The Relationship between Object Manipulation and Language Development in Broca's Area: A Connectionist Simulation of Greenfield's Hypothesis

Ronan G. Reilly
Department of Computer Science
National University of Ireland, Dublin
Belfield, Dublin 4
Ireland

Abstract

In her *Behavioral and Brain Sciences* target article Greenfield (1991) proposed that early in a child's development Broca's region may serve the dual function of coordinating object assembly and organizing the production of structured utterances. As development progresses, the upper and lower regions of Broca's area become increasingly specialized for motor coordination and speech, respectively. This commentary presents a connectionist simulation of aspects of this proposal. The results of the simulation confirm the main thrust of Greenfield's argument, and suggest that an important impetus for the developmental differentiation in Broca's area may be the increasing complexity of the computational demands made upon it.

Introduction

In her target article in *Behavioral and Brain Sciences*, Greenfield (1991) proposed that there are parallels in the developmental complexity of speech and object manipulation. In studying the object manipulation of children aged 11-36 months, she observed that the increase in complexity of their object combination abilities mirrored the phonological and syllabic complexity of their speech production. There are two possible explanations for this phenomenon: (1) It represents analogous and parallel development mediated by separate neurological bases; or (2) the two processes are founded on a common neurological substrate. Greenfield (1991) adduced evidence from neurology, neuropsychology, and animal studies to support her view that the two processes are indeed built upon an initially common neurological foundation, which then divides into separate specialized areas as development progresses.

A significant part of Greenfield’s argument centered on the results of an earlier study by Greenfield, Nelson, and Saltzman (1972) in which children were asked to nest a set of cups of varying size. The very young children could do little more than pair cups. Slightly older children showed a capacity to nest the cups, but employed what Greenfield referred to as a “pot” strategy as their dominant approach. This entailed the child only ever moving one cup at a time to carry out the task. Still older children tended to favor a “sub-assembly” strategy in which the children moved pairs or triples of stacked cups. The difference between these two strategies is illustrated in Figure 1 below. At a given age one can detect a combination of different nesting strategies being employed, but with one strategy tending to dominate.

<Insert Figure 1 about here>

The first goal of the work reported here is to devise an abstract characterization of the speech and object assembly tasks that will allow the interaction of both activities to be explored within a connectionist framework. The essence of both tasks is the management of the linear expression of hierarchically organized concepts. In one case it is a hierarchically organized motor program, in the other, it is a hierarchically organized word and/or sentence structure. The second goal is to reproduce a range of Greenfield’s findings within this abstract formulation. The final goal is to provide some new insight into the process of homologous development and divergence.

Interdependence and differentiation

Greenfield has proposed that both language production and high-level motor programming are initially subserved by the same cortical region, which subsequently differentiates and specializes. Two questions raised by Greenfield’s hypothesis will be explored in this commentary. The first is whether there is an interdependence between the computational demands of object assembly and of language production, such that language production requires the computational foundation initially laid down for object assembly tasks. The second question relates to the nature of the process of differentiation. One possibility is that the maturation of Broca’s region follows a genetically determined timetable, and differentiates relatively independently of an individual’s experience. This is the view favored by Greenfield herself, who, while not denying the necessity of environmental input, puts more emphasis on the role of epigenetically driven changes in circuitry within and around Broca’s area. Another possibility is that differentiation occurs as a result of the growing processing demands of both language and object assembly activities. Within this scenario, more emphasis is put on the role of the environment and experience in shaping cortical connectivity.

The set of simulations described here seeks to explore these two aspects of Greenfield’s hypothesis. With regard to interdependence, the simulations examine the relative benefits of constructing a language production mechanism on a system for doing object assembly as compared to other tasks. The process of differentiation is studied by examining the impact on the neural network model of increasing the complexity of the task to be learned.

<Insert Figure 2 about here>

A Simulation Framework

As observed by Dean and Cruse (1995), the current dominant metaphor for understanding motor behavior is the concept of a motor *program*. This has proved a useful concept, so long as the notion of “program” is loosely interpreted. That is, we must see the “program” as capable of operating without input, of being able to modify itself, and so on. Furthermore, Van Essen, Anderson, and Olshausen (1996) have argued that motor coordination is too computationally demanding to have separate circuits for each motor program available for all motor “modalities” (e.g., hands, legs, eyes). Consequently, they propose that motor programs are represented centrally in some abstract form, where they are accessed and “interpreted” by the relevant low-level effectors. Indeed, one could argue that it is the very abstractness of these representations that permits them to be used in contexts other than those for which they were designed, such as language, if we assume Greenfield’s hypothesis to be correct. There is good evidence from positron emission tomography (PET) studies that a likely location for this central repository of abstract motor programs is the ventral region of the left frontal lobe, commonly called Broca’s area (Fox, Petersen, Posner, & Raichle, 1988).

What form might these abstract representations take? Well, it is generally accepted (Houghton, 1990) that a hierarchically structured neural representation is the basis for the generation of complex sequential behavior. The view that such behavior might be generated from a linearly organized sequence, the elements of which are chained together by a simple associationistic mechanism has been viewed as untenable for some time (Lashley, 1951).

In keeping with these observations, an overall modeling framework for this commentary is presented in Figure 2. At a very general level this aims to capture the idea of hierarchically organized abstract representation of goal state connected to a response system for generating an action sequence to achieve the goal. It is based on Rumelhart and Norman’s (1982) model of skilled typing. They used the term “schema” to refer to the hierarchical representation. In their model, a schema for a particular word comprised sub-schemata for each of the word’s letters. These sub-schemata, in turn, were used to generate the appropriate hand and finger movements that initiated the key presses. Rumelhart and Norman’s typing model was modality specific, with motor programs specifically wired-in for carrying out movement of the hands and fingers. However, as has been argued above, we need to be able to represent motor programs at a more general level. The framework in Figure 2 is intended to be a generalization of their model. Figure 2 is neutral on how exactly the schemata get represented. Nonetheless, if we are to simulate faithfully the neural basis of the Greenfield hypothesis, there are constraints on the selection of representational candidates that should be taken into account. It will be desirable to devise a hierarchical representation that has at least two main properties: (1) It must represent with equal facility both motor and linguistic programs; and (2) it should have some degree of neural plausibility.

Connectionist hierarchical structures

One candidate for representing hierarchical structures within a connectionist framework is to use the recursive auto-associative memory (RAAM) method developed by Jordan Pollack (1990). Pollack’s RAAM is based on a connectionist architecture called an encoder. The idea behind the encoder is quite simple. A three-layer feedforward network is trained to reproduce on its output units the same pattern of activation that is on its input units. It is trained to do this for a given set of training patterns, using a supervised learning algorithm such as backpropagation (Rumelhart, Hinton, & Williams, 1986).

A key feature of encoder networks is that the number of intervening hidden units is less than the number of input units (the number of output units is the same as the number of input units). This means that the network must learn a compact encoding of the input patterns, but one that is sufficiently precise to permit a relatively faithful reconstruction of the same set of patterns on the output units. Thus, for every input pattern there is a unique pattern of activation on the network's hidden units which is that pattern's compact encoding[1]. An 8-4-8 encoder (8 input units, 4 hidden units, and 8 output units), for example, must learn to encode the pattern of activity on its eight input units, while simultaneously learning to decode it back into its original form on its output units. In fact, an "encoder" network is a combination of encoder *and* decoder. The set of weights between the input and hidden layers of units carries out the encoding, while the set of weights from the hidden layer to the output layer does the decoding. Pollack's (1990) innovations were (1) to permit the input units of an encoder to represent not just one element, but pairs or triples of elements, and (2) to use the compact encodings themselves as inputs. These two developments gave him a means of recursively encoding hierarchical structures into fixed-width distributed representations.

