Parsing with fLIF Neurons

Christian Huyck & Yulei Fan, Middlesex University
London, UK. c.huyck@mdx.ac.uk

Abstract – *This paper presents a natural language parsing system based solely on fatiguing Leaky Integrate and Fire neurons, a relatively faithful model of biological neurons. The parser implements cell assemblies, sequences, finite state automaton and a stack. The stack enables the system to parse context free grammars. The system uses variable binding to apply the rules, and implement the stack. A novel form of variable binding based on short term potentiation is presented. The output of the system is a semantic frame of the sentence that was parsed. The symbolic interpretation is derived from the underlying neural firing by a simple count of neurons that fire in a particular cycle. The system parses over 99% of the tested sentences correctly.*

Keywords: fatiguing LIF neuron, Cell Assembly, Natural Language Processing, Parsing.

1 Introduction

Perhaps the most complex task that humans perform is to understand language. Virtually every human can do this, but no machine approaches the general linguistic abilities of the typical primary school student. One of the fundamental processes involved in understanding language is to take a sentence of words, and combine the meanings of the words in an order dependent fashion to get the meaning of the entire sentence. This is typically called parsing the sentence. While a great deal of effort has been put into developing parsers, the best parsers still function below 90% efficiency in generating the syntax trees of sentences [2]. There has been less work on semantic evaluation, but again parsers are imperfect.

From this earlier parsing work, it is clear that ambiguities exist in natural language and semantics are needed to resolve these ambiguities. Unfortunately, computational interpretation of semantics is poor. One way forward may be to mimic human processing at the neural level. This may allow a system to learn semantics in a fashion similar to humans.

This paper presents a natural language parser implemented in fatiguing Leaky Integrate and Fire (fLIF) neurons. The model is a variation on the widely used Leaky Integrate and Fire (LIF) neuron [21], but these neurons fatigue when fire repeatedly; this fatigue makes it more difficult for the neurons to fire. This improves the dynamics of the system, by making it easier to move from one state to another, and adds an extra degree of biological faithfulness. The actual fLIF model is not explained in this paper but can be found elsewhere [8, 12]. While the fLIF model is an imperfect model of biological neurons, it does closely approximate the basic functions of neurons. The model is not perfect with discrete cycles that correspond roughly to 10 ms. of biological time. The loss of biological faithfulness is partially compensated by the efficiency of the model allowing a large number of neurons to be simulated in real time. The parser takes advantage of Cell Assemblies (CAs) [5], a concept that unifies neural and psychological theory (see section 2).

The parser has a small grammar and lexicon, but is able to repeatedly generate the correct semantic representation for a number of sentences. Moreover the parser is one component of a larger video game agent. The agent is an assistant to a user and takes text commands using the parser. The agent also can view the environment and take its own actions. The entire agent is implemented in fLIF neurons.

The parser is a partial implementation of a psychologically plausible symbolic parser [9]. This work is similar to a Marcus parser [17, 13], and is based around a stack, preferences for rule selection, limited look ahead, and grammar rules that combine syntax and semantics.

Crucially, implementation of the stack allows the parser to interpret context free languages [1]. It is widely agreed that natural languages are at least context free [19]. A neural parser is rather unique. While other connectionist parsers have been built (e.g. [18, 7]) the authors are aware of only one other neural parser [14]; however, Knoblauch and Palm's parser does not have a stack and thus is limited to regular languages.

While parsing, variable binding is used to bind nodes to stack elements, and to bind slot fillers to verb frames. The semantic representation is thus a verb frame [3]. The system described in this paper uses a novel form of variable binding based on short-term synaptic plasticity.

