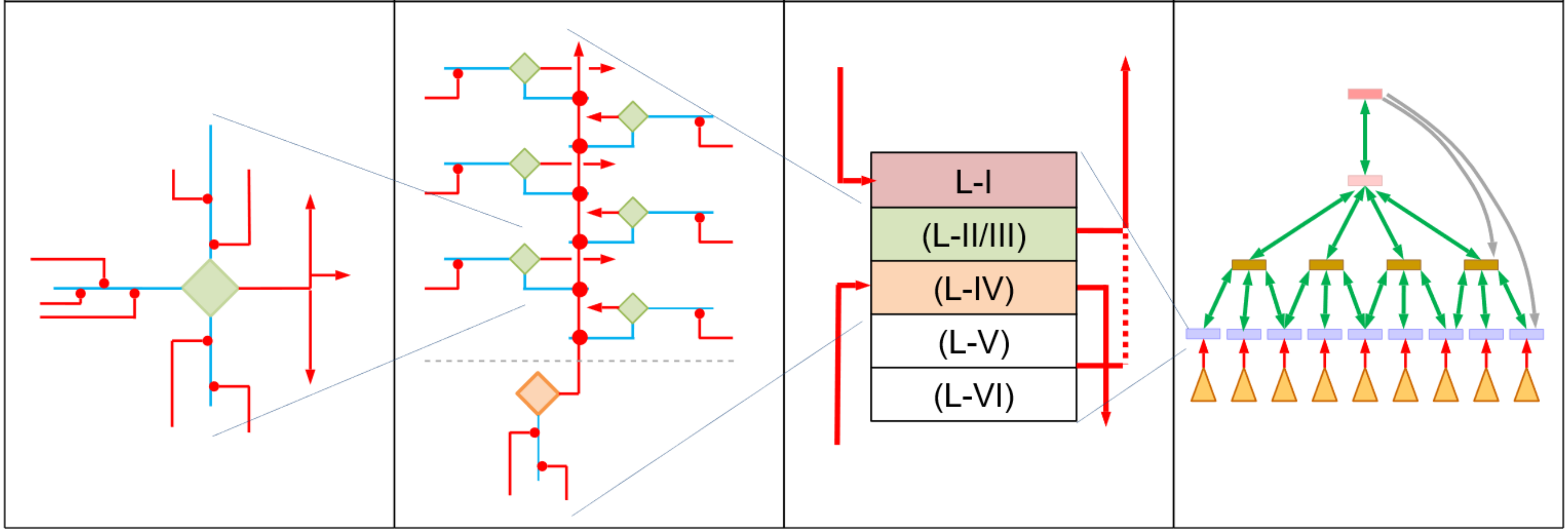
# Synaptic plasticity in an artificial Hebbian network exhibiting continuous, unsupervised, rapid learning

J. Campbell Scott, IBM Research Almaden (jcscott@us.ibm.com)
with Thomas F. Hayes, Ahmet S. Ozcan, Winfried W. Wilcke,

- Review / overview of the CAL network
  - Context Aware Learning
- Evolution of synapse population
- Learning from few examples
- Less forgetting

# Architecture of CAL: Context Aware Learning

| **Cellular scale**<br>Neuron, axons, dendrites, synapses | **Micro scale**<br>Mini-column | **Meso scale**<br>“Region” of layered cortical sheet | **Macro scale**<br>Hierarchical network of cortical “regions” |
|---|---|---|---|



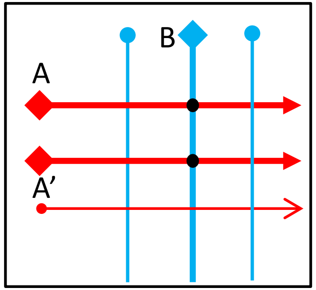
# CAL / DNN Comparison

(the top four)

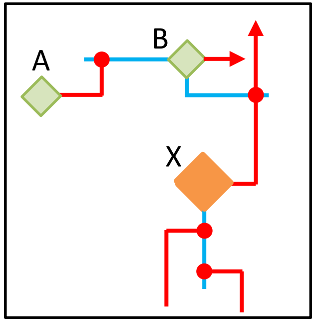
| | **DNN** | **CAL** |
|---|---|---|
| Data representation | Analog (differentiable) neurons<br>Compact real vectors | Binary neurons<br>Sparse binary vectors |
| Learning | Global cost function<br>Back-propagation of errors<br>Gradient descent | Strictly Hebbian (local)<br>Neurons that fire together,<br>wire together |
| Synapse generation | Connections defined by network design | Plastic synapses: generated, updated and removed in response to data |
| Consequences | Slow learning<br>Large data sets<br>Catastrophic forgetting | Learns rapidly, in real time<br>Few examples needed<br>Long term retention of most relevant |

# Previously demonstrated with CAL

1. Correlation via Hebb
   - Input (A) fires then output (B) fires; synapse is strengthened
   - Two or more inputs (A, A') fire, then output fires; inputs connect to the same output
     Learning: "coincidence detection"
   - Inference: firing output signals correlation of inputs

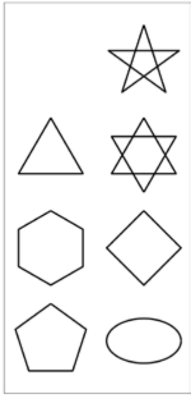


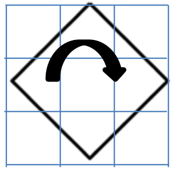

2. Learning sequences via Hebb
   - Prediction: modulating input from neuron(s) A reduces firing threshold of B
   - Verification: input from active neuron X causes B to fire (in the context of A)
   - Prediction is verified, synapse is strengthened
   - (Sub-)Sequence A→B is remembered
   - B contributes to prediction of C