<Insert Figure 3 about here>

Pollack's technique is illustrated in Figure 3. Using a 2*k*-*k*-2*k* encoder, the tree in Figure 3 can be encoded into a fixed-width representation by recursively training the encoder network on the set of patterns given in Table 1. In this case, the three terminals A, B, and C are represented by 1-in-4 bit vectors (e.g., A={0,0,0,1}, B={0,0,1,0}, C={0,1,0,0}) and the two non-terminals are four-element vectors of real numbers (e.g., R_1={0.2, 0.9, 0.1, 0.8}, R_2={0.7, 0.2, 0.3, 0.1}). At the beginning of training, the R_n vectors are initialized to random values in the range 0.0 to 1.0. As training proceeds, these get replaced with the relevant patterns of activation from the network's hidden units.

<Insert Table 1 about here>

The input at each training step consists of two adjacent vectors (i.e., the daughters of a node). For a given epoch, all of the training patterns are presented. Obviously, the R_n vectors will be changing as training proceeds. Consequently, these patterns present something of a moving target to the learning algorithm. Nevertheless, given a suitable choice of learning parameters, the representations will eventually converge and stabilize. At the end of the training procedure, the pattern R_2 can be said to represent the entire tree structure (A (B C)). The elements of this structure can be recursively unpacked by taking R_2, inserting it into the hidden units of the encoder network, and using it to generate the patterns A and R_1 on the output units. R_1, in turn, can be used to unpack the remaining terminal elements of the tree.

[1] Obviously, this encoding is tied to the weights of a given network, and would not be usable in another network.

The advantage of the RAAM technique is that encoded structures can be of varying complexity, yet still be represented in a fixed-width vector of real numbers. Furthermore, RAAMs can be generalized from binary to *n*-ary trees. The RAAM application used in this commentary employs a ternary encoder, where the 3*k* inputs are mapped to 3*k* output through *k* hidden units.

RAAM representations satisfy the two criteria set out above for a suitable neural representation of hierarchical structure. They are a general scheme, and thus can cope equally well with both linguistic and motoric representations. They are neurally plausible on several counts. The representations are distributed, and neural representations appear also to be distributed. They preserve similarity relationships (Pollack, 1990): RAAM encodings of structures that are similar, are themselves similar. This also appears to be a pervasive feature of neural representations (Sejnowski, 1986.). In contrast, a system of representation that simply associated an arbitrary bit pattern, say, with a given motor program would be less neurally plausible. The fact that RAAM representations are of fixed-width also gives them some degree of neural plausibility, since the neural pathways (i.e., cortico-cortical projections) along which their neural counterparts are transmitted, are also of fixed width.

<Insert Figure 4 about here>

Generating action sequences

Assuming that RAAM representations are an acceptable means of encoding hierarchical structures, a way then needs to be found to generate action and speech sequences from these representations. The solution proposed here is to use a simple recurrent network (SRN; Elman, 1990) which takes a RAAM representation as input and generates an appropriate sequence of actions as output (see Figure 4). Note that both the input to the SRN and its output are RAAM representations of tree structures. Thus the model described here is constructed in two phases: (1) The development of RAAM representations of action and speech goals; and (2) the generation of action and speech sequences from RAAM goal representations. The main focus of this study will be on the latter phase. Among the aspects of generation that will be focused on will be the relative ease with which certain classes of action sequences can be learned. A particular test of the approach used here will be whether in the nested cups task the *pot* strategy proves easier to learn than the *sub-assembly* strategy. A central issue, however, is whether there is some computational advantage from constructing a language system in an area that has some specialization for object assembly. As has already been suggested, it may not be entirely accidental that the neurological center for dealing with the hierarchical structure of language develops where it does.

<Insert Figure 5 about here>

The cups task

The original experiment by Greenfield et al. (1972) involved asking children of varying ages to assemble a set of five nested cups. In the case of the simulation, four rather than five cups were used in order to reduce the number of structures for which RAAM representations needed to be created. As will become clearer when the RAAM training procedure is discussed, training becomes increasingly more difficult with an increase in the number and complexity of structures to be encoded.

A child demonstrates a number of different strategies in assembling the cups: pairing, pot, and sub-assembly. The pot and sub-assembly strategies, however, will be the focus of simulation described here, since this is where the most significant difference is expected. There are of course other strategies a child might use, but which would not achieve a successful completion of the task (i.e., a set of fully nested cups). In some cases a child will insert the smallest cup into the largest, but will then be unable to complete the nesting task. In other cases noted by Greenfield et al. (1972), the child will rest a large cup on top of a smaller one, even employing a sub-assembly strategy where the large cup contains other cups. Again, these actions will not lead to a successful completion of the task. An underlying assumption of the simulations described here is that the child has some conceptualization of what s/he has to achieve, since the experimenter has already demonstrated the correct way of doing the task. So providing training on only these correct strategies is remaining faithful to the original experiment being simulated.

The cup-task representation comprises two parts: (1) A representation of the end-state to be achieved, and (2) a representation of the actions needed to be carried out to achieve the end (or goal) state. In both cases, a ternary RAAM representation is used. Where a node in the representation does not have three branches, a *null* terminal is used. Figure 5 represents the end states of the cups all being nested together using (a) a pure pot strategy, (b) a pure sub-assembly, and (c) a mixture of the two. Note that these end-states capture to some degree the route taken to achieve them, without detailing the actions required to do so.

In order to achieve the end states illustrated in Figure 5, a series of actions need to be carried out. These involve “getting” and “putting” cups or assemblies of cups. A general representational structure of the form *(<actor> <action> <acted upon>)* was used to represent these actions. Now it can be argued that this structure is to some degree arbitrary. However, the intention was to capture the fundamental notion of actor, action, and acted-upon, since Greenfield, Nelson, and Saltzman (1972) suggested that a child’s increasing flexibility in assigning cups to these categories was the critical element in their mastery of the more advanced sub-assembly strategy.

To achieve the configuration in Figure 5(a), one could have the following sequences of actions: *(null Get cupc), (cupc Put cupd), (null Get cupb), (cupb Put (cupc In cupd)), (null Get cupa), (cupa Put (cupb In (cupc In cupd))).* Alternatively, to end up with the configuration in Figure 5(c), the following sequence would be necessary: *(null Get cupa), (cupa Put cupb), (null Get cupc), (cupc Put cupd), (null Get (cupa In cupb)), ((cupa In cupb) Put (cupc In cupd)).* In this example, the first and second pair of actions are interchangeable. Note also that in the representation scheme as a whole, there is an implicit left-to-right polarity. So that *(cupa In cupb)* means that *cupa* is in *cupb*, and *(cupc Put cupd)* means that *cupc* is put into *cupd*.

<Insert Figure 6 about here>

The speech task

The nature of the speech production task paralleled that of object assembly. A RAAM encoded speech “goal” was used as input to the SRN, which then used it as the basis for the generation of speech actions (i.e., phonemes). The assumption underlying the speech “goal” is that its structure corresponds to some form of underlying representation that the speaker uses as a basis for speech generation. In order to capture the psychologically realistic details of the speech structure an onset/rhyme underlying representation was assumed for structure of the speech goal. The choice of appropriate representation here is to some degree moot. Nonetheless, there is good support for the psychological reality of the onset/rhyme structure from, for example, the study of speech errors (Stemberger, 1983; Dell, 1988). The output speech actions were also represented as RAAM encodings.

In utterances involving more than one word, a simple phrase structure was used as the overarching structure. A sample of speech was taken from Higginson corpus (Higginson,

1984) of the CHILDES child language database (MacWhinney & Snow, 1990; MacWhinney, 1993). The sample comprised three sets of 10 utterances sampled at 11 months, 20 months, and 35 months (see the Appendix for a complete list of utterances). The reason for choosing this particular corpus was because it gave a representative spread of language over the age ranges that Greenfield et al. (1972) had studied for the cups task. For SRN training, the entire pool of 30 utterances was divided into two equal sized groups.