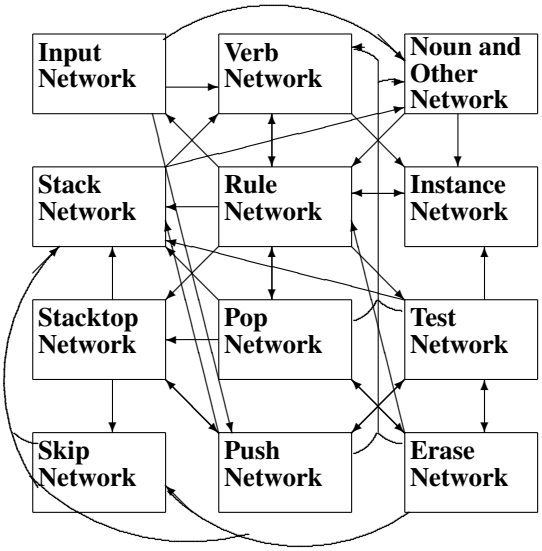

Figure 1: Topology of the Parsing System

2 State Change with the System

The network consists of 13 subnets shown in Figure 1. The action of the system is more fully explained in section 3. It makes extensive use of CAs for stable states, and pseudo-CAs for organising simple sequences of actions.

CAs are sets of neurons that have high mutual synaptic strength. When a small number of these neurons fire, they spread activation to the others causing a cascade of firing that enables the complete CA circuit to remain persistently active practically indefinitely [22]. Most of the subnets in the system are broken up into CAs. The process of activating a CA so that it continues to fire persistently is called CA ignition or simply ignition.

In a given fLIF network, the ignition and persistence of a CA puts the system into a state. Most of our prior work (e.g. [10, 12]) has focused on these single state systems. The network is presented an object, the object causes a CA to ignite, this categorises the object as a member of that CA, and this state persists indefinitely. To categorise a new object, the system is reset with each neuron's activity and fatigue levels reset to zero, then the new object is presented.

Unlike these single state systems, the system discussed in this paper is a multi-state system. Depending on the current state and the current input, the system moves to a new state.

It is relatively simple to see how a fLIF network can move from one state to the next independently of input. This occurs in the Push, Pop, Test and Erase sub-networks. In each of these, the subnet is broken up into a series of pseudo-CAs. Neurons in each have high mutual synaptic strength so they cause each other to fire. However, in a given sequence of pseudo-CAs, neurons have excitatory connections to neurons in the next pseudo-CA and inhibitory connections to neurons in the prior one. So after a few cycles, the next pseudo-CA in the sequence ignites; firing in that pseudo-CA causes the initial pseudo-CA to cease firing because of the inhibitory connections. Fatigue also plays a component in shutting down the prior pseudo-CA. As these pseudo-CAs do not persist for a significant time, they are not true CAs.

This chain of activation enables the system to execute a sequence of events. Unfortunately, it does not allow the system to perform different actions depending on the input. Fortunately, a simple mechanism exists to enable this type of change. One set of subnets encodes the current state. The current state sends activation to the possible next states. This activation is enough to prime those next states, but not enough to ignite them. Similarly, the inputs also send activation to those next states; again this activation is enough to prime but not enough to ignite. The combination of state and input is, however, enough to ignite the next state. This pragmatically, allows the system to function as a finite state automata [15]. Combined with the stack, the system functions as a push down automata.

3 Overall Architecture

The architecture is based on 13 subnetworks shown in Figure 1, which shows 12 subnets and the connections between them. In the figure, the noun subnet and the other word subnet are combined as their coarse external connections are identical.

Figure 1 is quite complex. The entire network implements the simple parsing algorithm:

1. Start by pushing a word onto the stack
2. Repeat
   (a) Test rules
   (b) If no rule succeeds,
      i. Push new word on stack
   (c) else (a rule succeeds)
      i. Apply the rule
      ii. Pop stack
3. Until $VP \rightarrow VPPeriod$ rule applied

An explanation of a parse of the simple sentence in example 1 is provided to highlight important aspects of the parser. For clarity, the explanation follows the algorithm.

