Brain Algorithms journal club
“Assembly calculus theory in the brain”
Papadimitriou et al. 2020, Mitropolsky et al. 2021 2022

Sabrina Drammis 02/20/22

Outline

1. Vision
2. Model definition
3. Functions and primitives
4. Model implementation for parsing language

Vision

“We do not have a logic for the transformation of neural activity into thought and action. I view discerning [this] logic as the most important future direction of neuroscience.” - Richard Axel

Assembly calculus is a proposed formal computational model of this sought “logic”. The model is based on assemblies (populations) of neurons and their interactions at an intermediate level (between individual neurons/synapses and whole-brain models).

Model definition

- Finite number of brain areas: *A*, *B*, *C*, etc.
    - Areas are regions of cortex

Model definition

- Finite number of brain areas: A, B, C, etc.
  - Areas are regions of cortex
- n excitatory neurons per brain area

Model definition

- Finite number of brain areas: *A*, *B*, *C*, etc.
  - Areas are regions of cortex
- n excitatory neurons per brain area
- Random recurrent connections between neurons within brain areas and across connected brain areas with probability p

Model definition

- Finite number of brain areas: *A, B, C,* etc.
    - Areas are regions of cortex
- *n* excitatory neurons per brain area
- Random recurrent connections between neurons within brain areas and across connected brain areas with probability *p*
- Multiplicative Hebbian learning rule: $w_{ij}^{t+1} = w_{ij}^{t}(1 + f_i^t f_j^{t+1}\beta)$
    - For each synapse *(i,j)* synaptic weight increases by a factor of $1+\beta$ if the post-syanptic neuron fires at time *t+1* and the pre-synaptic neuron fired at time *t*.
    - $\beta > 0$, a higher β causes faster convergence

Model definition

- Finite number of brain areas: *A, B, C,* etc.
  - Areas are regions of cortex
- *n* excitatory neurons per brain area
- Random recurrent connections between neurons within brain areas and across connected brain areas with probability *p*
- Multiplicative Hebbian learning rule: $w_{ij}^{t+1} = w_{ij}^{t}(1 + f_i^t f_j^{t+1} \beta)$
  - For each synapse *(i,j)* synaptic weight increases by a factor of $1+\beta$ if the post-syanptic neuron fires at time *t+1* and the pre-synaptic neuron fired at time *t*.
  - $\beta > 0$, a higher β causes faster convergence
- Only a fixed number of *k* of the *n* excitatory neurons in any area fire (*k*-cap)

Model definition

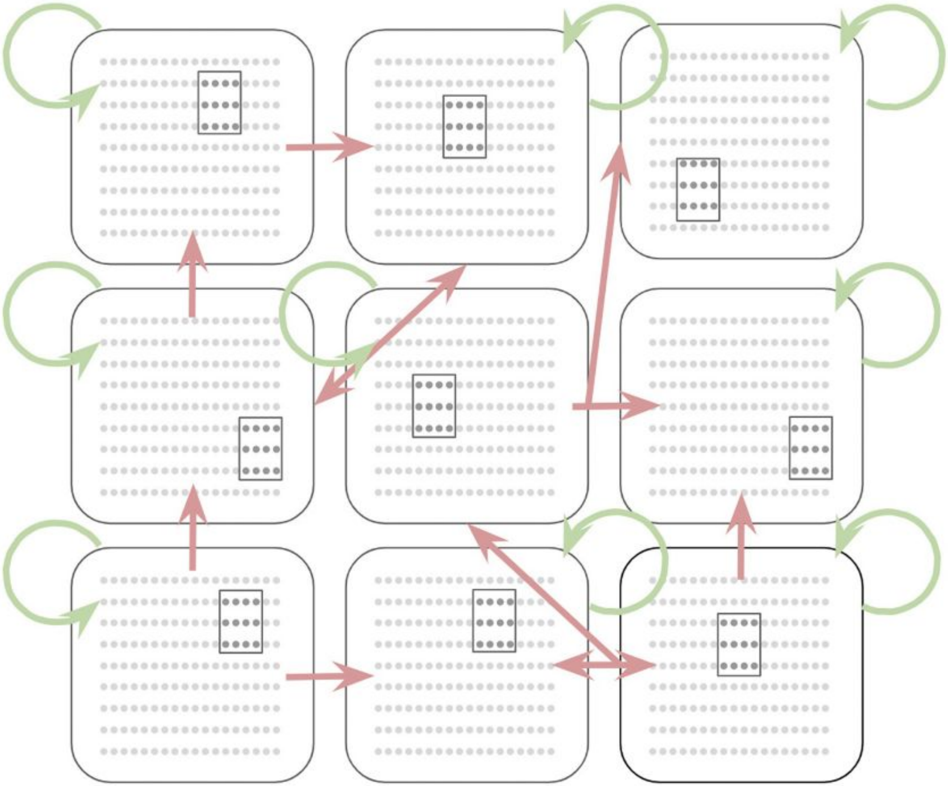
- Finite number of brain areas: *A, B, C,* etc.
    - Areas are regions of cortex
- *n* excitatory neurons per brain area
- Random recurrent connections between neurons within brain areas and across connected brain areas with probability *p*
- Multiplicative Hebbian learning rule: $w_{ij}^{t+1} = w_{ij}^{t}(1 + f_i^t f_j^{t+1}\beta)$
    - For each synapse *(i,j)* synaptic weight increases by a factor of $1+\beta$ if the post-syanptic neuron fires at time *t+1* and the pre-synaptic neuron fired at time *t*.
    - $\beta > 0$, a higher β causes faster convergence
- Only a fixed number of *k* of the *n* excitatory neurons in any area fire (*k*-cap)
- Synaptic weights are renormalized, at a slower timescale, to ensure relative stability.

