \sigma_{k-1} \rangle$$

These transformations were chosen because of their simplicity. For strings longer than 4 symbols, on which we focus our training and testing, these transformations do not interact with one another. Although we are interested in ultimately testing the viability of connectionist networks as models of language, we have chosen for this first foray into combinatorics to avoid the complexities of transformations that make crucial reference to abstract hierarchical structure, looking instead at transformations that manipulate sequences on the basis of the linear positions of their elements.[3]

[1] Even those who deny the necessity of transformational operations in grammar must nonetheless accept that the grammar somehow characterizes the systematic relation between certain forms, such a between active and passive sentences. Without such a characterization, the parallelism between these forms would be unexplained.

[2] Much ink has been spilled in the linguistics literature about how multiple transformations apply to a representation, whether in a parallel or serial fashion, and if the latter, what determines their ordering. In this paper, we avoid this issue, but look forward to exploring in future work the question of whether neural networks could disentangle such issues.

[3] There is a range of previous work that has focused on the question of inducing and representing grammatical structure-dependent

IDENT:

Output					a	b	c	d
Input	a	b	c	d	#			

FRONT:

Output					b	a	c	d
Input	a	b	c	d	{#, FRONT}			

BACK:

Output					a	b	d	c
Input	a	b	c	d	{#, BACK}			

COMB:

Output					b	a	d	c
Input	a	b	c	d	{#, FRONT, BACK}			

Figure 1: Sample transformational inputs and outputs

Because we are interested in allowing FRONT and BACK to apply to strings of arbitrary length, we cannot make use of a simple feed-forward network with a fixed set of input and output units. Rather, we encode sequences through their temporal extent in a Simple Recurrent Network (SRN, Elman (1990)), as proposed by Botvinick and Plaut (2006) in their model of serial recall. Botvinick and Plaut show that an SRN can be trained to take a temporally extended sequence of input elements, one at a time, and output this sequence in response to a cue, again one element at a time. Frank and Mathis (2007) show that this same SRN architecture can induce single transformational mappings with great accuracy: they trained an SRN to output a given sequence in either its original ordering, as in Botvinick and Plaut's work, or in the reverse ordering, depending on the cue that is presented at the conclusion of the input sequence. For novel sequences of length eight, their network correctly output the target sequence in more than 97% of the cases. This paper seeks to explore whether systems of transformations can be learned, and if so, how the network generalizes their application to contexts involving a novel combination of transformations.

In the remainder of this paper, we report the details of our experiments in training an SRN to carry out the FRONT and BACK transformations, as well as their combination. If an SRN is successful in inducing some abstract representation for these transformational operations separately, it should be capable of combining the operations, generalizing its knowledge to a case where it is asked to generate a string in which both transformations apply. Similarly, if an SRN is trained to apply one of the transformations alone, as well as both of them together, it should be able to determine the properties of the other transformation on which it was not trained alone.

Simulation details

The connectionist model used for all of our experiments article is an SRN with seven input units, 100 hidden and context units and four output units.[4] The input and output units were used for to encode localist representations of the input and output symbols, as well as the end of string symbol and transformation cues (see below). Hidden and context units used a standard sigmoid activation function while output units had a soft-max activation function. Initial random weights were assigned from the range [-1, 1]. Weight updates were carried out using the Back Propagation through Time algorithm with a learning rate of .001, no momentum, and a batch size of 51. This batch size was chosen for two reasons. First, it was large enough so that each weight update was made on the basis of a sample of sentences that are reasonably representative of the training data as a whole; learning is less successful and slower in our experience with smaller batches. Secondly, the size was chosen so as not be a factor of the size of the training set, 100,000 in this case. Since the simulator cycles through the training set, this property ensures that a wider range of samples is presented during training.