Example 1. *Follow me.*

Start by pushing word on stack

Initially, portions of two subnets are activated. The two CAs that receive external activation are the *Follow* CA in the Input net and the 0 element in the Stacktop net. This causes ignition of the appropriate reverberating circuit in these nets, and many neurons in these circuits will continue to fire until some other net causes them to stop firing via inhibitory connections.

The Input, Verb, Noun, Other, Instance, Stack, Skip, and Rule subnets are each broken into several CAs (see section 2). In these subnets the CAs are orthogonal so each neuron is in only one CA.

Firing in the 0 element of the stacktop and any input CA causes ignition of the Push subnetwork. In essence, this is the start state of the automata. The Push, Pop, Test and Erase subnetworks execute processes via neural firing; they are not CAs per se, but are quite similar (see section 2). The activation of the Push subnetwork (Pop, Test and Erase) causes a sequence of events to happen with neural firing in the subnetwork shutting down automatically at the end of the sequence. This sequence is implemented by sets of neurons, similar to CAs, that are connected to the next set of neurons in the subnetwork. These connections are excitatory, while reverse connections, in the set, are inhibitory. With the Push net, the sequence is to increment the Stacktop, ignite the appropriate Stack element in cooperation with the Stacktop, ignite the appropriate word CA (in this case a Verb CA), and start the Test subnetwork.

The Stacktop is incremented by sending activation to particular neurons in it [1].

The appropriate Stack element is ignited by a combined activation from push and stacktop. Push sends activation to all the items. Stacktop sends activation to the item that corresponds to the active Stacktop CA. In this case, the only CA that is receiving activation from both is the first Stack element. This ignites. Similarly, all of the word CAs in the Verb, Noun and Other Word nets are sent activation, but only the Follow CA receives activation from the Input and it is the only one that ignites. Additionally, the only symbolic act occurs when the Push net ceases to fire; the Input net moves onto the next word, in this case *me*. This is the only symbolic act the system performs. This symbolic act is necessary as there is no reading component to the system.

The first Stack element is now ignited as is the Verb's *Follow* CA. The Stack element has some fast bind neurons (see Section 4) and is now bound to *Follow*.

Test rules

The Test subnetwork then becomes active. It has a sequence of steps where it shuts down activation in the Stack and the words followed by igniting elements of the stack. It activates each of the first three Stack elements preceded and followed by turning them off. It does this twice. If the Stack elements are bound, the corresponding words are ignited. Activation in the word nets causes the appropriate rule to fire. In this case, no rule corresponds to a verb by itself and no Rule CA is ignited.

If no rule succeeds

Test finishes by activating Push. Push follows a similar process to push *me* on to the stack as its second element. The second stack element and the noun are both ignited. There simultaneous firing causes them to be bound.

Push new word onto stack

[1]The keen observer will note that the stacktop is the only subnet that has not been specified as a sequence pseudo-CA or a subnet of orthogonal CAs. This is because it consists of five CAs, that are linked with push and pop sequence manipulating neurons. The push neurons associated with a particular stacktop element prime the next stack element. When a push is generated, activation is sent to all the elements; only the primed element ignites; when it ignites it suppresses the prior element. A similar mechanism works for pushing.

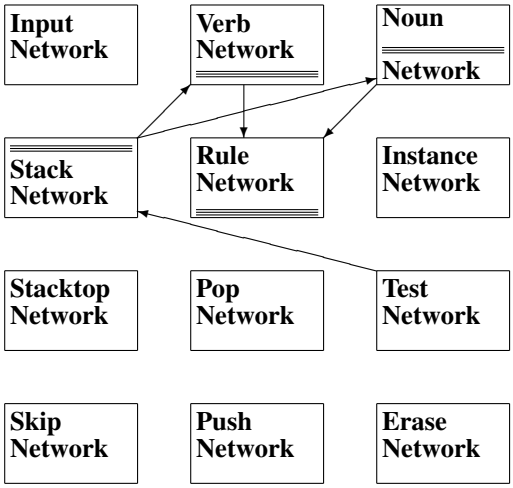

Figure 2: Rule Activated in System

When push finishes, it starts test running (and makes the period the input).

else (a rule succeeds)

This time the ignition of Verb Noun causes the VP → VP NP-object rule to fire. Note that in the rule subnet the leak is quite low so that a significant amount of activation is retained. This is important so that the activity of multiple stack elements can be integrated into the ignition of a given rule. The retention of activation (the part that remains after leakage) is a form of memory that is particularly important in the rule selection of this system.