Model definition

- Finite number of brain areas: *A, B, C,* etc.
  - Areas are regions of cortex
- n excitatory neurons per brain area
- Random recurrent connections between neurons within brain areas and across connected brain areas with probability p
- Multiplicative Hebbian learning rule: $w_{ij}^{t+1} = w_{ij}^{t}(1 + f_i^t f_j^{t+1}\beta)$
  - For each synapse (i,j) synaptic weight increases by a factor of $1+\beta$ if the post-syanptic neuron fires at time $t+1$ and the pre-synaptic neuron fired at time t.
  - $\beta > 0$, a higher β causes faster convergence
- Only a fixed number of k of the n excitatory neurons in any area fire (k-cap)
- Synaptic weights are renormalized, at a slower timescale, to ensure relative stability.

Model is defined by 4 params: n, p, k, β

Model definition



Functions and primitives

Functions:

- `project(x,B,y)`
- `associate(x,y)`
- `merge(x,y,A,z)`
- `reciprocal.project(x,A,y)`

Primitives:

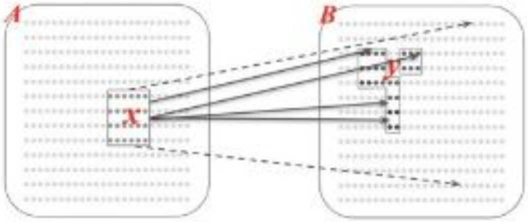
- `area(x)`
- `parent(y)`
- `inhibit(A)`
- `disinhibit(A)`

project(x,B,y)

Creates assembly *y* in a downstream area *B* as a “copy” of *x*.

Every time *x* fires, *y* will fire.

disinhibit(B)
repeat T **times: fire**(x)
inhibit(B).



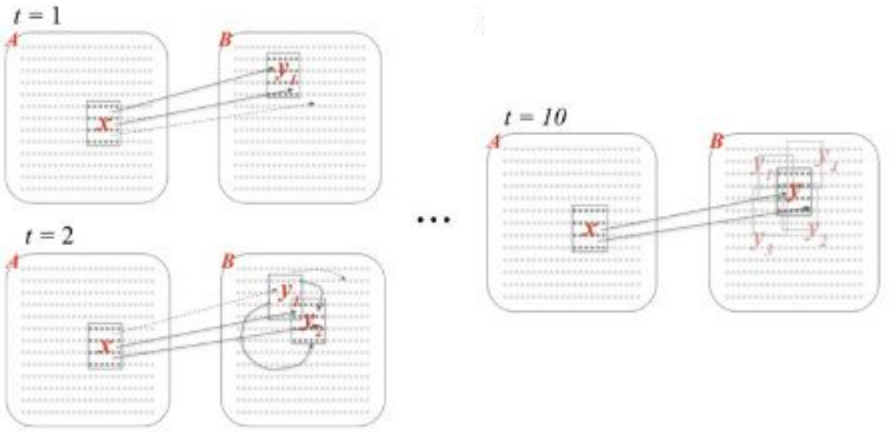
Convergence is exponentially fast
Typical value is *T=10*

project(x,B,y)

Creates assembly *y* in a downstream area *B* as a “copy” of *x*.

Every time *x* fires, *y* will fire.

disinhibit(B)
repeat T times: fire(x)
inhibit(B).



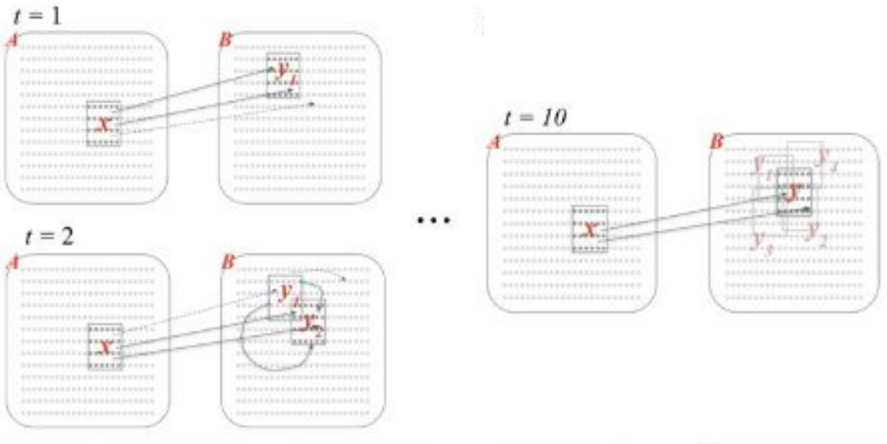
Convergence is exponentially fast
Typical value is *T=10*

project(x,B,y)

Creates assembly *y* in a downstream area *B* as a “copy” of *x*.

Every time *x* fires, *y* will fire.

```
disinhibit(B)
repeat T times: fire(x)
inhibit(B).
```



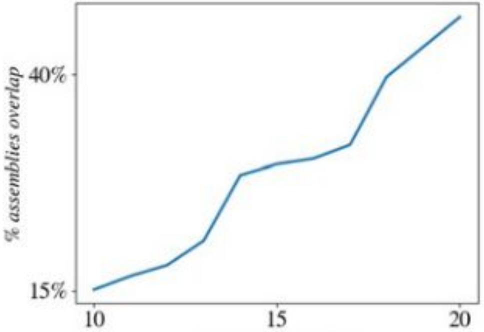
Motivation: medial temporal lobe (MTL) population response to a familiar face is hypothesized to be from projections from inferotemporal cortex encoding the face as a whole object.

