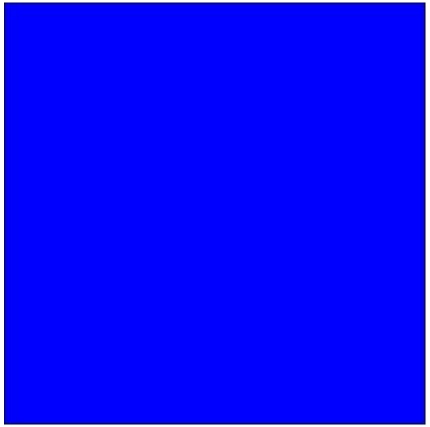
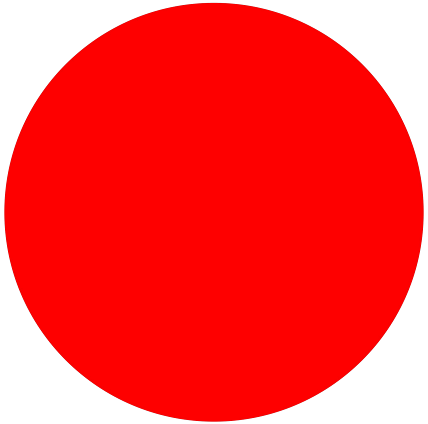
Computational Models of Cortical Function

Eric Chu and Archana Ram

Symbols

- Atomic
- Composite



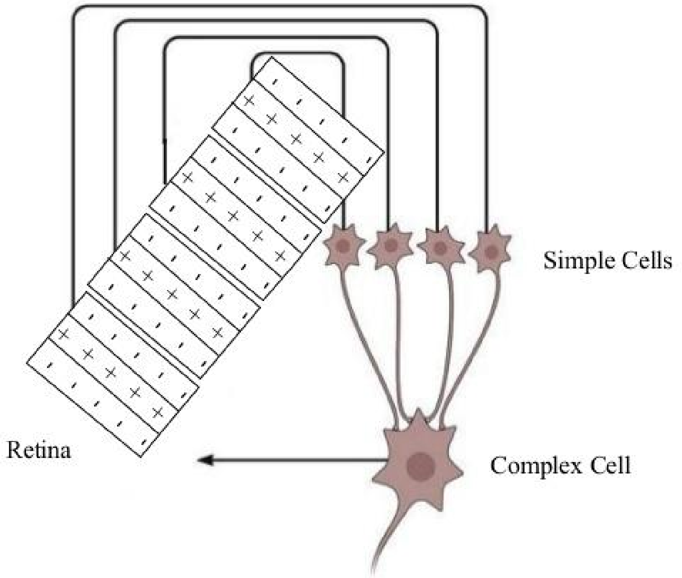


Symbol encoding

- Unique patterns over sets of neurons

Visual object representation

- Feature hierarchy
- Simple cells
- Complex cells
- 1+ object in field/background

