Explorations in Distributed Recurrent Biological Parsing

Terrence C. Stewart (tcstewar@uwaterloo.ca)
Peter Blouw (pblouw@uwaterloo.ca)
Chris Eliasmith (celiasmith@uwaterloo.ca)
Centre for Theoretical Neuroscience
University of Waterloo, Waterloo, ON, Canada N2L 3G1

Abstract

Our ongoing investigations into biologically plausible syntactic and semantic parsing have identified a novel methodology for processing complex structured information. This approach combines Vector Symbolic Architectures (a method for representing sentence structures as distributed vectors), the Neural Engineering Framework (a method for organizing biologically realistic neurons to approximate algorithms), and constraint-based parsing (a method for creating dynamic systems that converge to correct parsings). Here, we present some of our initial findings that show the promise of this approach for explaining the complex, flexible, and scalable parsing abilities found in humans.

Keywords: Neural engineering framework; parsing; localist representation; distributed representation; vector symbolic architectures; holographic reduced representation

Parsing with Traditional Localist Networks

Neural networks have often been explored as mechanisms for parsing language. In many of these approaches (e.g. Cottrell, 1985), a single connectionist node is created not only for each term in the language, but also for each possible usage of that term, leading to a combinatorial explosion of nodes (Figure 1). While these sorts of models provide accurate parsing and show some performance characteristics similar to humans (e.g. Waltz & Pollack, 1985), this exponential growth of components means that they would require more nodes than there are neurons in the human brain. This makes it difficult to see how such an algorithm could be instantiated within the brain.

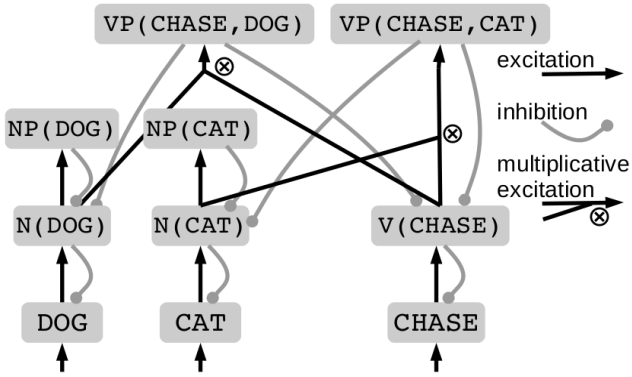

Figure 1: An example of a localist parsing network. A node exists for each possible combination of terms, leading to an exponential growth in resources required. The nodes for S(DOG,CHASE,CAT) and the three other possible sentences are omitted for clarity.