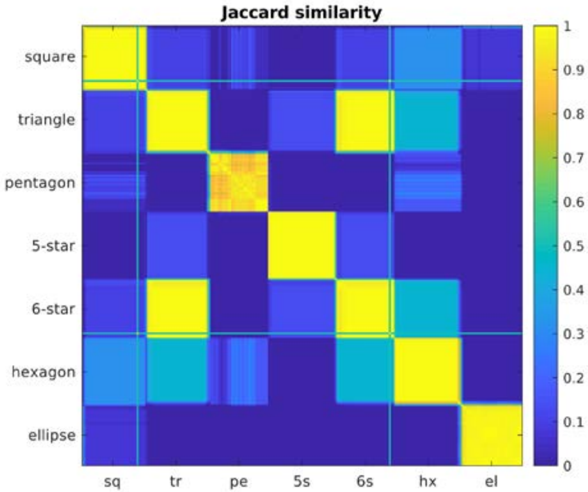


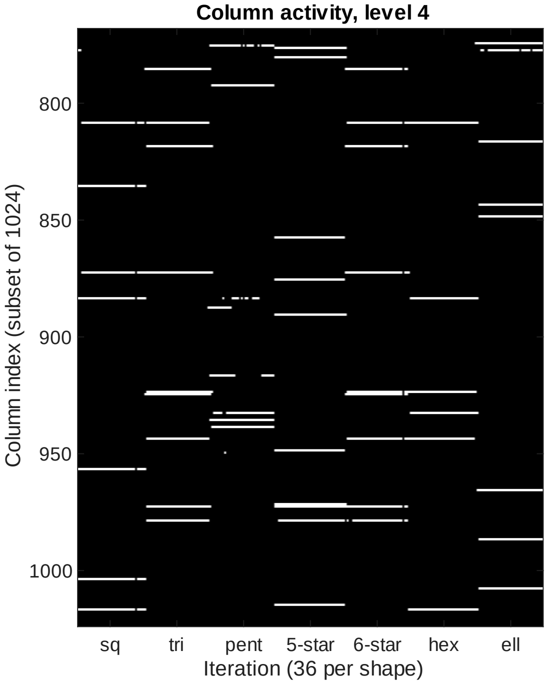
# Previously demonstrated with CAL (cont.)

3. Generation of stable representations
   - 4 level hierarchy (9, 4, 1, 1 regions)
   - Input binary video (rotating shapes)
   - Feed-forward verified predictions
   - Temporal pooling
   - $4^{th}$ level output: stable during each clip
   - Similar / orthogonal







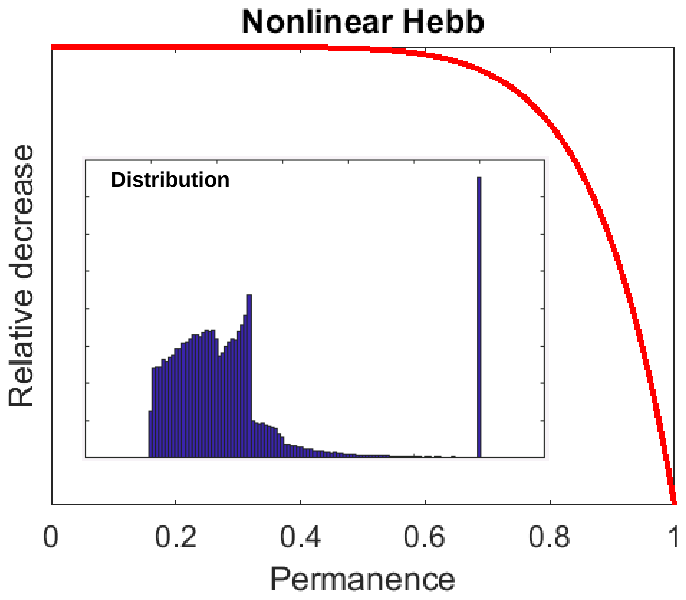
# Previously demonstrated with CAL (cont.)

4. Proposed method to avoid catastrophic forgetting

   – Non-linear permanence decrements reaching zero at maximum permanence

   – Leads to two populations of synapse: plastic and permanent

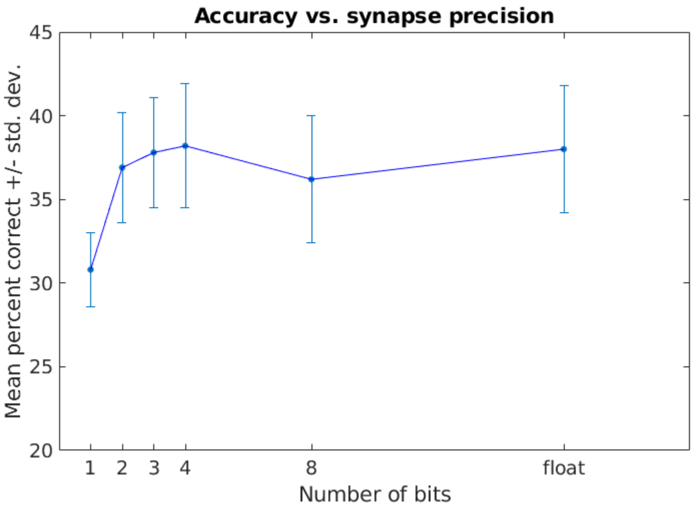
   – (… more to follow …)

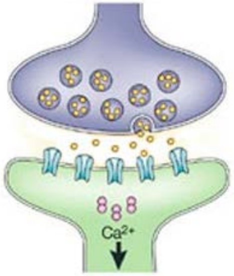


# What is new with CAL?

**Algorithms**

- More precise synaptic weights
  - Beyond binary
  - 4-bits virtually the same as double precision
  - cf. 15 ion-channels?
  - Permit hardware acceleration
- Synapses initialized with zero weight
  - No “seeding” to initiate learning
  - Synapses generated in response to neural activity
  - Ensures that new synapses are “relevant”
- Connect newly active axons to dendrites with fewest synapses