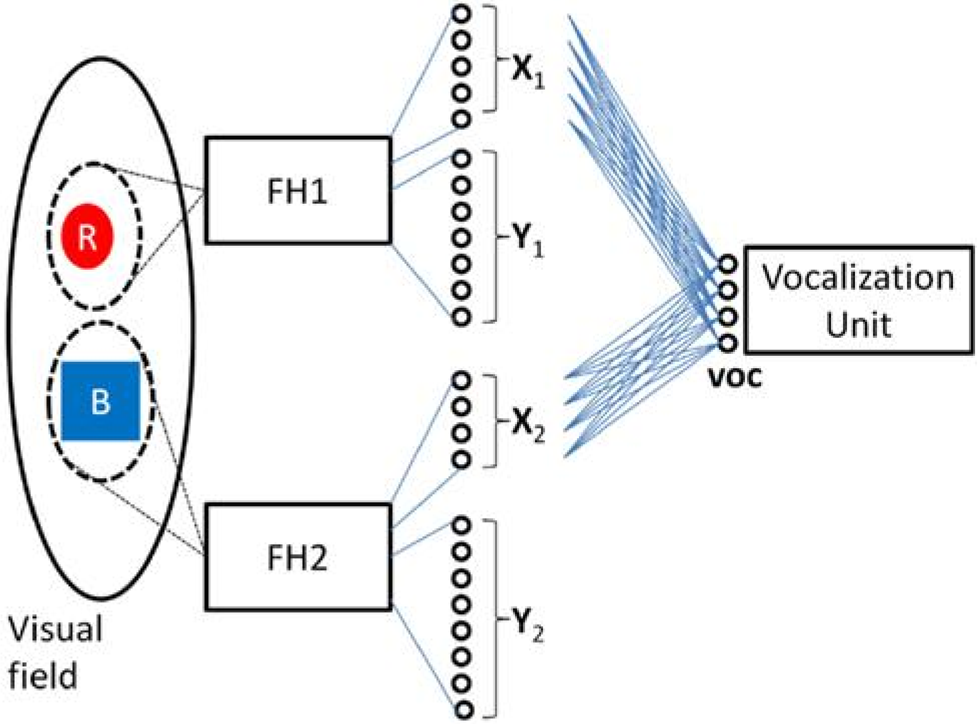


Variable binding problem

- “red circle and blue square”
- “blue circle and red square”
- Symbol binding
  - *Red* circle
  - “Mary loves John”

Variable binding in vision

- 2+ visual objects
- Single v. multiple “spotlights”
- Feature hierarchies
- No information generalized
- Role-filler independence



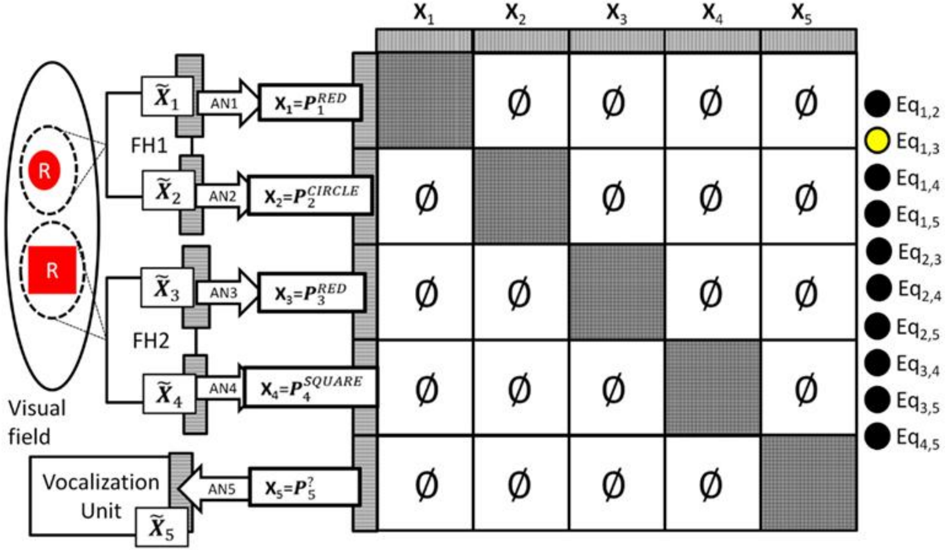
1. How could the larger neural system within this anatomical binding scheme determine whether object 1 and object 2 had the same or different colors (or shapes, etc.)?
2. How could the system vocalize the color of the object represented by X_2 by making use of the associations learned on X_1?

DPAAN solution

- Synchronized training
- “Universal translator” / “universal language”
- Dynamically partitionable autoassociative network

Perceptual binding with a DPAAN

- DPAAN with 5 partitions
- All spotlights lock on object
- Stable patterns are symbols
- “Red,” “circle,” “square”



1. How could the larger neural system within this anatomical binding scheme determine whether object 1 and object 2 had the same or different colors (or shapes, etc.)?
2. How could the system vocalize the color of the object represented by X_2 by making use of the associations learned on X_1?