Figure 2 shows this rule igniting over several cycles. The test network activates the stack items in sequence. The boxes represent ignited CAs consisting of several hundred neurons. In the figure, stack item 1 is ignited and is igniting the verb Follow. Before that stack item 2 had ignited the noun Follow. The Rule CA has accumulated activation from the noun, and now that the verb has ignited, enough activation has accumulated for the rule to ignite.

Apply the rule

The ignition of the rule causes the first and second Stack elements to be activated simultaneously, and shuts off the Test. The Noun *me* CA has also activated an instance of *me* in the Instance net. The rule also sends activation to the object slot in all the verbs; as the only verb that is active is *Follow*, its object slot also ignites. This slot has fast bind neurons that bind it with the Verb and bind it to the *me* CA in the instance net. However, a semantic trace is left in binding of the object slot to the instance net CA.

Pop stack

The rule then causes the Pop net to become active. This pops the stack and starts the Erase net.

One could ask why there is an instance net as there is a one to one correspondence between instances and nouns. The answer lies in the connection between the noun net and the rule net. While the active noun CA sends activation to

rules, the active instance does not. So, once the noun has been included into a parse and is removed from the stack, it no longer helps to apply grammar rules.

The Erase net is then activated to complete the clearing of the second stack element. This will be described in section 4.

Until $VP \rightarrow VPPeriod$ **rule applied**

The result at this point is that activating the first Stack element will ignite the Verb *Follow* which ignites its object *me*. Reading these activations provides a symbolic semantic interpretation of the sentence. The parse will continue to push the period, and eventually to run the VP → VP period rule which stops processing of the sentence.

This entire process can be viewed as an automata. The state of the parser is held by the stack, and the position in the processing subnets (push, test, pop and erase). Push modifies the stack and thus the state. Test merely inspects the stack, but if the right elements are on the stack in the right order, they will cause a rule to fire. This causes pop to modify the stack. Pop then passes control onto erase which completes the modification process. Erase finishes by passing control back to the test subnet.

4 Variable Binding

One of the fundamental problems of neural processing and indeed connectionist systems in general is variable binding. In brief, a variable needs to be given a value that it retains for some significant period of time. Later, the system needs to be able to give that variable a new value. Variable binding is needed to give the system compositional syntax and semantics [4].

This is a simple process in typical computers. A variable is associated with a portion of memory. The memory corresponds to a value in a circuit. As long as the electricity is on, the circuit, and thus the variable, will retain that value.

It is more difficult in neural systems. One commonly used mechanism for binding is via synchrony [20]. Two sets of neurons are bound together if they tend to fire at the same time. Unfortunately, this has problems supporting more than a small number of bindings.

Another option is to bind by permanently changing synaptic weights [11]. Most simulation of neurons involves learning by changing synaptic weights permanently. This is equated to the biological phenomena of Long-Term Potentiation. However, a mechanism to replace the binding with another binding is still required. This can be done with spontaneous neural activation, but this takes many simulation cycles.

Another biological phenomena is Short-Term Potentiation. Coactivation of neurons leads to increased synaptic weights that revert back to the original weights over time [6].

This phenomena has been modelled in this simulation by neurons that we have coined fast bind neurons. These change synaptic weights by a simple addition of a constant when two neurons are coactive; this weight is clipped at a maximum. Each cycle the neurons do not cofire, the weight is reduced until it reaches zero.