Faster and more accurate learning

# Synapse plasticity

Three phases

1. ~ 100 iterations: Creation of new synapses
   First correct prediction @30
   Prediction accuracy improves rapidly ( >95% @300)

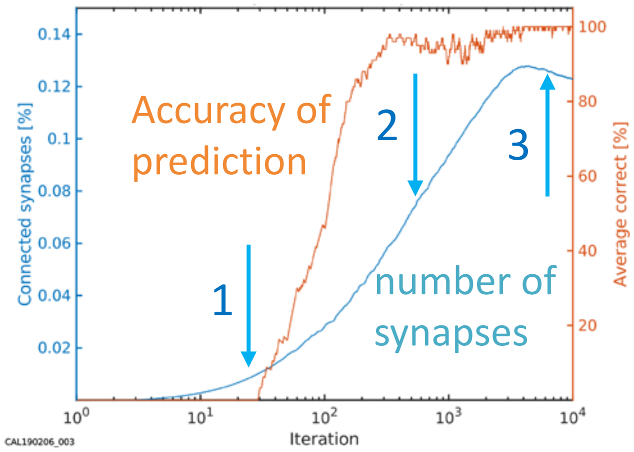
2. ~ 100 – 4000: Pruning starts
   Minor loss of accuracy from ~ 400

3. >~ 4000:  Prune weakest
   Some neurons become permanent – never forget
   Accuracy reaches 100%

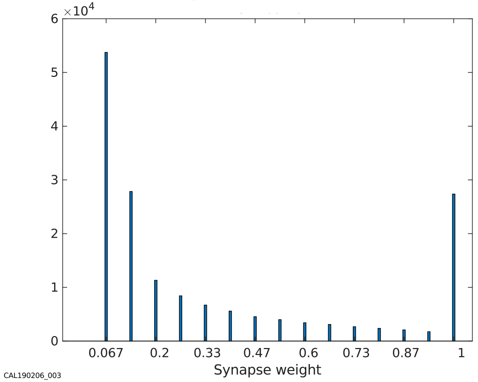
After 10k iterations

- Fewer than 0.14% of possible connections are made
- Many synapses are permanent (weight=1)
- Remainder are tending weaker

[Input data is quasi-chaotic, non-repeating sequence from population equation.
Weights have 4-bit precision]




Weight distribution



# Fast learning – compare conventional RNN

Input: text sequence (Ch. 1 of *Alice in Wonderland*)
Single pass (11,263 characters)
Include spaces, punctuation, new-line, etc. (hard)

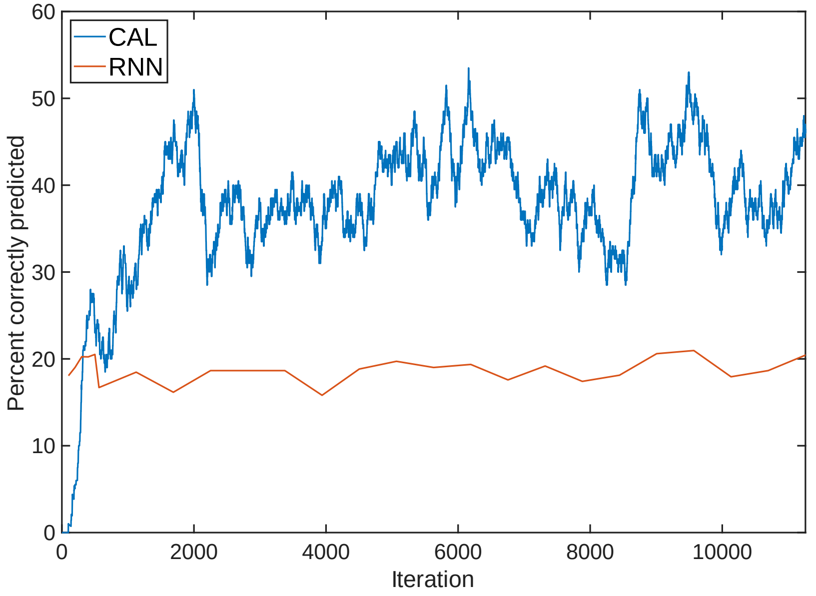
One input character per iteration, predict next.
Metric: percent correctly predicted.

Single region of CAL.

- First correct prediction @80,
- 20% accuracy @700,
- 30% @900, 40% @1500
- Accuracy ~ 2.5x previous version of CAL

RNN: Elman network, one hidden layer

- Also trained one character at a time.
- Minimize cross-entropy
- Reaches 20% at @9000 and 20.5% @100k, but …



# What is being learned in text example?

CAL predictions are clearly based on context: initially short words and common syllables.
Spaces often correctly placed, e.g. after “…ing,” “and”

RNN initially predicts based frequency
e.g. 18% are spaces, reaches ~18% accuracy by predicting all spaces. Then too many ‘t’s.

correct in context

```
Iterations 51 to 100
Input 'y her sister on the bank, and of having nothing to'
CAL   '  t yytit idVbfLsierse   Gonk on ben!ng td ieng ki'
RNN   '                                                   '

Iterations 1951 to 2000
Input 'e came upon a heap of sticks and dry leaves, and t'
CAL   'e tate ttG  ttte t tu th nen tt  t   _l te   ttB t'
RNN   ' t  t   t     t  t    t     tt   t               t'

Iterations 10951 to 11000
Input 'she remained the same size: to be sure, this gener'
CAL   'the t hesngI  the thne thnXr th tertht   then t rd '
RNN   '  w        t   t    t     s  s  s     s    s      '
```

# Learn rapidly or … ?

# Towards immortal memory

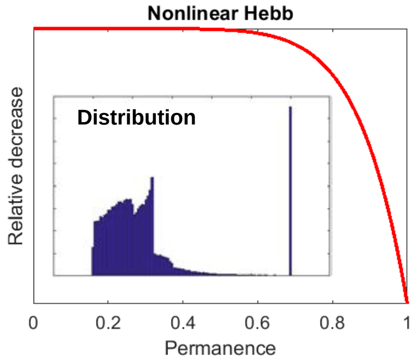
Do nonlinear Hebb updates minimize forgetting?

