Vectors, Matrices, and Associative Memory

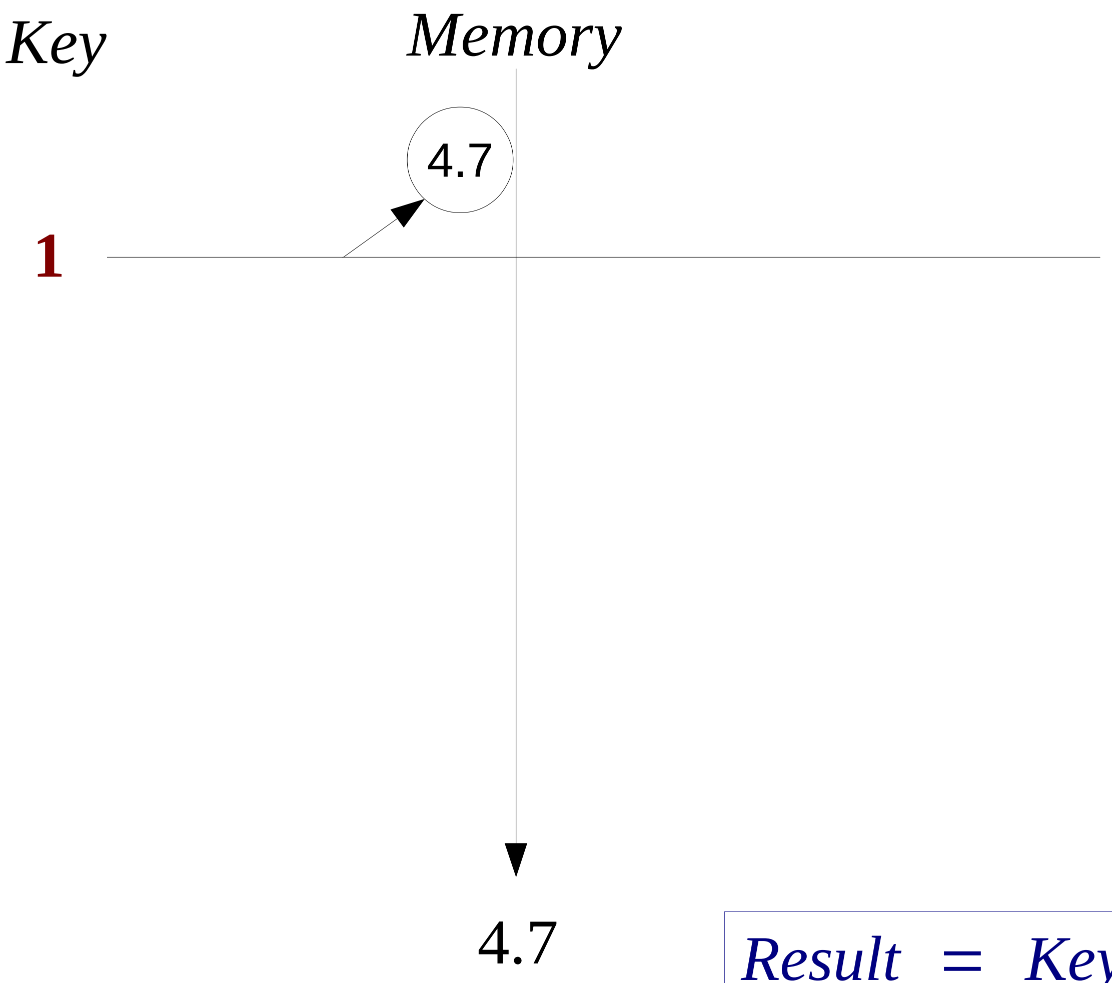
Computational Models of Neural Systems

Lecture 3.1