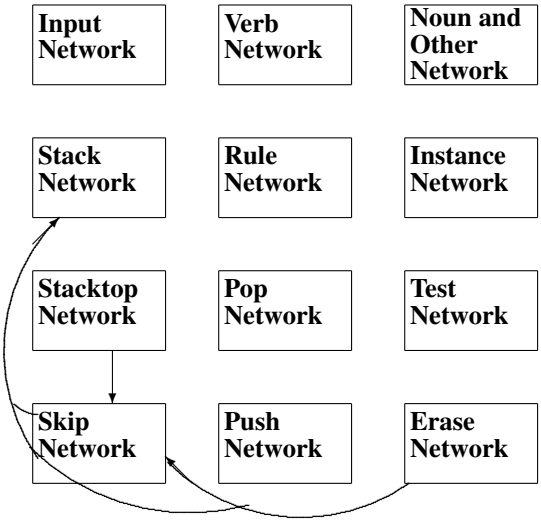

Figure 3: Skipping Binding During Erase

When a CA that has fast bind neurons is coactivated with another CA that it is connected to, the two are rapidly bound. Subsequent ignition of the initial CA will then cause ignition of the bound CA. After a period of no activation of the initial CA, the binding will be gradually but automatically erased.

So, when a Stack element is activated, the simultaneous activation of a word CA causes the element's fast bind neurons to rapidly increase weight. If the element is turned off and then on again, as the Test net does, the fast bind neurons will ignite the word CA. As it is only bound to one word, only that word is ignited. A similar process happens when Verb slots are bound to Instance CAs.

The erase net ignites the active Stack elements, but only those at or below the stacktop. It does this in sequence so that, if the Stacktop is two, the first Stack element is ignited, then shut, then the second Element. This is repeated three times. This ignition supports the binding by allowing the now reduced weights to regain their strength. The elements above the Stacktop are automatically erased as they are not supported. Skip comes in here to inhibit the activation of Stack elements above the Stacktop. This is done by having the skip CA inhibit the second stack element. If the skip CA is on, it prevents the second stack element from igniting. The skip CA is only turned on when the stacktop is one, and erase is trying to activate the second stack element. Figure 3 shows this. Erase typically activates stack items. However, if the stacktop is one, then the skip net comes on (just one CA) when erase is about to activate stack element two. This suppresses the ignition of stack element 2.

5 Results

The parser stores a semantic representation of the sentence in the form of a verb frame. This subsymbolic verb

frame can be readily translated into a symbolic frame.

To retrieve the semantic interpretation of the sentence after parsing has completed, the first stack element is ignited. Activation is sent to the item for 10 cycles. This ignites it and, in each case tested, causes ignition of the verb CA to which it is bound. Since the verb CAs are orthogonal, a calculation of which particular CA is ignited is simply a matter of counting the neurons in that CA that fire in the tenth cycle. For this measurement, if any neuron fires, the CA is on. If multiple verb CAs are on, it is considered an error.

Similarly, the verb CA has 60 fast bind neurons associated with each of its three slots (actor, object and location). If the slot has neurons firing, it is considered on. Again, multiple or absent slots are considered an error. These slot neurons have propagated activation onto the slot fillers that they are bound to. These slot fillers are also orthogonal and are measured in the same fashion as the verbs and slots.

The parser was tested on 17 sentences, uses five grammar rules, and 16 words that fall into five lexical categories. There is some randomness in creating the system, and a random system gets roughly 90% of the semantic representations complete. The networks can be saved and rerun.

The best system has over 99% performance with the calculated error being lost on slot filler absences. In this case, the system was tested on each of the 17 sentences 32 times. The system only failed on 2 instances of the sentences for a performance of 542/544 or .9963.

The system can be found at http://www.cwa.mdx.ac.uk/CABot/CANT.html. It is written in Java and we are happy to respond to queries for help in installing and running the system.