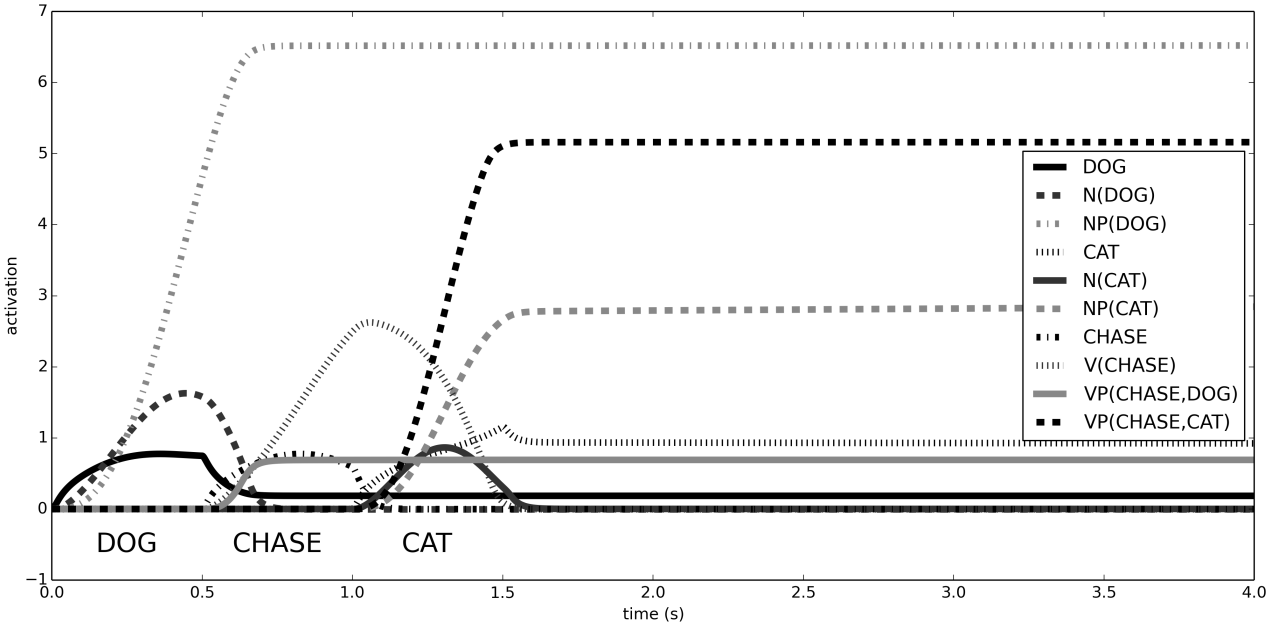

Figure 2: Activity level of each node in Figure 1 as the network is presented with the input DOG for the first 0.5s, CHASE for the next 0.5s, and then CAT for the next 0.5s. The result is the activation of NP(DOG) and VP(CHASE, CAT). The network can also successfully parse "DOG CHASE DOG", "CAT CHASE DOG", and "CAT CHASE CAT".

The network in Figure 1 is capable of parsing sentences from the following toy grammar:

S → NP, VP
NP → N VP → V N
N → DOG or CAT V → CHASE

Excitatory and inhibitory connections increase and decrease (respectively) the activity in the target node proportional to the activity in the source node. The multiplicative excitation connection increases activity based on the product of the activities in the two source nodes. The four nodes for the four possible sentences (S(DOG, CHASE, CAT), S(DOG, CHASE, DOG), and so on) are not shown, but are implemented similarly to the VP nodes, with connections to the corresponding NP and VP nodes.

Distributed Representation

As an alternative to localist representation, other approaches make use of a distributed representation. The idea is to represent content not as the activity of a single node, but rather each node has an equivalent set of numbers (a vector). For example, instead of a single node being active to represent DOG, this might be represented as the vector [0.1, -0.4, 0.7, 0.3, 0.2]. CAT would be another vector, and the presence of both terms would be represented as the sum of those vectors. The particular vectors used for each term might be chosen via some learning process that imposes similarity between vectors, such as DOG being more similar to CAT than it is to CAR. However, for the purposes of this paper we follow the standard process of randomly choosing these vectors.

With this approach, it is possible to re-describe any traditional localist model in a distributed manner. For example, we can take the nodes in Figure 1 and replace each one with a vector. To get the overall state of the system, we add together the vectors for each node, weighted by the activity level of that node.

To implement the connections, instead of using the activity of the source node, we must compute the *similarity* between the overall state vector and the ideal state vector for the source of the connection. Here, we use the dot product operation to compute similarity. So to implement an excitatory connection A → B, we take the state vector x and compute (x • A)B. The result is a vector indicating how much x should be changed. This can also be written mathematically as:

$$dx/dt = BA^T x$$

Importantly, the number of dimensions in the distributed representation's vector can be much less than the number of nodes in the localist representation, at the cost of a slight decrease in accuracy as the vectors slightly interfere with each other. This will work best when only a few of the nodes are active at any given time (i.e. when the distributed vector is formed by the combination of a few basic term vectors). Plate (2003) shows that with vectors with 1000 dimensions one can represent states with 8 nodes active out of a total of 50,000,000,000,000 nodes with 95% accuracy. The distributed representation thus avoids the problem of exponential growth.