- 3 (initially identical) networks distinguished by first task

  A. Easy: random sequence of length 100;
  100% accurate after ~10 epochs

  B. Moderate: 3 sentences in random order;
  87% accurate after 1 epoch (34x3 sentences)

  C. Hard (*Alice in Wonderland*):
  40% accurate after single epoch (11,263 characters)




- Then each network cycles through all tasks
  (still updating synapses)

# Learning not to forget

Task 1: sequence of 100 random characters
Task 2: three sentences in random order
Task 3: *Alice in Wonderland*, Chapter 1

Network A learns task 1 first, 100 % accurate
Network B learns task 2 first, 87%
Network C learns task 3 first, 40%

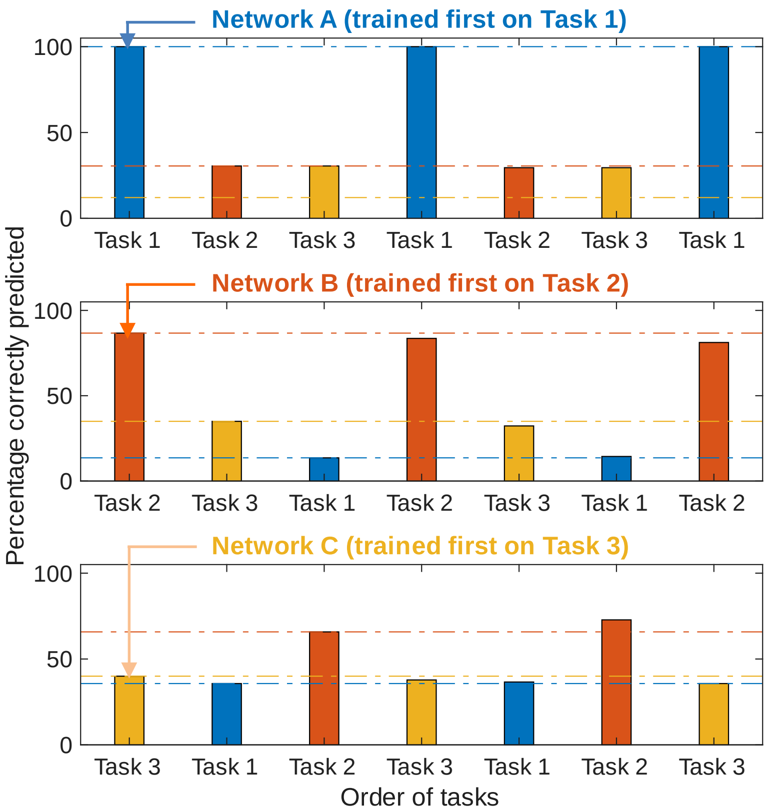
All networks show

- Small or no drop returning to first learned task
- Small change (+/-) returning to $2^{nd}$, $3^{rd}$ tasks

CAL may forget “gracefully” - not catastrophically

Loss of capacity after first task learned

Network size was selected for single task and fast execution



# Capacity is an issue


"Mr. Osborne, may I be excused? My brain is full."

[Gary Larsen, Far Side]

# Conclusions

- (In CAL) Memories are retained in synapses
  - Generated and retrieved by neuron activity
- Synapse plasticity
  - Structural: new connections made, irrelevant ones removed
  - Weight adjustment: based on **local** neural activity
  - No plasticity: reach full permanence
- Leading to
  - Fast learning
  - In context via modulating synapses/dendrites

“It is important to make the right connections”

(Hugh Whitemore, *Breaking the Code* – a play about life of Turing)

# Acknowledgments

IBM

Hernan Badenes
Charles Cox
Pritish Narayanan
David Pease
Tomasz Kornuta
Jayram Thathachar
Alexis Asseman

**Numenta**
Jeff Hawkins
Subutai Ahmad

Canon
Takamasa Tsunoda

Ryusei Shingaki

SAMSUNG
Hyong-Euk (Luke) Lee

Thank you!

# Backup

# Summary

- CAL learns rapidly from every input, in real time
  - Synapse weights change in response to local activity (Hebb)
  - Not regression to minimize a loss function
  - Multimodal input: binary images, text (integers), real numbers, … can be mixed
- CAL learns sequences via context provided by prior data
- CAL generates representations of sequences in upper levels
- Nonlinear Hebb reduces forgetting
- Feedback via apical synapses is predictive

# What next?

- How to apply predictive feedback?
  - Provide longer term context
- Interpretation via correlator
  - E.g. text and video input
- More general modulation
  - Not all neurotransmitters are ionic, potentiating
  - e.g. dopamine modulates learning rate (magnitude of synapse updates)
  - etc.
- etc.