The output phonetic representation used is an ASCII version of the IPA called UNIBET. So, for example, the utterance /dad✞/ is represented in UNIBET form as dadA, and in RAAM form as *((d (a null null) null) (d (A null null) null) null)*. The process of speech production was conceptualized as involving the generation of an action sequence derived from the RAAM representation of the entire utterance. As illustrated in Figure 6, given an utterance "goal" such as *((d (a null null) null) (d (A null null),* this would be the input to the SRN and would give rise to the following sequence of speech actions: *(null Say d), (null Say a), (null Say d), (null Say A).* The structure of each speech action was intended to be similar in form to those for object assembly, with "Say" in the same location in the speech structure as "Get" and "Put" in the action structure. Again, note that both the input goal and output actions are in the form of RAAM representations.

Simulation details

The simulations described here were programmed in C using the library of functions supplied with the SNNS simulator from the University of Stuttgart (Mamier & Wieland, 1996). The RAAM training involved taking both the object assembly and speech structures, and training a network to encode and decode these representations. This involved altogether 102 tree structures, comprising 35 object assembly ones and 67 language ones. The object assembly structures consisted of five goals, and 30 actions, while the 67 language structures consisted of 30 goals, and 37 actions. A 150-50-150 RAAM network was trained to encode them. Altogether, the RAAM network was trained on a total of 191 training patterns. There were more training patterns than tree structures because the basic element of training was the node of the tree, so there were as many training patterns as nodes.

The 45 terminals, because of their relatively high number, were encoded using a 6-bit binary representation, rather than the 1-in-*n* bit encoding used by Pollack (1990). There was no distinction made between the language terminals and the object assembly terminals; they were all arbitrary bit patterns. In principle one could have used a 1-in-50 bit code for the terminals, but one needs to use significantly more input units than are required simply to encode the terminal elements of the tree structures. This is because the excess units are needed to carry information about the recursively encoded tree structure. The RAAM network was trained for 116,000 epochs. An annealing schedule of learning rates was used. As training progressed, learning rate was varied from 0.1 through to 0.005, and momentum was varied from 0.5 and 0.9. In training a RAAM network, an important issue is the criteria that one uses to determine when training should cease. In this case an error tolerance of 0.05 was used for each element of a non-terminal vector, 0.2 for the binary encoding part of a terminal vector, and 0.1 for the empty part of that vector.

The SRN comprised 50 input units, 50 hidden units, 50 output units, and 50 context units. The context units were used to contain activation values of the hidden units from the previous time step. The end of each sequence (object assembly or speech) the activation values of the context units were reset to 0.5. A single learning rate of 0.01 and momentum of 0.9 was used for all SRN training trials. All SRN networks were trained for 1000 epochs per stage, where a *stage* is defined below.

The task of the SRN was to take a given RAAM-encoded goal and output a sequence of RAAM-encoded actions designed to achieve it. These goals could involve object manipulation or language generation. The training corpus was divided up into three parts. The first comprised the object assembly part (30 training patterns), the second involved all of the month 11 language corpus sample, and half of the month 20 sample (46 training patterns in total), while the third comprised the other half of the month 20 sample, and initially all of the month 35 sample (79 training patterns). The idea here was to create three stages in the learning process: an object assembly stage, a simple language stage, and a complex language stage. It is important to note that at each training stage, the training tasks from the preceding stages were also included. So that by the third training stage, the network was learning to produce action sequences along with both simple and complex speech sequences.

Simulation results

RAAM Encoding

Of the 102 RAAM structures, two proved particularly difficult to train. These were two of the multi-word utterances from the 35 month part of the Higginson corpus (see Appendix).

These utterances were then omitted from the later SRN stage of the study. By these criteria, the RAAM network learned to output terminals to 100% accuracy, but was less successful at accurately outputting non-terminal vectors (96%). Nevertheless, at 96% accuracy, the network was able to correctly encode and decode 98% of the training forms by using a nearest neighbor criterion. This meant that a possible terminal vector was tested for its distance to the set of known terminals, and then assumed to be the one it was closest to in terms of vector distance. So while not achieving 100% by the tolerance criteria, the network achieved an acceptable level of performance by the nearest neighbor criteria.

SRN-based generation

The results of the SRN experiments can be broken down into three parts: (1) an exploration of the relative difficulty of the carrying out the pot and sub-assembly strategies; (2) the advantage for language learning in having prior training on an object assembly task; and (3) the evidence for a trend in separation of functions as language learning becomes more complex.

In all of the results reported here, the data presented are averaged over 10 replications involving different random initial weight settings. In each case the SRN successfully learned to generate the RAAM representations of the action sequences. This was verified by using the RAAM network to decode the SRN output.

<Insert Figure 7 about here>

POT VS. SUB-ASSEMBLY

Greenfield et al. (1972) argued that one of the features that differentiates the pot strategy from the sub-assembly strategy is the need in the sub-assembly strategy to switch one’s perception of the role of an object from that of an actor in an action, to something acted upon. In the case of the pot strategy, each cup is manipulated individually, and added to the growing assembly, without the need for role switching. On the other hand, in the sub-assembly strategy, an assembly is created by one action, and then manipulated as the “actor” in the next action, thus having its role switched from *acted upon* to *actor.*

There is a developmental lag in the emergence of the sub-assembly strategy. We should therefore expect to see this lag reflected in some way in the ability of the SRN to learn each strategy. Figure 7 is a histogram representing an average of 10 replications of the average mean squared error (MSE) for the "Get" action in the sequence of actions for each strategy. As well as the pure pot and sub-assembly strategy, there is a number of possible mixed strategies: balanced, left, and right. These are so called because of the structure of the tree representing the end state (cf. the X-axis of Figure 7). The "Get" action is graphed because it is the critical action that determines the role of the manipulated object. Results are given for networks trained to 500 and 1000 epochs.

A 2x5 (Training Phase x Strategy) analysis of variance with repeated measures on the Strategy factor, was carried out on the data in Figure 7. Both Training Phase and Strategy factors were statistically significant ($F(1,18)=395.04$, $p<0.001$; and $F(4,15)=11.5$, $p<0.001$, respectively), as was the Training Phase by Strategy interaction ($F(4,15)=5.29$, $p<0.01$). At the 500 epoch stage, the largest difference between strategies was between the pot and sub-assembly (one-tailed t=5.69, df=9, p<0.001). In fact the pot strategy was significantly different from all other strategies except the "mixed right" strategy. However, by 1000 epochs these differences disappear. This initial difference and its later disappearance is exactly what we would expect to see if the abstract model of the strategies captures some of the essential features of the empirical data.

<Insert Figure 8a & b about here>

INTERDEPENDENCE

One of the central hypotheses that this commentary aims to explore is that there is some computational benefit from constructing a language processing system on a pre-existing motor control system. To test this idea, four sets of 10 SRNs were subjected to different pre-conditioning regimes. The first was the condition of primary interest: A set of networks was trained firstly to generate a sequence of object assembly actions from a RAAM-encoded goal. Then, in addition, it was trained to generate speech actions from a set of RAAM-encoded speech goals derived from the simple language corpus. Finally, and also in addition, it was trained to do the same for the complex language corpus. Note that at each stage, the training corpus involved not only the new material associated with that phase, but also material from preceding phases.

In order to test the hypothesis that there is an advantage to prior object assembly training, we need a number of control conditions with which to compare our results. Three such control conditions were devised. The first involved initializing the network to random values that had the same distributional properties as the weights in the 10 networks trained for 1000 epochs on the object assembly task. These weights were found to have a normal distribution with a mean close to zero and a standard deviation of approximately 0.7. Ten networks were initialized to different random weight settings with these distributional properties.

The second control condition involved training the network on a task that was similar to the object assembly task in terms of the number of training patterns, the length of sequences, and so on. This task involved producing a vector of outputs which was the reverse of that used in the object assembly task. Therefore, the first element in the training vector for this *reverse* control condition was the last element in the corresponding object assembly training vector, and so on. It is important to note that this vector is undecodable and quite meaningless in RAAM terms. It does not, for example, represent the mirror image of the encoded tree structure. It is merely a vector with similar numerical characteristics to a genuine RAAM vector.