Since the parser, described above, was completed, it has been integrated into our CABot1 agent. This is an agent in a Crystal Space video game. The agent assists a user in the 3D environment. The user issues commands that the agent tries to follow. In addition to the parser there is a visual subsystem, a goal propagation subsystem and a control subsystem. The agent goes through a loop (the control system) of accepting the user's command, parsing it, setting a goal based on the command, completing the goal, and then erasing the stack. Again, except for reading a new word and sending a command to the game, all of this is done in fLIF neurons. This parser has a similar performance though it now has an extra rule, a few more words and parses a few more sentences.

6 Discussion

The parser is effective though it is clearly not an industrial grade parser. The construction of the system involves some randomness in synaptic connections and neural properties. The system can be rerun and it is highly likely that a better network can be found. Moreover, the existing network could be modified to improve performance. That is, the randomness could be eliminated. However, at this stage, perfect performance is not the major point of the parser.

Instead the point is to develop neural parsing algorithms and integrate them with other neural systems as has been done with the CABot1 games agent. The integrated system will then function in an environment. Neural learning, not used in the parser presented in this paper, will then allow the system to learn the semantics of the environment. This in turn will enable the system to parse more effectively than current symbolic parsers.

As few multi-state simulations have been done with neural systems, building the techniques to develop these systems is a complex task. By using sequences, finite state automaton and push down automaton, implemented in fLIF neurons, this system shows how these things can be implemented. More formal treatment of these mechanisms will be explored.

This parser has also shown some weaknesses that need to be explained. Firstly, the system does not properly account for a verb that has two or more slot fillers. One would expect that the activation of a single slot in a verb frame would only activate the associated slot filler. Unfortunately, when the verb ignites, all of its bound slots also ignite along with all the slot fillers. This means that they are all bound together. For instance, the semantic representation of example 2 should have the object slot filled with pyramid and the location slot filled with door. When all of these are coactive, pyramid is also bound to location via the fast bind synapses. So when *Move*'s location slot alone is activated, both pyramid and door will ignite.

Example 2. *Move the pyramid to the door.*

We hope to solve this problem using a different form of binding for the slot fillers. This is based on a compensatory learning mechanism and spontaneous activation [11].

Secondly, the parsing takes too long from a psychological perspective. Each cycle in the fLIF model corresponds to roughly 10 ms. This allows the system to ignore refractory periods and synaptic transmission delays as all of these things happen below the 10 ms. level. Unfortunately, parsing a typical sentence takes on the order of 1000 cycles, or 10 seconds of simulated time.

Like symbolic parsers, neural parsers can also produce semantic output and account for human parsing failures. One of the advantages of a neural model of parsing is it can easily give timing data for sentence parsing. All of these can be compared with psycholinguistic data to reproduce more human-like parsers.

It seems that the problem with the parser is largely around the time that it takes to erase bindings. The stack mechanism requires the bindings to be erased. Over half the time is spent on erasing. Similarly, most of the remaining time is spent running the test. A better parser would avoid the use of a stack all together. Many modern parsers avoid the use of stacks and merely have a set of active items that can be used during the parse [16]. Implementing this kind of parser with fLIF neurons seems viable. In this type of system neural firing levels within a CA would probably become more important and the system would have to deal with multiple items being simultaneously active.

Finally learning is a key factor in neural systems. Indeed it is one of the main reasons or hopes why neural subsymbolic systems will surpass symbolic systems. Learning needs to be incorporated into the system. Prior work shows that the fLIF neurons can be used to learn categories [10, 12]. These techniques should be readily applied to learning parsing preferences.

7 Conclusion

This paper has shown a natural language parser implemented in fLIF neurons. This parser functions above the 99% rate. The parser takes advantage of the intermediate concepts of CAs, sequences, and finite state automaton. It was shown how sets of fLIF neurons can implement all of these.

The parser shows that a traditionally symbolic task, natural language parsing, can be implemented in relatively biologically accurate simulated neurons. We hope that this system will allow us to explore the acquisition of grammar rules, semantics and language understanding.

