How Language Could Have Evolved

Ken Del Signore

North Aurora, Il. 60542, USA,
kendelsignore@gmail.com

Abstract. This paper begins to develop a biologically inspired computational model of the Human language faculty and some associated thought processes. This model is developed starting from a simple proto-language, which humans are assumed to have inherited at speciation. This proto-language consists of single symbol exchange using a small set of symbols; similar to the observed gestural communication systems in the existing Great Ape families. Computationally, the model is built using a single class with the form of a Markov graph node. Instances of this node class are used to symbolically represent words. The model is built iteratively in main() as a single graph. Nodes are added to this graph using a merge, or conjunctive join, operation between any two existing nodes, notionally labeled as head and copy. A simple first order graph is developed which is hypothesized to be common to all Mammals and to generate shared mammal behaviors. This graph is then extended to allow for more complex human language and thought processes.

Keywords: great leap theory of mind faculty of language

1 Introduction

We take the simplest possible communication to be a single symbol, exchanged from one communicator to another. As an example, consider a hiker that is lost in the woods and builds a pile of rocks, and then moves on. If a second hiker subsequently finds the rocks, then we can say that a single symbol has been exchanged. The information conveyed is the same as if the first hiker had just stood next to the second and said "here", except that the hiker said "here" some long time prior and the symbol (the rocks) held the information through time.

To modify this example, if while the hiker is piling the rocks, s/he hears the second hiker coming straight on and calls out "here", then the second hiker will have, at the simplest, one additional quanta of information, namely a measured value of the distance. This second quanta is analogous to a floating point variable that can take on a continuous value.

Single symbol dialogue systems are observed throughout the animal and plant kingdoms. The symbol "here" is often the exchanged symbol in these systems. A flower can be interpreted as a single symbol, exchanged between a plant and its pollinators, with the meaning of the flower being "here" (Chomsky 2015). A symbol is defined to contain two quanta of information: label and value; with the value being possibly zero or unspecified. In the examples above, the second

communicator measures the value locally to itself using the externalization made by the first communicator.

The neocortex gives mammals the ability to store and recall sequences of symbols with relative ease. This ability gives mammals considerably more complex behavior relative to non mammals. Mammals can input and store sequences of symbols in combinations never experienced before and later recall and utilize this information; an example of which would be the second hiker remembering the path out of the woods and walking out with the first hiker.

All known mammal dialogue, excluding human, uses or can be easily reduced to single symbol exchange. The great apes have possibly the most developed system; using a gestural vocabulary of approximately 80 gestures to convey $\sim$15 unique meanings as commands and questions (Bryne 2017). The meanings loosely correspond to the hypernym forms of various parts of speech categories [here, no, give, on, on?, play?], which are discussed further below.

Among the Hominins, stone tools provide the first evidence for advancement in behavior. The initial Mode I tools are currently dated to 3.3 Mya and remained at a relatively fixed level of design and refinement for over a million and a half years. The early hominins did not evidently undergo much generational change; contrary to the current human cliche "kids these days...". Mode II tools were then developed and these spread slowly throughout the existing hominin range over the next million years. When humans first speciated, they inherited a Mode II toolkit. They had animal hides, cord, knots, hafted hand-axes, spears, and cooked meals, to name a few of their initial conveniences. Hominins had been hunting elephants and hippopotamuses since at least 400 Kya and early human sites dated 200 Kya also contain evidence that they subsisted on these animals.

Humans then began making rapid advances to their toolkits (Henshilwood 2018) and then left Africa approximately 75 Kya. The sudden change to the rate of change of the toolkit just prior to leaving Africa is suggestive that, at the simplest, a single change could have taken place in the hominins to allow them to make these advances. Our current human language (faculty) is argued to be this change (Bolhuis 2014). The use of complex sentences would presumably have allowed the hominins to more easily accumulate knowledge and transfer it to each other and their children. Daily storage and recall of unique sequences would also permit hominins the ability to mentally reconstruct scenarios after they occurred, which would allow them to explain them to others and explore possible solutions when time permitted.

All available evidence indicates that the current human language faculty and cognitive functionality was completely formed before humans left Africa and that it hasn't changed since (Bolhuis 2014); which would be a signature of the single change in Africa hypothesis. The argument for this is that babies from any culture can grow up in any new culture and will readily acquire the new culture and language, which is taken to imply that no changes to the human language faculty or other cognitive functions have occurred in humans since we left Africa.

It is also worth noting that not all humans in Africa obtained the new toolkit. Human sites dated as recent as 30 Kya have been found that do not show evidence of advancement beyond that of the inherited toolkit (Scerri 2021).

For the present inquiry, the two main historical developments of interest are the mammalian neocortex and the human toolkit change. The neocortex is viewed as having endowed mammals with the ability to conceptualize symbols, to form vocabularies of these symbols, and to input, store, recall, and utilize random sequences of these symbols. The human toolkit change is viewed as happening when the neocortex became large enough to support a new thought process and/or a new thought process was developed.

Sections 2 - 12 cover the derivation and evolution of the graph model in an approximate temporal order that it is posited to have developed in. Section 13 contains a discussion and comparison with similar work.

2 The Interface

We begin with one of the main minimalist assumptions of Linguistics, namely that communicators have some common internal neurological/symbolic representation of each word in their vocabulary. This is shown as the two blue "interface parcels" in the diagram in figure 1, which borrows from similar diagrams in (Pinker 1999 and Berwick 2013). In this example, while walking out of the woods, the two hikers roust a duck, which causes the duck symbol to activate in each hiker's interface parcel. We assume that each hiker processes their unique neural input of the event and each conceptualizes, or activates, an internal symbol that corresponds to "duck". Also note that if one of the hikers is replaced by another mammal, such as a dog, the interface parcel representation would still be valid.

For our initial purposes we assume that some analogous neural parcel exists that contains many neural attractor states that represent words or symbols. Such a neural parcel can be thought of as analogous to a two dimensional optical character recognition neural network, wherein each character is a unique attractor state of the OCR neural network. Each character corresponds to a subset of nodes in the 2d array that fire and lock into the active state when the array is input a noisy bitmap of a scanned character and allowed to run freely into the nearest attractor state.

Such a subset of nodes is analogous to a Cell Assembly (CA). Here we assume each CA holds the floating point value as the spiking rate and the sign as the phase of the spikes (Huyck 2013).

The symbols in this interface parcel, once formed, are assumed capable of subsequent re-activation on similar input. Furthermore, once activated, we assume that the symbols retain this state information for a short time and are more easily re-activated (Hubel 1980).

The interface parcel can be abstracted to represent all of the symbols that we are physically able to internalize and/or externalize. The temporal sequence of all such symbols can be thought of as our stream of consciousness.

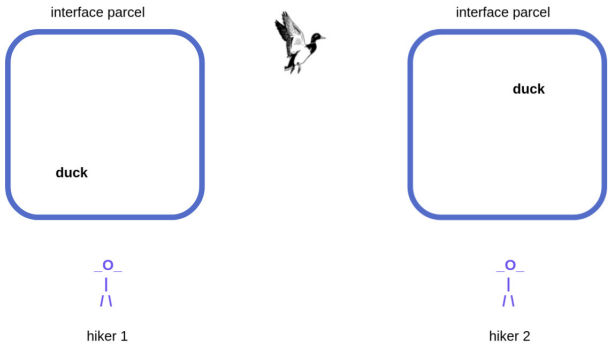

Fig. 1. The rousting of the duck provides unique sensory input to each hiker's interface parcel, which we assume causes the "duck" symbol to fire (or conceptualize) therein. This is posited to occur as some subset of the nodes in the interface parcel firing into a stable attractor state.

3 Short term memory, (stm) sequences and recall

Behaviors involving short term memory are easily observable in mammals and many other animals. In humans, we can easily form stm associations between any two randomly picked symbols in our vocabularies.

As an example, we can extend the duck scenario above such that the surprise of the duck rousting causes the first hiker to sneeze. Following this, the second hiker's interface parcel would contain the activated symbols "duck" and "sneeze" and these would be stm bound such that if some short while later a second duck was rousted, then this would cause the second hiker to recall the sneeze symbol and to expect the first hiker to sneeze again. We can say that the second duck caused the hiker to "think of" the sneeze symbol.

Such a random two symbol stm mechanism can be built by the current model using the assumed node functionality. We (hypothetically) introduce many additional nodes to the interface parcel, referred to as stm nodes. These stm nodes are assumed to be equivalent to the symbol nodes except that they are unlabeled. The stm nodes are assumed to be randomly and sparsely connected to the labeled symbol nodes (Hawkins 2005).

When two random symbol nodes, such as duck and sneeze, are activated and fire, they each provide input activation to their respective stm nodes. If a subset of these stm nodes is common to both symbols, this subset can become activated over background due to having 2x more input activation than the stm nodes that are not common. The elevated input level is then assumed to persist for some time interval, allowing for subsequent short term associative recall.

Computationally, stm memory can be implemented as a single node created by a merge function between a root node "ip" (interface parcel) and its child nodes, as shown in figure 3. The (stm) nodes are created by the merge function

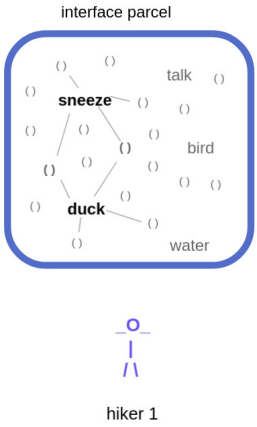

Fig. 2. Many sparsely connected (stm) nodes are capable (under appropriate conditions) of forming a short term association between any two symbols in the parcel.

that runs each time a node fires. Bidirectional connectivity is assumed possible in all connections.

The merge operation can be applied recursively between other recent stm nodes to create the (d s) node in figure 3, top right. This node then stores the short term association between duck and sneeze.

The stm nodes are created at run time using a merge constructor function of the Node class as shown in figure 4. This function takes two nodes as input, labeled "head" and "copy". Bidirectional links are set up between the head and copy nodes in the merge constructor function.

The sequence: "hiker rocks duck sneeze" is input with the "touch" function calls in main() as shown in figure 5.

The structure formed in (stm) memory can then be used for output of the hiker, rock, duck, sneeze symbols. Using the recursive algorithm shown in the pseudo code in figure 6, the symbols can be output in the order that they occurred.

4 Protolanguage, Mammalian single symbol exchange

Following the suggestion of Chomsky (2015), we assume a simple (truncated English) corpus of the form: [here now me thing get do go]. This corpus is drawn from the hypernym forms of the parts of speech: [adverb noun verb].