A third form of control involved training a set of 10 networks on the simple language task first, and then on the combined simple language and object assembly tasks. The motivation for this was to discover if there was something specific to the object-assembly task that provided a foundation for language generation. If this were the case, then there should be an asymmetry in the benefits from prior training on object assembly as compared to simple language, with object assembly providing the greater benefit.

Figures 8a and 8b show the average performance of the 10 SRN networks as a function of different pre-conditioning[1] regimes. In both graphs, the error bars define a 95% confidence interval (the more compact error bars tend to be obscured by the plot symbols). Figure 8a is a graph of the pattern of the average MSE over 10 replications following the addition of the simple language corpus. As might be expected, error is initially greatest for the random control condition. Early in training, object-assembly pre-training gives an advantage over the various control conditions. A noteworthy feature of this graph is how the error for pre-training on the simple language corpus remains relatively high throughout. Why this might be, will be taken up in the discussion. Furthermore, despite its structural similarity to object

[1] The term pre-conditioning is used here as distinct from pre-training, since the random condition does not involve training.

assembly training, the reverse control condition is consistently less effective as a pre-training regime up to the point of convergence with the random control.

Figure 8b gives the error patterns for the four forms of pre-conditioning following the addition of the complex language corpus. The networks are still disadvantaged from the random initialization, even this far into training (i.e., after 1000 epochs, if pre-training occurred). As one might expect, simple-language followed by object assembly pre-training is indistinguishable from object assembly followed by simple-language pre-training. Nonetheless, any pre-training involving object assembly is better than the other control conditions.

These results suggest, therefore, that the abstract characterization of the language and object assembly tasks captures some the features of their real-world counterparts. What is striking about the results is how beneficial object assembly training is, even when compared to control tasks that are very similar to it in structure. Possible reasons for this will be addressed in the discussion.

Differentiation

The third and final part of this analysis was to see if something like the functional differentiation that Greenfield has suggested occurs in Broca's region can also be the SRNs. This was done by examining the pattern of hidden unit activations of object assembly networks, firstly after combined object assembly and simple language training, and secondly after the addition of the complex language corpus. If a separation of function or specialization has occurred, we would expect to see this reflected in the pattern of hidden unit activations of these networks at the two stages. What is of particular interest is whether it is possible to detect an increase in representational distance between the object assembly tasks and language tasks as the language task becomes more complex.

If one were dealing with just one network, the obvious approach would be to carry out a cluster analysis of the hidden unit activations associated with each pattern set. However, because a *set* of networks is being examined, a cluster analysis is not feasible. Instead, the process was analyzed the following way. Two sets of 10 networks were of interest here: (1) those trained on object assembly and then on the simple language corpus (the "simple" set), and (2) those trained on object assembly and both simple and complex language corpuses (the "complex" set). Hidden unit vectors (50 in length) were collected from the simple set for the object assembly and simple language training patterns. From the complex set, hidden unit vectors were collected for the object assembly, simple language, and complex language training patterns. An average vector was calculated for each of these five sets of hidden unit vectors. At the end of this process there were 10 groups of five average vectors.

<Insert Figure 9 here about >

We can conceptualize the hidden unit vectors as specifying a point in high-dimensional space, and the set of vectors associated with a particular sequence of patterns, for example, those associated with the pot assembly strategy, as describing a trajectory through this high-dimensional space. If we take an average of these trajectories we can get some indication of the region in space which this particular set of trajectories predominantly occupies. We can then measure the Euclidean distance between this and other regions of representational space. The hypothesis under discussion here is that there should be an increase in distance between the object assembly region and language region as the language task demands increase. The main way to test this hypothesis is to see if the Euclidean distance between OA and SL varies as a function of language training complexity. The pattern of distance measures is illustrated in Figure 9. The difference between the OA-SL distance and the OA'-SL' distance is statistically significant in the predicted direction (one-tailed, matched pair t=7.27, df=9, p<.001). The other distance measures shown in Figure 9 are provided to illustrate that the measure as a whole behaves consistently. So, for example, the distance between SL' and CL, as one would expect, is very small. While the difference between OA' and CL is the largest of all, indicating that the distance measures are indeed behaving consistently.

How does this distance measure relate to Greenfield's observation of a physical differentiation of function in Broca's region? The relationship is actually a fairly direct one. When Euclidean distance increases between two hidden unit vectors, one finds that the active units in vector A, say, tend *not* to be the ones that are active in vector B. This is directly analogous to the process of spatial separation of function that Greenfield proposes occurs in Broca's area during its development.

Discussion

This commentary has shown that by using an abstract connectionist characterization of language processing and object manipulation one can demonstrate a pattern of development similar to that posited to occur during the emergence of spoken language. The simulations described here indicate a computational advantage for networks that have had prior training on a simulated object assembly task, when compared with various control conditions. This is taken as support for Patricia Greenfield's view of a functional homology underlying both language and object assembly tasks.

A possible criticism is that the style of representation used for language and action goals, namely the RAAM, is not sufficiently biologically realistic. Such a view, however, indicates a misunderstanding of the modeling process. The aim of any form of computational model is to abstract from the target phenomenon those features deemed theoretically relevant. As pointed by Green (1998), it is important to be as explicit as possible in identifying the theoretically relevant components and processes of one's model. In the case of the model described here, one important relevant feature is the concept of a superposed, distributed representation. There are many varieties of such representation, the RAAM being just one. The main test of the RAAM's plausibility is not whether its precise neurobiological equivalent can be found, but whether it affords the same kind of operations that its neurobiological counterpart does, whether it succeeds or fails in similar sorts of ways. Given that the main evidence used in evaluating the model is based on error patterns during learning, then another relevant feature of the model is that its representations change during learning in ways that are similar to their biological counterparts. In other words, both the static and dynamic properties of the model's representations should map onto their real counterparts at some level of abstraction.

Another possible criticism is that because of the structural similarity between the motor and language RAAM representations, the simulation results are in some way built into the model from the start. This criticism to a large extent misses the point. Greenfield's premise is that the neural substrates for language production and object assembly are *initially* one and the same. It is therefore reasonable to assume that the basic representational building blocks of both action plans and utterance plans are similar, if not identical. However, as the language becomes more complex, there is a corresponding divergence in the structural complexity of the representations. What is of importance from a modeling point of view in the simulation is not the starting point, but the manner in which the representational space evolves during training under pressure from increasingly more demanding language processing requirements. Furthermore, and as will be discussed in more detail below, the way in which the model's behavior varies under the different control conditions is perhaps its most informative aspect.

To understand why the object assembly task provided a significant training advantage, it is useful to look at the third control condition, where prior training was on simple language and then followed by object assembly. In this case, the language training did not benefit the object assembly task. This suggests that the training advantage provided by initial training on object assembly may be another case of the "importance of starting small" in the sense of Elman (1993). The tree structures associated with the object assembly task are relatively simple, with few deep embeddings. The simple language structures are more complex in this respect. Elman (1993) demonstrated in his grammar-learning studies that if a complex grammar is to be learned by an SRN, the network must first be trained on a simpler version of it. In the model described here, when initial training was provided on the language task, the network was being trained on a complex task first and then a simpler one, the reverse of the approach that Elman found to be effective. It is unsurprising, therefore, that training is retarded on the combined object-assembly and simple-language training set.

A key evaluation criterion for any model, computational or otherwise, is whether the chosen abstraction, the functions it performs, and the outcomes it produces, can as a whole be plausibly mapped onto the target phenomenon. In this respect, it is argued that the model described here has been successful. In the first place, the model reproduced the relative difficulty that children have in producing the pot and sub-assembly strategies. Second, there appeared to be a divergence in the exploitation of hidden-unit space as the demands of the language task increased. Earlier, two possible divergence scenarios were suggested, one in which Broca's area differentiated according to a genetically determined schedule, another in which the differentiation occurred as a side-effect of increasing task demands. The model presented here suggests that task demands, of themselves, provide an impetus for the restructuring that may occur in the region. This suggests that environmental factors may play a greater role in differentiation than allowed for by Greenfield.

