RAAM: Recursive Auto-Associative Memory

My most famous paper
Before those out of control robots

Minsky & Papert

Perceptrons, 1988

General Principle for Machine Learning:

"No machine can learn to recognize X unless it possesses, at least potentially, some scheme for representing X"

"Perceptrons had no way to represent knowledge"

Fodor & Pylyshyn (1988)

- The Language of Thought is necessarily compositional, combinatorial, and systematic.
- The only known adequate systems are rule- or syntax-based
- Therefore, Connectionism is mere implementation detail, or wrong.

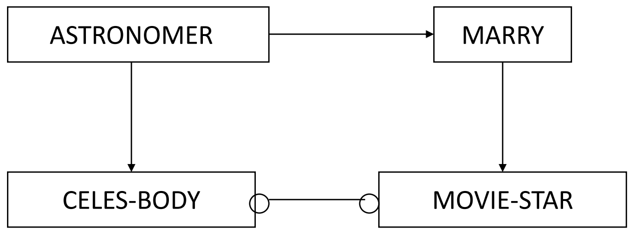
An Open Problem in 1986...

- Geoff Hinton's "Reduced Descriptions"

"A part of the distributed representation for Horse, should be a reduced description of the distributed representation of the Horse's Leg...

A linear "image" which would have great representational properties...How?

Local Representations



- *Without Labels on Links*
  Semantic Networks are Powerless

Distributed Representations

- Pattern of Activation per concept rather Unit per concept
  - equivalent to Feature Vectors
  - Sometimes called "Micro-features"
  - Cannot represent multiple concepts

Coarse-Coding

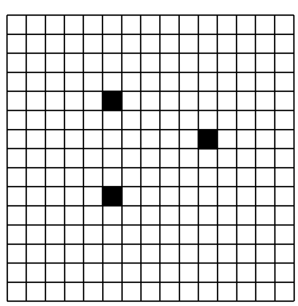
Hinton (1984)

Trades accuracy for accessibility

- Each Unit codes ambiguously
- Multiple units provide redundancy
- Graceful Degradation

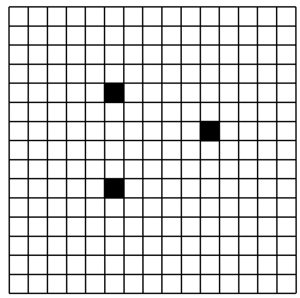
Imagine a set of concepts arrayed

in D dimensions and N possibilities



Imagine a set of concepts arrayed

in D dimensions and N possibilities

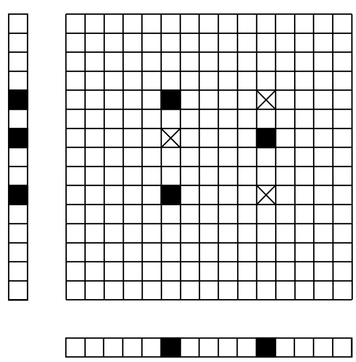


Local: N^D units

Represents any subset

Fully Distributed System

Uses a Register for each dimension

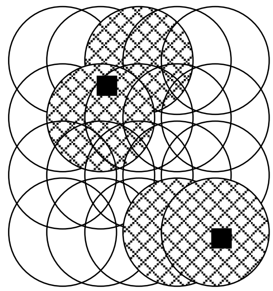


Only DN Units!

No accuracy for simultaneous access

Coarse-Coded System

Many to Many Coding



If each unit covers an area...
each concept hits several units...

And several concepts can coexist

Exploiting Coarse Coding

Sequence by superposition

Touretzky BOLTZWORK, M&R
Past-Tense

Code multiple triples of letters

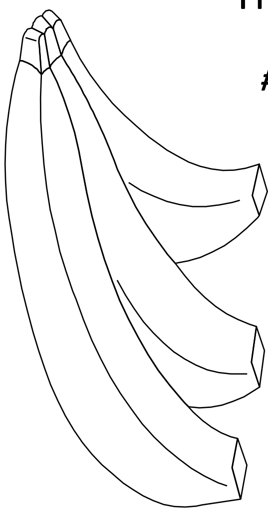
Represent a sequence by overlap:

#STORE#

is represented by collection:

#ST + STO + TOR + ORE + RE#

The Banana Problem:



#BA BAN ANA NAN ANA NA#

OH OH!

