Rule-learning in recurrent neural networks[*]

W.H. Zuidema
student CKI, nr. 9340424
Theoretical Biology, Utrecht University[†]

December 15, 1999

1 Introduction

Opponents of connectionist modeling often argue that neural networks are incapable of inferring "rules" from data. Gary Marcus (1999), for example, shows that seven months old infants can infer simple rules from an artificial language, and claims that neural networks are incapable to do the same. This failure is explained, according to Marcus, because such rules require generalization beyond the "training space", which networks simply cannot provide.

This analysis is very questionable. Marcus fails to provide clear definitions of "rule" and "training space". He induces a very general claim from a very limited experiment and he fails to explain the apparent contradiction with theoretical results, such as the universal function approximation properties (Cybenko, 1989) and, similarly, the Turing machine properties of neural networks (Pollack, 1988, cited by Elman 1991).

It is, however, indeed difficult to obtain neural network behavior that, intuitively, would be described as "rule-based". Interesting studies on this issue have been performed by Jeff Elman (i.e. 1991) and John Batali (1994, 1997), using simple recurrent neural networks. These networks are similar to standard feed-forward neural networks, with an additional set of recurrent nodes that feed to the hidden units like the input nodes, and themselves obtain their values from the hidden units. This recurrent loop allows the network to use information about input provided one or several time steps earlier (see figure 1).

Results from these studies show that recurrent neural networks can in fact induce "rules" from training data. In these experiments the simple recurrent networks learn, using the back-propagation algorithm and several additional didactics, to recognize simple formal languages. Formal languages are an excellent domain for such studies, as they form the archetypes of rule-based systems, can easily be classified, and are, in many ways, relevant for the analysis of natural language.

The mini-project reported in this paper, reproduces some of the results of Batali (1994) and Elman (1991). It aims at better understanding rule induction in neural networks, and at obtaining some intuition on the reproduction of results in formal languages (such as Zuidema, 1999) in a connectionist framework. This paper by no means tries to model cognitive phenomena.

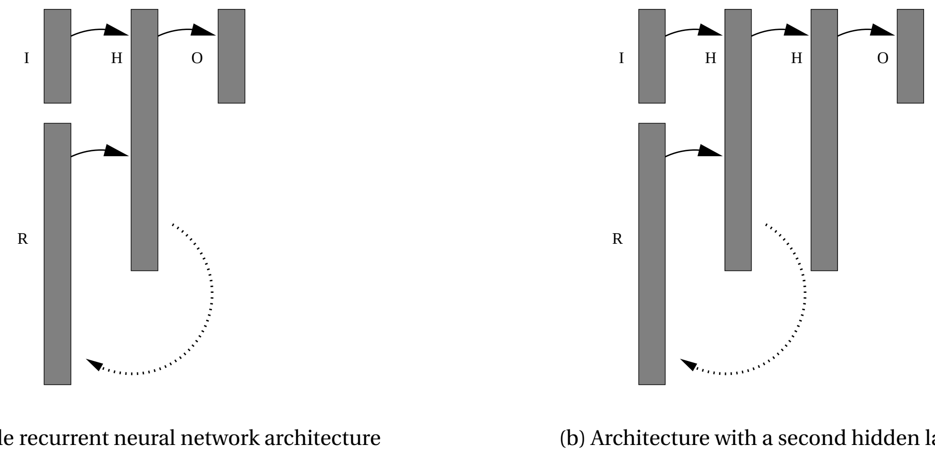
2 Experimental setup

2.1 architecture

The architecture used in the experiments reported in this paper, is the "simple recurrent neural network" (Elman, 1991). The network consists of four types of nodes: input nodes, hidden nodes, output nodes and recurrent nodes. The input layer and the recurrent layer feed to the hidden layer; the hidden feeds to

[*] mini-project "connectionisme" supervised by Dr. Jaap Murre
[†] Padualaan 8, 3584 CH Utrecht, the Netherlands, http://www-binf.bio.uu.nl/~jelle, jelle.zuidema@phil.uu.nl

the output layer, all in a fully connected manner. The recurrent nodes obtain the activation of the hidden nodes; they form a copy of the hidden layer in the previous time step. An alternative architecture, used in some of the experiments, contains an extra hidden layer, between the first hidden layer and the output layer.