This initial corpus is similar to the reported meanings exchanged in chimpanzee gestural dialogues. The gesture meanings given in table 1 are derived from a large video corpus of wild chimpanzee single symbol gestural exchanges (Hoabaiter 2014). The meanings are mapped to a part of speech and then to a hypernym word for that POS.

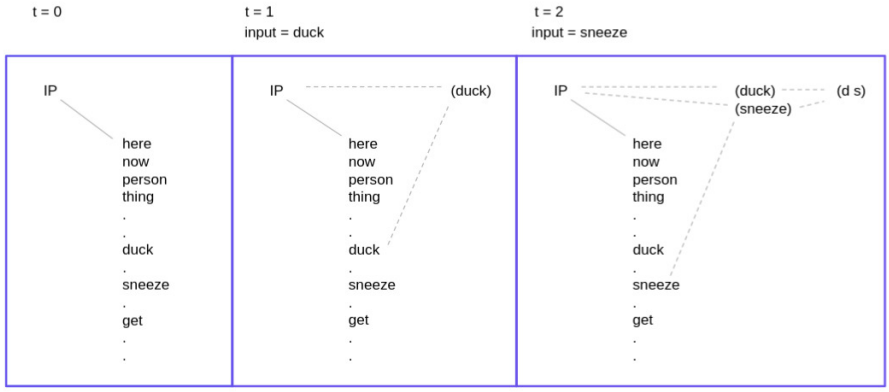

Fig. 3. The child nodes fire based on external input and then touch and fire the ip head node. The ip node performs the merge between itself and the node that touched it. This creates the (duck) and (sneeze) nodes. The (sneeze) node fires after it is created and it performs a merge with itself and its next youngest sibling to create the (d s) node.

```
class  Node {

   string      symbol
   float       value                    //  [-1, 0, 1]  =  F,  ?,  T

   int         input_level              // current input value
   int         threshhold               // threshold to fire


         pair< Node*,  int weight >     head
         pair< Node*,  int weight >     copy
    list < pair< Node*,  int weight > >  branches


   Node()                         // base    constructor

   Node( string, head )           // string  constructor

   Node( head,   copy )           // merge   constructor


   touch  ( weight, depth )

   output ( tabs,   depth )

};
```

```
main() {

   Node   ip( "ip", null   )

   Node   head( "head", ip )

   Node   copy( "copy", ip )

 Node   child( head, copy )

};
```


Fig. 4. The Node class and the use of the constructor functions in main().

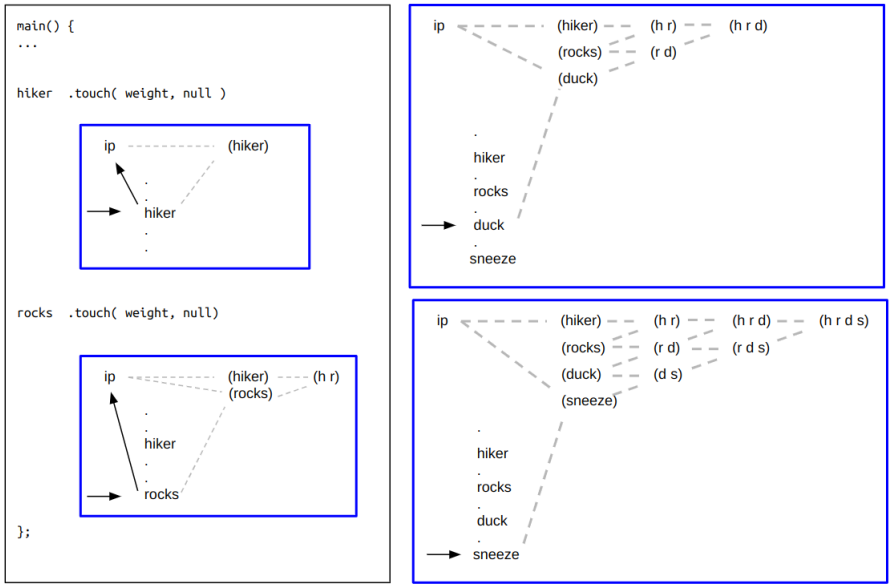

Fig. 5. The hiker.touch() function call causes the hiker node to fire, which then touches and fires the ip node. The stm node is created by a merge between the ip node and the node that touched it. The stm structure then grows iteratively as more symbols are input.

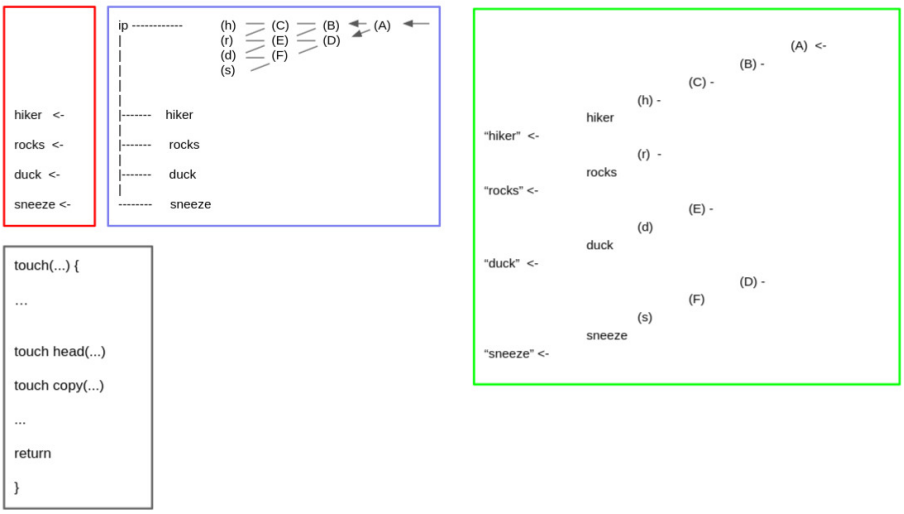

Fig. 6. A c++ pointer to the rightmost (stm) node, labeled "(A)" above, can be returned to main() and used to output the stored node sequence using the touch() function.

Table 1. The chimpanzees exchange approximately 15 unique meanings using gestures. The hypernym forms are mapped to the closest meaning and can all be nominally matched. No grammar or random combinatoric use of gestures is observed. All species of Great Apes share a common set of approximately 100 gestures with each species using a subset of ∼60 gestures, however, the gesture to meaning mapping is different in every species.

Gesture	Meaning	Part of Speech	Hypernym Form
grab,...	stop that	negative	"no"
mouth stroke, ...	acquire object	verb - get	"give"
bite,...	contact (affection)	verb - feel - ?	touching
big loud scratch, ...	init grooming	verb	"do"
arm swing, ...	move away	verb - go	"go"
beckon,...	move closer	verb - come	come
big loud scratch, ...	travel with me	prep	with
jump, ...	follow me (sex)	prep	with
foot present, ...	climb on me	prep	on
reach	climb on you?	prep - ?	on?
present location	groom here	adverb	here
leaf clipping, punch ground	sexual attention - male	?	flirt
leaf clipping	sexual attention - female	?	flirt

Switching back to humans, our understanding of the neocortex is expanding at a great rate using many types of experimental methods. fMRI experiments in (Epstein 1998) and (Huth 2013) have identified two voxels ($< 2mm^3$) that fire in response to words that are hyponyms of ∼(person/place) and (thing). Movies are shown to volunteers and the hypernym mappings from WordNet are used to tag 1800 nouns from the movie dialogues to person, place, or thing. In all volunteers, these two voxels can be identified in similar locations on a neocortex flatmap and show activation when the corresponding hyponym words are used in the movie dialogues.

The symbols of the proto-language are assumed to be formed as child nodes in the interface parcel as shown in figures 7-9. The stm memory allows for storage of state information and for simple dialogues.

5 Movement, boys eat what? what boys eat?

A mechanism for movement can be implemented by using the stored values in the (stm) nodes and modifying the pseudocode as shown in figure 10. The input value of zero is propagated to the (stm) nodes as shown and then can be used to causes movement in the subsequent output order of the stored symbols.

This scenario is implemented in the c++ prototype as shown in figure 11. The blocks of text separated by horizontal dashed lines are static printouts of the ip graph (no arrows) or runtime graph flow diagrams (with arrows). The input sequence is "boys eat what", where "what" = thing:0. Following input, a

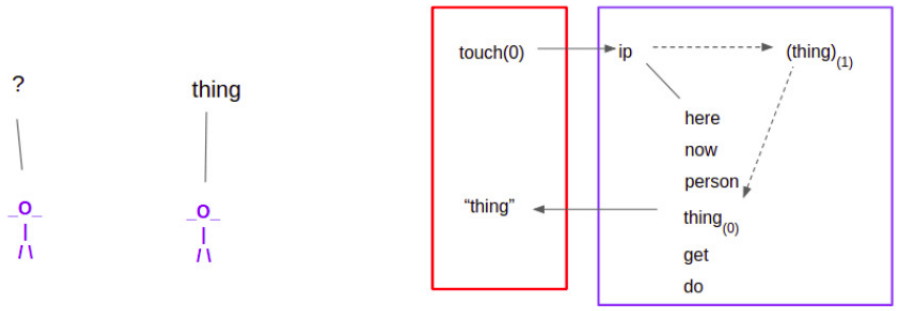

Fig. 7. The value of zero passed in the touch(0) function indicates a question; -1, 0, 1 = [no, ?, yes]. The simplest input form of a question is the function call: ip.touch(0). The stm (thing) node is assumed to have been previously formed.

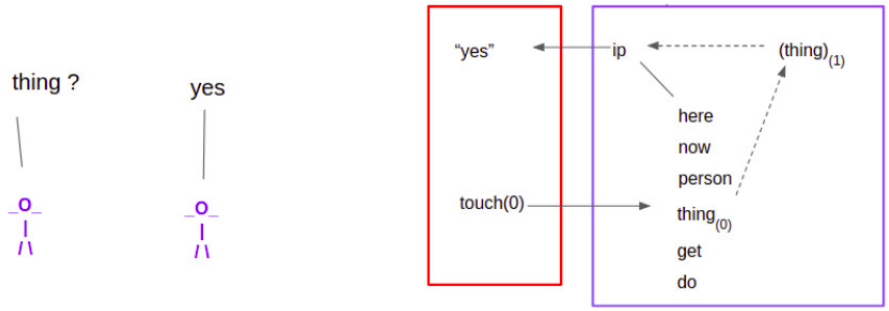

Fig. 8. All symbols can be input as questions with the touch(0) function call as: thing.touch(0) .