Convergence is exponentially fast
Typical value is *T=10*

associate(x,y)

Two assemblies adapt to observed affinity of inputs and increase their overlap (while other cells leave the assemblies to maintain its size to *k*).

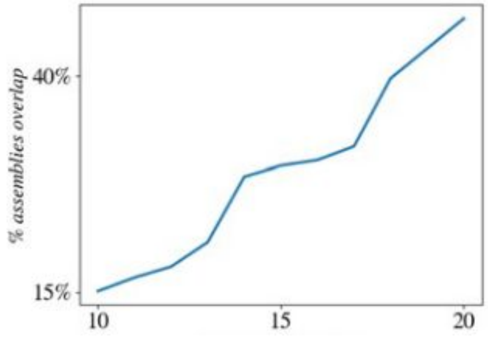
disinhibit(A)
repeat T **times: do in parallel** {**fire** (**parent**((x)),
  fire(**parent**(y))}
inhibit(A).



associate(x,y)

Two assemblies adapt to observed affinity of inputs and increase their overlap (while other cells leave the assemblies to maintain its size to *k*).

```
disinhibit(A)
repeat T times: do in parallel {fire (parent((x)),
  fire(parent(y))}
inhibit(A).
```



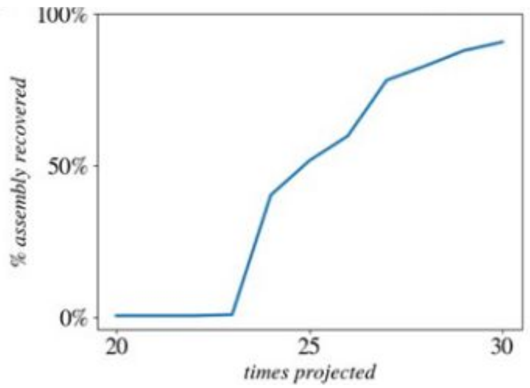
Motivation: Neurons in MTL consistently responding to the image a particular familiar place will also respond to the image of a particular familiar person when shown in an image combined with the place.

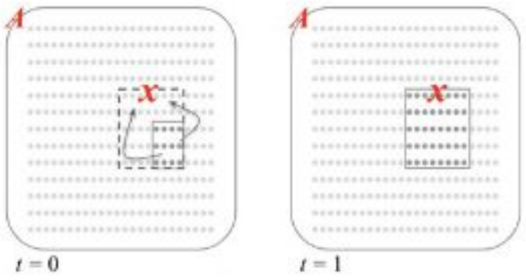
associate(x,y)

Two assemblies adapt to observed affinity of inputs and increase their overlap (while other cells leave the assemblies to maintain its size to *k*).

<u>Pattern completion</u> (content addressable memory)

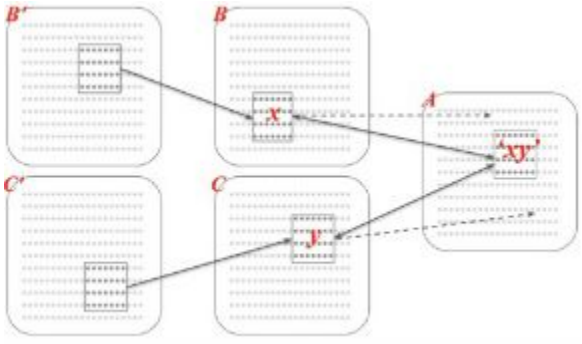
40% of parent is shown





merge(x,y,A,z)

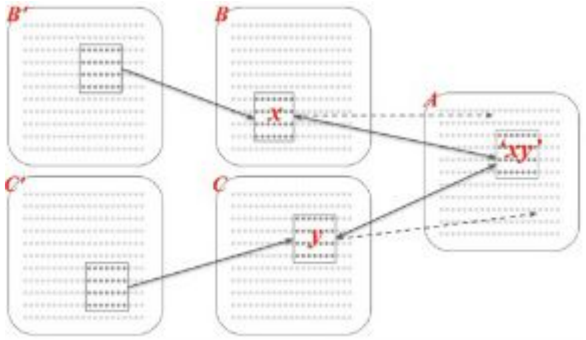
Most complex function (requiring 5 brain areas).



`merge(x,y,A,z)`

Most complex function (requiring 5 brain areas).

Creates a new representation in a new area with two-way connections to the parent assemblies.



Motivation: Linguists have long predicted that the human brain constructs recursive tree representations (hierarchies) in the brain when parsing language and outside of language in deduction, planning, etc.