(a) The simple recurrent neural network architecture

(b) Architecture with a second hidden layer

Figure 1: *Architectures used in the experiments. Full arrows indicate weighted, full connectivity. Broken arrows indicate the recurrent connection, which consists of copying the hidden nodes' activations to the recurrent layer.*

Activation from the input layer and recurrent nodes is propagated through the network in the standard way. Activation of the hidden nodes is thus given by:

$$O_i^H = f\left(\left(\sum_j O_j^I W_{ij}^H\right) + \left(\sum_j O_j^R W_{ij}^R\right) - \theta_i\right) \quad (1)$$

$$O_i^O = f\left(\sum_j O_j^H W_{ij}^O) - \theta_i\right) \quad (2)$$

Where θ is a threshold value, and f is a simple sigmoidal function, given by:

$$f(x) = \frac{1}{1+e^{-x}} \quad (3)$$

All weights and thresholds are initialized with some random value between $-\alpha$ and $+\alpha$ (here: 4.0). Weights are updated using the "back-propagation of error" algorithm. A "delta-value" is attributed to the output and hidden nodes:

$$\delta_i^O = O_i^O\left(1 - O_i^O\right) error_i \quad (4)$$

$$\delta_i^H = \sum_j \left(O_i^H\left(1 - O_i^H\right) W_{ij}^O \delta_j^O\right) \quad (5)$$

Where $error_i$ is given by the difference between O_i^O and the target value. Statistically, in some domains $\pm error^2$ is a more appropriate error-measure, but it is not (yet) implemented here.

The values of the weights are changed, proportional to the delta-values, the connected node's activation and a learning rate r (here: 0.1 or 0.05). The thresholds are updated proportional to the learning rate and the delta values:

$$W^O_{ij} = W^O_{ij} + r\delta^O_i O^H_j \quad (6)$$
$$W^H_{ij} = W^H_{ij} + r\delta^H_i O^I_j \quad (7)$$
$$W^H_{ik} = W^H_{ik} + r\delta^H_i O^R_k \quad (8)$$
$$\theta_i = \theta_i - r\delta_i \quad (9)$$

The implementation allows for switching off the recurrent connections. As a test, under these circumstances the network was trained to fit a simple 2D function and showed excellent fit after several hundreds of training examples. In recurrent mode, the network received repeatedly symbols from a string "010010", and trained to predict the next symbol. Also here, the network showed perfect predictions after several thousands of steps.

2.2 training data

The training data consists of strings from a simple formal language 0^n1^n. Those strings thus consist of a sequence of 0's and a sequence of 1's of equal length. This is probably the simplest example of a context free language, because it needs at least a context free grammar to generate it. For instance:

$$S \mapsto 01 \quad (10)$$
$$S \mapsto 0S1 \quad (11)$$

(This grammar is context free, because the right-hand side of the second rule has a terminal symbol both to the left and to the right of the nonterminal symbol. Regular grammars can have the terminal symbols either to the left or to the right, consistently over the whole grammar.)

Following Batali (1994), we trained a recurrent neural network with strings from this language, with the string length ranging from 4 to 26. Symbols of a string are presented sequentially to the network. The three input nodes are set to $\langle 1,0,0\rangle$, $\langle 0,1,0\rangle$ and $\langle 0,0,1\rangle$ for the symbols 0, 1 and X ("space") respectively, where the space is used both to indicate the beginning and the end of a string. The output nodes represent in the same way the prediction for the next symbol in the string. Output values range between 0 and 1; the highest value is taken to be the networks prediction. The network is trained using the target vector (as above) and the back-propagation algorithm.

In some of the experiments reported here only one string from this language was presented. In other experiments an alternative context free grammar was used.

2.3 performance measures

We will discuss the networks performance using two simple error measures. The first is the "root mean squared error" of the output nodes; that is, the square-root of the average squared distance between an output nodes' activation and the target value (0 or 1). The second is the average prediction error; that is, the number of times the highest output value does not correspond to the target, divided by the number of symbols presented to the network.