Furthermore, Plate's approach and other similar Vector Symbolic Architectures (Gayler, 2003) supply a method for combining two vectors to generate a new vector. That is, instead of randomly generating a vector for N(DOG), it can be computed based on the vector for DOG and the vector for NOUN. For this, we follow Plate and use the mathematical operation of circular convolution (⊛). That is, we set N(DOG) = NOUN⊛DOG.

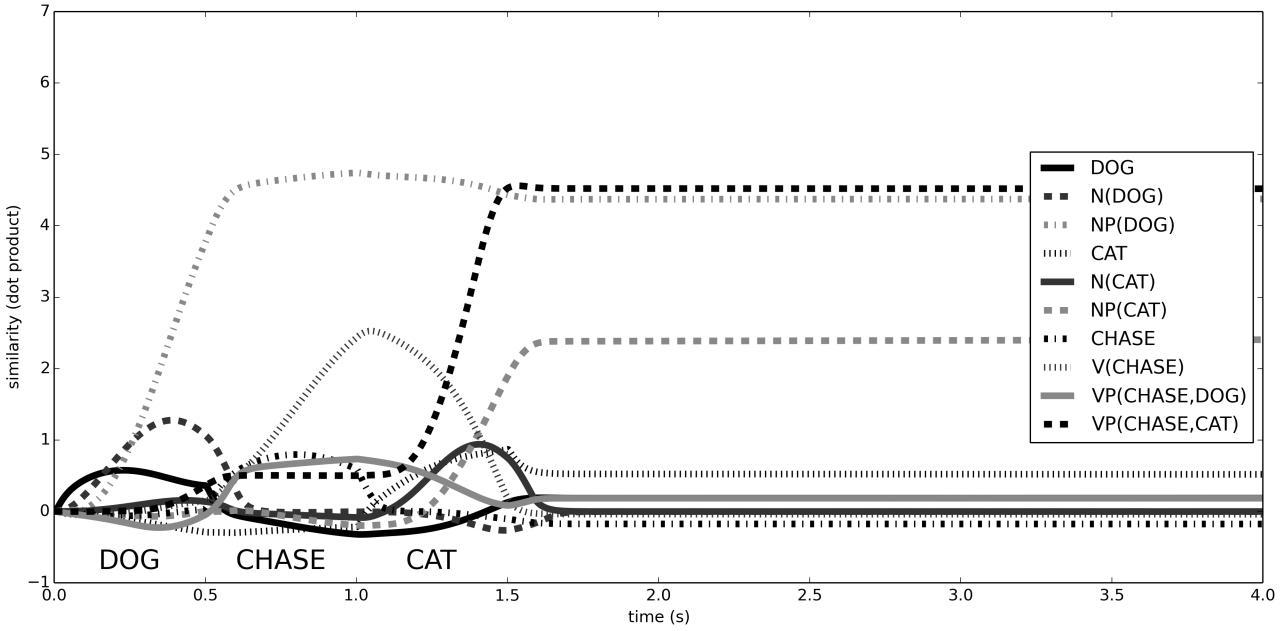

Figure 3: A distributed version of parsing. The state of the system is a single vector *x*. The lines are the dot product of x with the vectors for each node. The network behaves similarly to Figure 2 without requiring an exponential growth in node count.

Biological Recurrence

In the localist representation, the actual parsing process is done by changing the activity level of each node based on its connections and the activity levels of the nodes that connect into it. This can be thought of as a differential equation $dx/dt=f(x)$, where x is overall state vector (the activity level of each node) and f is a function that implements the effects of the connections.

This same idea is true for the distributed approach as well. We take each connection and replace it with a function. As discussed above, the connection causing the DOG node's activity to increase the activity of the N(DOG) node would be expressed as $dx/dt = (x \bullet \text{DOG})(\text{NOUN} \circledast \text{DOG})$. That is, we compute the similarity (dot product) of the current state x with DOG and multiply the result by NOUN⊛DOG. The result is the change in x caused by this connection. The change caused by all of these connections can be found by generating a single function that is the sum of each connection's function.

Now that we have expressed the parser as a differential equation, we can go one step farther and determine how biologically realistic neurons could implement that equation. That is, rather than dealing with the idealized connectionist nodes or simply computing the math of the distributed approach, we can create a model where each component is a spiking neuron, and the differential equation is approximated by the synaptic connections between neurons.