`reciprocal.project(x,A,y)`

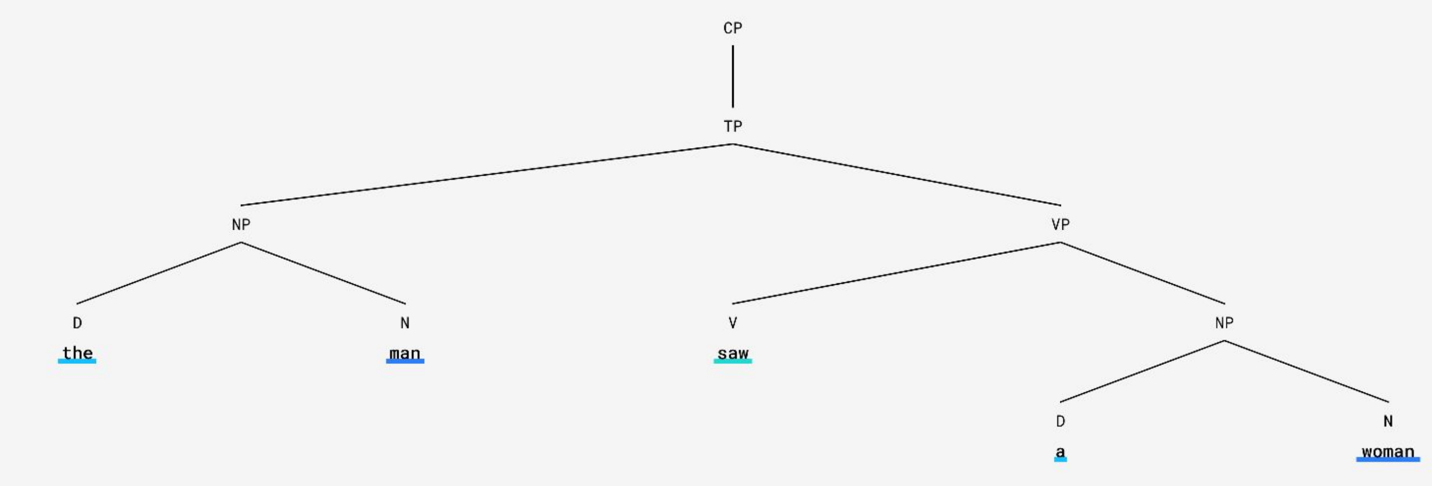
An extension of `project(x,A,y)` but now y has strong backward synaptic connectivity to x.

Motivation: Reciprocal project has been hypothesized for variable binding. I.e. assigning “cats” as the subject in the sentence “cats chase mice.”

Model implementation for parsing language

Goal: Construct an algorithm using AC to parse sentences.

Ie. “the man saw a woman”



Parser overview

- Brain areas: LEX, VERB, SUBJ, OBJ, DET, ADJ, ADV, PREP and, PREPP

Parser overview

- Brain areas: LEX, VERB, SUBJ, OBJ, DET, ADJ, ADV, PREP and, PREPP
- LEX (lexicon) contains fixed assemblies x_w for all words in the language.

Parser overview

- Brain areas: LEX, VERB, SUBJ, OBJ, DET, ADJ, ADV, PREP and, PREPP
- LEX (lexicon) contains fixed assemblies x_w for all words in the language.
- The firing of x_w has the ordinary effects of firing in the model but also invokes the execution of a short program α_w, specific to word w.
    - These short programs essentially define the devices grammar.

Parser overview

- Brain areas: LEX, VERB, SUBJ, OBJ, DET, ADJ, ADV, PREP and, PREPP
- LEX (lexicon) contains fixed assemblies x_w for all words in the language.
- The firing of x_w has the ordinary effects of firing in the model but also invokes the execution of a short program α_w, specific to word w.
  - These short programs essentially define the devices grammar.
- An action α_w contains two sets of `inhibit` and `disinhibit` commands for specific brain areas or fibers. The first set is executed before `project*` and the second after.

Parser overview

- Brain areas: LEX, VERB, SUBJ, OBJ, DET, ADJ, ADV, PREP and, PREPP
- LEX (lexicon) contains fixed assemblies x_w for all words in the language.
- The firing of x_w has the ordinary effects of firing in the model but also invokes the execution of a short program α_w, specific to word w.
  - These short programs essentially define the devices grammar.
- An action α_w contains two sets of `inhibit` and `disinhibit` commands for specific brain areas or fibers. The first set is executed before `project*` and the second after.
- `project*` fires the entire system 20 times for assembles to form and converge.

Parser overview

Example action for transitive word "saw":

$$\alpha_{\text{saw}} = \left\{ \begin{array}{l} \textit{Pre-commands} = \\ \text{disinhibit((LEX, VERB), 0)} \\ \text{disinhibit((VERB, SUBJ), 0)} \end{array} \;\vdots\; \begin{array}{l} \textit{Post-commands} = \\ \text{inhibit(SUBJ), 0)} \\ \text{disinhibit(OBJ), 0)} \\ \text{inhibit((LEX, VERB), 0)} \end{array} \right\}$$

Parser overview

Words are processed sequentially by the parser.

For each word:

1. Activate its assembly in LEX
2. Apply the pre-commands
3. `project*`
4. Apply the post-commands

Parser overview

Words are processed sequentially by the parser.

For each word:

1. Activate its assembly in LEX
2. Apply the pre-commands
3. `project*`
4. Apply the post-commands

Algorithm 2: Parser, main loop.

input : a sentence s
output: representation of dependency parse of s, rooted in VERB

disinhibit(LEX, 0) ;
disinhibit(SUBJ, 0) ;
disinhibit(VERB, 0) ;
foreach word w in s **do**
    activate assembly x_w in LEX ;
    foreach pre-rule (Dis)inhibit($\square$, i) in $\alpha_w \rightarrow$ Pre-Commands **do**
        (Dis)inhibit($\square$, i) ;
    *project** ;
    foreach post-rule (Dis)inhibit($\square$, i) in $\alpha_w \rightarrow$ Post-Commands **do**
        (Dis)inhibit($\square$, i)

Parser overview

Words are processed sequentially by the parser.