David S. Touretzky

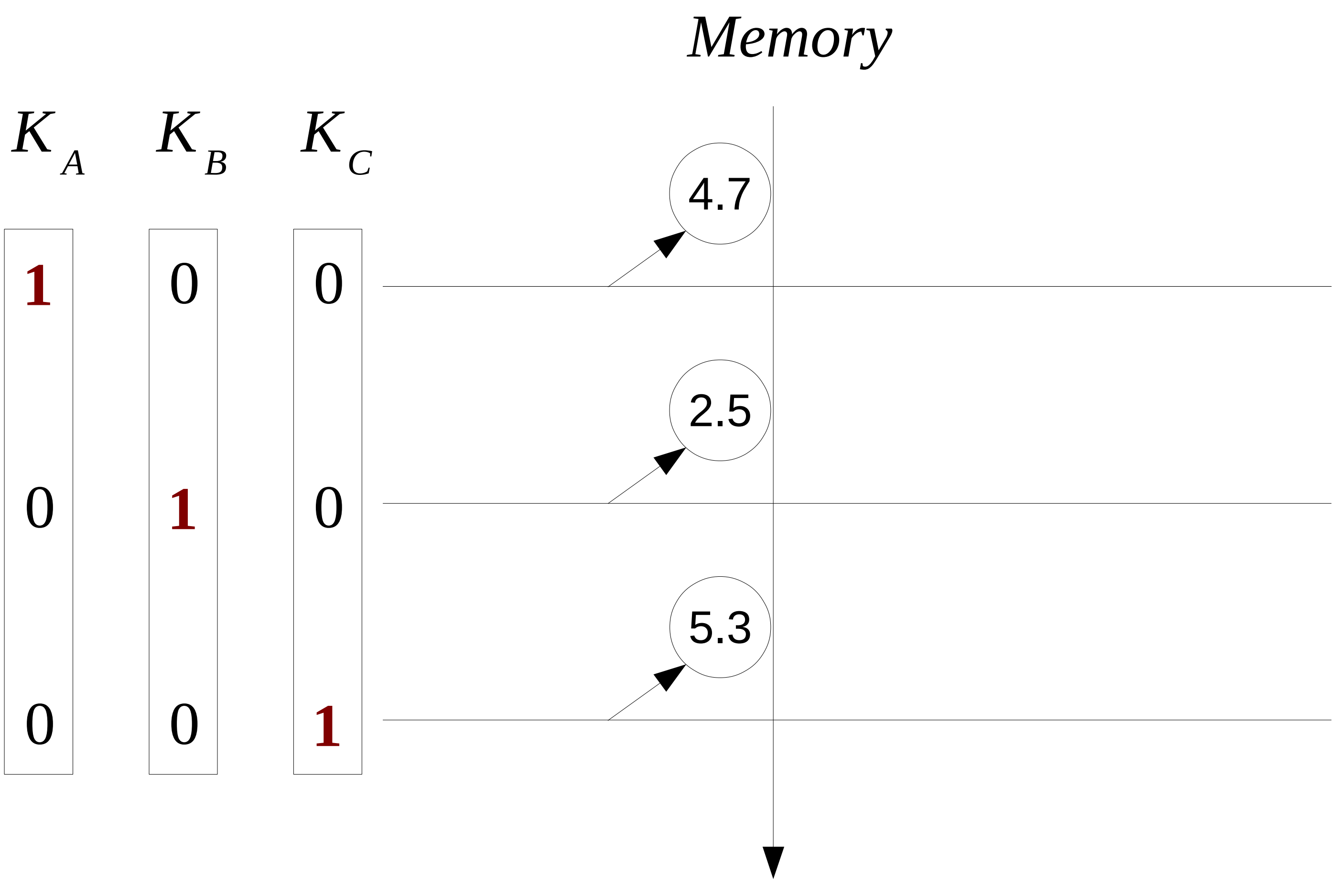
September, 2019

A Simple Memory



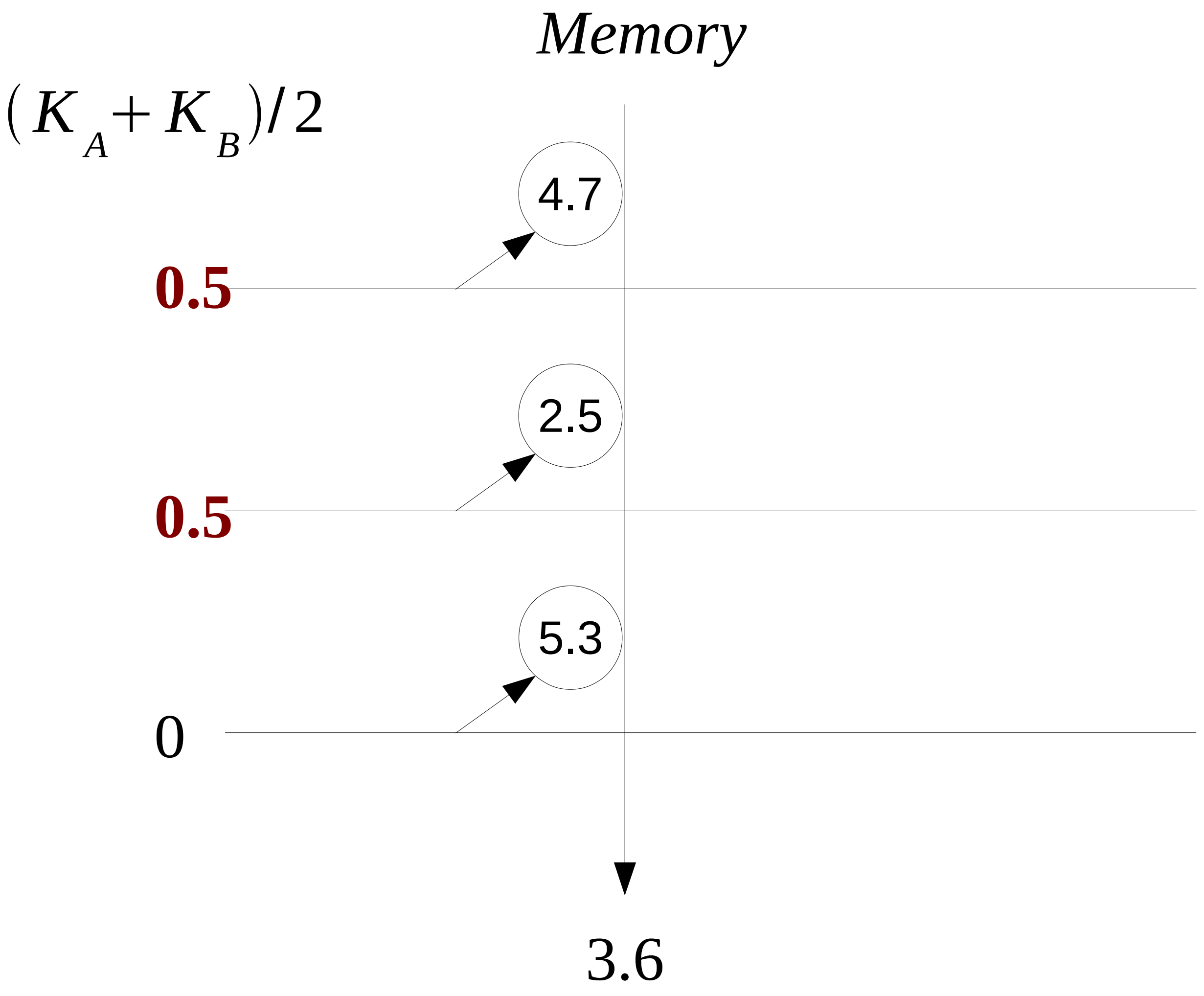
$$Result = Key \times Memory$$

Storing Multiple Memories

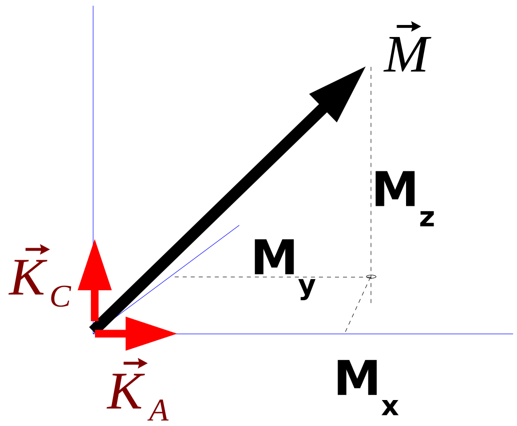


Each input line activates a particular memory.