To do this, we use the Neural Engineering Framework (NEF; Eliasmith & Anderson, 2003). This provides a method whereby the activity in a group of neurons represents a vector, connections between groups of neurons implement functions on those vectors, and recurrent connections implement differential equations on those vectors. In each case, the neurons only *approximate* the desired function. Given a enough neurons, this approximation can be made arbitrarily close to the ideal function. However, given realistic biological constraints the resulting behaviour will not be identical to the mathematic version. This provides a natural competence/performance distinction.

To implement this parsing model using the NEF, we use a population of 4,000 LIF neurons. The activity of these neurons will represent a 128-dimensional vector. The neurons have randomly chosen biologically realistic properties in terms of their background current, sensitivity to input, and their "preferred" input stimulus (much like how neurons in visual cortex have particular visual stimuli to which they respond most strongly). This forms a distributed representation of our distributed vector.

Next, we transform the desired differential equation into a form that takes into account intrinsic neuron properties such as the post-synaptic time constant (the amount of time it takes neurotransmitters to be reabsorbed). For common recurrent connections in cortex, this is ~0.1s. The NEF can then be used to solve for an all-to-all recurrent connection weight matrix between all 4,000 neurons that will optimally approximate the given differential equation (Eliasmith & Anderson, 2003).

Importantly, this allows standard linear connection weights to approximate nonlinear functions. In this case, to implement multiplicative excitatory connections, we need $dx/dt = (x \bullet \text{V(CHASE)})(x \bullet \text{N(CAT)})(\text{VP(CHASE,CAT)})$ and other similar functions. These functions will be less accurately implemented than the ones for simple linear