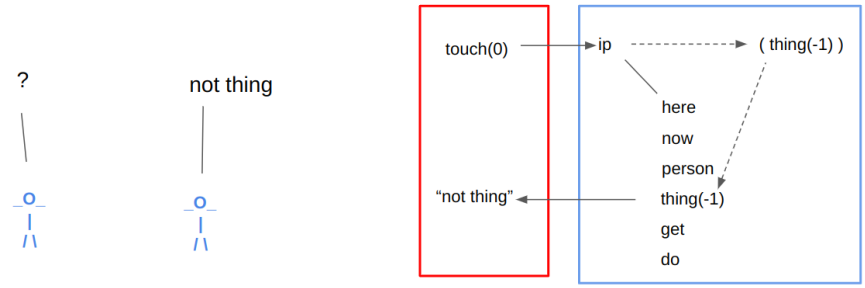

Fig. 9. An stm node having a value = -1 indicates negation. In this scenario, (thing(-1)) was previously created via an stm merge.

pointer the the (b e t) node is returned to main() and this is used in the call: (b e t)->touch() to invoke the output: “what boys eat”

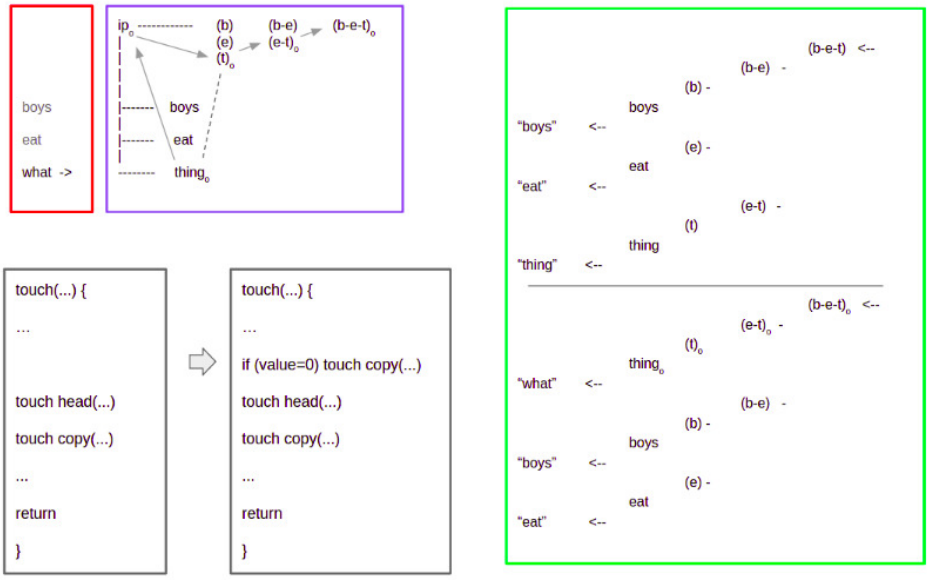

Fig. 10. The single line modification to the pseudocode will alter the output ordering from “boys eat what” to “what boys eat”.

6 Adverb periodicity, here I now eat daily

The set of words in a language can be bifurcated and mapped into two symbols “a” or “n”, which correspond to the adverbs/adjectives and the nouns/verbs/prepositions.

The Wordnet English corpus of 120K definitions and 60K glossary sentences can be encoded into such reduced sequences and used as input as shown in figure 12. These sequences can be considered a “Truncated English”. The terminal nodes of the ip graph are then bifurcated further to form a “Less Truncated English”. Oscillation of the a-n graph is posited to produce the ubiquitous phenomenon of adverb periodicity in human language: quickly I eat; I quickly eat; I eat quickly.

7 Compare function at the merge

here -here -HERE

A compare mechanism exists such that similar sensations, separated in time, can be compared. A familiar example is hearing the first two intermittent sounds

```
kwd1:code20$ c++ graph1.cpp
kwd1:code20$ ./a.out
------------------------------------------------------------
IP
                boys  0
                thing 0
                eat   0
------------------------------------------------------------

                boys:1  <--
IP:1  <--
|
(b)

                eat:1  <--
IP:1  <--
|
(e) (b e)

                thing:0  <--
IP:0  <--
|
(t) (e t) (b e t)

------------------------------------------------------------

IP              (b) 1           (b e) 1         (b e t) 0
                (e) 1           (e t) 0
                (t) 0

                boys  1
                thing 0
                eat   1
------------------------------------------------------------
time interval occurs

                                            (b e t):0  <--
                                (e t):0  <--
                (t):0  <--
  thing:0  <--
                                (b e):1  <--
                (b):1  <--
  boys:1   <--

                (e):1  <--
  eat:1    <--
------------------------------------------------------------
```

Fig. 11. c++ prototype output of "what boys eat?" Here "thing:0" is defined to be "what"

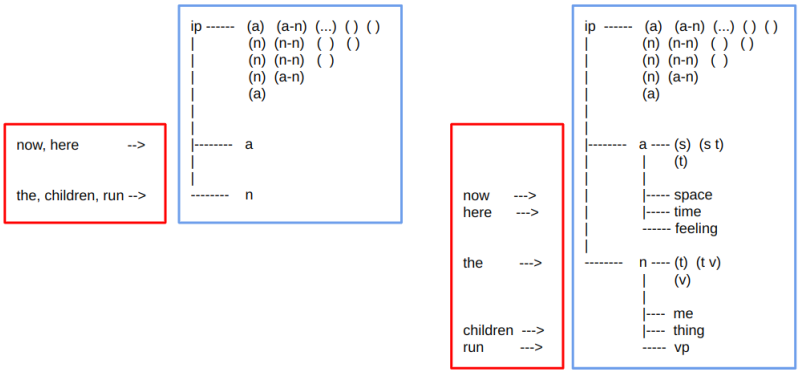

Fig. 12. The ∼120K unique words in the WordNet corpus can be mapped to 'a' or 'n'. "Now the children run here" would map to ip (stm) memory as shown. The a and n branches are then bifurcated further as shown. The words in the corpus are mapped to one of the six hypernym forms. The stm memory mechanism functions within each branch of the graph.

of crunching leaves when someone or something is moving in the woods, relative to a fixed observer. The change in intensity is available at the interface parcel as an internal feedback to the observer.

The ability to compare two of the same sensory inputs separated in time is an important evolutionary advantage to all animals. The initial measured information from each sensation must be stored through time and then compared with the second measurement at a later time. This functionality can be achieved at the second level (stm) merge, shown in the adverb branch of figure 13, using the values of the two parent nodes.

This function, storing a value and using it in a compare operation later in time, is similar to that of the Reichardt model used by Hubel to explain directionally sensitive neural circuits in V1 (Hubel 1980).

8 Conjunctions, illicit conjunctions, movement of conjunctions

An "and" node is added to the IP graph, with the corresponding (stm) symbol labeled (+), as shown in figure 14. As introduced, this node would have no additional properties not already described.

An additional recursive touch() call is then added to the touch() function to touch the (stm) node's head as each node in an (stm) diagonal layer is added. This is shown by the arrows in figure 15. In figure 15, after "time" is input, the horizontal sequence of nodes that terminates in "||" forms a closed loop, which can be detected by the touch() function, allowing it to return an enhanced return value of 2 as indicated.

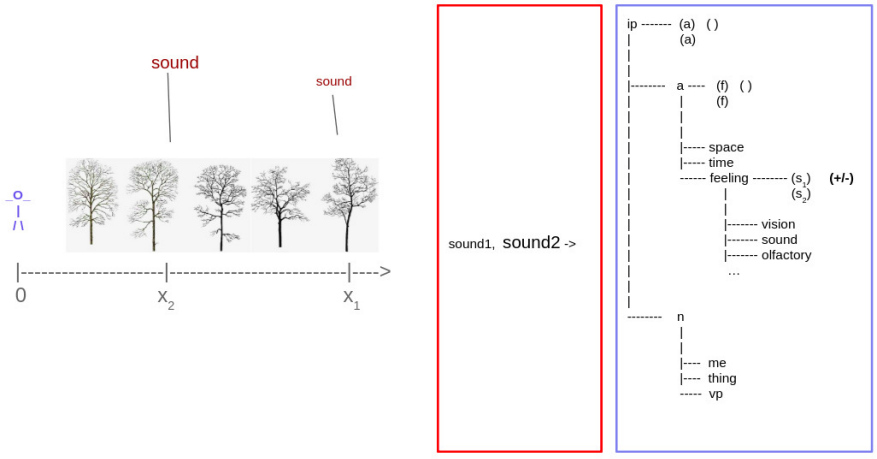

Fig. 13. State information is stored in the adverb branch and can be used in a subsequent compare operation.

The return value of 2 can then be detected in the touch function and used to trigger a Hebbian enhancement of the weights between the calling and called nodes in (stm) memory.

An illicit conjunction sequence can be created with the model by introducing a single logical change to the (stm) nodes that are copy-rooted by the (+) node, namely, that if the return value in the Hebb loop is <2, then return -1. This is shown in figure 16, where the (a +) node would detect the non closed loop condition (RETURN=1) and change the return value to -1.

The value=-1 is retained in the (stm) structure and can be propagated to subsequent (stm) recall. Such a mechanism could be used during sleep to exclude the stored (stm) sequence from normal [ltm] sequence storage.

Movement of a conjunction (when and where boys eat?) is possible by using the enhanced connection weight values stored in the closed loop. In the c++ prototype output shown in figure 17, the input sequence: "thing space:0 and time:0" (= boys where and when?) is input to the IP graph. The (n a + a) node is then touched from main() to cause the output sequence corresponding to: "when and where boys?"

The conjunction node functions to create a layer of (stm) nodes that returns a false signal if a closed loop is not detected. Additionally, once a closed loop is detected, the subsequent (stm) nodes in the conjunction layer (example node: (n a +) in figure 17) can be disabled by storing a value=2 in the (stm) nodes. This process occurs locally, as each (stm) node is touched in the Hebbian loop that runs for each row of (stm) memory.