To analyze what function from input to output the network has actually learned, we also occasionally analyze the weights and activations of the nodes in the network. As in formal grammars, we can systematically study the set of strings the network can produce or accept. The procedure to find a network's "derive language" can be as follows:

```
1. initialize all activations to zero
2. present X to the network
3. propagate
4. choose a symbol 0,1 or X with probabilities
   proportional to the corresponding output values
```

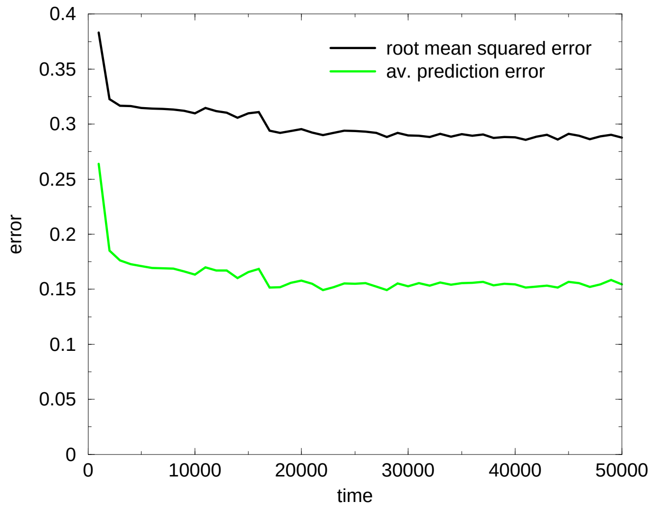
```
5. present that symbol to the network
6. repeat step 3-5 until an X is chosen or
   the maximum string length is reached.
7. repeat step 1-6 many times
```

Similar to the analysis one can apply to formal grammars and their languages, one can measure the expressiveness (the number of distinct strings), compositionality (non-random repetitions of substrings between strings) and recursiveness (non-random repetitions of substrings within strings) of such languages (Zuidema, 1999). However, as it turns out, learning performance is so poor in the tasks of this report, that these languages seem to reflect more randomness than formal regularities.

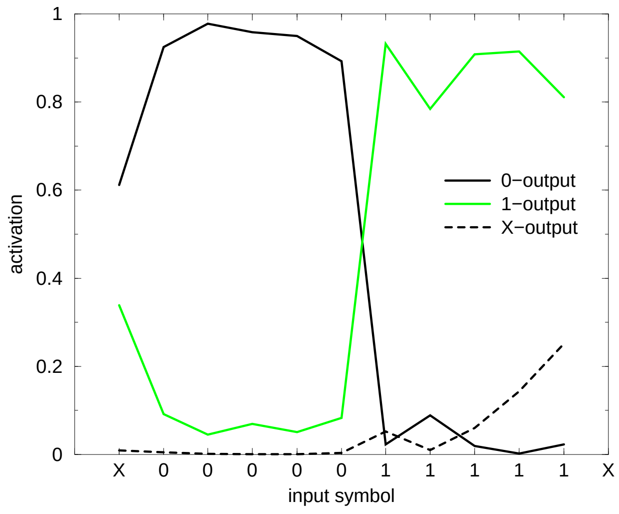
3 Results & Discussion

3.1 full set of strings

Results of training on the full set of strings, ranging in length from 4 to 26, are rather disappointing, as was reported by Batali (1994). After an initial period of around 20,000 strings in which performance gets better, the network settles in a suboptimal, local maximum, where the prediction error is around 15% (see figure 2).



(a) Performance of the network on the full language



(b) Output node activations for one example string

Figure 2: *The behavior of a simple recurrent network when trained on a language* 0^n1^n*, where* $2 < n \leq 13$*. (a) Both error measures converge to a suboptimal value within 20,000 time steps (one string per time step). The average prediction error is close to the prediction error one expects for a trivial solution. (b) This trivial solution, returning the input as output, is indeed what one can observe in an example network, after being trained on 100,000 strings. As one can see, the X-output activation does rise towards the end of the string.*

85% accuracy is a typical outcome of a successful machine learning task. Batali (1994) reports that combining genetic optimization and back-propagation led to an decrease in error to around .12. At first reading, this appears to be a nice result. However, as one can see in figure 2(a), the behavior of our network after 100,000 training examples is rather trivial. It simply returns the input it receives, and shows no expectancy of the "0 to 1" transition, and hardly any for the end of the string.