Mixtures (Linear Combinations) of Memories



Memories As Vectors



This memory can store three things.

$$\vec{M} = \langle 4.7,\ \ 2.5,\ \ 5.3 \rangle$$

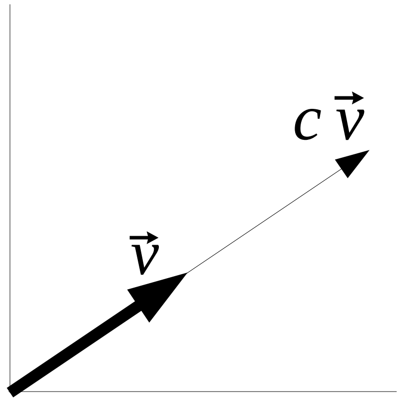
Basis unit vectors:

$$\vec{K}_A = \langle 1,0,0 \rangle = \text{x axis}$$

$$\vec{K}_B = \langle 0,1,0 \rangle = \text{y axis}$$

$$\vec{K}_C = \langle 0,0,1 \rangle = \text{z axis}$$

Length of a Vector

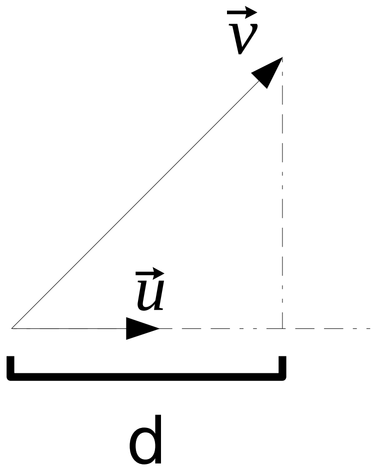


Let $\|\vec{v}\| =$ length of $\vec{v}$.

Then $\|c\,\vec{v}\| = c\,\|\vec{v}\|$

$$\frac{\vec{v}}{\|\vec{v}\|} = \text{a unit vector in the direction of } \vec{v}.$$

Dot Product: Axioms



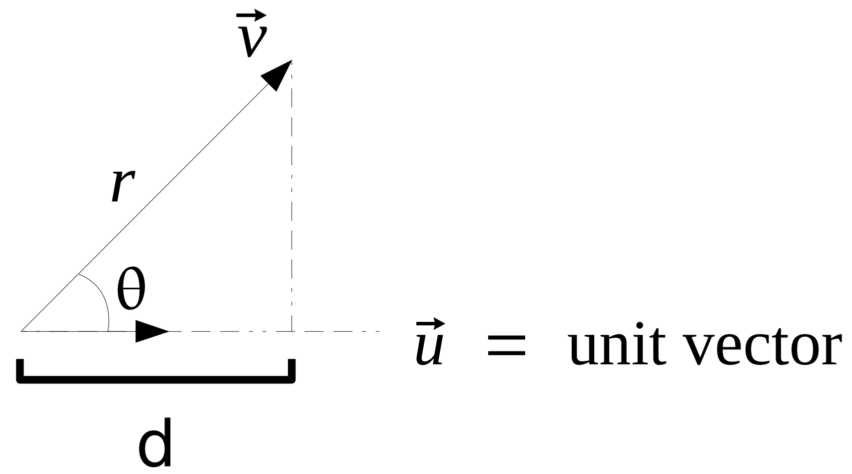
Let $\vec{v}$ be a vector and $\vec{u}$ be a unit vector.

Two axioms for dot product:

$$\vec{v} \cdot \vec{u} = d$$

$$c\,\vec{v}_1 \cdot \vec{v}_2 = c\left(\vec{v}_1 \cdot \vec{v}_2\right) = \vec{v}_1 \cdot c\,\vec{v}_2$$

Dot Product: Geometric Definition



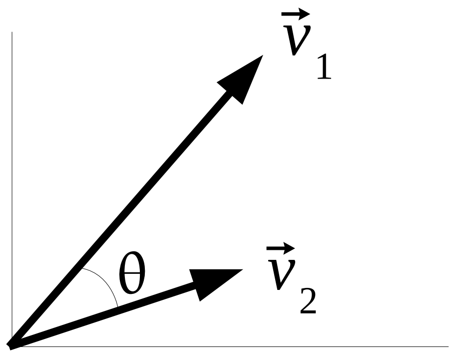
$$\vec{v} \cdot \vec{u} \;=\; d \;=\; r\cos\theta$$