In summary, the evidence from the modeling described here supports the argument that there are good computational reasons for building a language production system on an object-assembly foundation.

Acknowledgments

This work was completed while the author was a Fellow at the Netherlands Institute for Advanced Studies, Wassenaar, The Netherlands.

References

Dean, J., & Cruse, H. (1995). Motor pattern generation. In M. I. Arbib (Ed.), *The handbook of brain theory and neural networks.* Cambridge, MA: MIT Press.

Dell, G. (1988). Retrieval of phonological forms in production: Tests of predictions from a connectionist model *Journal of Memory and Language, 25,* 124-142.

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

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

Fox, P., Petersen, Posner, M., & Raichle, M. (1988) Is Broca's area language-specific? *Human Neurobiology, 38,* 1-172.

Green, C.D. (1998). Are connectionist models theories of cognition. *Psycoloquy, 9.* ftp://ftp.princeton.edu/pub/harnad/Psycoloquy/1998.volume.9/ psyc.98.9.04.connectionist-explanation.1.green.

Greenfield, P. (1991). Language, tool and brain: The ontogeny and phylogeny of hierarchically organized sequential behavior. *Behavioral and Brain Sciences, 14,* 531-595.

Greenfield, P., Nelson, K, & Saltzman, E. (1972). *Cognitive Psychology, 3,* 291-310.

Higginson, R. (1985). *Fixing-assimilation in language acquisition.* Unpublished doctoral dissertation, Washington State University.

Houghton G., (1990). The problem of serial order: A neural network model of sequence learning and recall. In R. Dale, C. Mellish, & M. Zock (Eds.), *Current research in natural language generation.* London: Academic Press.

Lashley, K.S. (1951). The problem of serial order in behavior. In L.A. Jefress (Ed.), *Cerebral mechanisms in behavior.* New York: Wiley.

MacWhinney, B. & Snow, C. (1990). The child language data exchange system: An update. *Journal of Child Language, 17,* 457-472. .

MacWhinney, B. (1993). *The CHILDES Database: Second Edition*. Dublin, OH: Discovery Systems.

Mamier, G. & Wieland, J. (1996). *Stuttgart Neural Network Simulator*. Institute for Parallel and Distributed High Performance Systems, University of Stuttgart.

Pollack, J. B. (1990). Recursive distributed representations. *Artificial Intelligence, 46,* I77-105.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning internal representations by error propagation. In D. E. Rumelhart, J. L. McClelland, & The PDP Research Group (Eds.), *Parallel distributed processing. Explorations in the microstructure of cognition. Volume 1: Foundations.* Cambridge, MA: MIT Press.

Rumlehart, D. E., & Norman, D. (1982). Simulating a skilled typist: A study of skilled cognitive-motor performance. *Cognitive Science, 6,* 1-36.

Sejnowski, T. J. (1986). Open questions about computation in the cerebral cortex. In J. L. McClelland, D. E. Rumelhart, & The PDP Research Group (Eds.), *Parallel distributed processing. Explorations in the microstructure of cognition. Volume 2: Psychological and biological models.* Cambridge, MA: MIT Press.

Stemberger, J. P. (1983). *Speech errors and theoretical phonology: A review*. Bloomington: Indiana University Linguistics Club.

Van Essen, D.C., Anderson, C.H., & Olshausen, B.A. (1996). Dynamic routing strategies in sensory, motor, and cognitive processing. In C. Koch, & J.L. Davis (Eds.), Large-scale neuronal theories of the brain. Cambridge, MA: MIT Press.

Appendix: Language Corpus

The language sample is taken from the Higginson (1985) corpus of the CHILDES database (MacWhinney, 1993). The original corpus was sampled randomly. The omitted utterances were not RAAM encodable, and were not used in SRN training. The simple and complex corpus comprised 14 utterances each, though in the case of the simple corpus there was some repetition of utterance types in the sample drawn.

Simple

Dada, Nana, uh, baba, baby, um, duck, Sue, Cleo.

Complex

read; cuckoo; baby; us; piggy; oh a story; a crayon; yacht; dollar; no; colour crayons; what's he have; do a puzzle; look 't I found;

Omitted

kitties are in the back; might get all 'n gooey.

Tables

Table 1

A set of training patterns for a simple RAAM network to encode the tree (A (B C)). The terminals A, B, and C are represented as 1-in-*k* bit vectors, and the R_n vectors comprise real numbers.

input pattern	**hidden pattern**	**output pattern**
$(B\ C)$	R_1	$(B'\ C')$
$(A\ R_1)$	R_2	$(A'\ R_1')$

Figures

Figure 1

A comparison of the pot and sub-assembly strategies in the cup nesting task involving three cups. The gray arrows indicate the object manipulated (single cup or assembly), and where it was put. Black arrows indicate the resulting change in state.

Figure 2

Schematic representation of a model for generating sequential behavior from hierarchically organized goal representations. The goal structure corresponds to the RAAM representations used in the model, and the response generator corresponds to the SRN used to produce action sequences.

Figure 3

An illustrative example of how to encode a binary tree using an 8-4-8 RAAM network. The outputs A', B', and C', and R1' are used to indicate that these are approximations within a certain tolerance of A, B, C, and R1. R2 can be decoded by placing it on the *hidden* units of the encoder network and recursively decoding the resulting output.

Figure 4

A schematic representation of an SRN used to generate action sequences from RAAM-based goals. Note that the output of the SRN was also a RAAM representation. The context units contain the activation values of the hidden units at the previous time step and serve as limited capacity memory for the network.

Figure 5

Goal-state representations for (a) a pure pot strategy; (b) a pure sub-assembly strategy; and (c) a mixed "balanced" strategy.

Figure 6

An illustration of the generation of a sequence of speech actions from the RAAM representation of the utterances [dadA]. An onset-rhyme format is used to represent the structure of the syllables. Empty slots in the RAAM structure are indicated by "0". For multi-word utterances, the syllable structure is embedded in a simple phrase structure.

Figure 7

Comparison of the error on the “Get” action for different assembly strategies after 500 epochs and 1000 epochs of training. These data are an average of 10 trials.

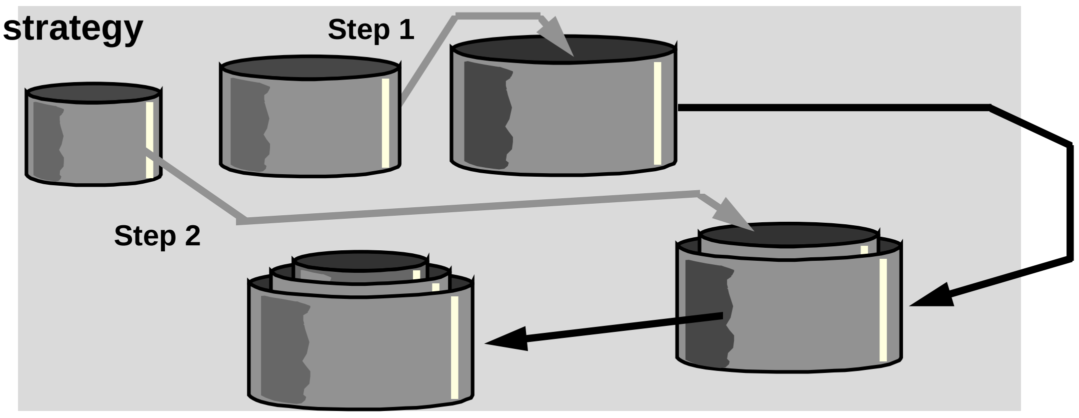
Figure 8a & b

Comparison of effects of different forms of pre-conditioning on ease of learning of (a) the simple language corpus and (b) the complex language corpus. Note that the training sets are cumulative, incorporating the training sets (if any) from earlier stages. Error bars indicate a 95% confidence interval. OA=object assembly and SL=simple language.