The transition in the center of all strings of course is "fundamentally" uncertain; it occurs somewhere between the third and 13th symbol, but the 0's at the beginning contain no information about its position. The end of string, though, can be predicted correctly. This gives us two non-trivial "events" when processing a string. With an average string length of 14, trivial solutions thus produce an average prediction error of $\frac{2}{14} \approx 14.4$. Non-trivial solutions should approximate $\frac{1}{14} \approx 7.2$, which is the upper bound for performance.

Batali's "brute force" combination of a genetic algorithm and back-propagation thus yields solutions that are slightly better than trivial, but still far from accurate. His report shows some graphs like figure 2 of networks that do correctly predict the end of a string, but the high prediction error seems to suggest that these are in fact exceptions.

3.2 inductive bias

The behavior of a learning algorithm can be described with its *inductive bias* (Mitchell, 1997). Any learning algorithm has such a bias; the challenge of machine learning is to use it as a positive tool. The bias can be divided in two components: the representation bias and the search bias.

Representation bias refers to which solutions are possible in the representational language of an algorithm. Solutions that cannot be represented can obviously not be learned. Search bias refers to the inherent ordering in which these possible solutions are evaluated.

Simple feed-forward neural networks have been shown to be universal function approximators, if sufficient hidden units are present. Even if the number of hidden units is relatively small, the representational power of multiple layer neural networks is huge. Representation bias therefore, is seldom a problem. Search bias is more problematic, because known learning rules frequently get "stuck" in local optima.

To understand the failure of learning the full language, we first study the way in which the network can represent information of the sort necessary for this task, by teaching a single word to the network. Thereafter, we will discuss some possible set-ups to provide a different search bias.

3.3 learning a single word

We trained the same network a before with only one string, "00001111", and evaluated its behavior. This task is easy, and the network finds a perfect solution within 2000 - 5000 training samples. There are many possibilities for the network to accomplish this task. In any such solution, the recurrent connections must implement some kind of counter, that counts how many 0's and 1's are already processed.

It's not hard to envision such a counter. For instance, a hidden unit with large negative weights connecting it to the 1 and X-input node, and a small positive weight connecting it to the 0-input node, and a large positive weight connecting it to its recurrent counterpart. With such a setup, the recurrent node's activation increases slightly every time a 0 is presented to the network.

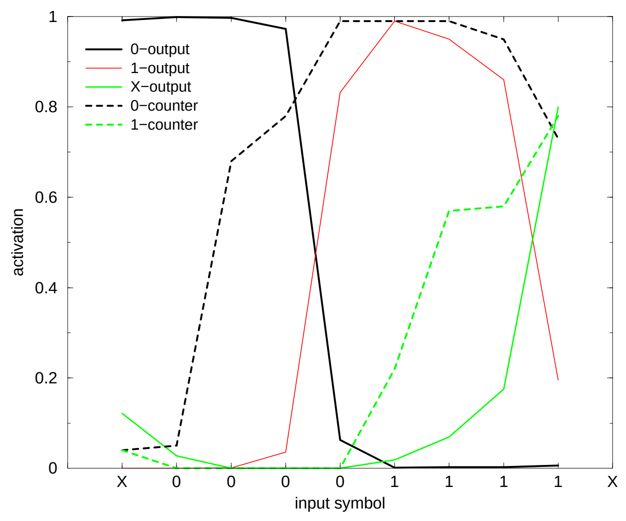
If one has a 0-counter and a 1-counter, the end of the string can be predicted if both values reach equal levels. This can be done by subtracting the 1-counter's activation from the 0-counter's activation, and detecting whether it is above (output 0 or 1) or below (output X) a certain threshold. Interestingly, this seems to be the solution learned by the network (see figure 3).

3.4 alternative search biases

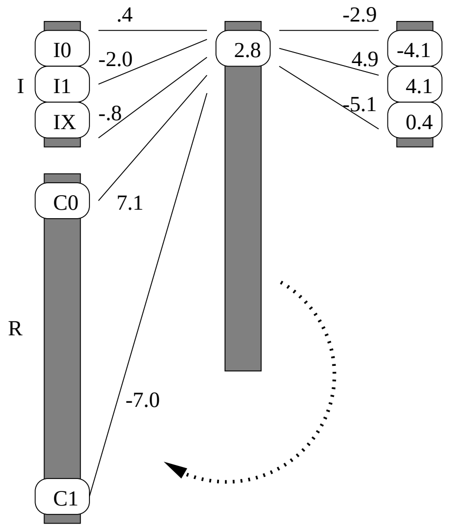
Elman (1991) showed that learning simple formal language is hard for simple recurrent networks, but can be facilitated providing a "beginning small" environment. He trained networks with a carefully designed "didactic" in which in four stages the training samples became more and more complicated. Learning was very successful under these circumstances. Next, he showed that the same results can be obtained by disturbing the recurrent nodes with decreasing frequency. Because the recurrent nodes form a memory, disturbing them destroys the temporal information and allows the networks to focus on the simple task.