Variable binding

- Agent, verb, patient
- Role -> filler
- “I want to table you”
- Pointers

Physiological evidence

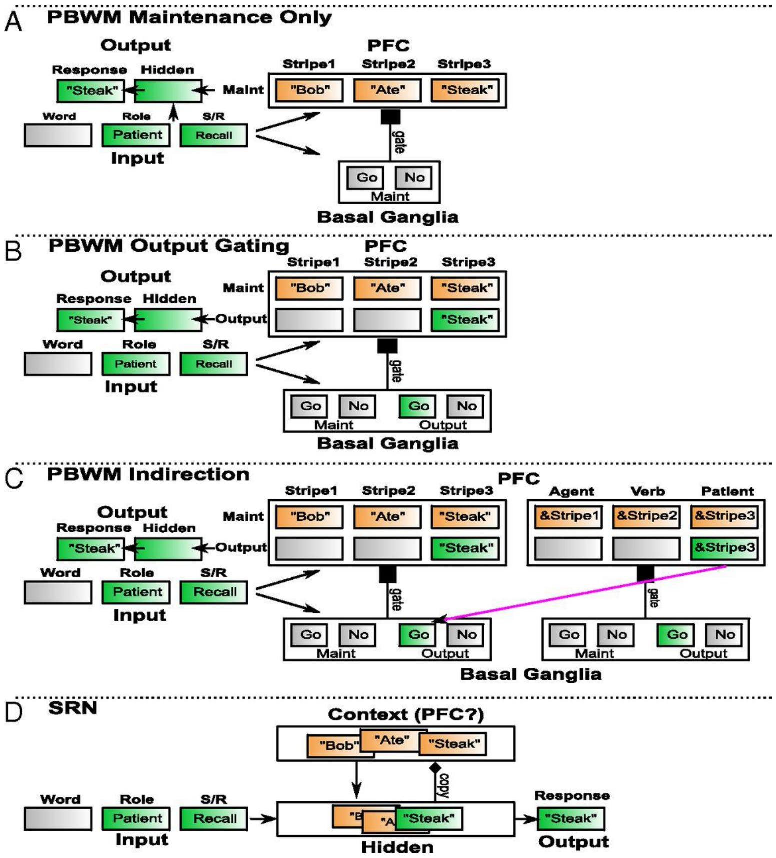
- Stripe-like patches in PFC
  - Inhibitory between, excitatory within
- Project to basal ganglia (BG)
  - Update to encode information
  - Maintain information
  - Output encoded information to drive other processes
- Hierarchy of stripes

Sentence decoding

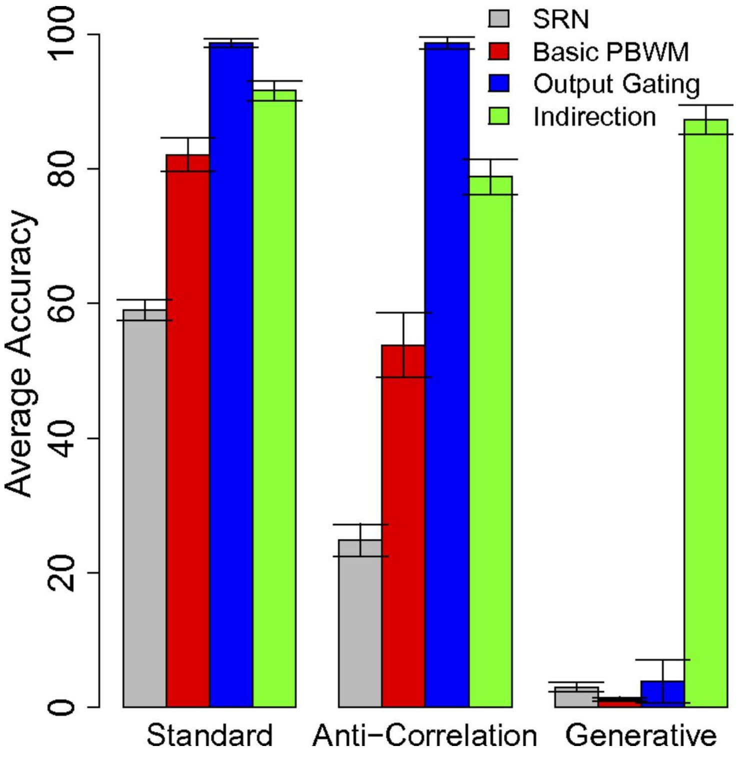
- Agent, verb, patient
  - “She tabled him”
- Presented sentence word-by-word
- Recall
- Anticorrelation

Models

- Words encoded in separate stripes
- Multiple items in working memory
- 2 networks + output gating
    - Filler-specific
    - Role-specific
- Simple recurrent network