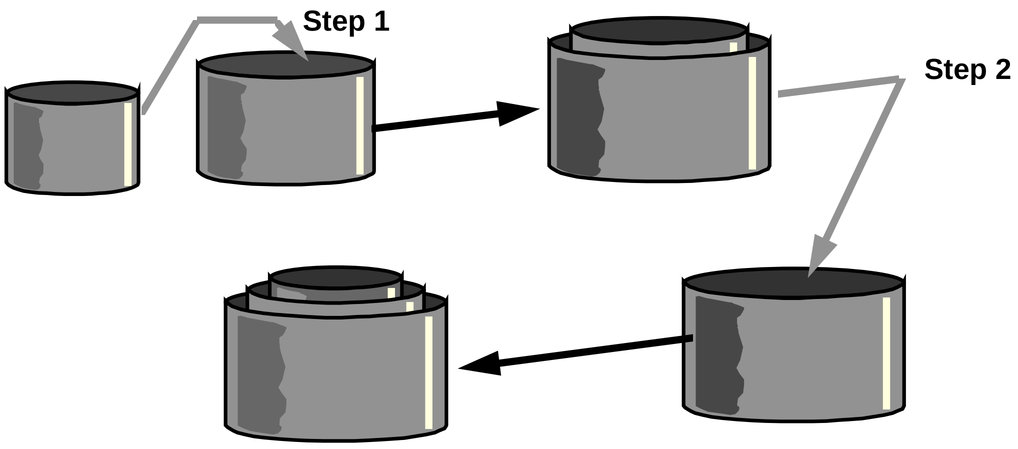
Figure 9

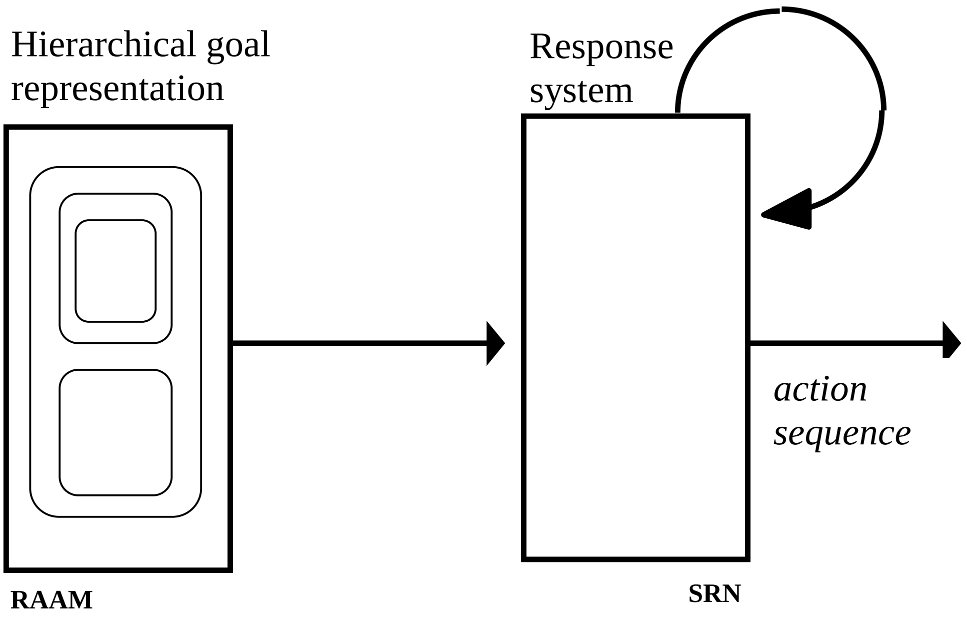
Schematic representation of distances between average vectors for different training sets in simple and complex language contexts. Note that the lines are not to scale. OA = average vector for object assembly training patterns in networks trained on simple language corpus; . OA’ = object assembly in networks trained on simple and complex language corpuses; SL = simple language training patterns in networks trained on simple language corpus; SL’ = simple language training patterns in networks trained on the simple and complex language corpuses; and CL = complex language training patterns in networks trained on simple and complex language corpuses.