We hypothesized that a similar mechanism could be beneficial for the network to learn the simple context free language of this paper. We designed several "didactics", such as starting with one word and increasing the language size slowly and starting with frequent disturbance of the recurrent node activations and decreasing in until no disturbance is left.

We trained the network with a range of different parameters (including the 4-layer architecture of figure 1(b)), but were unsuccessful to teach the network more than two words. The network in figure 4



(a) Activations of the output nodes and two recurrent nodes when presented the target string



(b) Some of the weights and thresholds of the solution the network learned

Figure 3: *The network successfully learns to predict the next symbol when trained and tested with only the string "00001111". (a) Some of the recurrent nodes function as counters for 0's and 1's. (b) The weights and thresholds of this network in fact implement the hypothesized solution (see text).*

successfully learned the string "0000011111". At time-step 60,000 the second string ("00001111") is introduced, leading to a rapid increase in error. Occasionally, the network performs pretty good, with a prediction error of 0.05, which is close to the upper bound for two strings of length 8 and 10. The solution is not very stable, though. After the introduction of the third string, the network never reaches a non-trivial level of performance.

An other, but related idea put forward in the literature (Hillis, 1991; Pagie and Hogeweg, 1997), is that "coevolutionary learning" can be successful, where learning with a fixed target fails. Success stories include Hillis' coevolution of sorters and sorting problems, and the backgammon program that (using reinforcement learning) beated the human world champion after having played millions of games with a copy of itself.

We studied the dynamics of a system with two recurrent networks, one of which, the "speaker" produces strings and the other, the "hearer", is trained to predict the first network's next symbol. We tried a variety of settings. For instance:

i. the speaker is trained to avoid being predicted by the hearer;

ii. the speaker is trained to increase predictability;

iii. speaker and hearer alternate roles;

iv. every round the hearer becomes the new speaker, and a untrained network becomes the new hearer.

The results from this work are far from conclusive. No recognizable formal language arises in any of the experiments. But the dynamics seem to be very rich and unpredictable. In fact, the set-up is very similar to the "coupled dynamical recognizers" studied by Takashi Hashimoto and Jordan Pollack. Their analysis is too complicated for the scope of this paper and the mathematical skills of the present author. We leave this as future work.