# Some key definitions

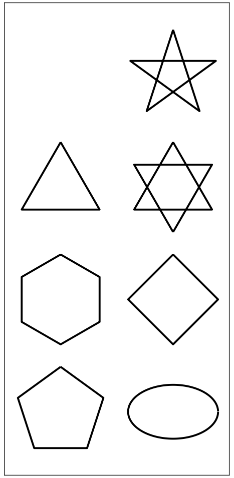
- **encoder**:  encodes analog values (from sensor) as sparse binary vector
- **binary correlator**:  signals when any pair of axons are frequently active at the same time
- **sequence memory**:  predicts which neurons are expected to be active at the next time step, and strengthens synapses if they are indeed active
- **overlap**:  the number of active axons which have synaptic connections to the same dendrite

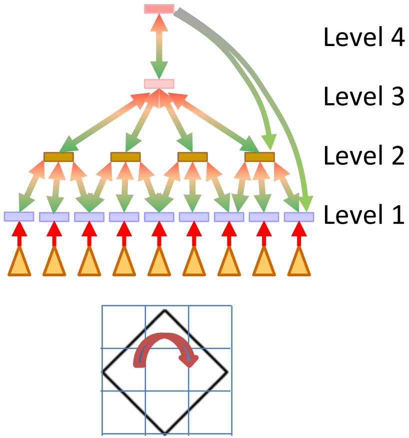
# Feed-forward (FF) upwards in the hierarchy

- In each region, temporal pooling of feed-forward data (sparse binary vectors)
  - Union (logical OR) of consecutive iterations
  - Input to correlator
  - i.e. correlator “compares” consecutive FF vectors

Input data:

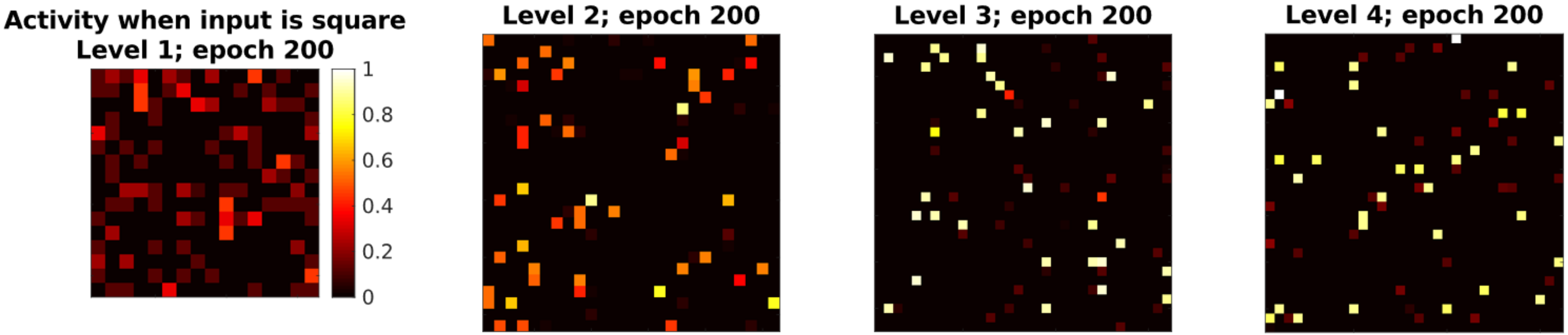
- sequence of binary images
- 9 receptive fields
- 7 rotating shapes
- 36 frames per shape
- i.e 7x36 = 252 iterations / epoch





# Representation of sequences is spontaneous

- As the data propagate upward,
  column activity becomes increasingly stable.
- At level-4, the same mini-columns remain active for each shape



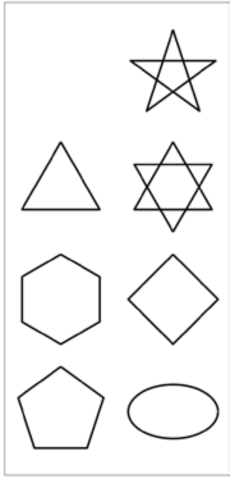

Each pixel corresponds to one mini-column
Color shows fraction of time it is active for a single shape

pattern means
“rotating square”

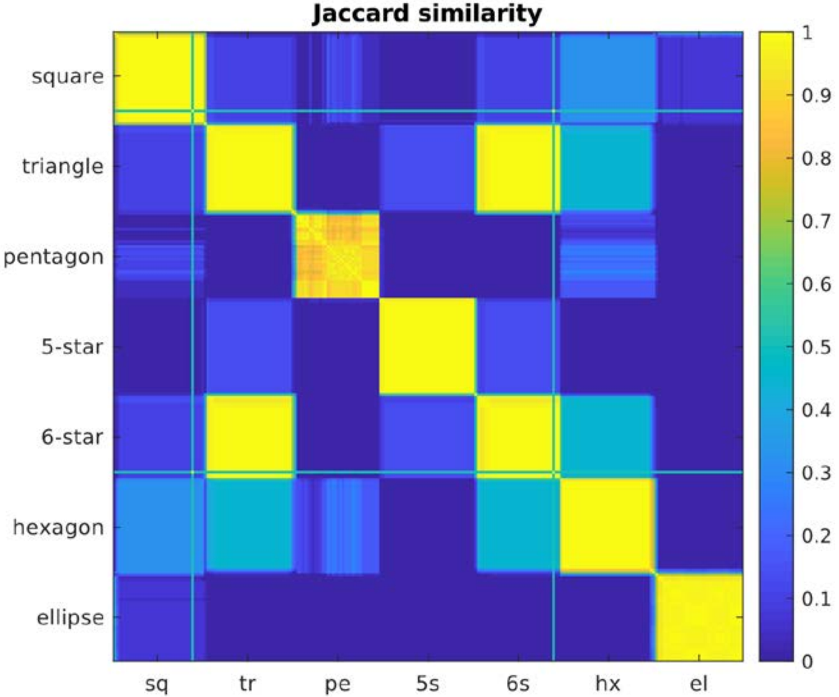
# Representations range in similarity / orthogonality

Jaccard similarity of binary vectors, A B, is overlap normalized by union.