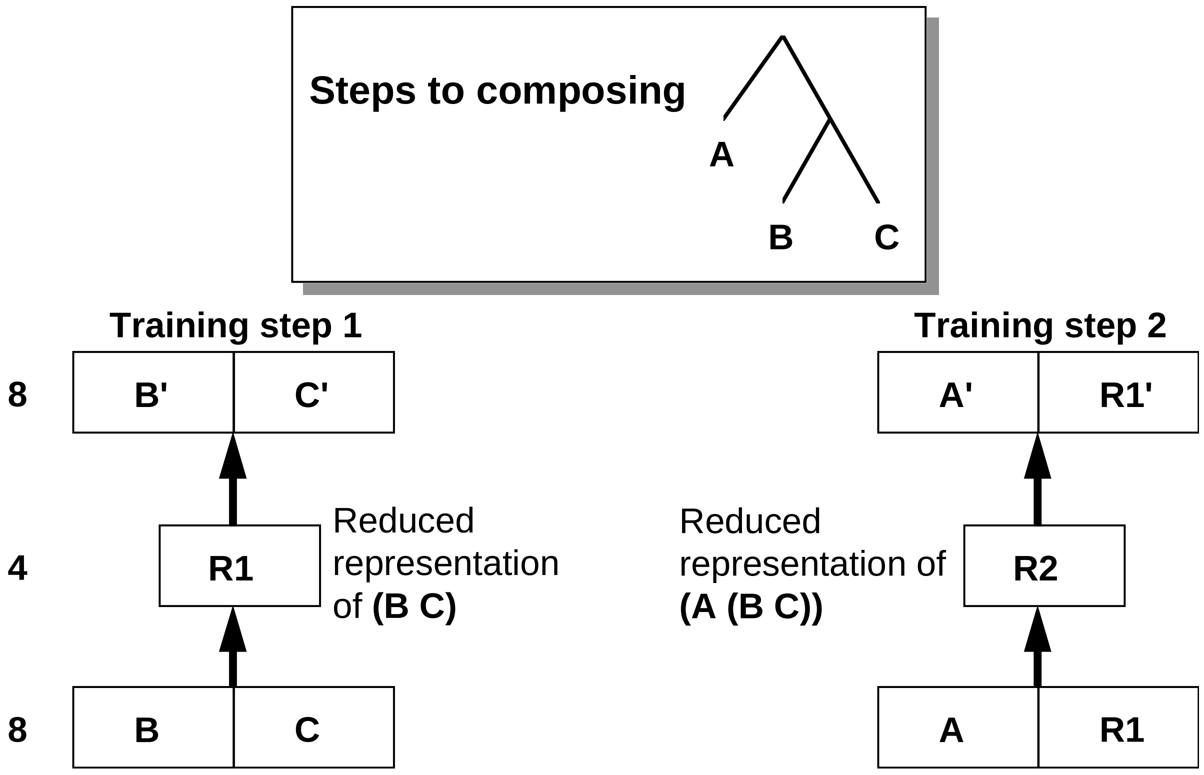
Pot strategy

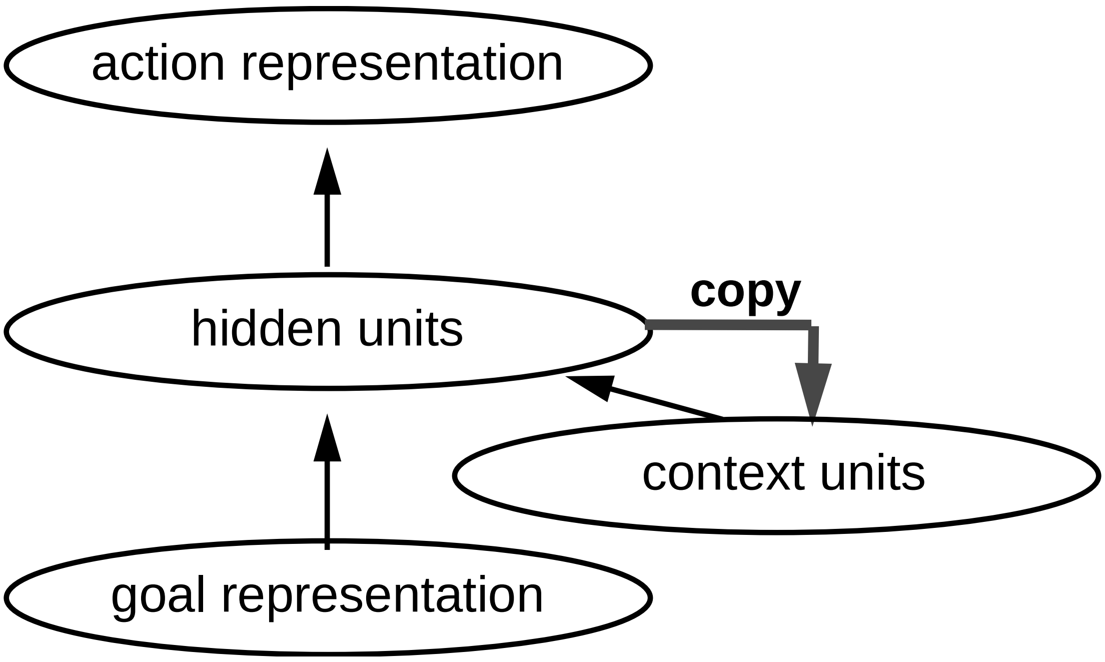


Sub-assembly strategy

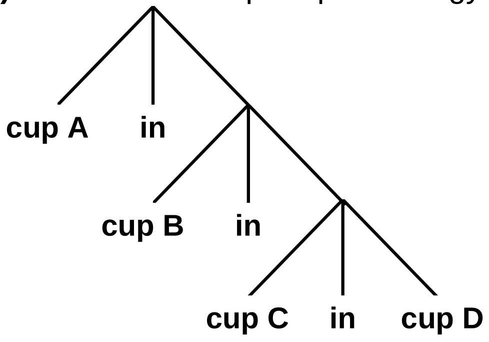



Hierarchical goal representation
Response system
action sequence
RAAM
SRN

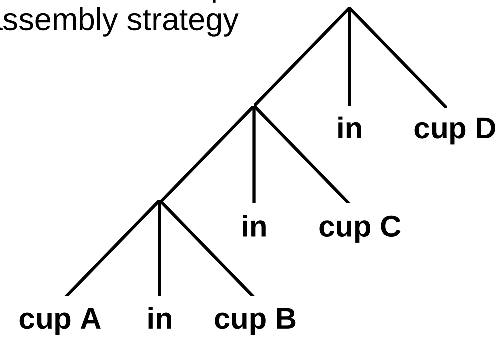

Steps to composing
A
B
C
Training step 1
8
B'
C'
4
R1
Reduced representation of (B C)
8
B
C
Training step 2
A'
R1'
Reduced representation of (A (B C))
R2
A
R1