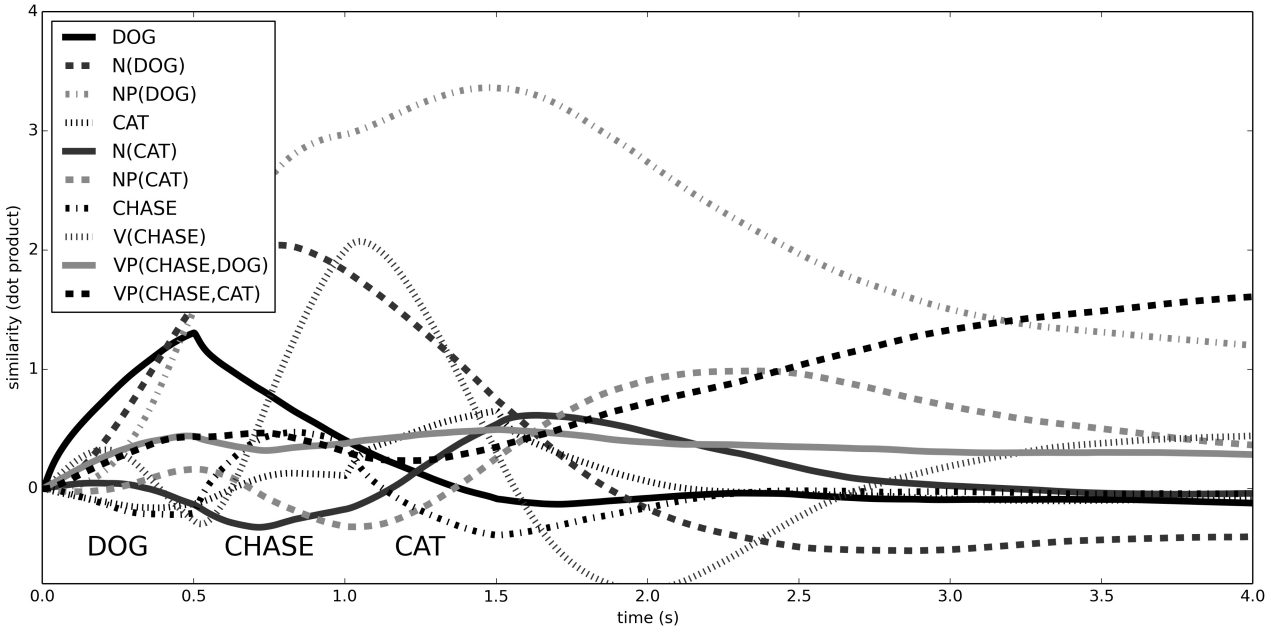

Figure 4: Parsing "DOGS CHASE CATS" in a network of 4000 leaky integrate-and-fire neurons, using distributed vectors of 128 dimensions. The input is the vector for DOG, then CHASE, and then CAT, for 0.5 seconds each. The graph shows the similarity of the represented vector x to the vectors for each indicated term. The network successfully parses the sentence.

connections. However, the ability to approximate this sort of complex connection allows for a wide range of new types of constraint models that we are only beginning to explore.

Semantic Pointer Architecture

It should be noted that the recurrent parsing networks described here use the same approach to representation as in our previous work with building neuron models to perform inductive reasoning, remember sequences, and exhibit cognitive control. This capability formed the core of Spaun (Eliasmith et al., 2012), the first large-scale brain simulation capable of performing multiple tasks. We call this general approach the Semantic Pointer Architecture (Eliasmith, 2013). The vectors are thought of as Semantic Pointers because they not only form a compressed representation of multiple pieces of information (so that the original information can be accessed by only having the compressed vector, much like a pointer in computer science), but also the vector itself contains similarity information, making it useful for making semantic decisions (so the vector value is not arbitrary, as in a computer science pointer).

In previous work, we have used a model of the cortex-basal ganglia-thalamus loop to control changes to these semantic pointers (vectors). In (Stewart, Choo, & Eliasmith, 2014), we showed that a left-corner parser could be implemented with this loop. However, the approach taken in this paper is to implement language processing with a dedicated recurrent network, rather than relying on the general-purpose (and less neurally efficient) basal ganglia. It is possible that this sort of system is involved in the dedicated language-oriented parts of the human brain.

Constraint Satisfaction in Recurrent Networks

Instead of implementing traditional rewrite rule grammars, as above, another way to think about parsing is that grammatical knowledge can be represented by a set of interacting constraints that favor and penalize the co-occurrence of certain structural features in the representation of a linguistic expression (Smolensky & Legendre, 2006). This approach can also be directly mapped into recurrent biologically plausible networks via the NEF.

To achieve this, we note that a constraint can be thought of as a bidirectional connection of the form seen in the distributed parsing model. That is, we can encode each constraint as a weighted outer product of two vectors (Smolensky et al., 2013). Then, we construct a transformation matrix that is sum of the outer products corresponding to entire collection of constraints under consideration. If, for example, there was just a positive constraint between two representations A and B, then the function the neural network needs to approximate would be:

$$dx/dt = (BA^T + AB^T)x$$

For more constraints, more outer products would be summed together. Each of these outer products can be thought of as a projection matrix that maps an input to a scaled version of a vector used to define the outer product. For instance, BA^T maps to B scaled by the dot product of the input x and A^T, by virtue of the linearity of matrix multiplication and the fact that the column-space of BA^T is all scalar multiples of B.

To build a biologically plausible implementation of this constraint satisfaction network, we use the NEF (Eliasmith & Anderson, 2003) in the same way as the previous example. 4000 neurons are configured to represent a 128 dimensional vector, and the NEF is used to find the optimal set of recurrent synaptic connection weights on all of those neurons that will best approximate this function.

The result of defining this mapping between the soft constraints and synaptic weights is that each pattern of neural activity in the population can be assigned a single scalar value (i.e. the value of a *harmony function*; Smolensky and Legendre, 2006) that reflects the degree to which the constraints in question are being satisfied. Over time, the state of the system will gravitate towards a position that maximizes this value and thereby involves a minimal degree of constraint violation. In the case that the constraints correspond to grammatical knowledge, one can think of this trajectory through the model's state space as a parallelized execution of the rules defining the grammar.