$$J = \frac{|A \cap B|}{|A \cup B|}$$



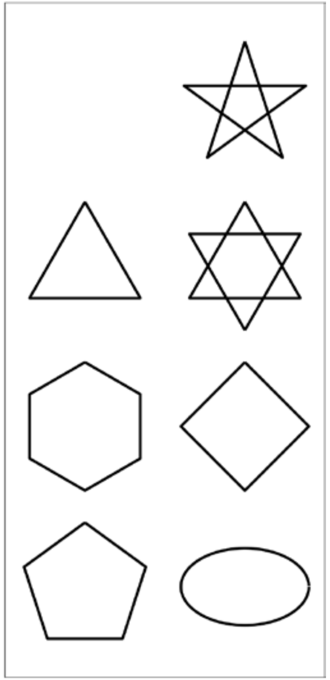
$J$=0, orthogonal; $J$=1, identical. Compare outputs of level-4 correlator at pairs of iteration.

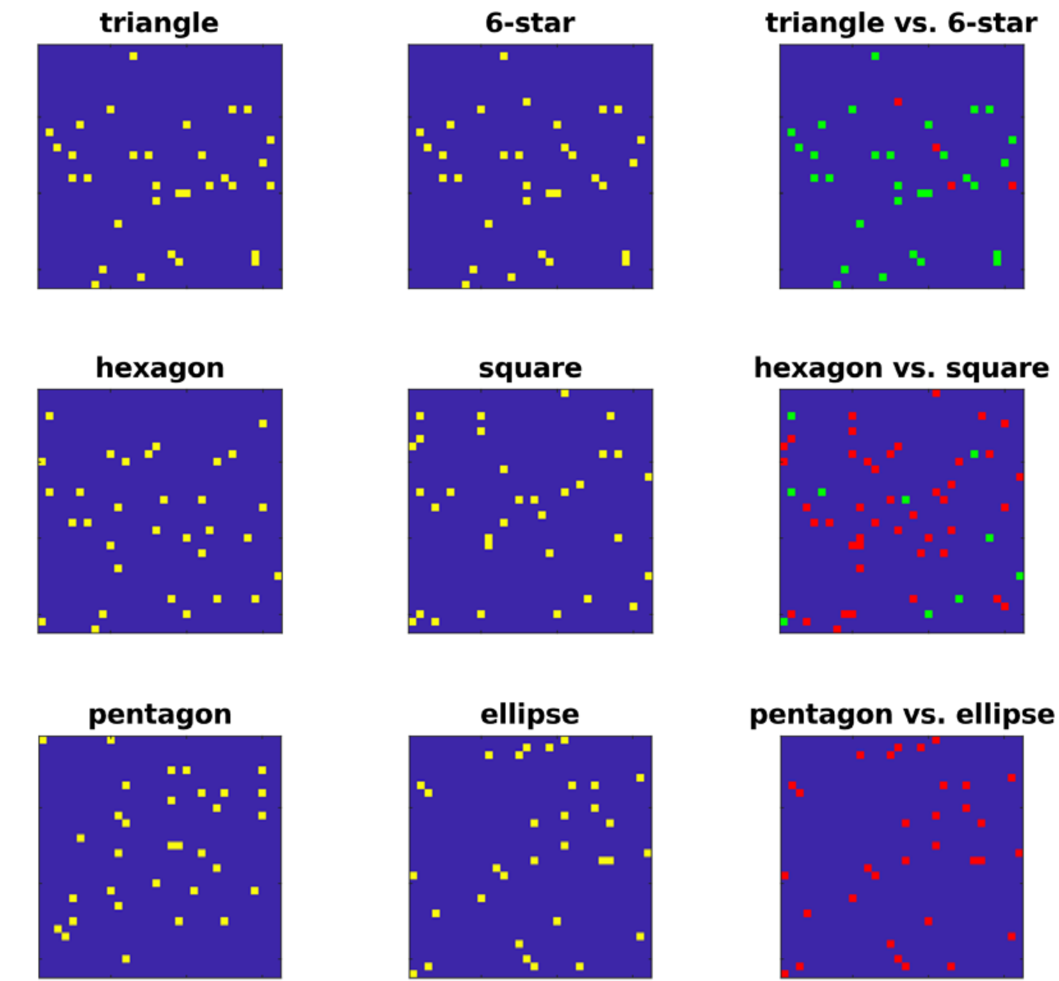



6-pointed star is most like triangle (it is two triangles)

Ellipse is virtually orthogonal to everything else

# Visualization of similarity






AND XOR

Quite similar
($J = 0.88$)

Some similarity
($J = 0.19$)

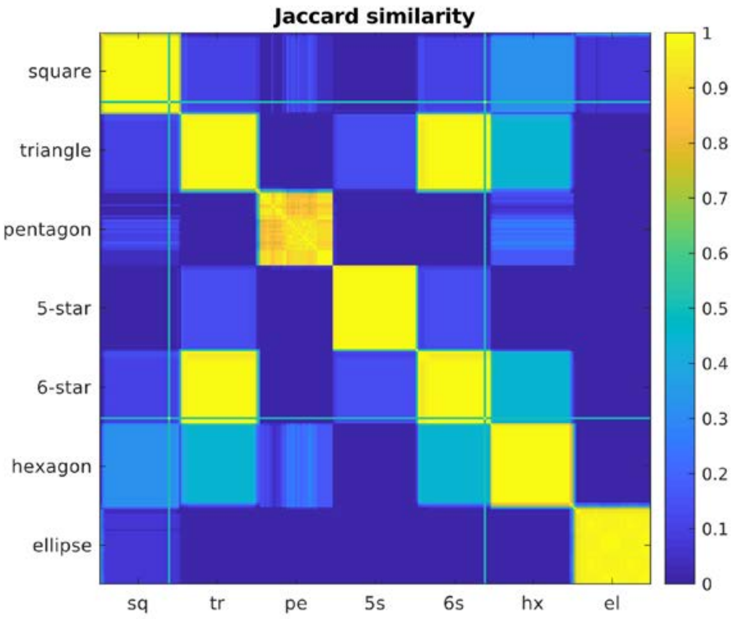
Orthogonal
($J = 0$)