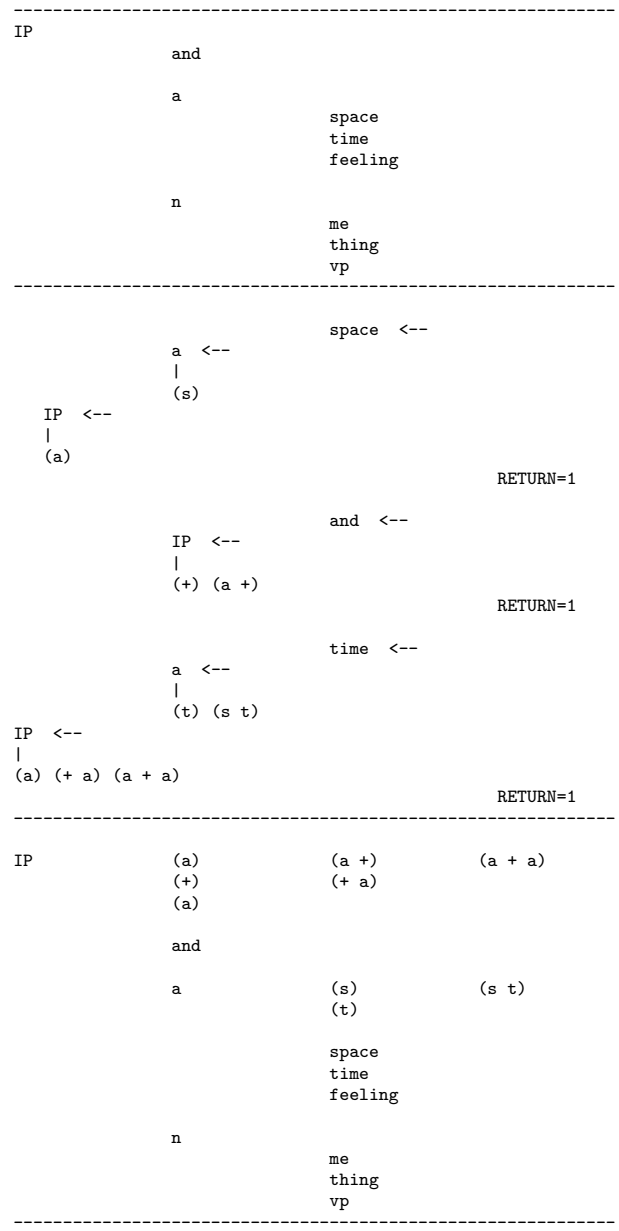

Fig. 14. Inputting "space and time".

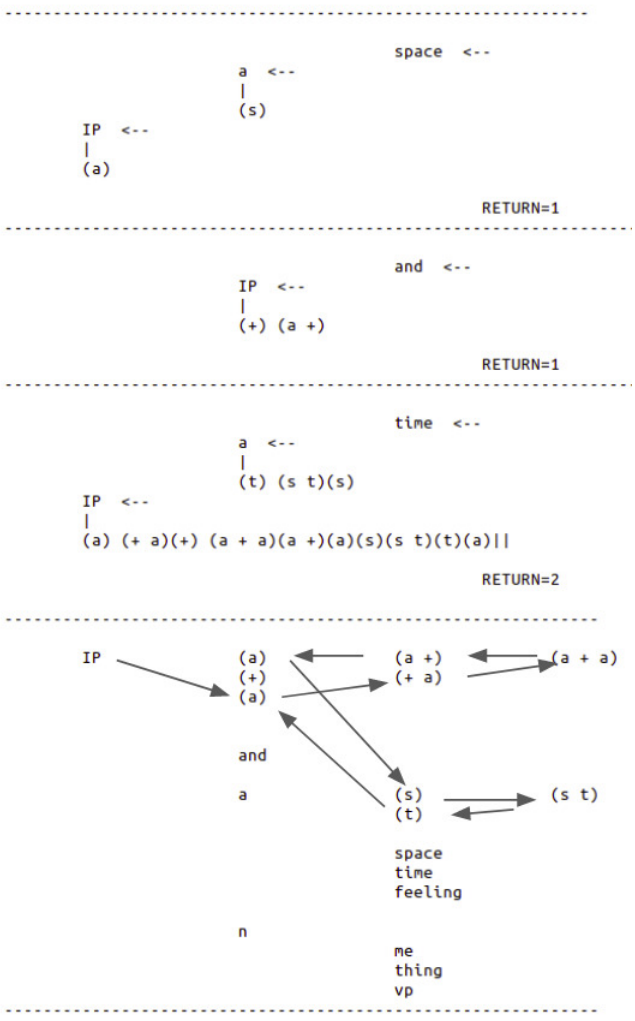

Fig. 15. Closed loop detection and Hebbian one shot binding. The touch() function builds a list of node pointers as it recursively calls itself and can detect the closed loop and return an enhanced return value.

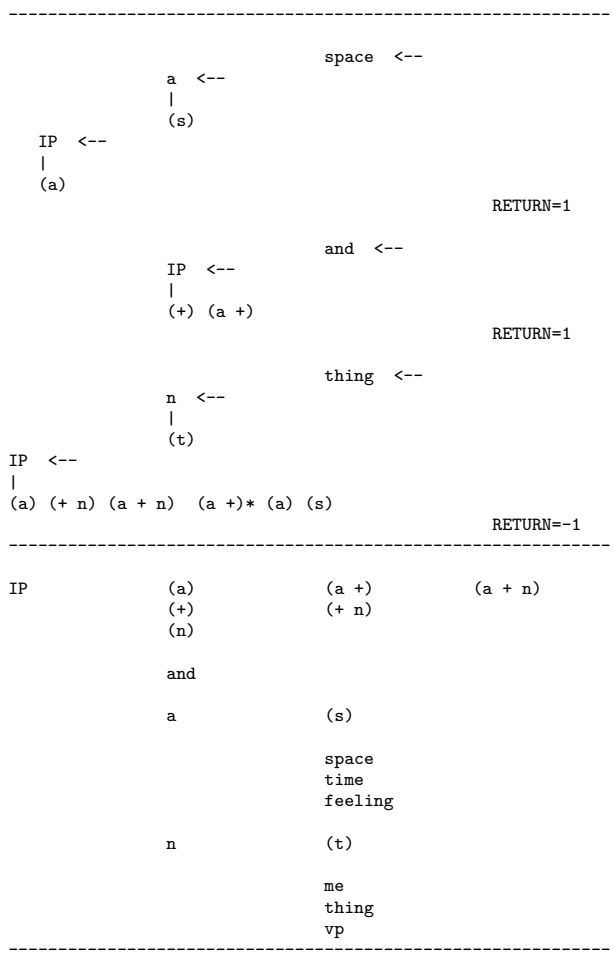

Fig. 16. Illicit conjunction: space and *thing. A RETURN=-1, marked by the '*', is generated by the (a +) node following the input of "thing", based on the non closed loop condition (RETURN=1 from (a)).

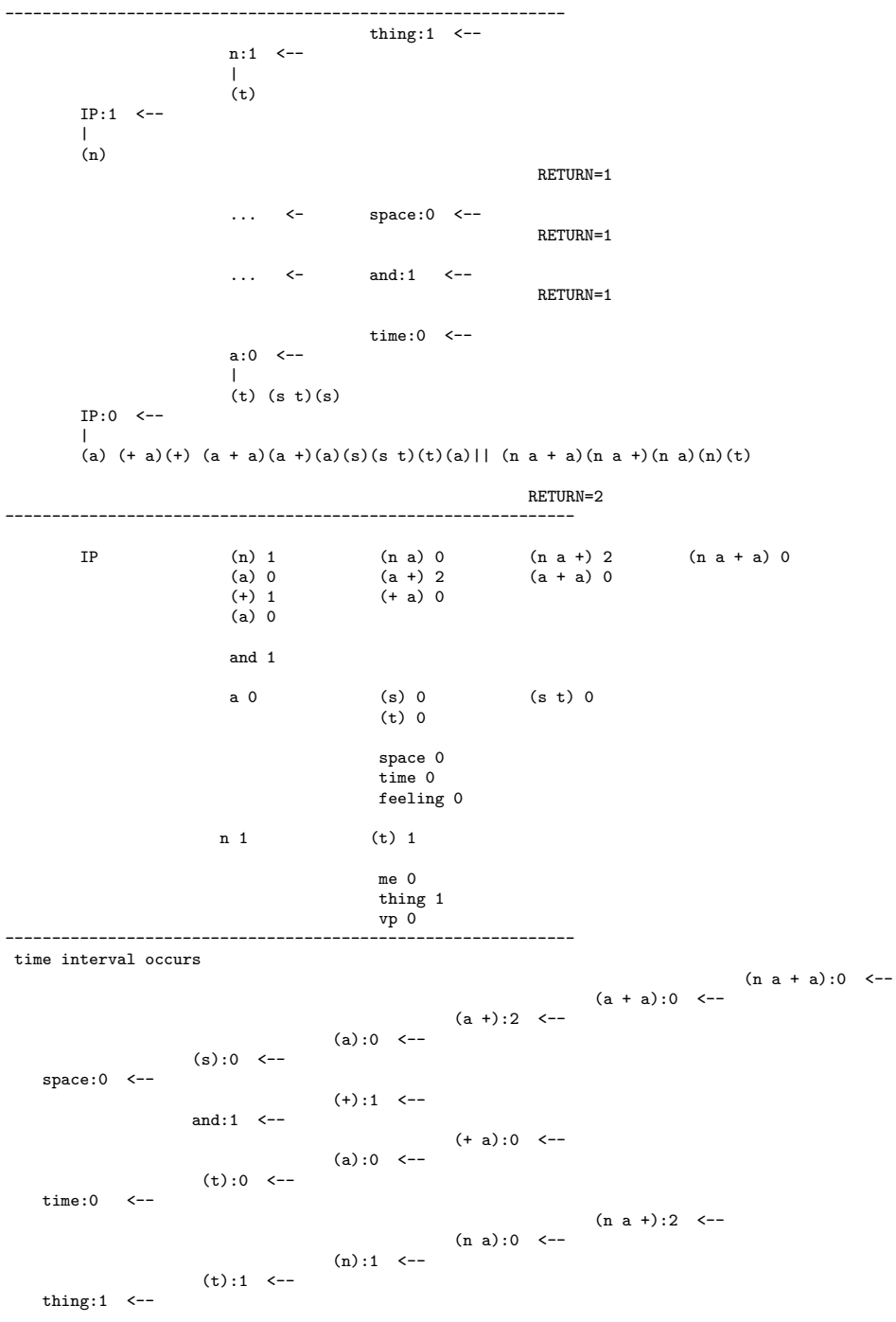

Fig. 17. Movement of a conjunction is accomplished using the modified bindings and the value of zero stored in IP (stm) memory. The truncated English input sentence "thing where and when?" is input and then output as "where and when thing?", where 'space:0' and 'time:0' are defined to be where and when.