To test the scalability of this approach to performing constraint optimization in neural systems, we generate a vocabulary of 200 representations and generate random all-to-all constraints between them, yielding a transformation matrix encoding a total of 40,000 constraints. We then generate a neural population that computes the function described by the transformation matrix through its recurrent connections. Figure 5 depicts the similarity between the representational state encoded by this population and each of the 200 representations over time, as a fixed set of vocabulary items are presented as constant input. The stability achieved after approximately 300ms indicates that the system is able to rapidly compute a local solution to the problem of optimizing all 40,000 constraints.

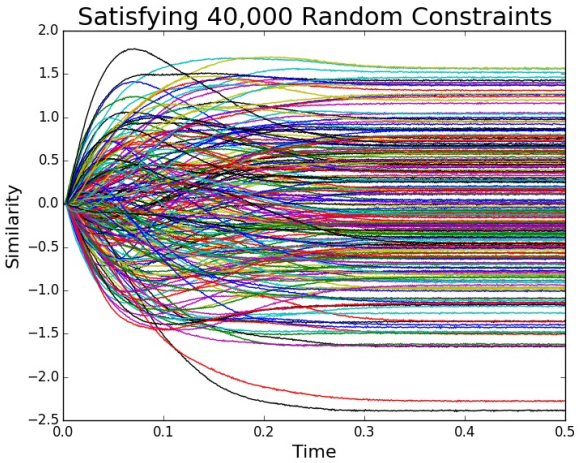

Figure 5: A recurrent network of 4000 neurons representing a 128-dimensional vector solving a randomly generated constraint satisfaction problem. The similarity of the resulting vector with the 200 basic vectors is shown. The network settles to a final result after ~300ms.

Completing Parse Trees

We can further use this approach to take partial parse trees and complete them. Given a set of rewrite rules for a grammar, we can identify grammatical constituents that can and cannot occur together. These form a set of constraints on the final vector representation. Traditionally, these constraints can be seen as connections between localist nodes. However, as noted in the previous section, these constraints can again be converted into differential equations that act on the distributed state vector.

To demonstrate this, consider the following toy grammar:

S → NP, VP	S → AUX, NP, VP
NP → DET, N	VP → V
VP → V	VP → V, NP

After building a network to implement these constraints, we can present partial tree to the system and it will generate the correct consistent components for the parse tree. For example, Figure 6 shows what happens when the input is the partial tree S(?,NP(DET,?),VP(?,V,?)). The network successfully identifies the missing components, resulting in S(AUX,NP(DET,N),VP(V)) as the final parse.

Stability Analysis of a Parser as a Dynamical System

Given that we use a recurrently connected neural ensemble to perform constraint satisfaction, we can treat the ensemble as a dynamical system and do a mathematical analysis of its behaviour. This behaviour is governed by the linear transformation matrix, T, where the computed function is

$$dx/dt = Tx$$

We can factor T into three matrices using eigen-decomposition:

$$T = S\Lambda S^{-1}$$

where S is an matrix whose columns are the eigenvectors of T, and Λ is a diagonal matrix containing the eigenvalues of T. Because the eigenvectors in S are orthogonal for symmetric matrices (and thus form a basis for the vector space), we can rewrite the starting state of the system as a linear combination of eigenvectors:

$$x_0 = c_1 v_1 + c_2 v_2 + ... + c_n v_n$$

where v_i is the i^{th} eigenvector, and c_i is a weight on this vector. This description of the initial condition of the system allows us to solve for the state of the system at arbitrary times given that the transform matrix simply scales each of its eigenvectors by a corresponding eigenvalue. If we treat the transform as a matrix differential equation (as is appropriate in the case that the constraint satisfaction process is implemented in neurons), we can then solve for the state of the system in the following manner:

$$x_t = c_1 e^{\lambda_1 t} v_1 + c_2 e^{\lambda_2 t} v_2 + ... + c_n e^{\lambda_n t} v_n$$