action representation
copy
hidden units
context units
goal representation

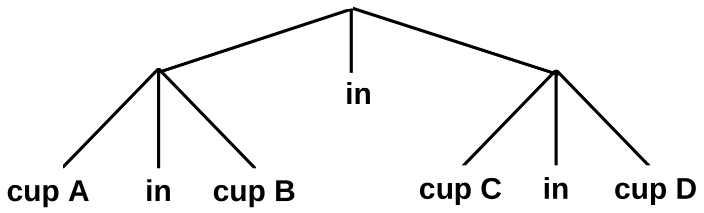
(a) Goal-state for a pure pot strategy

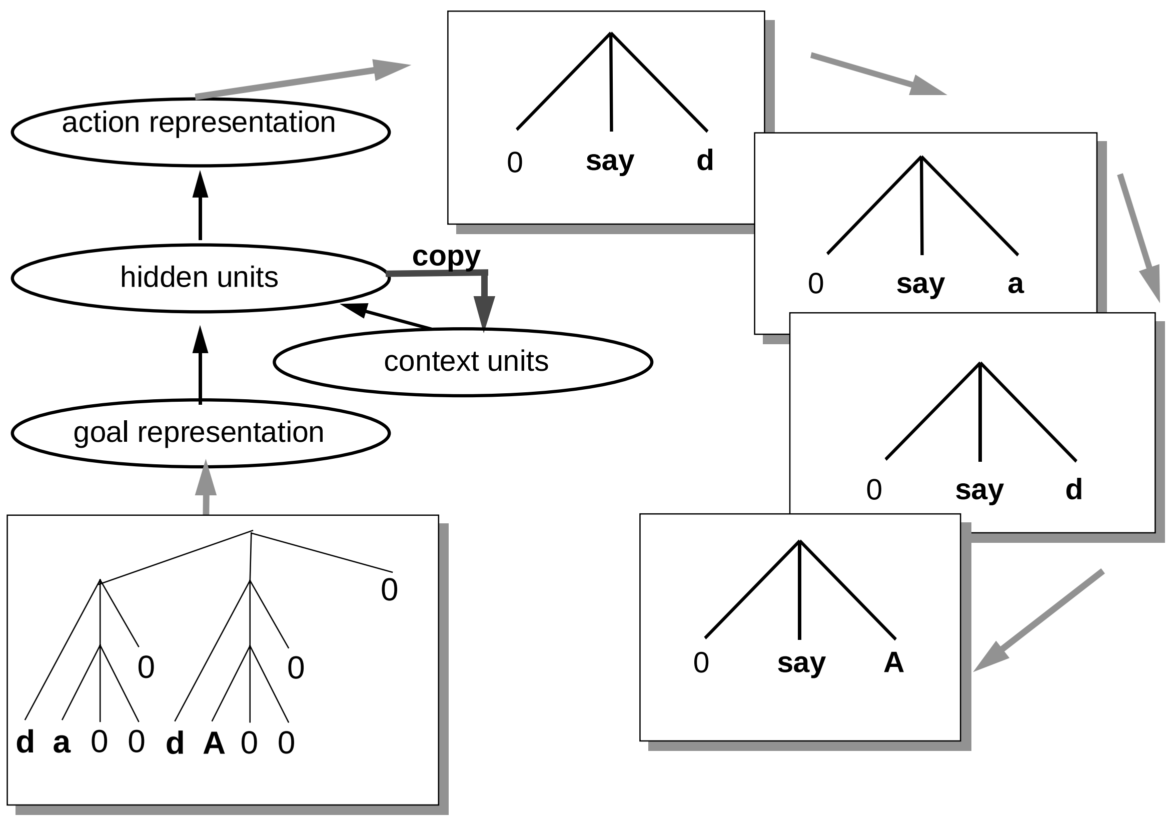


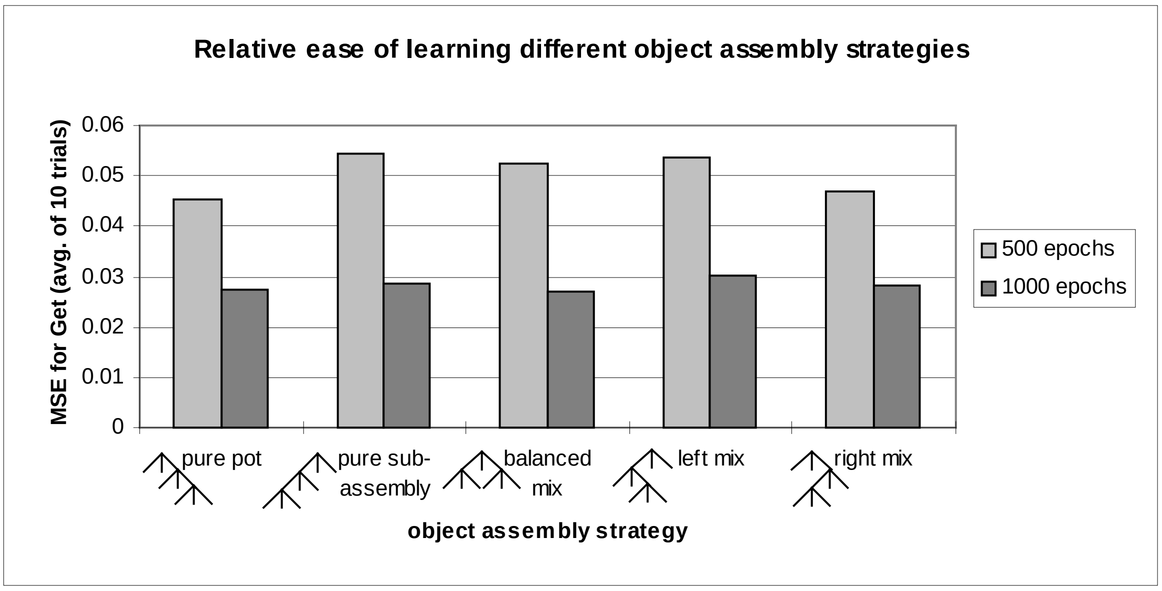
(b) Goal state for a pure sub-assembly strategy

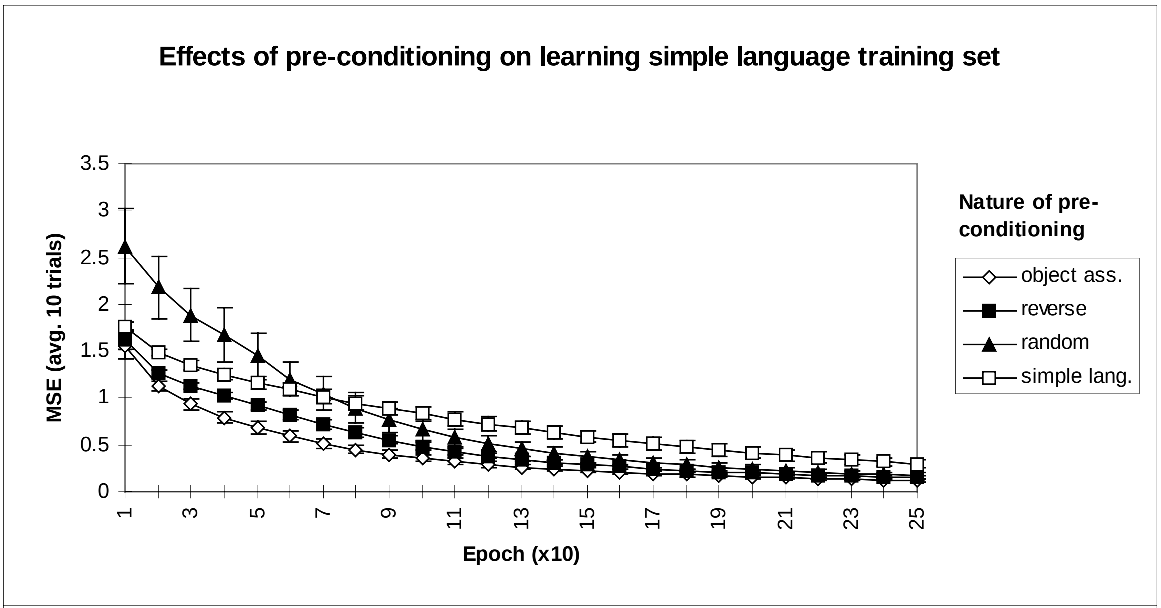


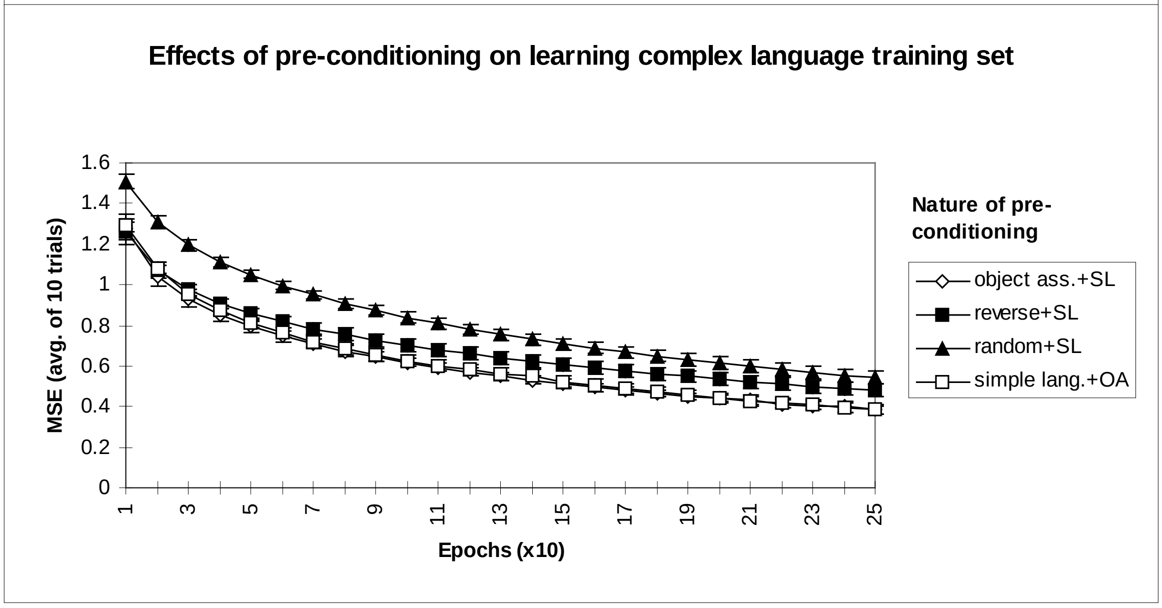
(c) Goal state for a mixed strategy

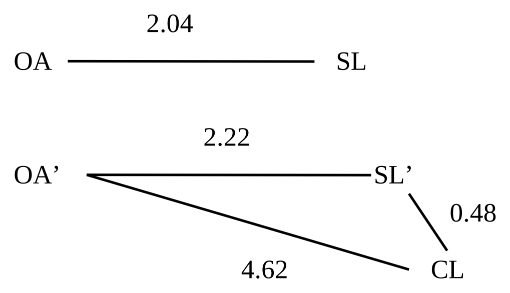



action representation
hidden units
copy
context units
goal representation
d a 0 0 d A 0 0
0
0
0
0 say d
0 say a
0 say d
0 say A


Relative ease of learning different object assembly strategies
MSE for Get (avg. of 10 trials)
0.06
0.05
0.04
0.03
0.02
0.01
0
pure pot
pure sub-assembly
balanced mix
left mix
right mix
object assembly strategy
500 epochs
1000 epochs


Effects of pre-conditioning on learning simple language training set
MSE (avg. 10 trials)
3.5
3
2.5
2
1.5
1
0.5
0
1
3
5
7
9
11
13
15
17
19
21
23
25
Epoch (x10)
Nature of pre-conditioning
object ass.
reverse
random
simple lang.


Effects of pre-conditioning on learning complex language training set
MSE (avg. of 10 trials)
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
1
3
5
7
9
11
13
15
17
19
21
23
25
Epochs (x10)
Nature of pre-conditioning
object ass.+SL
reverse+SL
random+SL
simple lang.+OA


2.04
OA
SL
2.22
OA'
SL'
0.48
4.62
CL