9 Long Term Memory, Sleep, Prediction

A permanent long term memory node can be made at the time the (stm) node is made by the merge() function, as shown in figure 18 using the square bracket notation: [ltm]. The (stm) and [ltm] nodes can also be connected by the merge function. The [ltm] node retains the label of the copy node as shown.

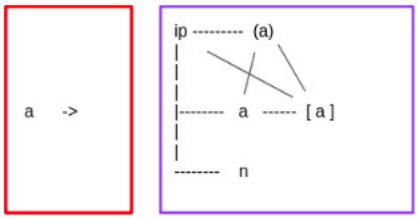

Fig. 18. A long term memory node is formed by a second merge operation with modified parameters.

In the touch() function, the creation of [ltm] nodes by the merge() function can be toggled on and off with a single global variable. Once created, [ltm] nodes can be called by the (stm) nodes at the end of the existing touch() function using the link that was created at merge() time. This input sequence is shown in figure 19. The call to the [ltm] node can also be toggled on and off; which would be analogous to inputting into (stm) rapidly without time to process the touch() call to the [ltm] node.

A dreaming mode is introduced in main() to replay the accumulated (stm) sequences and to build the associated sequences in [ltm] memory. This allows the [a n] node to be created as shown in figure 20.

Once [ltm] nodes have been created, the (stm) nodes can be cleared from the graph and the [a n] node can be used to reproduce the node firing sequence that will rebuild the (stm) pattern in memory.

A simple prediction process can occur using the stored [a n] sequence, namely to predict an (n) node in (stm) memory following input of an "a" symbol. This is shown in the graph flow shown in figure 22, where the symbol "space" is input, leading to the [a] -> [a n] -> [n] -> n -> IP -> (n).9 -> (a n).9 sequence to fire. Here the predicted (stm) nodes are created with value = .9 to differentiate them.

10 Verbs and prepositions

The head-connection weight of the [vp] node is manually modified in main() to cause its head node (the vp node) to fire when the [vp] node fires. This causes a second layer of (stm) nodes to be built in (stm) memory upon input of "vp",

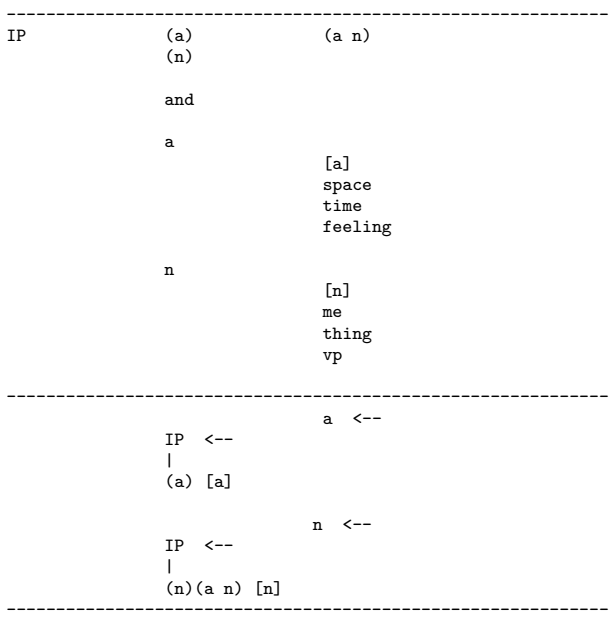

Fig. 19. The [ltm] nodes fire at the end of the existing (stm) node sequences using the link set up at (stm) merge time.

as shown in figure 23. This second (stm) layer can then alter subsequent graph operations, similarly to the "and" node.

In figure 24, "I eat food" is input, causing the (stm) layers to be built as shown. Input of "food" causes a closed loop to form (return=2) in nvp and ip (stm) memory. This would permit enhanced binding of the subject and object, using the same mechanism as the conjunction node.

We introduce a modification to the touch() function for the layer of (stm) nodes introduced by the second [vp] loop. This is shown in figure 24 following the input of "food"; a closed loop is detected in the nvp (stm) memory and a return value = 2 is returned to the calling nodes. This is posited to cause the $()_2$ (stm) node to touch() its copy node when the return value = 2 from the closed loop. This causes the graph flow to follow the (stm) memory back to the specific verb node and would allow for one shot Hebbian enhancement of the (stm) bindings between the verb and the subject+object.

Illicit direct object constructions can be detected to cause a return value of -1. A purpose of this function would be to keep the illicit input node from forming enhanced bindings in (stm) memory by inhibiting the Hebbian one shot weight enhancement mechanism. This function can be implemented using the same mechanism as the conjunction node. In the graph flow shown upper right in figure 25, the input "I eat *drink" is depicted. Input of "drink" causes the

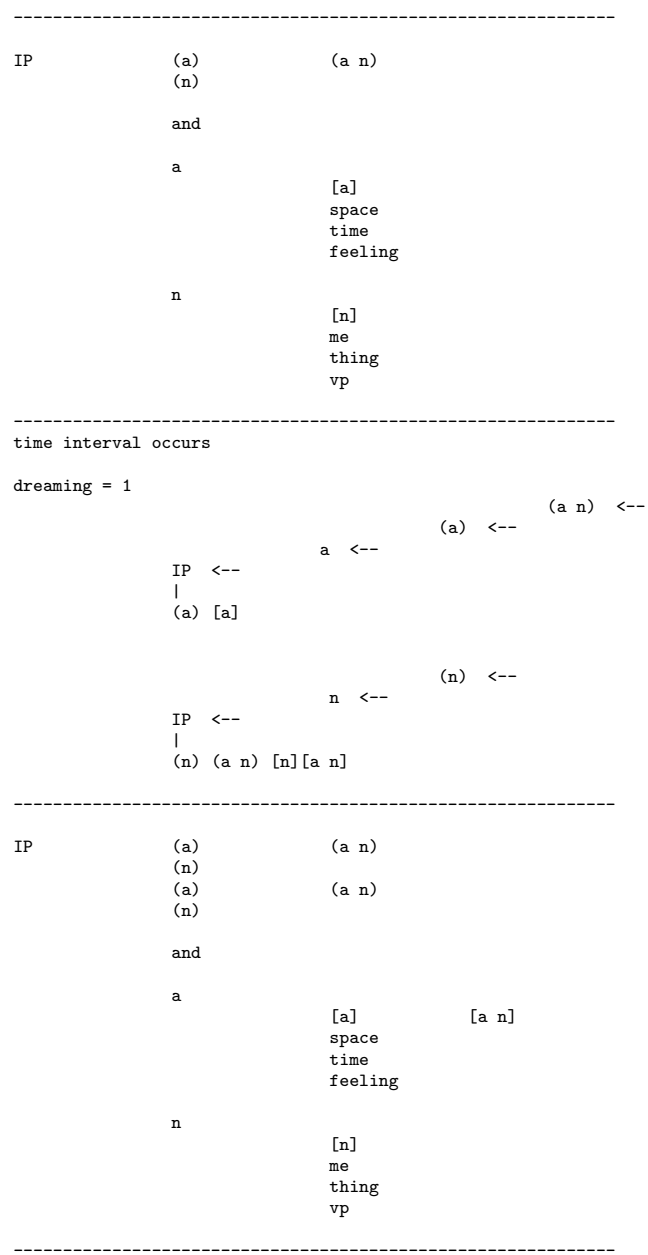

Fig. 20. A dreaming mode allows previously created stm memory structures to duplicate in [ltm] memory by replaying the stored (stm) pattern with modified network parameters for the merge function.

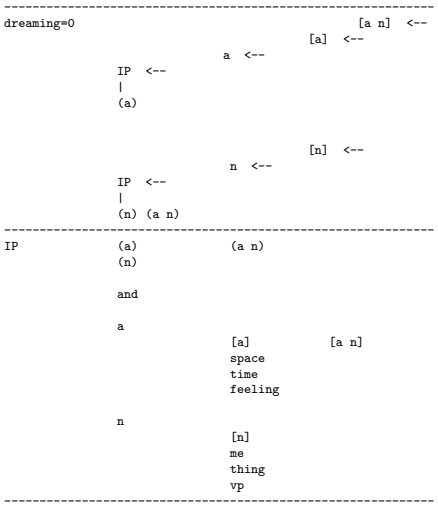

Fig. 21. The [a n] node can then be used to recall the (a) (n) pattern to (stm) memory.

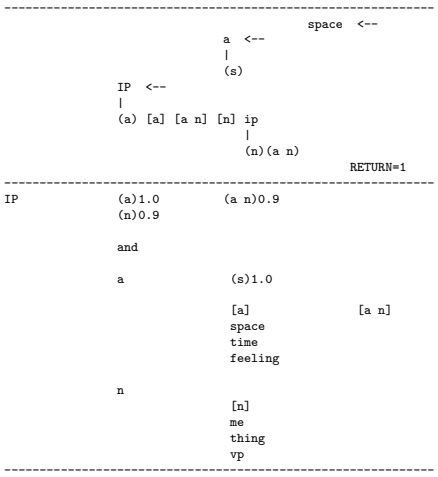

Fig. 22. Prediction of “n” after the input of “space”

```
------------------------------------------------------------
                                        vp  <--
                              n  <--
                              |
                              (v1) [vp] vp n(v2)(v1 v2) IP(n1)
                    IP  <--
                    |
                    (n2)(n1 n2) [n]
                                                  RETURN=1
------------------------------------------------------------
IP                  (n1)            (n1 n2)
                    (n2)

                    and

                    a
                                    [a]                [a n]
                                    space
                                    time
                                    feeling

                    n               (v1)               (v1 v2)
                                    (v2)

                                    [n]
                                    me
                                    thing
                                    vp                 [vp]
------------------------------------------------------------
```

Fig. 23. The vp node is posited to be the hypernym node for the verbs and prepositions. The weights of the [vp] node are such that it fires the vp node a second time, generating a second loop in (stm) memory. This second loop is posited to implement functions of the direct object.