As time grows, eigenvectors with negative eigenvalues will disappear from x_t, while those eigenvectors with positive eigenvalues will exponentially increase; eigenvectors with an eigenvalue of zero will remain unchanged in terms of the contribution they make to x_t. Stable states are therefore acquired only when the transformation matrix has non-positive eigenvalues. Nonetheless, in the case that the eigenvalues are positive,

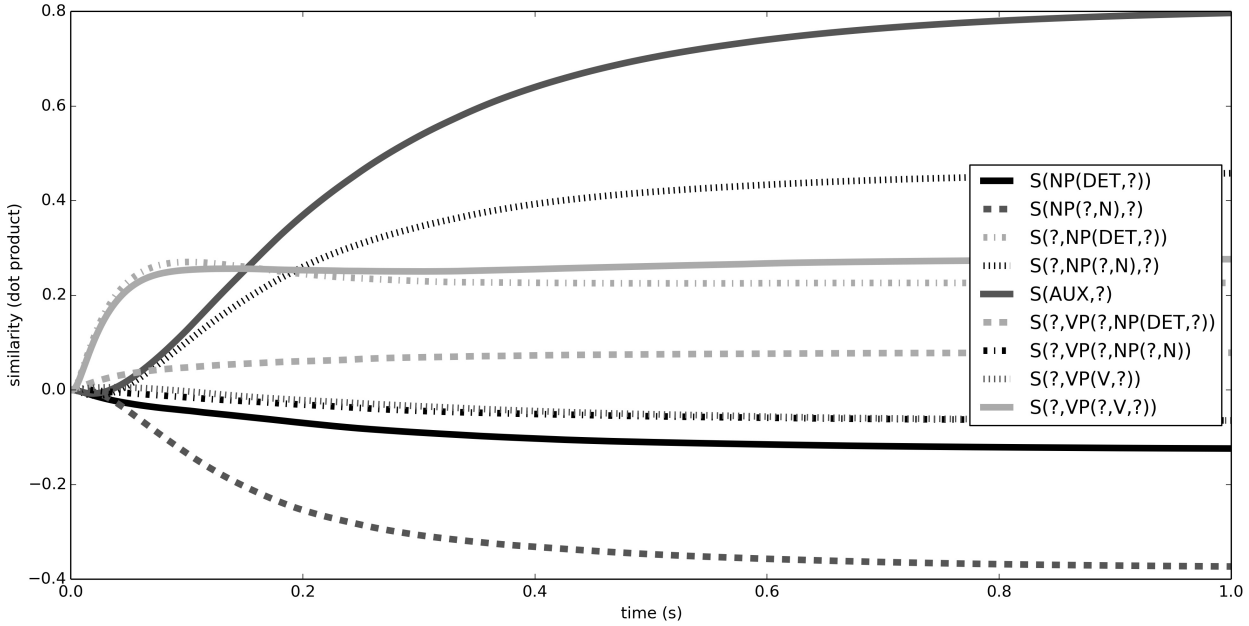

Figure 6: Completing a partial tree using grammar constraints encoded as recurrent connections in 4000 LIF neurons. Each possible grammar structure has its own vector, and the grammatical rules create positive and negative constraints. The input is the sum of the vectors for S(?,NP(DET,?),?) and S(?,VP(?,V,?)). After ~0.1 seconds, the two vectors for S(AUX,?) and S(?,NP(?,N),?) start to be represented, resulting in a stable parse of S(AUX,NP(DET,N),VP(V)).

the vector describing the system state converges on a particular direction in the vector space even as it grows without bound. Of course, when implemented using neurons with the NEF, this growth in numerical value will be balanced by the saturation behaviour that occurs when neurons reach their maximum firing limit.

The motivation for adopting this analysis is that it illustrates how our system might be harnessed to perform interesting computations with linguistic applications. In the case of parsing, the goal is to set the system to an initial state that encodes a set of words, and then have the dynamics of the system drive it towards a state that encodes an optimal parse of these words. The key insight offered by our analysis is that the representations over which our constraints are defined can all be represented as linear combinations of eigenvectors.