The input and output strings we presented to the network during testing and training were random sequences over an alphabet of four symbols {a, b, c, d}. At the outset of an example, an input string was presented, one symbol at a time. During this portion of the example, the network was given no target output, and in evaluation the output of the network in these time steps was ignored. At the time step following the last symbol in the input string, the input to the network consisted of an end of string symbol '#' together with a transformation cue. The transformation cue was represented by a pair of input units, one for the FRONT transformation and one for the BACK transformation. If activation is given to one of these transformation units, say the FRONT cue unit, the target outputs for the times steps from the transformation cue correspond to the transformed input string, with the first two symbols permitted in this case. If both the FRONT and BACK units are activated, which we call a COMB cue, the target outputs are the input with both transformations applied. Finally, if neither unit is activated, which we call the IDENT cue, the target outputs are identical to the input sequence. These are shown in Figure 1.

Input strings in examples ranged in length from four to ten symbols. Training sets consisted of 100,000 randomly generated sequences, where a length between four and ten symbols was chosen at random with equal probability, and each symbol in the string was chosen randomly. The transformation for such a sequence was chosen at random from among

transformations in a connectionist network (Chalmers, 1990; Niklasson & Gelder, 1994; Neumann, 2002; Smolensky, 1990). That work does not however address the question of inducing the hierarchical structure necessary for the proper characterization of the transformation. Frank and Mathis (2007) take on this issue, using a model like the one explored in this paper, with mixed results.

[4]The choice of 100 units for the hidden layer size was motivated by previous simulations, which had networks with 50 hidden units incapable of mastering the task of transformational mapping. Networks with 75 hidden units succeed at learning transformational mappings somewhat less successfully than networks with 100 hidden units, but on generalization studies show qualitatively similar results.

those that were included in a particular training set. Testing sets of 100,000 examples were randomly generated using the same method as the training sets, but any examples that were present in the training set were removed, which left between 45,000 and 60,000 examples. Finally, for the simulations in which some transformation or combination of transformations was withheld, an experimental set of 100,000 examples was generated, but each of these examples was distinct and the transformation specification was uniformly set to the withheld value. Because of the equiprobability of string lengths, the training set typically exhausted the strings of shorter length for non-withheld transformations, meaning that the testing set was overwhelmingly composed of longer strings. For similar reasons, the experimental set typically contained most if not all possible strings of shorter lengths, though they were not the necessarily majority.

Experiment 1: Comprehensive training

In our first set of simulations, we test whether SRNs could acquire the different types of transformations, alone and in combination. We trained ten networks with different initial weights on the same training set. Each SRN went through 1,000,000 weight updates, and weights were saved after every 1000 updates. To assess the performance of a network on a particular example, we compared the most active output unit to the target output during the second half of the example. Only if all of these units were identical did we score the example as correct. After training was complete, we found the point during training that yielded maximal performance on the testing set and used this set of weights for analysis.

Results

Performance was very consistent across all ten of the trained networks. None of the ten networks showed performance on the training set below 100.00%. Mean performance on the testing set (including only novel strings) was 99.90% (SD: 0.036). As seen in Table 1, there was a small numerical difference between performance on the testing set with shorter and longer strings: the network performs better on shorter strings than on longer ones. A one-way ANOVA confirmed that this result was significant ($F(5,54)=49.88$, $p < .001$). There was also a numerically small, and again significant ($p < .001$), difference in the network's testing accuracy at the start of the string (first two symbols) 100.00% (SD: 0), as compared to the end of the string, 99.94% (SD: 0.024).

From these results, we can safely conclude that the network architecture we are employing is highly successful in learning the transformational mapping task and in generalizing its knowledge to novel strings.

Experiment 2: Transformation combination

In our second set of simulations, we trained SRNs only on inputs with one or no transformations specified, withholding the COMB case involving both transformations. Training was performed as in the previous set of simulations. Reported performance was the point at which accuracy on the experimental set was maximal. In this set of simulations, we trained 30 SRNs and the data presented below is averaged over all of them unless otherwise specified. Three distinct training and testing example sets were used with ten SRNs trained on each example set. All networks were assigned different initial random weights.