Since tuples are not counted

For any fixed Breadth

There exists counterexamples

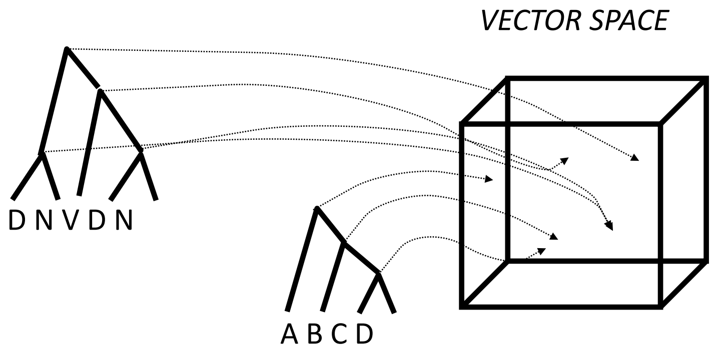
RAAM

- An automatic way to develop representations for sequences and trees.
- Develops the access mechanisms as well
- Built on top of Back-Propagation's ability to learn with "hidden" units

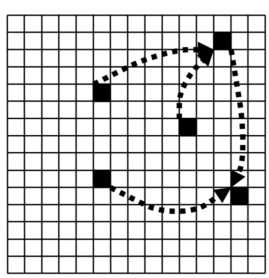
Recursive Auto-Associative Memory

- Solves 25-year-old problem:

How to pack symbolic structures into numeric patterns

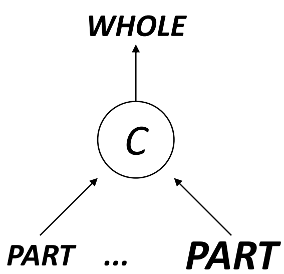


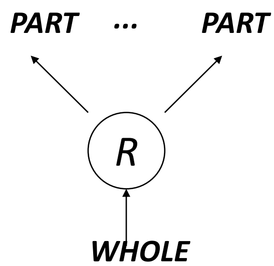
A New Way to get multiple items



Encode them into structures

Consider two Machines



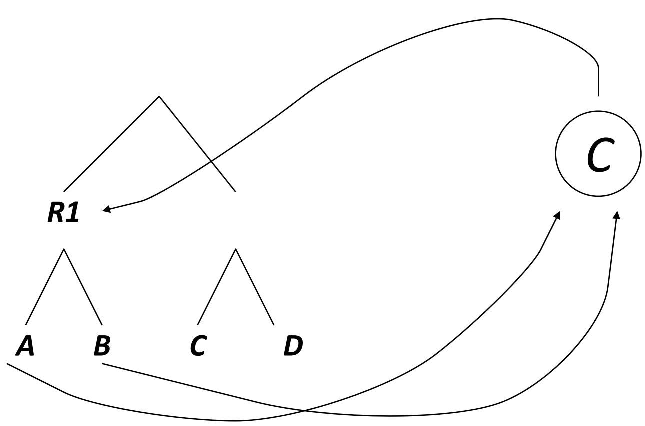


A "Compressor"
of parts into
a whole

a "Reconstructor"
from the whole to
facsimiles of parts

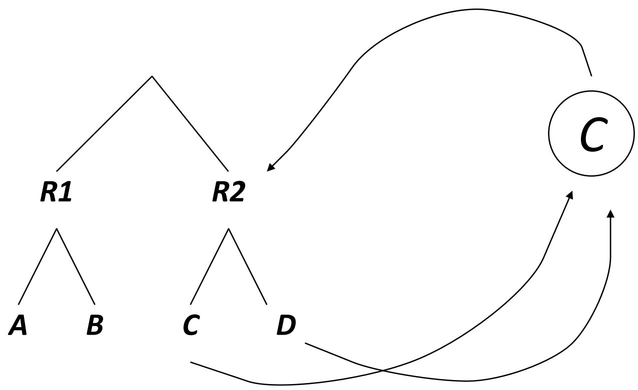
The Compressor

Encodes TREES Bottom up:



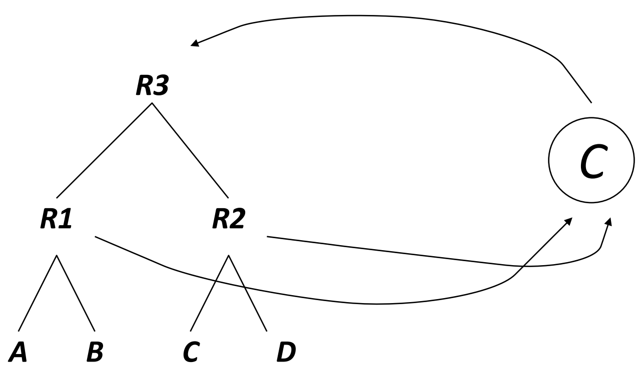
The Compressor

Encodes TREES Bottom up:



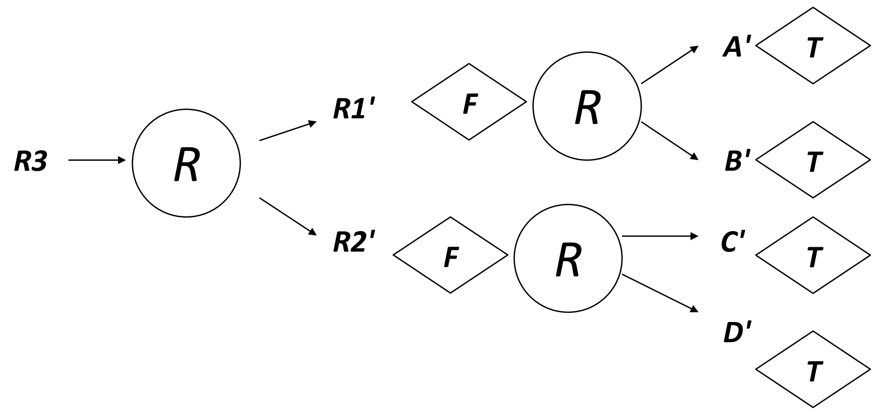
The Compressor

Encodes TREES Bottom up:

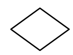


The Reconstructor

Decodes a tree

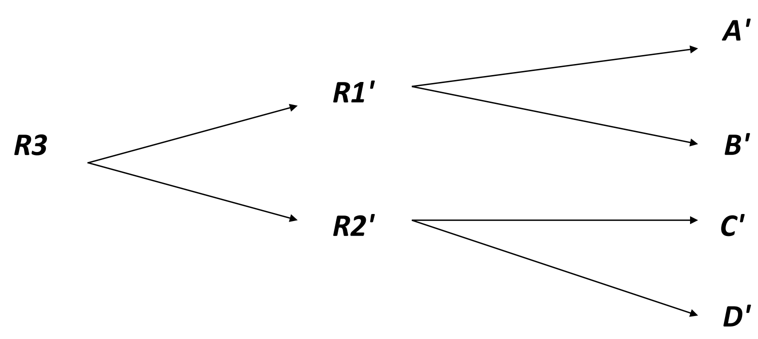


Using a "Terminal Test"



The Reconstructor

Decodes a tree

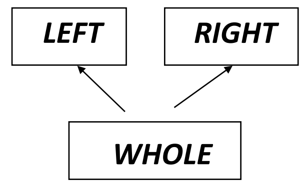


Returning only a Facsimile

Two Mechanisms are Co-Evolved

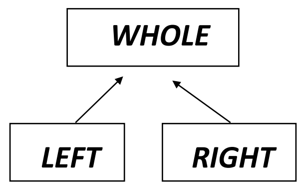
For Binary Trees:

Reconstructor



k->2k feed-forward

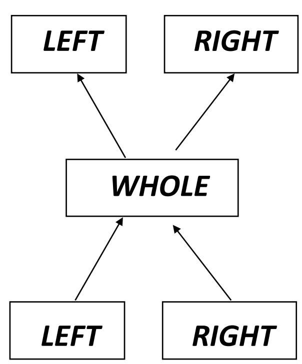
Compressor:



2k -> k Feed-forward

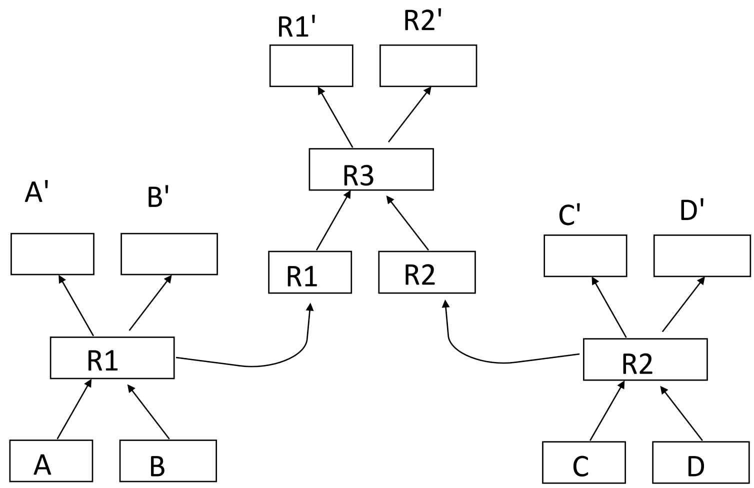
Two Mechanisms are Co-Evolved

As a single auto-associator



2k -> k -> 2k Feed-forward

Two Mechanisms are Co-Evolved



setup terminals

- (defun puthash (key table value)
-   (setf (gethash key table) value))