As such, we can predict which initial conditions will get mapped states that have particular degrees of similarity to the states that define particular representations. We could in theory choose eigenvalues that perform certain mappings of interest, and then reconstruct a transformation matrix with these eigenvalues. Overall, while more needs to be done to determine how sophisticated grammatical constraints can be encoded and processed using our recurrent network architecture, stability analysis of this sort offers a promising starting point.

Conclusions and Future Directions

Traditionally, grammatical knowledge has been thought of in terms of a set of fixed production rules that that are used to generate the sentences of a language. More recently, this knowledge has instead been characterized in terms of soft constraints on the well-formedness of linguistic expressions (Smolensky and Legendre, 2006). Our work suggests that it is possible to develop this latter approach to the study of grammatical knowledge in the context of detailed simulations of neural systems.

In order to extend our work to accommodate more sophisticated forms of language processing, a few outstanding problems need to be solved. First, it must demonstrated that the parsing capabilities of our simple dynamical systems can scale to more complex grammars. Second, a better understanding of the dynamic behavior associated with particular transformation matrices is required. For example, it might be useful to learn these matrices from examples of parse trees in much the same way that supervised learning techniques are used to train standard feed-forward networks. More generally, it would be very useful to be able to map desired properties of the system's behavior directly onto a set of constraints in a way that is consistent with what is known about how these constraints are likely organized.

Limitations aside, there are number of promising directions in which to extend this work. For example, many researchers are currently very interested in developing compositional distributional models of the meanings of arbitrary linguistic expressions (e.g. Socher et al., 2012). One possible approach to developing such models involves performing constraint optimization with recurrent networks over semantic features in addition to the syntactic features we are currently examining. Other interesting extensions involve incorporating hierarchical structure into the neural systems that perform constraint satisfaction, along with the incorporation of multiplicative interactions that modify the transformation matrix and allow the behavior of the recurrent network to be controlled by an external signal.

Overall, such extensions can help provide insight into the possible ways in which the sophisticated linguistic capabilities that are the hallmark of human intelligence are implemented in neural systems.

Acknowledgments

This work is funded thanks to the Social Sciences and Humanities Research Council of Canada and the U.S. Office of Naval Research.

References

Cottrell, G.W. (1985) Connectionist parsing. In *Proc. 7th Annu. Conf. Cogn. Sci. Soc.*, Erlbaum, 201–211.

Eliasmith, C. & Anderson, C. (2003). *Neural Engineering.* Cambridge: MIT Press.

Eliasmith, C., Stewart, T.C., Choo, X., Bekolay, T., DeWolf, T., Tang, Y., and Rasmussen, D. (2012). A large-scale model of the functioning brain. *Science, 338*:1202-1205.

Eliasmith, C. (2013). *How to build a brain.* Oxford University Press, New York, NY.

Gayler, R. (2003). Vector Symbolic Architectures Answer Jackendoff's Challenges for Cognitive Neuroscience, in Slezak, P. (ed). *Int. Conference on Cognitive Science*, Sydney: University of New South Wales, 133–138.

Plate, T. (2003). *Holographic Reduced Representations*, CSLI Publications, Stanford, CA.

Smolensky, Goldrick, and Mathis (2013). Optimization and quantization in gradient symbol systems: A framework for integrating the continuous and the discrete in cognition. *Cognitve Science, 38,*6, 1102-1138.

Smolensky, P. and Legendre, G. (2006). *The harmonic mind: From neural computation to optimality-theoretic grammar.* Volumes 1-2. Cambridge MA: MIT Press.

Socher, R., Huval, B., and Ng, A., and Manning, C. (2012). Semantic compositionality through recursive matrix-vector spaces. *Empirical Methods in Natural Language Processing.*

Stewart, T.C., Choo, X., and Eliasmith, C.. (2014). Sentence processing in spiking neurons: a biologically plausible left-corner parser. In *36th Annual Conference of the Cognitive Science Society*, 1533-1538.

Waltz, D. and Pollack, J. (1985). Massively parallel parsing: A strongly interactive model of natural language interpretation. *Cognitive Science 9,* 51–74.