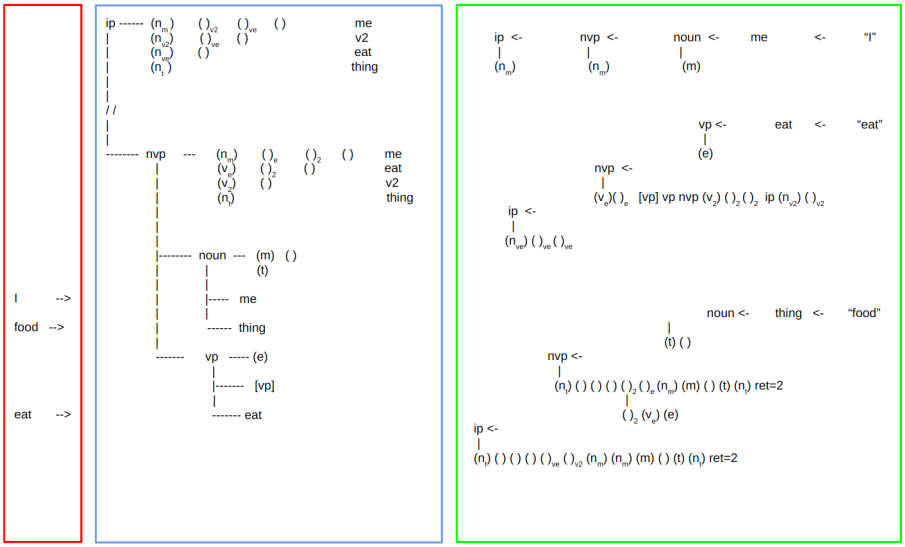

Fig. 24. I eat food.

$( )_{v2}$ node to return -1 by the same rule as used in the conjunction $( )_{+}$ node, namely if the return value (from the $( )_{ve}$ node) is < 2, return -1.

This similar function between the $( )_{v2}$ and $( )_{+}$ nodes indicates that this function could be latent in all (stm) nodes and enabled by change of a local variable. This information would be stored in the permanent [vp] and "and" nodes and passed to the (stm) nodes at run time.

The second construction shown is "I eat now *food". Following input of the "now" symbol, the $( )_{v2}$ node would return -1 on the existing logic, however "I eat now" is a valid construction. This indicates the $( )_{at}$ node must override the return value = -1, as indicated in the graph flow. However, following the input of "food", the $( )_{at}$ node must then produce a return value of -1. This logic could be based on the condition of (return value = 2) + (the Hebbian enhancement already run).

Here the process of making functional changes to the touch() function is based on the desired language outcome. The recursive touch function is such that a single line of code change to the touch function will produced the desired changes to the graph flow.

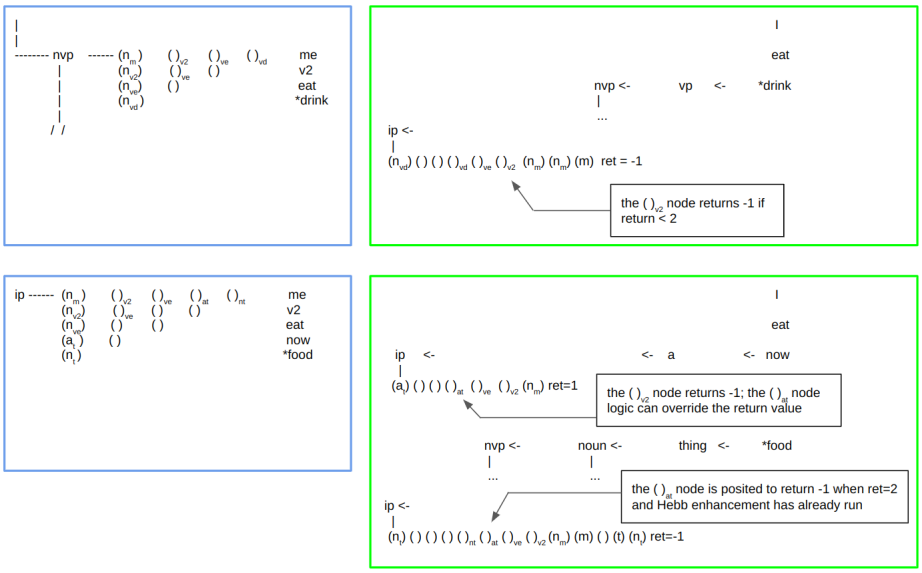

Fig. 25. Illicit direct object constructions, "I eat *drink" and "I eat now *food".

Prepositions are posited to be children of a single node "prep", which is a child of the vp node as shown in figure 26. The [p] node is posited to set an additional layer of (stm) nodes using the same mechanism as the [vp] node. This would allow the Hebbian one shot enhancement mechanism to run onto the "a"

branch when “house” is input as shown in figure 26. The preposition “at” is posited as made by a merge between the [p] node and the [s t] node.

Here we have combined space and time into a single node. This follows the pattern where the branches of the graph branch into two child branches.

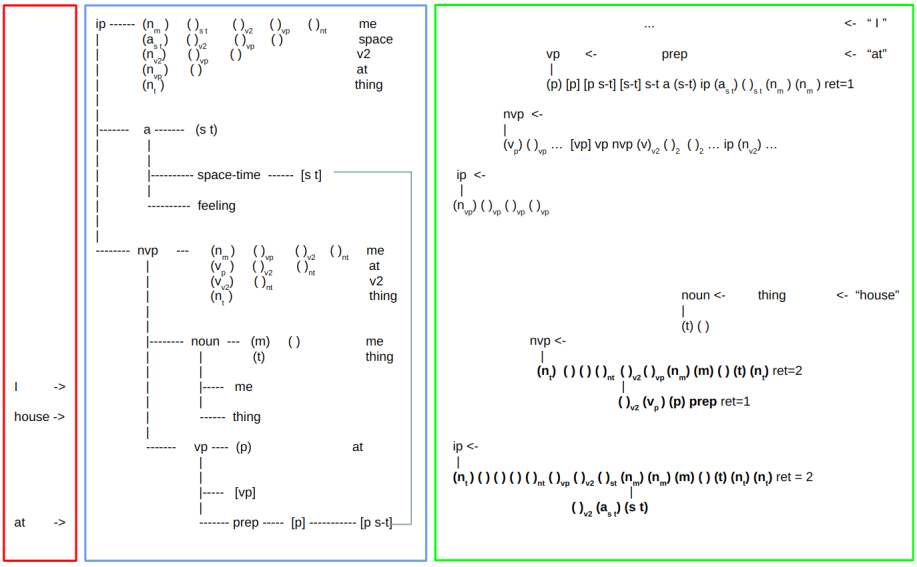

Fig. 26. A “prep” node is created as a child of the vp node. The weights of the head and copy nodes are set so as to produce the (stm) layer in IP memory. The ’at’ node is posited as being formed from a merge between the [p] and [s-t] nodes.

Further child nodes of the prep node would be: in, on, to, through,... The pattern would be to create each new preposition node by a merge operation in main() between the preposition’s archetype and some other high level [ltm] node in the graph.

11 Past and future, progressive and perfected, singular and plural

The [ltm] nodes of the nvp, noun, and vp nodes are posited to develop pairs of child nodes, differentiated by a +-1 stored value, which correspond to the progressive/perfected tenses, singular/plural, and the past/future tenses, respectively.

This value based mechanism allows for illicit grammar detection in all three categories using the same mechanism, as described below.

11.1 Past and future, past irregular verbs

We modify the graph to add two child nodes to the [vp] node as shown in figure 27. These nodes are assumed to carry default values of +-1, corresponding to the past and future tenses. The "will" and "ed" tense markers are mapped to either node as shown. Upon input, the (stm) memory nodes are assumed to carry the +-1 value, which permits detection of the illicit construction when the merge function creates the (*) node in figure 27.

The $[-1]_{\text{PAST}}$ and $[+1]_{\text{FUTURE}}$ nodes are posited to cause an additional (stm) diagonal layer to form as shown.

The future modal verbs can be modeled in sequence, $[+.6]_{\text{can}}$, $[+.8]_{\text{shall}}$, $[+.9]_{\text{must}}$, $[+1.0]_{\text{will}}$.

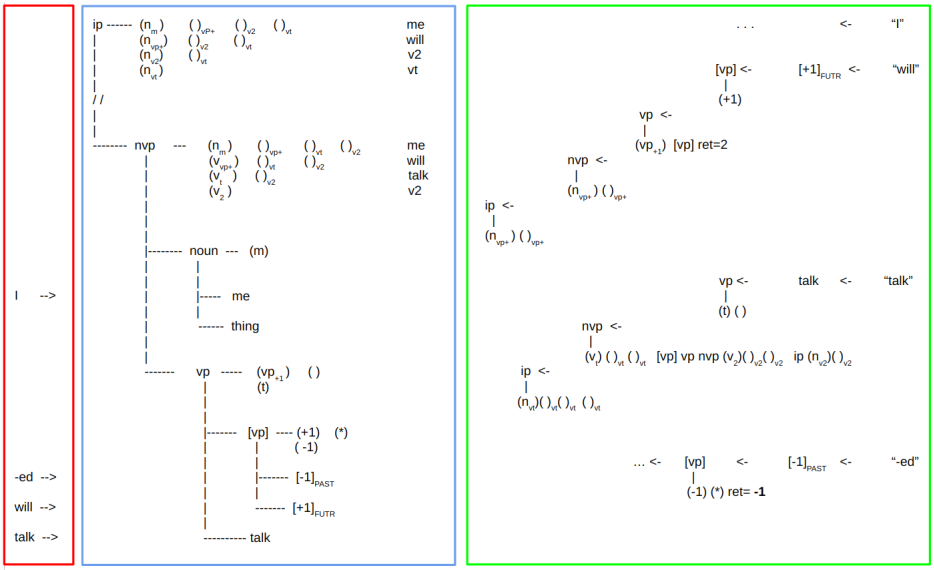

Fig. 27. The [vp] node bifurcates on value to form the past and future tense symbols. Illicit tense detection (I will talk *ed) is possible at the merge of the (+1) and (-1) child nodes of the [vp] node.

Irregular past tense verbs can be represented in the graph by a merge between a verb's [ltm] node and the $[-1]_{\text{PAST}}$ node. The graph flow at input is modified as shown in figure 28. The $[-1]_{\text{ATE}}$ node is assumed to produce the normal verb input but then to also follow its copy link to the $[-1]_{\text{PAST}}$ node, which produces an identical graph input response as the "-ed" verb ending.