Table 1: Experiment 1 – Length Effect on Test Set Performance across ten networks

Length	Accuracy	Standard Deviation	test set size
5	100.00%	0	229
6	100.00%	0	4223
7	99.99%	0.006	10421
8	99.98%	0.017	12935
9	99.95%	0.028	14170
10	99.85%	0.065	14267

Results

There was little difference in training set performance across the three example sets, with a mean accuracy of 99.81% (SD: 0.367) across all ten networks on each of three training sets. A similar result holds for the performance on the testing sets with mean accuracy of 99.27% (SD: 0.933) across all networks. Once again, the networks were very successful at learning the training set and at generalizing to examples not present in the training set. As before, performance at the start of the string (99.94%, SD: 0.126) was slightly better than at the end (99.76%, SD: 0.275), but this difference was nonetheless significant ($p < 0.01$).

On the experimental set data, which includes only examples with the COMB transformation cue, the performance of these networks was considerably poorer and somewhat more variable, with mean accuracy of 62.57% (SD: 14.761). Though performance is far below that on the training and testing set for all the networks that we trained, it is worth noting that it is still considerably above chance, which we calculate as the likelihood of outputting the correct string if the symbol occupying each position were chosen independently and with equal probability. Thus, for a string of length four in a language containing fours symbols, the chance of outputting any particular response, including the correct one, would be $.25^4 \approx .004$, or, .4% accuracy. This shows the network has been successful to some degree in generalizing its experience from the individual transformations to the combination of the two.

As noted above, because the experimental set is made up of unique examples, the vast majority of the examples it contains involve longer sequences. Since success on an example requires that the network output all of the symbols accurately, the task is more difficult for longer strings. As a result, we must entertain the possibility that the reduced performance we see here is because of the higher average length of the sequences involved in this set. To begin to factor this out,

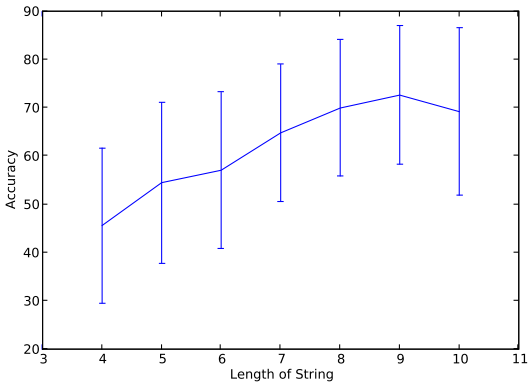

Figure 2: Experiment 2 – Effect of Input Length on the experimental set. Bars indicate SD across the thirty networks that were trained

we can compute performance at the left and right edges of the output strings, not penalizing errors in any but the network's first, second, penultimate and final outputs, which are the the loci of the elements that are to be transformed. As expected, network performance on this "edge accuracy" measure is higher than performance overall (60.03%, SD: 14.338), but even this level of performance is lower than that observed in the training and test sets.

We can approach the question of the effect of example length more directly by breaking performance down according to the length of the input sequence. Figure 2 shows accuracy by length over all the trained networks with bars showing one standard deviation above and below the mean. There is a substantial difference in accuracy between shorter strings and longer ones, confirmed by a one-way ANOVA ($F(6,203)=11.5696$, $p < .001$), with the surprising result that the network is more accurate for longer strings. This is precisely the opposite of the effect we saw in Experiment 1 where performance on test data was higher than on shorter strings than longer ones. Table 2 presents performance by string length broken down for accuracy at the left edge (i.e., the first two symbols were both correct) and at the right edge (i.e., the last two symbols were both correct). Here we see a combination of the effects that we have observed thus far: accuracy is higher at the beginning of the string, and longer strings are more accurately transformed than short ones.