- (defparameter *table* (make-hash-table :test 'equal))
- (puthash 'len *table* 5)
- (puthash 'd *table* '(1 0 0 0 0))
- (puthash 'n *table* '(0 1 0 0 0))
- (puthash 'v *table* '(0 0 1 0 0))
- (puthash 'p *table* '(0 0 0 1 0))
- (puthash 'a *table* '(0 0 0 0 1))

- (defun inverthash (table &aux out)
-   (maphash #'(lambda (k v) (push (cons k v) out)) table)
-   (loop for x in out do (puthash (cdr x) table (car x))))

- (inverthash *table*)

The Encoder

- (defun zeros (n) (loop for i from 1 to n collect 0))
- (defun pad (lst n) (append lst (zeros n)))

- (defun getatom (atom p) (pad (gethash atom *table*) p))

- (defun encode (tree network &aux p)
- (setf p (- (length (cadar network))(gethash 'len *table*)))
- (forward-pass network (append (or (and (atom (car tree))(getatom (car tree) p))
- (encode (car tree) network))
- (or (and (atom (cadr tree))(getatom (cadr tree) p))
- (encode (cadr tree) network))))
- (hidden network))

The reconstructor

- (defun terminalp (rep &optional (edgetol .2))
- (loop for x in rep always (or (< x edgetol)(> x (- 1.0 edgetol)))))

- (defun tobool (rep &aux p)
- (setf p (- (length rep)(gethash 'len *table*)))
- (gethash (butlast (loop for x in rep collect (if (< x .5) 0 1)) p) *table*))
-
- (defun decode (rep network &aux left right)
- (loop for node in (cadar network) as x in rep do
- (setf (bp-node-output node) x))
- (loop for node in (caddar network) do
- (getinput node)
- (setoutput node))
- (setf output (loop for node in (car (last (car network))) collect
- (bp-node-output node)))
- (setf left (butlast output (length (cadar network))))
- (setf right (last output (length (cadar network))))
- (list (or (and (terminalp left)(tobool left))(decode left network))
- (or (and (terminalp right)(tobool right))(decode right network))))

Why does it work?

As t -> infinity

A'(t) -> A

B'(t) -> B

C'(t) -> C

D'(t) -> D

R1'(t) -> R1(t)

R2'(t) -> R2(t)

R3(t) -> Representation for Tree

Details of operation

- Epoch Learning
- Need for close tolerance on non-terminals

|A - A'|

|R1 - R1'|

- Moving Target Learning means small learning rates

Experiments with RAAM

- Sequences
  - 3-bit sequences
  - Spelling Checker
- Trees
  - Syntactic Trees
  - Semantic Triples

Simple Environment of Binary Trees

((DN)V)

(V(DN))

(P(D(AN)))

((DN)(P(DN)))

(D(A(A(AN))))

((DN)(V(D(AN))))

((D(AN))(V(P(DN))))

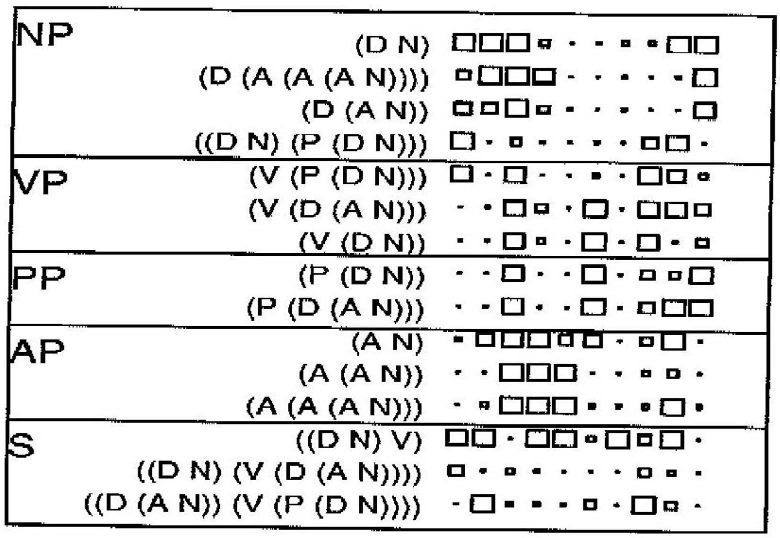
RAAM a tree into autoencoder

```
(defun bptree (tree network &optional (mu 0.3)(alpha 0.9) (tol .05)(edgetol .2)&aux p vec err)
 (setf p (- (length (cadar network))(gethash 'len *table*)))
 (forward-pass network
             (setf vec (append (or (and (atom (car tree))(getatom (car tree) p))
                                   (bptree (car tree) network mu alpha tol edgetol))
                               (or (and (atom (cadr tree))(getatom (cadr tree) p))
                                   (bptree (cadr tree) network mu alpha tol edgetol)))))
 (setf err (backward-pass network vec tol edgetol t))
 (values (hidden network) err))
```