11.2 Progressive and perfected

The progressive and perfected tense symbols "are" and "have" are implemented on the graph using the same +-1 branching mechanism from the [nvp] node as

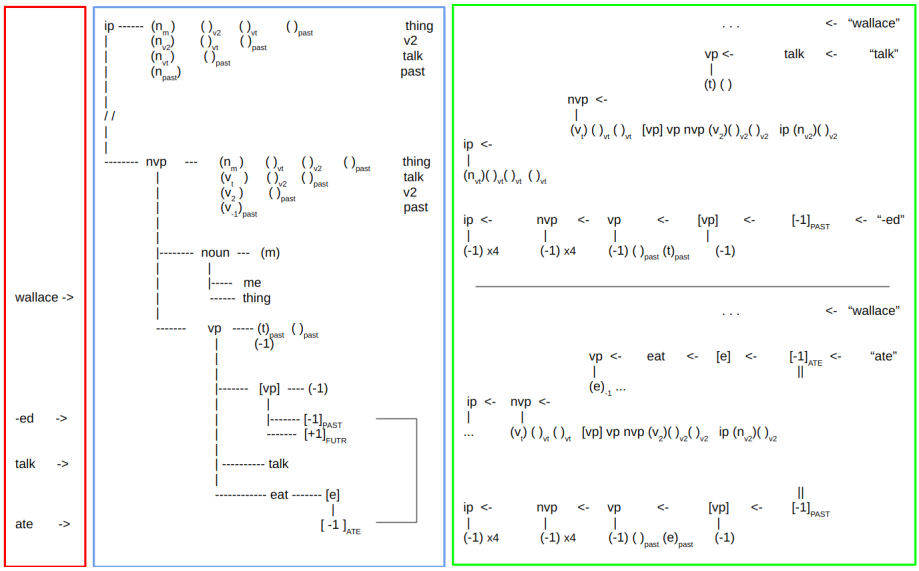

Fig. 28. Irregular past tense verbs are formed by a merge between the verb's [ltm] node and the $[-1]_{\text{PAST}}$ node.

shown in figure 29. Detection of illicit tense grammar occurs in a similar manner, as a merge between (stm) nodes with values=±1.

The $[-1]_{\text{PROG}}$ node is posited to merge with the [vp] node to create the [-ing] node. This permits a similar verb agreement check.

The $[-1]_{\text{PROG}}$ and $[+1]_{\text{PERF}}$ nodes can then form child nodes by merging with the $[-1]_{\text{PAST}}$, [me], and [thing] nodes as shown.

11.3 Singular/plural, my, the, a

Singular and Plural nodes are posited to form as value based branches of the [noun] node as shown in figure 30. "my" and "the" are posited to form as merge nodes between the [me]/[thing] nodes and the [noun] node. The [a] node is posited to form as a merge between the [the] node and the $[+1]_{\text{SINGULAR}}$ node.

12 Executive function in main(), input-predict-[respond], ponder-[respond], sleep

Input of a sentence sequence in main() proceeds as a series of calls to the touch() function of each node in the sequence. The touch() function calls cause (stm) memory structures to accumulate in the ip graph, possibly forming enhanced Hebbian bindings and/or returning any false signals.

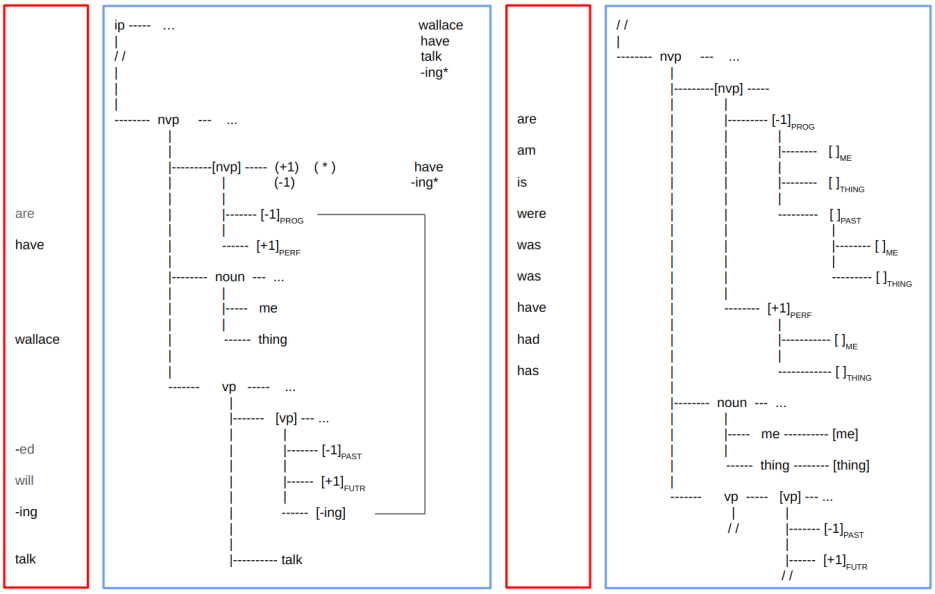

Fig. 29. Progressive and perfected nodes develop as children of the [nvp] node. A negative signal can be generated by the merge in the [nvp] node's (stm), as in: wallace have talk ing*.

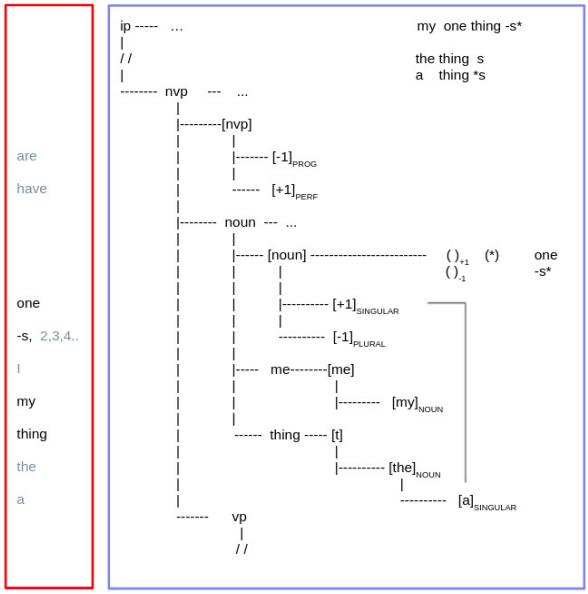

Fig. 30. Posited merge nodes created for "the", "a", and "my" nodes.

The (stm) memory structures that are built for each sentence can be probed from main() to return the rightmost (stm) node. These (stm) nodes can then be touch()-ed from main() to invoke sequence output.

Following an input sequence, the ip graph can be used to generate a prediction and/or a question based on its stored [ltm] and (stm) memory.

A ponder mode is posited whereby the same prediction mode would run, but with modified graph parameters to allow for deeper recursive function calls to probe [ltm] memory that could be recalled and utilized.

Repeating this process on a training corpus would then accumulate a set of (stm) nodes that can encode the sequences that have been processed since the last sleep cycle. A sleep cycle can then reprocess the (stm) memories into [ltm] structures.

The ip graph can then be extended manually in main() by doing merge() function calls between existing terminal nodes, following which, the appropriate words in the training corpus are mapped to the new terminal graph nodes and the process of corpus processing is repeated, allowing for additional graph terminal nodes to be added.

13 Discussion, Similar Work

The graph model derived herein draws heavily from the interdisciplinary field of Psycholinguistics which is a broad research area concerned with all aspects of human language (Fitch 2017).

The Strong Minimalist Thesis described in (Bolhuis 2014) proposes that a single recursive function, merge(head, copy), produces all observed human language and grammar. (Lieberman 2015) argues against the SMT model, citing that the current diversity in human languages would preclude a single model that evolved rapidly 100 Kya; arguing that the grammatical information necessary to encode the set of all human languages would have had to have been fully specified at a single point of evolution. Lieberman notes that the evolution of the human vocal tract can be traced to 500 Kya, and argues that language acquisition can be explained via similar mechanisms as motor control in the neocortex, which would also be common to all mammals. (Fitch 2017) discounts the link to vocal tract evolution, citing that all primates have a similar "speech ready" vocal tract and that complex language has only evolved in humans.

(Phillips 2013) argues against the right to left order of merge operations used by the SMT model to account for sentence object movement and instead that a left to right assembly of sentence components is most likely to represent the underlying physical processes at work. In the SMT model, the sentence is first premeditated as a directed graph (Bolhuis 2014, fig. 1). A merge operation is then posited to occur repeatedly at each vertex until the empty stub is encountered, causing the symbol to be linked in place. The sentence would then be externalized.

In the prototype c++ graph model, as shown in figure 11, the sentence is input as sequence of touch() calls from main() and then externalized via a subsequent function call: (b e t)->touch().

The Node class defines a single recursive runtime function, touch(). The prototype for this function is implemented as any node in a neural network would be. The touch function takes an input, adds it to an internally stored activation level, and if the node fires, it touches the nodes that it is connected to, as shown below:

```
touch( input, value ){

  if node fires:

     head.touch(...)

     copy.touch(...)

     loop child branches{
       branch->touch(...) }
}
```

A "value" parameter is also passed in the touch() function which is specific to the graph model. If the node fires, the passed in value will overwrite the node's current value. In the SMT model, the graph nodes do not have a value parameter. The value parameter adds an internal degree of freedom to the node class that can then be used to produce movement and other language phenomena using a compact recursive algorithm. The change to the touch function to cause movement is given as a single line change to the pseudocode as shown in figure 10.

Short term memory nodes are added to the IP graph via a merge operation that occurs between the nodes in the graph as they fire. We introduce a direction parameter to the call to the touch function and perform the merge with a different head node for the left and right directions as shown:

```
touch( ..., touched_by, direction ){

  if node fires:

  if direction == left
       new_child_node = merge( touched_by,      *this )

  if direction == right
       new_child_node = merge( head.last_child, *this )


  new_child_node-> touch( *this, right)

   ...
}
```

The left merge creates the first level (stm) nodes in figure 5, i.e. (hiker), (rock), (duck), and (sneeze). The right merge creates the higher order (stm) nodes, i.e. (h r), (r d), (h r d), etc.

Each newly created (stm) node is then touched and fired with direction = right. This allows the (stm) nodes to form in the ip graph as shown in figure 11.

Each subsequent input symbol causes a diagonal layer of (stm) nodes to be added to the half matrix of (stm) nodes. In the second block of runtime graph output in figure 11, an output mode flag is used to suppress the merge() operation between the (stm) nodes as they fire with direction=left.

The graph mechanism for coordination is then deduced from the structure of the graph and simple conjunction examples. The symbols in the ip graph are bifurcated into (adverbs/adjectives) and (nouns, verbs, and prepositions) as in figure 12, and then further bifurcated within each branch as shown. This allows for the closed loop detection algorithm to be implemented as shown in figure 15. When a closed loop is detected, the weights of all the nodes in the loop can be enhanced in the touch function when the nodes in the loop return in the order that they fired. The enhanced bindings between the nodes can then be used in movement of the conjunction, as in figure 17.

```
here and now              // true

Wallace and Gromit        // true

Wallace and *now         // false
```

Illicit conjunctions are posited to be detected as a non closed loop condition upon return of the (a +) node in figure 16. The (a +) node is copy rooted to the 'and' node, which would allow some local variable or flag to the inherited by the (a +) node and then subsequently to be used to invoke logic to return a false indication to main().