# Jaccard and Hamming

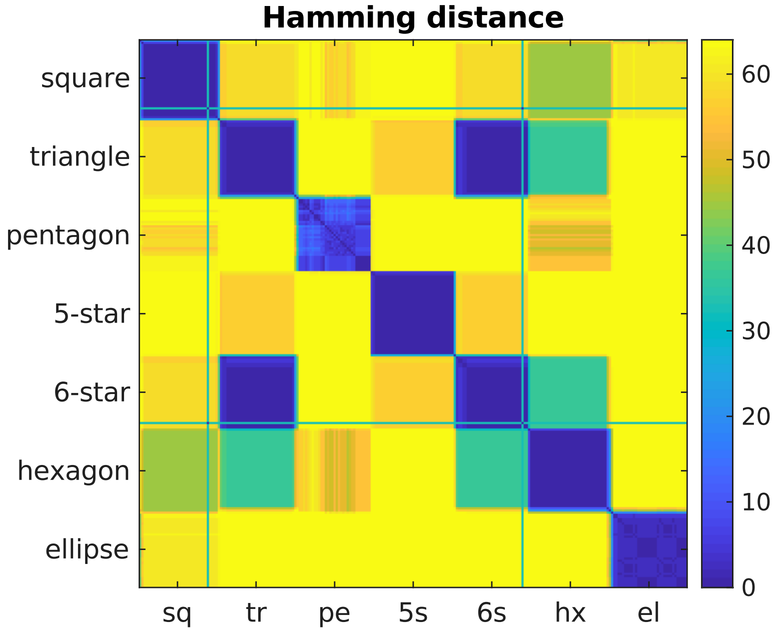
$$J = \frac{2N_a - H}{2N_a + H}$$

$N_a$ bits active in each binary vector (here $N_a$ = 32)

$$H = 2N_a \frac{1 - J}{1 + J}$$




CAL190301_1401




CAL190301_1401

# Full disclosure – capacity issue

Accuracy [%]

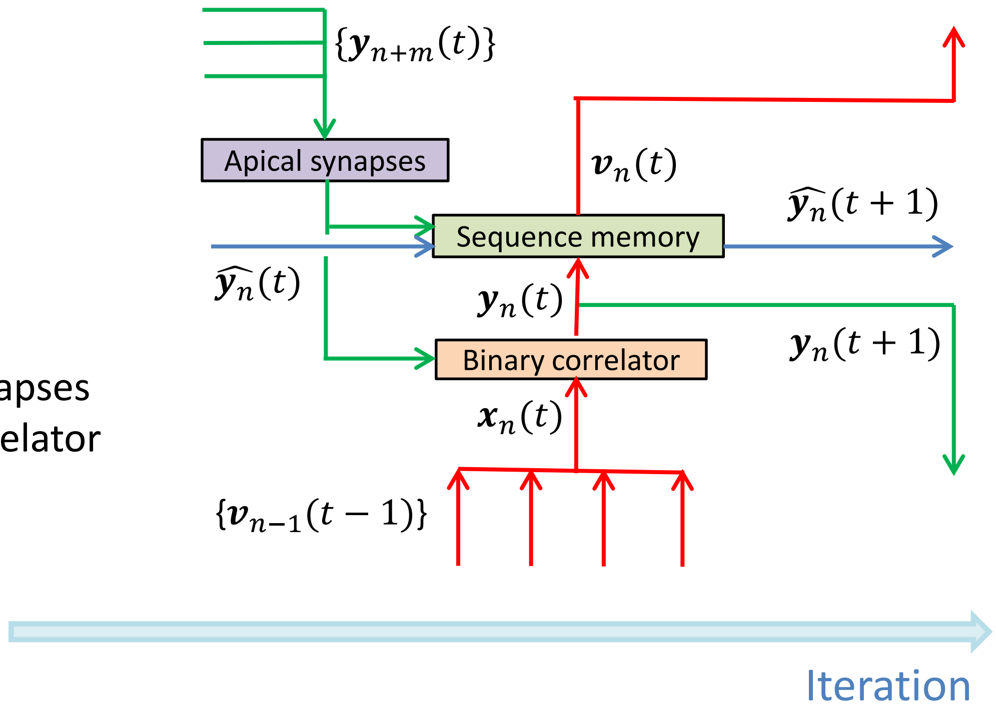
| | Network A | Network B | Network C |
|---|---|---|---|
| Task 1 | 100 | | |
| Task 2 | 30.5 | 86.7 | |
| Task 3 | 12.1 | 35.0 | 40.0 |
| Task 1 | 100 | 13.6 | 35.1 |
| Task 2 | 29.5 | 83.6 | 65.8 |
| Task 3 | 13.4 | 32.3 | 37.8 |
| Task 1 | 100 | 14.4 | 36.6 |
| Task 2 | | 81.2 | 72.8 |
| Task 3 | | | 35.7 |


"Mr. Osborne, may I be excused? My brain is full."

[Gary Larsen, Far Side]

# Data flow and timing: feed-forward and feedback

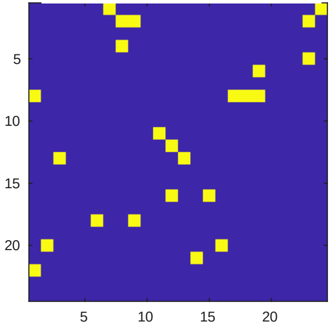
1. Data (vector) from region(s) below concatenated and enter correlator
2. Output from correlator passed to sequence memory, and fed back
3. Compared with previous prediction
   Verified neurons fire and feed-forward
   New prediction saved for next iteration
4. Feedback from upper levels to apical synapses
5. Modulate sequence memory and/or correlator
6. Next input is (concatenation of) verified neurons in level below.