Experiment 3: Transformation subtraction

In Experiment 2, we considered whether a SRN could learn to combine the effects of two independently learned transformations. In the current experiment, we consider the extent to which they can break apart the individual operations of combinations of transformations that are learned together. We do this by providing training examples involving one of the transformations alone (e.g., FRONT), no transformations (IDENT), or their combination (COMB), but withholding training data for the other transformation by itself (e.g., BACK). The logic here was similar to that in the previous experiment: if the network is successful at identifying the abstract operation associated with the BACK cue and learns the mapping associated with the combination of BACK and FRONT, it should be able to subtract out the effects of BACK from the latter to isolate the operation associated with the FRONT cue.

Table 2: Experiment 2 – Mean Left and Right Edge Accuracy by Length on COMB examples across all thirty networks

Length	Left	Right	test set size
4	75.96%	61.82%	256
5	78.82%	67.70%	1024
6	82.41%	69.69%	4092
7	85.52%	76.44%	13736
8	87.62%	79.93%	23937
9	88.33%	81.75%	28003
10	86.54%	76.99%	29216

Results

We divided this experiment into two parts, withholding the FRONT cue in one and the BACK cue in the other, Since we did not find substantial difference among training sets in Experiment 2, we restricted attention to a single training set. In each part, we trained 10 SRNs in the same fashion as in previous experiments. As before, the performance that is reported for each network reflects the point during the 1,000,000 weight updates of training when its performance is maximal for the experimental set.

Part I: Generalization to FRONT For networks in which the FRONT cue was withheld, mean training set performance was 99.77% with a SD of 0.486. Mean performance on testing data was at 99.69% with a SD of 0.267. Mean performance on the experimental set, consisting of 100,725 unique examples that contained only the FRONT cue, was 83.76% (SD: 10.928). Focusing only on the performance of these networks at the left and right edges, so as to reduce the effect of possible errors in the middle of the longer strings in the experimental set, increases accuracy to 87.34% (SD: 8.599).

As in Experiment 2, we find an effect of example length and string position. As seen in Table 3 and supported by a two-way ANOVA, accuracy is significantly affected by length ($F(6,126)=3.37$, $p < .01$), with better performance on longer strings, and by the position of the symbols ($F(1,126)=55.80$, $p < .001$). There was no reliable interaction between these factors. This result is surprising given that the withheld transformation concerned the front of the string.

Part II: Generalization to BACK For networks in which the BACK cue was withheld, mean training set performance was 99.99% with a SD of 0.033. Mean performance on testing data was at 99.84% with a SD of 0.122. Mean perfor-

Table 3: Experiment 3 (Part I) – Average Left and Right Edge Accuracy by Length for ten networks on the FRONT cue

Length	Left	Right	test set size
4	93.36%	71.37%	256
5	95.08%	80.80%	1024
6	98.76%	81.23%	4093
7	99.53%	84.09%	13715
8	99.49%	92.14%	23990
9	99.35%	90.42%	28243
10	98.51%	85.85%	29404

Table 4: Experiment 3 (Part II) – Average Left and Right Edge Accuracy by Length for ten networks on the BACK cue

Length	Left	Right	test set size
4	91.68%	77.30%	256
5	93.74%	81.25%	1024
6	95.71%	83.29%	4094
7	96.11%	89.61%	13721
8	96.63%	92.27%	23865
9	97.14%	92.43%	28060
10	96.62%	91.31%	29176

mance on the experimental set consisting of 100,196 unique examples was 85.32% (SD: 10.764), and edge accuracy was 88.27% (SD: 8.848). These results are not significantly different from that of the networks in Part I of this experiment.

If we break down performance by length and position, as illustrated in Table 4, we find the by now familiar results that performance is better at the beginning of the string and better for longer strings. Once again, a two-way ANOVA revealed that accuracy was significantly affected by input length ($F(6,126)=5.64$, $p < .001$) and the location of the transformation ($F(1,126)=45.84$, $p < .001$), but that there was no reliable interaction between these factors.