Model performance



Neural Random-Access Machines

Overview

- Architecture can manipulate and dereference **pointers** to an external variable-size random-access memory
- Tested on algorithms involving linked lists and binary trees

Related Work

- Deep networks with memory, augmented networks
- Examples:
  - Memory Network (Weston 2014): early attempt at explicitly separating memory
    - Tested on question- answering
  - Neural Turing Machine (Graves 2014)
    - Tested on algorithms

Related Work

- Examples (continued):
  - Attention model (Bahdanau 2014): variants have achieved SOA on machine translation, speech recognition, etc.
  - Grid-LSTM (Kalchbrenner 2015):
    - Tested on Wikipedia character prediction (SOA)
    - Tested on Chinese-to-English translation task
  - Pointer Network (Vinyals 2015): doesn’t have writable memory, sort of similar to attention model
    - Tested on computing planar convex hulls, computing Delaunay triangulations, and the Traveling Salesman Problem
  - Stack, queue, dequeue-based models (2015)

Related Work

- Key-point: depth, size of memory, and number of parameters are no longer confounded and can be altered independently
  - (in contrast to models like LSTM, whose number of parameters grows quadratically with the size of their short term memory)

Model (No External Memory)

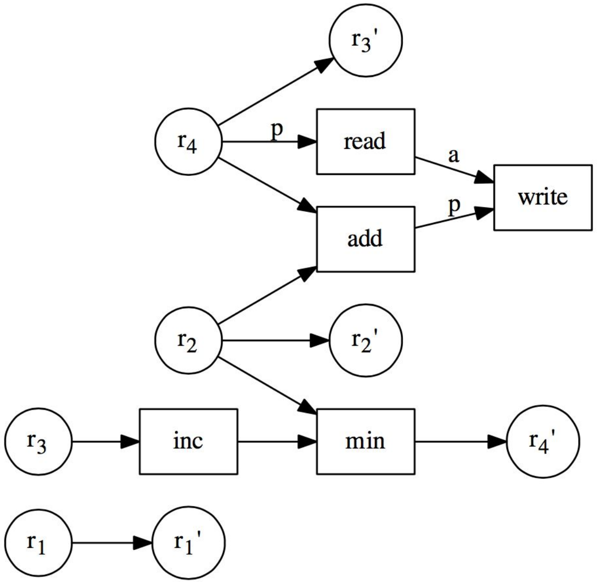
- Neural **controller** (“processor”)
  - Feedforward NN or LSTM
- R **registers**, each represents an integer as a distribution over {0,1,..., M-1}, for some constant M
- Controller interacts with registers through **modules (gates)** such as integer addition and equality test
  - Module m_i: {0,1..., M - 1} x {0, 1, ..., M-1} -> {0,1,..., M-1}

Model (No External Memory)

- Performs sequence of timesteps
    - 1) The controller gets some inputs depending on the values of the registers
    - 2) The controller updates its internal state if the controller is an LSTM
    - **3) The controller outputs the description of a "fuzzy circuit" with inputs r_1,..., r_R, gates m_1,...,m_Q and R outputs**
    - 4) The values of the registers are overwritten with the outputs of the circuit

Model (No External Memory)

- Example “exemplary” circuit for COPY task (step 3):