Long term memory nodes can be created in the graph model by introducing a dreaming mode and modifying the touch() function to do an additional merge to create an [ltm] node, if it does not already exist, as shown in figure 18. The dreaming mode is enabled from main() by setting a global flag. Sequences of symbols that have been input and stored in (stm) memory can be replayed from main() with the dreaming mode flag set. The existing logic in the touch() function can then be used to store the corresponding symbol sequence in [ltm] memory.

All mammals can store long term memories of sequences of symbols and recall and utilize this information after sleeping. Navigational path information about a mammal's local environment would be presumably stored as a sequence of symbols. For instance, a complex path to water/food from a mammal's home location would be used on a daily basis by the mammal.

The 'n' branch of the ip graph in figure 23 is bifurcated into me, thing, and 'vp'. The vp node is posited to be the hypernym node for all verbs and prepositions and to implement the functions of the direct object. The function of the direct object can be implemented with the ip graph model by modifying the connection weights to the head node of the [vp] node, i.e. the vp node. This causes the [vp] node to touch() and fire the vp node a second time, which then fires to the ip node, which has the effect of adding a second layer of (stm) nodes to the (stm) nodes that are already present, as shown in figure 23.

The direct object functionality is drawn in figure 24 using the example "I eat food". The input of the direct object causes a closed loop to be detected

and allows for Hebbian one pass weight enhancements of the nodes in the loop. Specific verbs and preposition nodes can be added to the graph as children to the vp node and will inherit this direct object functionality on input. The second layer of (stm) nodes can also be used to return a false signal for illicit direct object constructions, i.e. "I eat *quickly food", using the same mechanism as is used for the illicit conjunction.

A 'prep' node is posited as a child of the vp node, and to cause a third layer of (stm) nodes to be added to (stm) memory using the same mechanism at the [vp] node, as shown in figure 26. In figure 26, the [p s-t] node is posited to represent the preposition 'at'. The [p s-t] node is created via a merge call from main() between the existing [ltm] nodes for [prep] and [space-time].

The model is then extended by creating a value based bifurcation of the [vp] node as shown in figure 27. Here the past and future tense symbols "ed" and "will" are introduced as child nodes to the [vp] node. The inferred functionality is that these nodes produce an (stm) node with a ± 1 value upon input. This allows for detection of the illicit construction "I will talk *ed" as shown. Here the detection would occur in the merge() function that creates the (*) node in figure 27.

Irregular past tense verb symbols are formed as a merge between a verb's [ltm] node and the $[-1]_{PAST}$ node as shown in figure 28. When the $[-1]_{ATE}$ node fires on input, it fires its head and copy links and produces the same internal representation as the verb + 'ed' marker.

This value based bifurcation is duplicated to the [nvp] node in figure 29, where the [+1] and [-1] nodes are inferred to represent the progressive and perfected tense symbols. Here the invalid construction "Wallace has talk *ing" is detected in the merge of the (*) node in the [nvp] node's (stm) memory. In figure 29 right side, the terminal nodes of the current English grammar are created by doing merges between the $[+1]_{PROG}$ and $[-1]_{PERF}$ nodes and the [my], [thing], and $[-1]_{PAST}$ nodes. As before, these merge function calls are directly encoded in main().

This value based bifurcation is again duplicated to the [noun] node in figure 30, where the [+1] and [-1] nodes are inferred to represent singular and plural symbols. Here the invalid construction "my one thing *s" is detected in the merge of the (*) node in the [noun] node's (stm) memory. The [noun] branch could be expanded to encode mathematical relations. Logical relations would presumably have to occur in the top level ip (stm) memory due to the placement of the 'and' node.

The WordNet dictionary corpus contains 80K, 20K, and 10K, nouns, verbs, and adverbs/adjectives respectively. In the graph model, each of these words would map to a terminal node in the graph, at whatever stage of development of the graph. Approximately 30 terminal nodes are described herein. The sentence "I washed the car yesterday" would be reduced into a truncated English form: "me did thing before" at roughly the current development level of the graph.

If the graph model is representative of the human neocortex connectome, then we would expect there to be a unique terminal node for each word in the

WordNet corpus. This expansion of the model would occur as merge() function calls in main() between existing terminal nodes. This allows the child nodes to inherit all of the functionality of their parent nodes.

The key difference between humans and the other mammals could be in the level of bifurcation in the graph. If one considers an ip graph with all words as direct children and no hierarchical bifurcations, the graph model would still be able to encode simple dialogue interactions, as in figures 7, 8, and 9, and to encode sequences in (stm) and [ltm] memory, but the closed loop Hebbian weight enhancement process would not produce consistent results and could not be utilized by the mammal.

14 Acknowledgments

The author has received many helpful comments from many people who have reviewed early versions of this manuscript. The author also gratefully thanks the reviewers for their helpful comments.

References

1. Berwick, R.C., et al.: Evolution, brain, and the nature of language. In Trends Cogn Sci. 2013 Feb;17(2):89-98. Epub 2013 Jan 9. See fig. 1. (2013). doi: 10.1016/j.tics.2012.12.002.
2. Bolhuis, J.J., et al.: How Could Language Have Evolved? In PLOS Biology, (8/26 2014). http://dx.doi.org/10.1371/journal.pbio.1001934
3. Bryne, R.W., et al.: Great ape gestures: intentional communication with a rich set of innate signals. In Animimal Cogn. (2017). doi: 10.1007/s10071-017-1096-4
4. Chomsky, N.: Artificial Intelligence, in Navigating a Multispecies World: A Graduate Student Conference on the Species Turn. https://sts.hks.harvard.edu/events/workshops/navigating-a-multispecies-world/ (2015)
5. Chomsky, N.: Some Core Contested Concepts. In J. Psycholinguist Reseasch 44: 91. https://dspace.mit.edu/openaccess-disseminate/1721.1/103525 (2015). https://doi.org/10.1007/s10936-014-9331-5
6. Chomsky, N.: Language, Creativity, and the Limits of Understanding. (4-21-16). At https://www.youtube.com (2016)
7. Chomsky, N.: Language and the Cognitive Science Revolution(s), lecture given at Carleton University 2011. https://chomsky.info/20110408/ (2011)
8. Darling, C.: Guide to Grammar & Writing. https://www.guidetogrammar.org/grammar/
9. Del Signore, K.W.: Measuring and Simulating Cellular Switching System IP Traffic. In Bell Labs Tech. J., 18: 159–180. (2014). doi:10.1002/bltj.21651
10. Epstein R., Kanwisher N.: A cortical representation of the local visual environment. Nature. 1998;392(6676):598–601. The 'parahippocampal place area' (PPA) is described. (1998)
11. Fitch, W.T.: Empirical approaches to the study of language evolution. In Psychon Bull Rev. 2017 Feb;24(1):3-33. (2017). doi: 10.3758/s13423-017-1236-5

12. Goertzel, B.: Artificial General Intelligence: Now is the Time, Google tech talk. At https://www.youtube.com/watch?v=A-dycsiRwB4 (2007)
13. Hawkins, J., Blakeslee S.: On Intelligence. Published by Times Books. (2005)
14. Henshilwood, C., et al.: An abstract drawing from the 73,000-year-old levels at Blombos Cave, South Africa. In Nature 562, pgs. 115–118. (2018). https://doi.org/10.1038/s41586-018-0514-3
15. Hobaiter, C., Byrne, R.W.: The Meanings of Chimpanzee Gestures. In Current Biology Volume 24, Issue 14, 21 July 2014, Pages 1596-1600. (2014)
16. Hubel, D.H.: Eye Brain and Vision. At http://hubel.med.harvard.edu/index.html (1980)
17. Hubel, D.H.: Tungsten Microelectrode for Recording from Single Units. In Science 1957 Mar 22;125(3247):549-50. (1957). doi: 10.1126/science.125.3247.549.
18. Hubel, D.H., Wiesel, T.N.: Receptive fields of single neurones in the cat's striate cortex. In J Physiol. Oct 1959;148(3):574-91. (1959). doi: 10.1113/jphysiol.1959.sp006308.
19. Huth A.G., et al.: A Continuous Semantic Space Describes the Representation of Thousands of Object and Action Categories across the Human Brain. In Neuron. 2012 Dec 20; In 76(6): 1210–1224. See figure 2, PPA analysis. (2013). doi: 10.1016/j.neuron.2012.10.014,
20. Huyck, C.R., Passmore, P.J.: A review of cell assemblies. Biol Cybern 107, 263–288 (2013). https://doi.org/10.1007/s00422-013-0555-5
21. Kruger N., et al.: "Deep Hierarchies in the Primate Visual Cortex: What Can We Learn for Computer Vision?. In IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 35, no. 8, pp. 1847-1871, Aug. 2013. doi: 10.1109/TPAMI.2012.272.
22. Minsky, M.: A Framework for Representing Knowledge, MIT-AI Laboratory Memo 306, June, 1974. http://web.media.mit.edu/ minsky/papers/Frames/frames.html (1974)
23. Moore, Richard: Ape Gestures: Interpreting Chimpanzee and Bonobo Minds. In Cell Press. Volume 24, Issue 14, 21 July 2014, pages R645-R647. (2014)
24. Phillips, Collin: Linear Order and Constituency. In Linguistic Inquiry Vol. 34, No. 1 (Winter, 2003), pages 37-90. (2003)
25. Phillips, Collin: Derivational Order in Syntax: Evidence and Architectural Consequences. In Studies in Linguistics 6: 11-47. (2013)
26. Pinker, Steven: Words and Rules. At https://stevenpinker.com/files/pinker/files/edinburgh.pdf See figure 1. (1994)
27. Scerri, Eleanor M. L., et al.: Continuity of the Middle Stone Age into the Holocene, In Scientific Reports volume 11, Article number: 70 (2021)
28. Tattersall I.: A tentative framework for the acquisition of language and modern human cognition. In J. Anthropol. Sci. 2016 Jun 20;94:157-66. Epub 2015 Mar 25. PMID: 27014833. (2016). doi: 10.4436/JASS.94030
29. repository of code used to generate figures: https://github.com/kwd2/graph1