For each word:

1. Activate its assembly in LEX
2. Apply the pre-commands
3. `project*`
4. Apply the post-commands

Algorithm 2: Parser, main loop.

input : a sentence s
output: representation of dependency parse of s, rooted in VERB

disinhibit(LEX, 0) ;
disinhibit(SUBJ, 0) ;
disinhibit(VERB, 0) ;
foreach word w in s **do**
  activate assembly x_w in LEX ;
  foreach pre-rule (Dis)inhibit($\square$, i) in $\alpha_w \rightarrow$ Pre-Commands **do**
    (Dis)inhibit($\square$, i) ;
  *project** ;
  foreach post-rule (Dis)inhibit($\square$, i) in $\alpha_w \rightarrow$ Post-Commands **do**
    (Dis)inhibit($\square$, i)

This forms a parse/dependency tree of the sentence in the synaptic connectivity of the system.

Parser overview

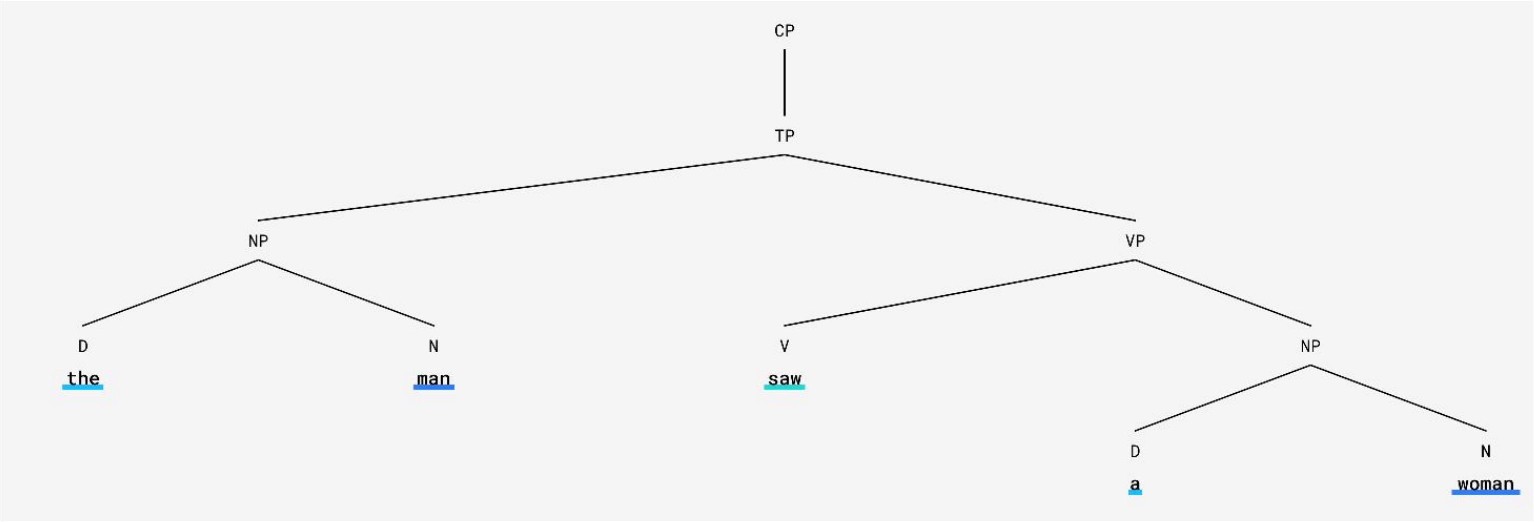
1. N V-INTRANS (people died)
2. N V N (dogs chase cats)
3. D N V-INTRANS (the boy cried)
4. D N V N or N V D N (the kids love toys)
5. D N V D N (the man saw the woman)
6. ADJ N V N or N V ADJ N (cats hate loud noises)
7. D ADJ N D ADJ N (the rich man bought a fancy car)
8. PRO V PRO (I love you)
9. {D} N V-INTRANS ADVERB (fish swim quickly)
10. {D} N ADVERB V-INTRANS (the cat gently meowed)
11. {D} N V-INTRANS ADVERB (green ideas sleep furiously)
12. {D} N ADVERB V {D} N (the cat voraciously ate the food)
13. {D} N V-INTRANS PP (the boy went to school)
14. {D} N V-INTRANS PP PP (he went to school with the backpack)
15. {D} N V {D} N PP (cats love the taste of tuna)
16. {D} N PP V N (the couple in the house saw the thief)
17. {D} N COPULA {D} N (geese are birds)

Parser overview

1. N V-INTRANS (people died)
2. N V N (dogs chase cats)
3. D N V-INTRANS (the boy cried)
4. D N V N or N V D N (the kids love toys)
5. D N V D N (the man saw the woman)
6. ADJ N V N or N V ADJ N (cats hate loud noises)
7. D ADJ N D ADJ N (the rich man bought a fancy car)
8. PRO V PRO (I love you)
9. {D} N V-INTRANS ADVERB (fish swim quickly)
10. {D} N ADVERB V-INTRANS (the cat gently meowed)
11. {D} N V-INTRANS ADVERB (green ideas sleep furiously)
12. {D} N ADVERB V {D} N (the cat voraciously ate the food)
13. {D} N V-INTRANS PP (the boy went to school)
14. {D} N V-INTRANS PP PP (he went to school with the backpack)
15. {D} N V {D} N PP (cats love the taste of tuna)
16. {D} N PP V N (the couple in the house saw the thief)
17. {D} N COPULA {D} N (geese are birds)