Discussion

A number of results stand out from the three experiments just described. First of all, and perhaps most impressively, we found in all of our experiments that our network model is successful in generalizing the transformation mappings on which it is trained to novel strings. This suggests that the networks have indeed internalized some kind of systematic operation, capable of applying across the class of representations that the network uses to encode the sequential information present in its input. At present, we do not have a clear idea about what this representation consists of or how the transformational mapping is encoded. In a network with as many hidden units as ours, such questions are extremely difficult to answer. We are at present exploring a variety of methods that we hope will allow us to identify representations in such large networks.[5] Nonetheless, we can glean something about the manner in which the network is solving the task by looking at the network's pattern of performance.

The results of Experiments 2 and 3 provide substantial evidence that however the transformational operations are represented, they are indeed represented in a way that supports operation combination. Performance on applying the combination of two independently learned transformations was well above chance, as was performance on the "subtraction" of two transformations. The transformations studied here were chosen so as to be able to apply independently of one another, in the sense that the operation of one does not influence the operation of the other, at least when understood as applying to symbolic representations. However, it was far from clear prior to our experiments that such independence would carry over to the distributed representations that the SRN uses to encode the input sequence. The fact that this independence does carry over suggests that the network's encoding of the sequence as well as of the transformational operations defined over it retain important representational dimensions present in symbolic models.

Even if the transformations support combination, it remains an open question as to whether they are encoded as abstract operations that apply independently of the particular symbols that appear in specific positions, as in the symbolic rule "switch the first two symbols," or whether they encode the specific symbols that are to be switched, as in, "when *ab* are in the first two positions and the FRONT cue appears, output *ba*." To begin to explore this question, we performed a small number of simulations like those in Experiment 2, but where the training data included no FRONT examples with the bigram *ab* in the first two positions. Performance on the resulting network varied on the basis of the random initial weight, but there were at least some networks exhibiting some generalization to FRONT examples involving the initial bigram *ab*, though not at the level of Experiment 2 to novel examples. Withholding from the training data the same bigram in the front two and last two positions for FRONT and BACK examples respecitively led to generalization that was as good or better than witholding from the FRONT examples alone. This suggests that in the FRONT case the network is not generalizing the swapping of the bigram *ab* at the back of the string to the front of the string in the FRONT hold-out examples. In contrast, withholding a particular symbol from second position in examples involving the FRONT transformation resulted in no generalization at all to held-out exam-

[5] Botvinick and Plaut (2006) use a logistic regression over presigmoid activation values to conclude that a model essentially identical to ours makes use of a superpositional encoding of content and position information. Don Mathis (p.c.) points out however that according to Botvinick and Plaut's method an untrained network can be shown to use the same encoding, suggesting at the least that this aspect of the representation is not induced during training. This type of behavior is also seen in Echo State Networks (Jaeger, 2002), where recurrent weights are not modified during training. As a reviewer points out, we should not expect anything similar during the "decoding" phase of our task, though Botvinick and Plaut only performed their analysis during the encoding phase.

ples, suggesting that the transformations are represented in a way that is sensitive to the individual symbols that are being transformed, though not in combination with one another.