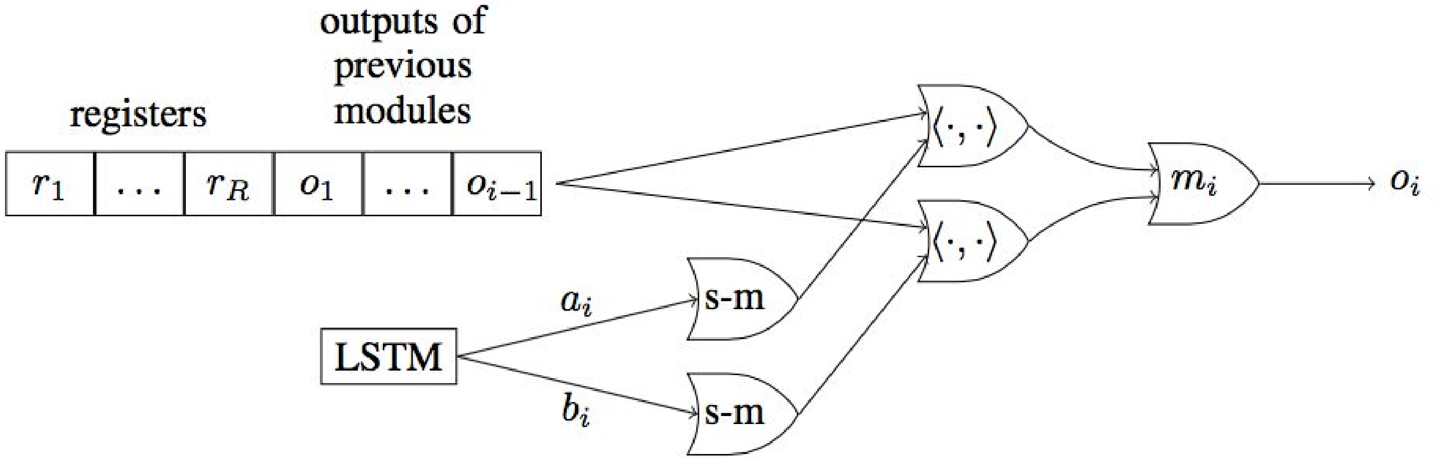


Step	0	1	2	3	4	5	6	7	8	9	10	11	r_1	r_2	r_3	r_4	READ	WRITE
1	6	2	10	6	8	9	0	0	0	0	0	0	0	0	0	0	p:0	p:0 a:6
2	6	2	10	6	8	9	0	0	0	0	0	0	0	5	0	1	p:1	p:6 a:2
3	6	2	10	6	8	9	2	0	0	0	0	0	0	5	1	1	p:1	p:6 a:2
4	6	2	10	6	8	9	2	0	0	0	0	0	0	5	1	2	p:2	p:7 a:10
5	6	2	10	6	8	9	2	10	0	0	0	0	0	5	2	2	p:2	p:7 a:10
6	6	2	10	6	8	9	2	10	0	0	0	0	0	5	2	3	p:3	p:8 a:6
7	6	2	10	6	8	9	2	10	6	0	0	0	0	5	3	3	p:3	p:8 a:6
8	6	2	10	6	8	9	2	10	6	0	0	0	0	5	3	4	p:4	p:9 a:8
9	6	2	10	6	8	9	2	10	6	8	0	0	0	5	4	4	p:4	p:9 a:8
10	6	2	10	6	8	9	2	10	6	8	0	0	0	5	4	5	p:5	p:10 a:9
11	6	2	10	6	8	9	2	10	6	8	9	0	0	5	5	5	p:5	p:10 a:9

Model (No External Memory)

- OK, how are **inputs for module m_i chosen? (This defines the circuit)**
- For each 1<= i <= Q, (Q is number of modules), i-th ouput determined by i-th gate according to:

$$o_i = m_i \left((r_1, \ldots, r_R, o_1, \ldots, o_{i-1})^T \mathbf{softmax}(a_i),\ (r_1, \ldots, r_R, o_1, \ldots, o_{i-1})^T \mathbf{softmax}(b_i) \right).$$



Model (No External Memory)

- R's are probability distributions, so inputs to m_i being weighted averages of probability distributions, are also probability distributions
    - How to take probability distribution as input and make output also probability distribution?

$$\forall_{0 \le c < M} \ \mathbb{P}(m_i(A,B) = c) = \sum_{0 \le a,b < M} \mathbb{P}(A = a)\mathbb{P}(B = b)[m_i(a,b) = c].$$

- Q outputs o_1,...,o_Q. Out of values currently in register and outputs ({r1,...,r_R,o_1,...,o_Q}), which should be stored in registers?
    - Controller outputs vectors c_i in R^R+Q for each register r_i, 1 <=i <= R

$$r_i := (r_1, \ldots, r_R, o_1, \ldots, o_Q)^T \mathbf{softmax}(c_i).$$

        - softmax(c_i) just picks out which value to store

Controller's Inputs

- For each register, controller (NN or LSTM) receives $P(r_i = 0)$ (as opposed to entire distribution)
- Motivation:
  - Decouples memory size from M
  - Limits amount of info available to controller, so problem is solved by modules and not controller (separation of duty)

Memory Tape (Pointers!)