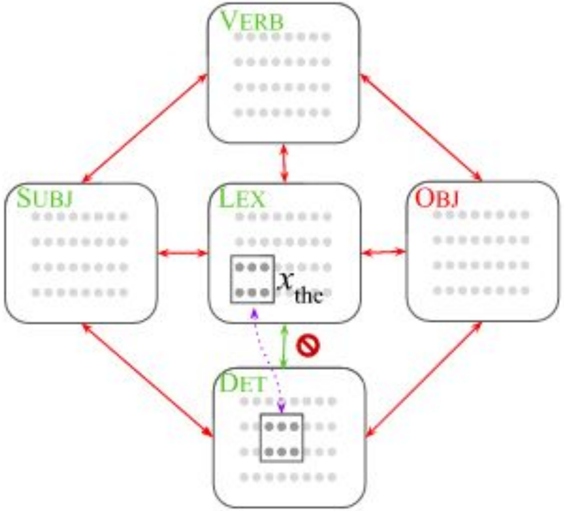
Parser overview

"The man saw the woman"



Parser overview

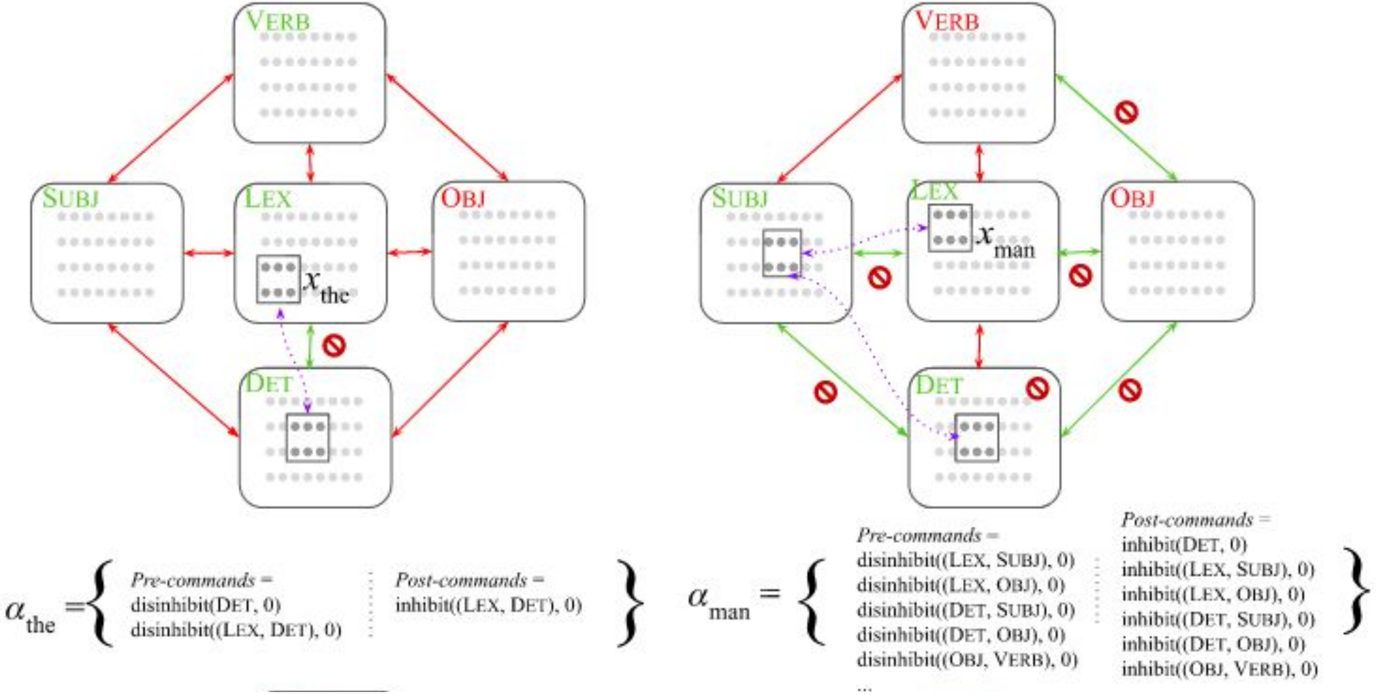
"The man saw the woman"



$$\alpha_{\text{the}} = \left\{ \begin{array}{l|l} \textit{Pre-commands} = & \textit{Post-commands} = \\ \text{disinhibit(DET, 0)} & \text{inhibit((LEX, DET), 0)} \\ \text{disinhibit((LEX, DET), 0)} & \end{array} \right\}$$

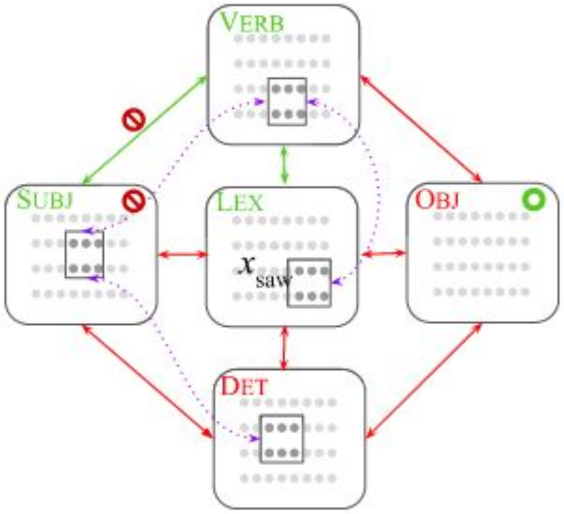
Parser overview

“The man saw the woman”