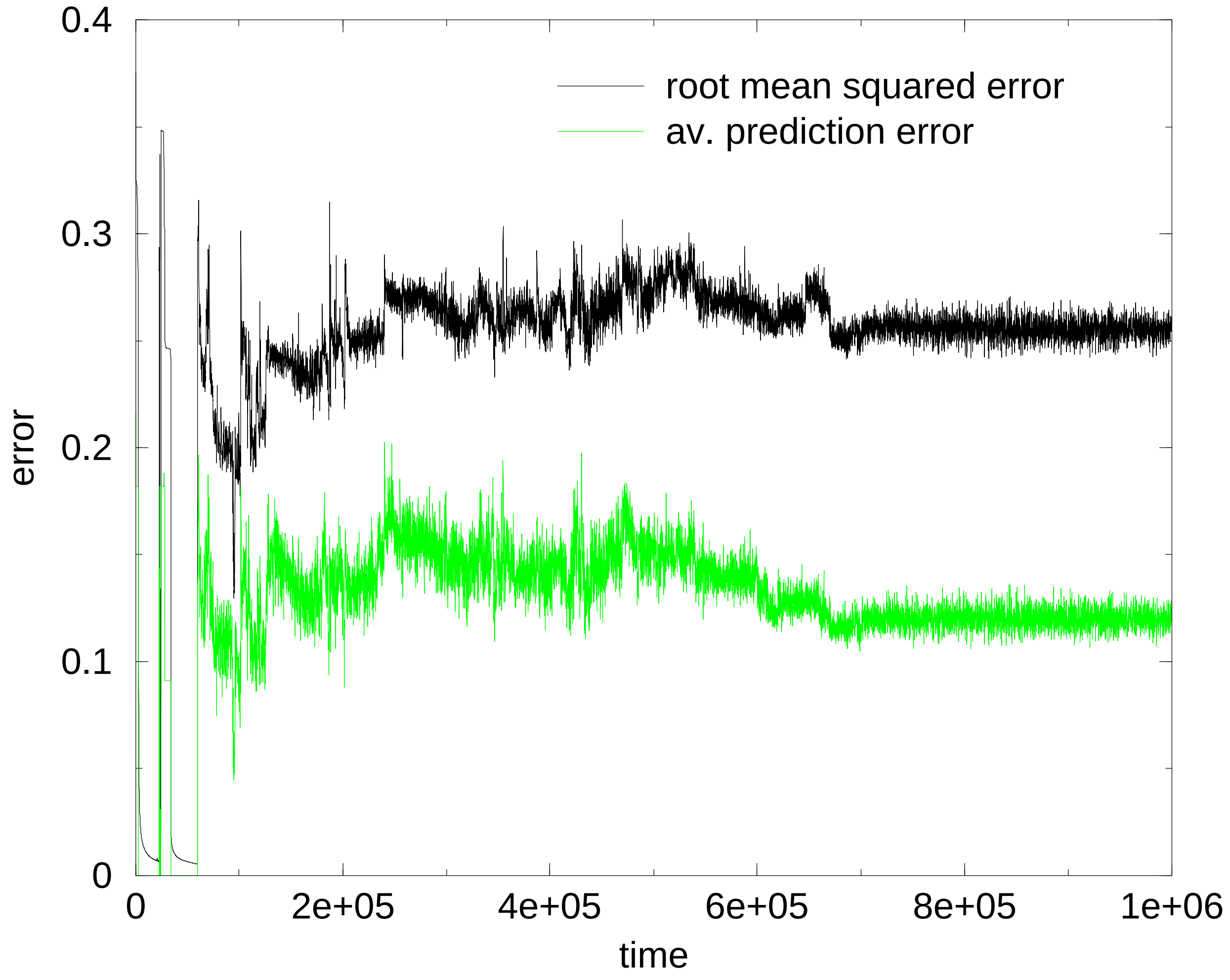


Figure 4: *Performance of the network when trained in an "beginning small" environment. Every 60,000 time-steps a new string is added to the training set. Only in the case of 1 and 2 strings a non-trivial level of performance is reached.*

4 Conclusion

We studied the behavior of simple recurrent networks when trained to learn a simple context free language. The most straight forward implementation fails to produce non-trivial solutions.

In trying to understand this failure, we identified two aspects of learning algorithms: the representation bias and the search bias. Analysis of the learning behavior if trained on only one string, indicates that recurrent networks are capable of representing the temporal information needed to recognize context free grammars. This is in agreement with other results (Elman, 1991; Batali, 1994).

Our efforts to find a different search bias that leads to better results, were unsuccessful. Previous work (Batali, 1994) with the same tasks, did not produce much better results. Maybe the particular task is simply very hard for the types of networks used in these experiments. While the target language is arguably among the simplest possible in the representations of formal language theory, it might be an unnatural and complicated task in the representational language of neural networks.

References

Batali, J. (1994) , In *Artifical Life IV.* (R. Brooks and P. Maes eds.), pp. 160–171, MIT Press

Batali, J. (1997) , In *Approaches to the evolution of language: social and cognitive bases.* (J. Hurford and M. Studdert-Kennedy eds.), Cambridge: Cambridge University Press

Cybenko (1989) , *?*

Elman, J. L. (1991) , *Machine Learning* **7**, 195

Hillis (1991) , *?*

Marcus, G. (1999) , *Science*

Mitchell, T. (1997) , *Machine learning,* McGraw-Hill

Pagie, L. and Hogeweg, P. (1997) , *Evolutionary computation* **5**, 401

Pollack, J. B. (1988) , In *Proc. of the Tenth Annual Conference of the Cognitive Science Society,* Lawrence Erlbaum

Zuidema, W. H. (1999) , *Evolution of grammar, a model study,* Master's Thesis (in prep.)