- OK, what we have right now can do LIMITED sequence-to-sequence transformation
    - 1) Initialize the registers with input sequence
    - 2) Train the model to produce the desired output sequence in its registers after a given number of timesteps

- PROBLEM: inability to generalize to longer sequences
    - length of sequence model can process is equal to number of registers
- SOLUTION:
    - Variable-size memory tape: **M memory cells**, each which stores a distribution over the set {0,1,...,M-1}

Memory Tape (Pointers!)

- **Memory cells** store values (again, probability over {0,1,...,M-1})
- Matrix **M**, value M_i,j is probability that the i-th cell holds the value j
- Registers can be seen as **fuzzy pointers** (distribution over {0,1,...,M-1}, M memory cells)
- **READ** module: takes as input a pointer, returns value in memory
    - If input is fuzzy pointer (distribution), then module returns (M^T)*p (distribution)
- **WRITE** module: takes as input a pointer p and a value a, stores the value a under the address p in the memory
    - Fuzzy form with matrices

Inputs and Outputs Handling

- Model memory initialized with input sequence, expected to produce the output in the memory
- Controller decides whether to continue the execution or finish it, in which case the current state of the memory is treated as the output
    - Controller outputs scalar take sigmoid to get value f_i in [0,1] as done or not
    - Can also add maximum timesteps T

Inputs and Outputs Handling

- **LOSS OF THE MODEL**: NLL of producing the correct output,

$$-\sum_{t=1}^{T}\left(p_t \cdot \sum_{i=1}^{M} \log(\mathcal{M}_{i,y_i}^{(t)})\right)$$

- For an **input-output pair x,y in {0,1,...,M-1}^M** (M length sequence, {0,1,...,M-1} is probability)
- Where **matrix M_i,j^(t)** is probability that the i-th memory cell holds the value j after timestep t
- **Vector p_t** = probability that output is produced at exactly the timestep t (f_i is probability of termination

$$p_t = f_t \cdot \prod_{i=1}^{t-1}(1 - f_i)$$

Discretization

- READ takes M^2 time because multiply of M and p
- Hypothesis (of what happens as the model learns):
    - Model naturally learns solutions in which the distributions of intermediate values have very low entropy. E.g. more like [0.01,0.95,0.01,0.01,0.01] than [0.2, 0.2, 0.2, 0.2, 0.2]
        - BECAUSE, if this wasn't the case, increased fuzziness would lead to more uncertainty, higher cost
    - **Basically, replace softmax with [0,0,0,1,0....] where 1 is at index with argmax**
        - Which speeds up computation
- Superforecasters

Experiments - Training

Q: Are these empirically-learned engineering tricks plausible in biological brains? How might they be occur?

- Adaptive learning rate optimization techniques
- Gradient clipping
- Curriculum Learning
- Noise
- Enforcing Distribution Constraints

Experiments - 14 Modules

- ZERO(a,b) = 0
- ONE(a,b) = 1
- TWO(a,b) = 2
- INC(a,b) = (a+1) mod M
- ADD(a,b) = (a+b) mod M
- SUB(a,b) = (a-b) mod M
- DEC(a,b) = (a-1) mod M
- LESS-THAN(a,b) = [a<b]
- LESS-OR-EQUAL-THAN(a,b) = [a<=b]
- EQUALITY-TEST(a,b) = [a=b]
- MIN(a,b) = min(a,b)
- MAX(a,b) = max(a,b)
- READ
- WRITE

Experiments - Tasks

- Input is given to the network in the memory tape
- Goal is to modify memory
- Final error is c/m
  - c is correctly written
  - m is total number of cells that should be modified
- EASY
  - Access, Increment, Copy, Reverse, Swap
- HARD
  - Permutation, ListK, ListSearch, Merge, WalkBST

Results

- 0% error for all except Merge and WalkBST (error <= 1 %)
- Generalizes to inputs longer than the ones seen during training
  - Train complexity: e.g. length of input array <= 20
  - Tested on inputs up to length 50^6
- Discretization hurts performance for all HARD tasks except Permutation
- Question: How long did it take to train?

Conclusion & Discussion

- Key points for memory networks, augmented networks:
  - Differentiability
  - Ability to handle variable length inputs and outputs
  - Generalizability to longer lengths than training data
  - Probabilistic memory
- What should the evaluation tasks be?