Parser overview

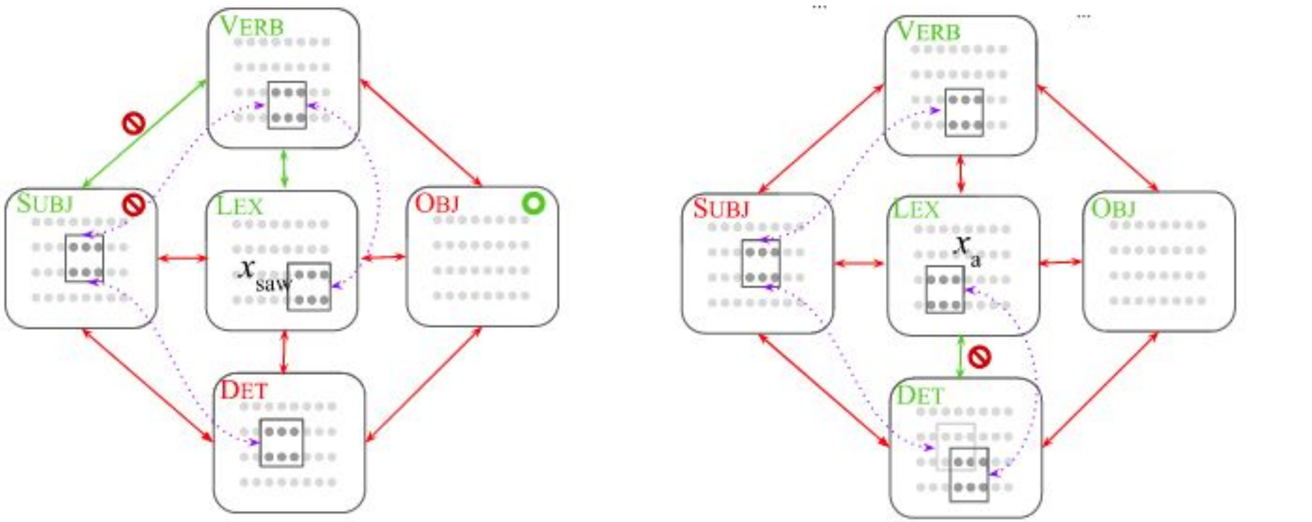
"The man saw the woman"



$$\alpha_{saw} = \left\{ \begin{array}{l|l} \textit{Pre-commands} = & \textit{Post-commands} = \\ \text{disinhibit}((\text{LEX}, \text{VERB}), 0) & \text{inhibit}(\text{SUBJ}, 0) \\ \text{disinhibit}((\text{VERB}, \text{SUBJ}), 0) & \text{disinhibit}(\text{OBJ}, 0) \\ & \text{inhibit}((\text{LEX}, \text{VERB}), 0) \end{array} \right\}$$

Parser overview

"The man saw the woman"

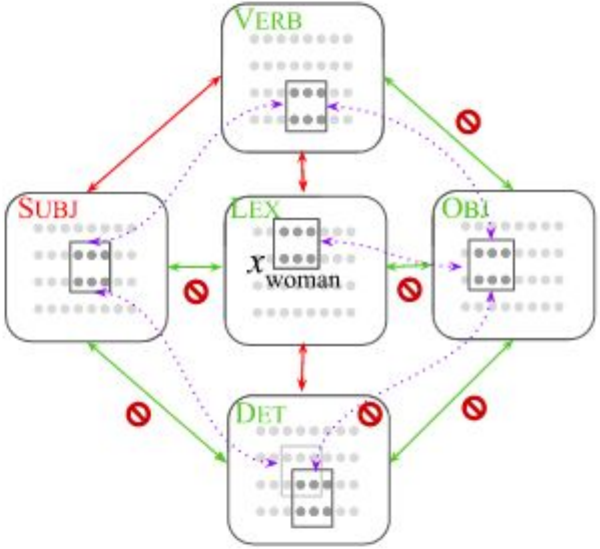


$$\alpha_{saw} = \left\{ \begin{array}{l} \textit{Pre-commands} = \\ \text{disinhibit((LEX, VERB), 0)} \\ \text{disinhibit((VERB, SUBJ), 0)} \end{array} \;\vdots\; \begin{array}{l} \textit{Post-commands} = \\ \text{inhibit(SUBJ, 0)} \\ \text{disinhibit(OBJ, 0)} \\ \text{inhibit((LEX, VERB), 0)} \end{array} \right\}$$

$$\alpha_{a} = \left\{ \begin{array}{l} \textit{Pre-commands} = \\ \text{disinhibit(DET, 0)} \\ \text{disinhibit((LEX, DET), 0)} \end{array} \;\vdots\; \begin{array}{l} \textit{Post-commands} = \\ \text{inhibit((LEX, DET), 0)} \end{array} \right\}$$

Parser overview

“The man saw the woman”



$\alpha_{\text{woman}} = \{$

Pre-commands =
disinhibit((LEX, SUBJ), 0)
disinhibit((LEX, OBJ), 0)
disinhibit((SUBJ, DET), 0)
disinhibit((OBJ, DET), 0)
disinhibit((VERB, OBJ), 0)
...

Post-commands =
inhibit(DET, 0)
inhibit((LEX, SUBJ), 0)
inhibit((LEX, OBJ), 0)
inhibit((SUBJ, DET), 0)
inhibit((OBJ, DET), 0)
inhibit((VERB, OBJ), 0)
...

$\}$

Parser overview

Problem: The current parser only parses simple sentences. But language is recursive…

Center embeddings: a harder problem

Def. center embedding: a phrase embedded in the middle of another phrase.