Acknowledgements: This work is supported by EPSRC grant EP/D059720.

References

[1] A. Aho, R. Sethi and J. Ullman (1986) *Compilers: Principles, Techniques and Tools.* Reading, MA: Addison-Wesley Publishing Co.

[2] Collins, M. (1997) Three Generative, Lexicalised Models for Statistical Parsing. In *Proceedings of the 35th Annual Meeting of the ACL* pp. 16-23.

[3] Filmore, C. (1968) The Case for Case. In *Universals in Linguistic Theory* E. Back and R. Harms (Eds.) Holt, Rinheart and Winston Inc.

[4] J. Fodor and Z. Pylyshyn (1988) Connectionism and Cognitive Architecture: A Critical Analysis. *Cognition* 28 pp. 3–71.

[5] D. Hebb (1949) *The Organization of Behavior*, John Wiley and Sons, New York.

[6] Hempel C., Hartman, K., Wang, X., Turrigiano, G, and Nelson, S. (2000) Multiple Forms of Short-Term Plasticity at Excitatory Synapses in Rat Medial Prefrontal Cortex *J. Neurophysiology* 83 pp. 3031–3041.

[7] J. Henderson (1994) Connectionist Syntactic Parsing Using Temporal Variable Binding. *Journal of Psycholinguistic Research* 23:5 pp. 353–379.

[8] C. Huyck (1999). Modelling Cell Assemblies. Middlesex University Technical Report ISSN 1462-0871 CS-99-07

[9] C. Huyck. (2000) A Practical System for Human-Like Parsing, Proc. of the 14th European Conference on Artificial Intelligence, Berlin, pp. 436–440 2000.

[10] C. Huyck (2007) Creating Hierarchical Categories Using Cell Assemblies. *Connection Science* 19:1 pp. 1–24.

[11] C. Huyck and R. Belavkin (2006) Counting with Neurons: Rule Application with Nets of Fatiguing Leaky Integrate and Fire Neurons. *Proceedings of the Seventh International Conference on Cognitive Modelling* pp. 142–147.

[12] C. Huyck and V. Orengo (2005) Information Retrieval and Categorisation using a Cell Assembly Network. *Neural Computing and Applications* 14 pp. 282–289.

[13] D. Jurafsky. (1992) An On-Line Computational Model of Human Sentence Interpretation: A Theory of the Representation of Linguistic Knowledge, PhD Thesis, University of California, Berkley CA.

[14] A. Knoblauch and G. Palm (2001). Pattern separation and synchronization in spiking associative memories and visual areas. *Neural Networks* 14 pp. 763–780.

[15] Lewis, Harry R. and Christos H. Papadimitriou. 1981. *Elements of the Theory of Computation.* Prentice-Hall, Inc. Englewood Cliffs, New Jersey.

[16] R. Lewis and S. Vasishth (2005) An activation-based model of sentence processing as a skilled memory retrieval. *Cognitive Science* 29 pp. 375–419.

[17] Marcus, M. 1980 *A Theory of Syntactic Recognition for Natural Language* Cambridge, MA: MIT Press.

[18] Mikkulainen, Risto. 1993. Subsymbolic Natural Language Processing Cambridge, MA: MIT Press.

[19] Pinker, S. (1994) The Language Instinct. Penguin Books, London.

[20] L. Shastri and V. Aijanagadde (1993) From Simple Associations to Systematic Reasoning: A Connectionist representation of rules, variables, and dynamic bindings using temporal synchrony. *Behaviour and Brain Science* 16 pp. 417–494.

[21] D. Tal and E. Schwartz (1997) Computing with the Leaky Integrate-and-Fire Neuron: Logarithmic Computation and Multiplication, *Neural Computation* 9:2 pp. 305–318.

[22] T. Wennekers, M. Garagnani, and F. Pulvermuller. (2006) Language models based on Hebbian cell assemblies. *Journal of Physiology-Paris* 100 pp 16–30