A very surprising result concerned the effect of string length on performance. In all of our experiments, when tested on transformational mappings on which they were trained, network performance degraded as the length of the input string increased, both for the strings present in the training data and for novel input strings. However, for novel transformations or novel combinations of transformations, this pattern was reversed: performance was best on longer strings and worst on shorter strings. One way of understanding this pattern is in terms of the network's representation of the input strings. If the network does not represent each symbol-position pairing distinctly, but rather uses an encoding of each such pairing, that also represents information about other symbols in the its context. This will mean that the inversion of the first two symbols will unavoidably affect, albeit to a lesser degree, the representation of subsequent symbols in the string. In Experiment 2, since both the first and last pair of symbols are inverted, moving these two pairs closer together is more likely to result in some sort of interference in a potentially unstable representation of the inverted symbols. If the representation of unchanged intervening symbols is more robust, the addition of noise resulting from the inversion of neighboring symbols would be less likely to give rise to error. We have begun exploring the influence of distance between the transformations, replicating Experiment 2 with a modified domain such that the BACK transformation applies to the third and fourth rather than the last two symbols. Under this set-up, there is no additional distance between the transformations for longer strings that could lessen any possible interference between them. What we found was that the dramatic length effect on the held out COMB examples in Experiment 2 was not replicated, suggesting that our interference hypothesis is on the right track. This idea is easily extended to Experiment 3, though the inclusion of the COMB cue could result in reduced interference from each transformation.

Another way of understanding this length effect is in terms of properties of the transformational mappings. Specifically, suppose that the transformation that the network induces is not as abstract as the symbolic counterpart, which simply manipulating symbols at the periphery of a representation, but also makes commitments about the stasis of nearby elements in the input. In other words, the transformation does not simply change what needs to be changed, but also specifies that other things do not change. Assuming there is a default mechanism of outputting the symbols in the order in which they were originally seen, encoded in the recurrent connections, such a transformation would be less abstract than necessary in the sense that it refers to parts of the string that do not need to change. When presented with the withheld transformations, of course, this alternative would be less successful, in precisely the ways observed in our simulations. If this is on the right track, it suggests that any shortcomings in network models that exhibit rule-like behavior come not from their inability to participate in systematic computation, but rather from a failure to induce maximally abstract characterizations of the patterns in the data that are presented during training.

Acknowledgments

For useful comments, discussion and statistical help, we thank Bill Badecker, Chris Kirov, Don Mathis, and four anonymous CogSci 2008 reviewers. This work was supported by NSF grants SBR-0446929 and SBR-0549379.

References

Altmann, G. (2002). Learning and development in neural networks: The importance of prior experience. *Cognition*, *85*, B43–B50.

Botvinick, M. M., & Plaut, D. C. (2006). Short-term memory for serial order: A recurrent neural network model. *Psychological Review*, *113*(2), 201–233.

Chalmers, D. J. (1990). Syntactic transformations on distributed representations. *Connection Science*, *2*(1& 2), 53–62.

Chomsky, N. (1957). *Syntactic structures*. The Hague: Mouton.

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

Frank, R., & Mathis, D. (2007). Transformational networks. In *Proceedings of the 3rd workshop on psychocomputational models of human language acquisition.* Nashville.

Jaeger, H. (2002). Adaptive nonlinear system identification with echo state networks. In S. Becker, S. Thrun, & K. Obermeyer (Eds.), *Advances in neural information processing systems 15* (pp. 593–600). Cambridge, MA: MIT Press.

Marcus, G. F. (2001). *The algebraic mind*. Cambridge, MA: MIT Press.

Neumann, J. (2002). Learning the systematic transformation of holographic reduced representations. *Cognitive Systems Research*, *3*(2), 227–235.

Niklasson, L. F., & Gelder, T. van. (1994). On being systematically connectionist. *Mind and Language*, *9*, 288–302.

Shultz, T. R., & Bale, A. C. (2001). Neural network simulation of infant familiarization to artificial sentences: Rule-like behavior without explicit rules and variables. *Infancy*, *2*, 501–536.

Shultz, T. R., & Bale, A. C. (2006). Neural networks discover a near-identity relation to distinguish simple syntactic forms. *Minds and Machines*, *16*(2), 107–139.

Smolensky, P. (1990). Tensor product variable binding and the representation of symbolic structures in connectionist systems. *Artificial Intelligence*, *46*, 159–216.

Vilcu, M., & Hadley, R. H. (2005). Two apparent 'counterexamples' to Marcus: A closer look. *Minds and Machines*, *15*(3–4), 359–382.