Train a set of Trees into RAAM

```
(defun learntrees (trees network &optional (mu 0.3)(alpha 0.9)(limit 1000)(tol .05)(edgetol 0.2))
  (loop for link in (cdr network) do
        (setf (bp-link-previous-delta link) 0.0))
   (format t "~%")
  (loop for cycle from 1 to limit do
        (loop for link in (cdr network) do
              (setf (bp-link-delta link) 0.0))
        (loop for tree in trees do
              (bptree tree network mu alpha tol edgetol))
        (format t "Cycle: ~d ~%" cycle)
            (loop for link in (cdr network) do
         (incf (bp-link-weight link)
                (- (setf (bp-link-previous-delta link)
                         (+ (* mu (bp-link-delta link))
                            (* alpha (bp-link-previous-delta link)))))))))
```

Hinton Diagram



Exploring a RAAM

- Bottom up searching to see what the representational capacity is
- After training, try
  - All pairs
    - (D N)(N V)(D V)(P D)....etc
    - Triples with represented pairs
      - ((D N) V)(V (D N))...etc
    - Quads
    - etc

Additional Trees coded by RAAM

New NP's

((D (A N)) (P (D N)))
((D N) (P (D (A N))))
((D (A N)) (P (D (A N))))

New VP's

(V ((D N) (P (D N))))
(V ((D (A N)) (P (D N))))
(V ((D N) (P (D (A N)))))
(V ((D (A N)) (P (D (A N)))))

New S's

((D N) (V (D N)))
(((D N) (P (D N))) V)
((D N) (V ((D N) (P (D N)))))
(((D N) (P (D N))) (V (D N)))
((D N) (V ((D (A N)) (P (D N)))))
((D N) (V ((D N) (P (D (A N))))))
(((D N) (P (D N))) (V (D (A N))))
((D N) (V ((D (A N)) (P (D (A N))))))
(((D N) (P (D N))) (V ((D N) (P (D N)))))
(((D N) (P (D N))) (V ((D N) (P (D (A N))))))
(((D N) (P (D N))) (V ((D (A N)) (P (D N)))))
(((D N) (P (D N))) (V ((D (A N)) (P (D (A N))))))

Philosophical Uses of RDR's

Van Gelder, Horgan, Chalmers, Niklassen,

Limerick Contest at Midwest Connectfest

Deep Stuff 1

The folks from the journal Cognition

Weren't more than a bad apparition.

All's left of Fodor

Is just a bad odor,

And nobody's missin' Pylyshyn.

-- *Tim van Gelder*

Deep Stuff 2

Pollack has made this admission

Of his neural net's true composition:

"I recursively RAAM it

With symbols, Goddamnit!

So don't pay no mind to Pylyshyn."

-- *Dave Touretzky*

Deep Stuff 3

These guys Fodor and Pylyshyn

Came to town with a mission.

But they got in a jam

When faced off with RAAM

Instead they should just have gone fishin'.

-- *Dave Chalmers*

Open Problems

- Scaling up more than 6 levels
  - i.e. converge really tightly
  - discover design methodology - eschew training
- Theoretical Capacity for Infinite Language (with unbounded rationals)
- Killer Application which proves RAAM's are better than anything else at
  - Speech, Handwriting
  - Higher Dimensions (Vision), Continuous Version
- How to learn structures from statistics of data, rather than as given.

Conclusions

- How symbolic structures are represented in the brain is as yet unknown.
- JBP's RAAM model showed how to recursively code trees into patterns, but didnt scale
  - Capacity problems may stem from Terminal Test.
- Many advances in Deep Learning built on top of RAAM's advance
- Practical uses for this technology are still out there.