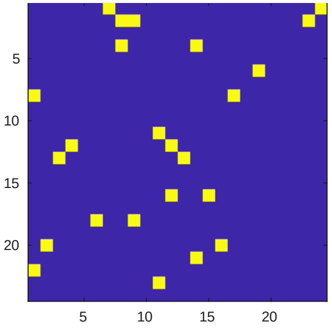
# Learning in apical synapse array

Compare active
apical dendrites
with next column activity

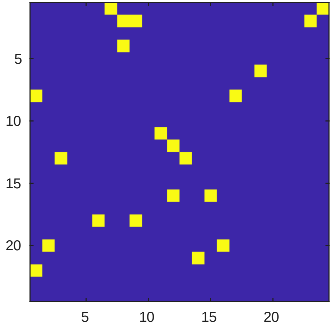
Apical activity



Active columns



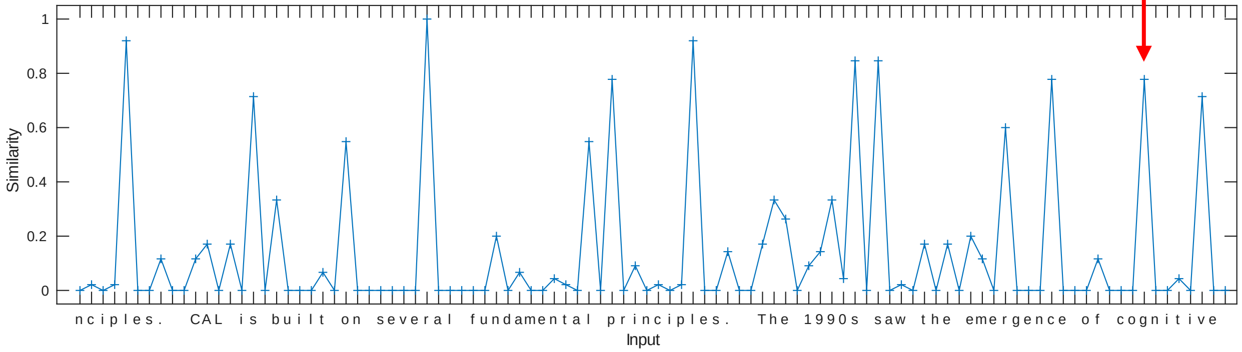
Match (AND)



Jaccard similarity of binary vectors, A B, is overlap normalized by union: $\frac{|A \cap B|}{|A \cup B|}$

Apical feedback
predicts next input.
(not every iteration)

Long term context




CAL190301_1401

# Binary correlation

- Correlation is a time average showing how often a pair of bits are active at the same time, vs. being active at different times
- The correlation between two bits, $x_i, x_j$ of binary vector $\boldsymbol{x}(t)$, $t = 1 \ldots N$ is

$$\chi(x_i, x_j) = \frac{\sum_t [x_i(t) \wedge x_j(t) - x_i(t) \otimes x_j(t)]}{\sum_t [x_i(t) | x_j(t)]}$$

where the numerator is

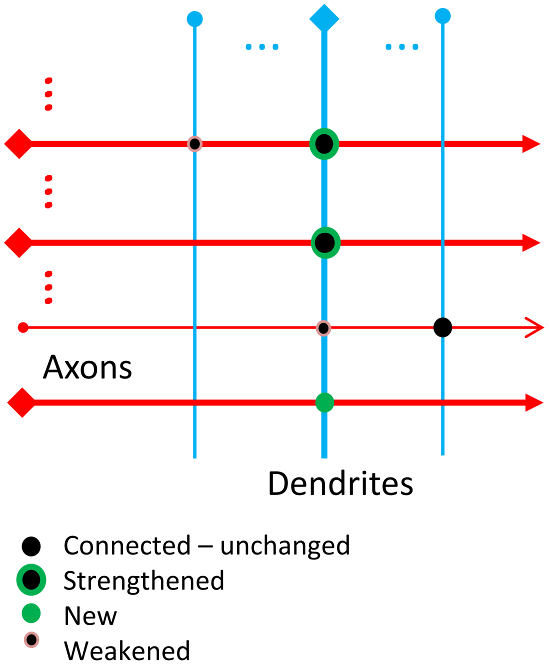
positive, +1, ($\wedge \equiv$ AND) if both bits are on

negative, -1, ($\otimes \equiv$ XOR) if only one bit is on

and the denominator is unity ($| \equiv$ OR) when either one is on, and normalizes $-1 \leq \chi \leq 1$.

- Reduces to

$$\chi(x_i, x_j) = \frac{\sum_t [3x_i(t)x_j(t) - x_i(t) - x_j(t)]}{\sum_t [x_i(t) + x_j(t) - x_i(t)x_j(t)]}$$



# Binary correlation

- The correlation between two bits, $x_i, x_j$ of binary vector $\boldsymbol{x}(t)$**,** $t = 1 \ldots N$ is

$$\chi(x_i, x_j) = \frac{\sum_t [x_i(t) \wedge x_j(t) - x_i(t) \otimes x_j(t)]}{\sum_t [x_i(t) |\, x_j(t)]}$$

- Reduces to

$$\chi(x_i, x_j) = \frac{\sum_t [3x_i(t)x_j(t) - x_i(t) - x_j(t)]}{\sum_t [x_i(t) + x_j(t) - x_i(t)x_j(t)]}$$

# Lateral connections provide context