“Dogs, when they run, chase cats”

Extender parser for center embeddings

- New brain area: DS (dependent segment)

Extender parser for center embeddings

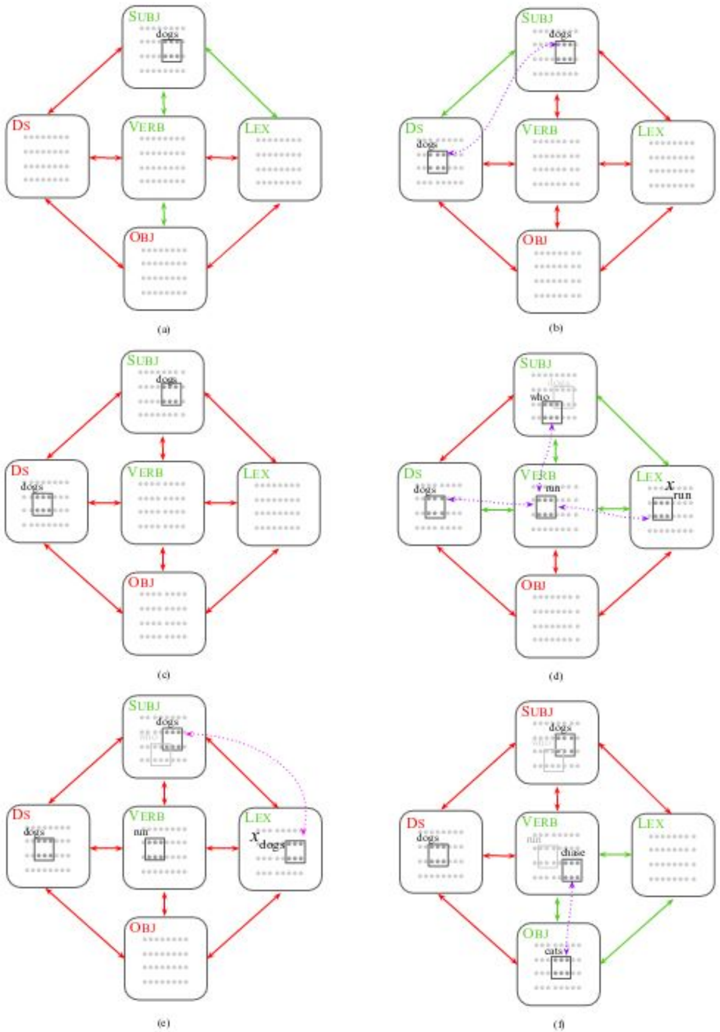
- New brain area: DS (dependent segment)
- Idea:
  - The Parser stores the part of the utterance already parsed in working memory.
  - After the embedded clause has been parsed, the Parser returns to the beginning of the outer sentence in the working memory and processes it to restore the state.
  - Note: On the first time through the outer sentence (before the comma), weights are learned. The second time through is then much faster.

Extender parser for center embeddings

- New brain area: DS (dependent segment)
- Idea:
    - The Parser stores the part of the utterance already parsed in working memory.
    - After the embedded clause has been parsed, the Parser returns to the beginning of the outer sentence in the working memory and processes it to restore the state.
    - Note: On the first time through the outer sentence (before the comma), weights are learned. The second time through is then much faster.
- The algorithm is recursive. (I.e. the embedded clause can have an embedded clause.)

Extender parser for center embeddings

1. Parse the outer clause until detecting a center embedding. (Assume the parser can always recognize the beginning of a dependent sentence.)
2. Project the last word from the outer clause to DS
3. When a center embedding is detected, the Parser state is reinitialized
4. The embedded clause is parsed and linked to DS areas.
    a. On the verb of the inner clause, project from LEX and DS to VERB. (This allows recovery of the root verb.)
5. The Parser restores its last state when it was parsing the outer clause.
    a. The state is reinitialized and the beginning of the outer clause is reprocessed from working memory.
6. The remainder of the outer sentence is parsed



Algorithm 1: Enhanced Parser, main loop

---

input : a sentence s, depth d
output : representation of dependency parse of s, rooted in VERB

Function *parse* $(s, d \leftarrow 0)$ **:**
- **foreach** word w in s **do**
  - **if** $(d+1)$-depth clause begins after w **then**
    - disinhibit(DS) ;
    - disinhibit(DS, AREA(w)) ;
    - *project** ;
    - inhibit(DS, AREA(w)) ;
    - inhibit(DS) ;
  - **else if** w begins (d+1) depth clause **then**
    - $d \leftarrow d+1$;
    - clear the slate ;
  - **else if** w ends embedded sentence **then**
    - $d \leftarrow d-1$;
    - **foreach** word y before w **do**
      - activate y in LEX ;
      - *fire* DISINHIBITED AREAS ;
  - **if** $d > 0$, AREA(w) = VERB **then**
    - disinhibit(DS) ;
    - disinhibit(DS, VERB) ;
  - execute w actions and *project** ;
  - inhibit(DS) ;
  - inhibit(DS, VERB) ;

Extended parser accepts context free languages

The extended parser can be seen as a finite-state automata with extra capabilities of (a) marking the current input symbol and (b) reverting from the current input symbol to the previously marked symbol closest to the current one.

Cal this a *fallback automata* (FBA).

They prove that FBAs accept context-free languages (CFLs).

Fallback automata

$A = (\Sigma, K, I, F, \Delta)$

- Σ is nonempty finite set of symbols
- K is set of states
- I is set of starting states
- F is set of accepting states
- Δ is the transition function
    - $\Delta \subseteq ((\Sigma \times T \times K) \times (K \times \{s, ✓, \leftarrow\})$, where $T = \{f,s\} \cup K$

Discussion

- How does this generalize? Humans are trained on all possible sentences.
- Isn’t language parsing happening too quickly for synaptic plasticity?