$$r \;=\; \left\|\vec{v}\right\|$$

$$\vec{v} \cdot \vec{u} \;=\; \left\|\vec{v}\right\| \cos\theta$$

Dot Product of Two Arbitrary Vectors

$$\vec{v}_1 \cdot \vec{v}_2 = \|\vec{v}_1\| \; \|\vec{v}_2\| \; \cos\theta$$



Proof:

$$\vec{v}_2 = \left(\frac{\vec{v}_2}{\|\vec{v}_2\|} \right) \|\vec{v}_2\|$$

Unit vector

$$\vec{v}_1 \cdot \vec{v}_2 = \left(\vec{v}_1 \cdot \frac{\vec{v}_2}{\|\vec{v}_2\|} \right) \|\vec{v}_2\|$$

$$= \left(\|\vec{v}_1\| \; \cos\theta \right) \|\vec{v}_2\|$$

$$= \|\vec{v}_1\| \; \|\vec{v}_2\| \; \cos\theta$$

Dot Product: Algebraic Definition

Let $\vec{v} = \langle v_1, v_2 \rangle$ and $\vec{w} = \langle w_1, w_2 \rangle$

$$\vec{v} \cdot \vec{w} = v_1 w_1 + v_2 w_2$$

But also:

$$\vec{v} \cdot \vec{w} = \|\vec{v}\| \, \|\vec{w}\| \cos\theta$$

Can we reconcile these two definitions?

See the proof in the Jordan (optional) reading.

Length and Dot Product

$$\vec{v} \cdot \vec{v} = \|\vec{v}\|^2$$

Proof:

$$\vec{v} \cdot \vec{v} = \|\vec{v}\| \, \|\vec{v}\| \cos\theta$$

The angle $\theta = 0$, so $\cos\theta = 1.$

$$\vec{v} \cdot \vec{v} = \|\vec{v}\| \, \|\vec{v}\| = \|\vec{v}\|^2$$

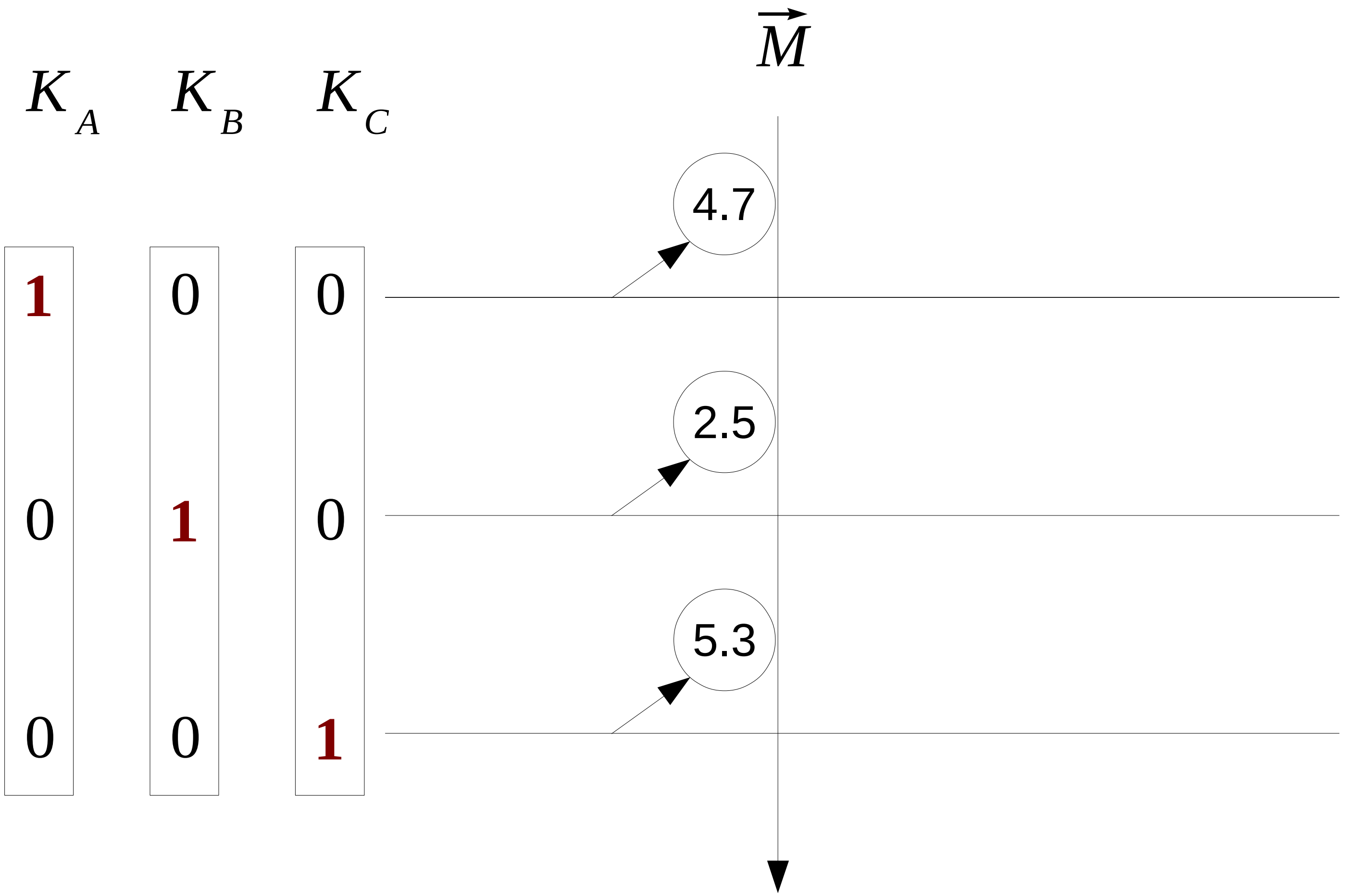
And also:

$$\vec{v} \cdot \vec{v} = v_x v_x + v_y v_y = \|\vec{v}\|^2$$

so we have:

$$\|\vec{v}\| = \sqrt{v_x^2 + v_y^2}$$

Associative Retrieval as Dot Product



Retrieving memory A is equivalent to computing $\vec{K}_A \cdot \vec{M}$

This works for mixtures of memories as well:

$$\vec{K}_{AB} = 0.5\vec{K}_A + 0.5\vec{K}_B$$

Orthogonal Keys

The key vectors are mutually orthogonal.

$$K_A = \langle 1, 0, 0 \rangle$$
$$K_B = \langle 0, 1, 0 \rangle$$
$$K_C = \langle 0, 0, 1 \rangle$$

$$K_A \cdot K_B = 1 \cdot 0 + 0 \cdot 1 + 0 \cdot 0 = 0$$

$$\theta_{AB} = \arccos 0 = 90^o$$

We don't have to use vectors of form $\langle \ldots, 0,1,0, \ldots \rangle$.

Any set of mutually orthogonal unit vectors will do.

Keys Not Aligned With the Axes

$$K_A = \langle 1, 0, 0 \rangle \qquad K_B = \langle 0, 1, 0 \rangle \qquad K_C = \langle 0, 0, 1 \rangle$$

Rotate the keys by 45 degrees about the x axis, then 30 degrees about the z axis.

This gives a new set of keys, still mutually orthogonal:

$$J_A = < \; 0.87, \; 0.49, \; 0 \; >$$

$$J_B = < \; -0.35, \; 0.61, \; 0.71 \; >$$

$$J_C = < \; 0.35, \; -0.61, \; 0.71 \; >$$

$$J_A \cdot J_A = (0.87)^2 + (0.49)^2 + (0)^2 = 1$$

$$J_A \cdot J_B = (0.87) \cdot (-0.35) + (0.49) \cdot (0.61) + 0 \cdot (0.71) = 0$$

Setting the Weights

How do we set the memory weights when the keys are mutually orthogonal unit vectors but aren't aligned with the axes?

$$\vec{M} = \left(m_A \vec{J}_A\right) + \left(m_B \vec{J}_B\right) + \left(m_C \vec{J}_C\right)$$

Prove that this is correct:

$\vec{J}_A \cdot \vec{M} = m_A$ because:

$$\vec{J}_A \cdot \vec{M} = J_A \cdot \left(\vec{J}_A m_A + \vec{J}_B m_B + \vec{J}_C m_C\right)$$

$$= \left(\vec{J}_A \cdot \vec{J}_A\right) \cdot m_A + \left(\vec{J}_A \cdot \vec{J}_B\right) \cdot m_B + \left(\vec{J}_A \cdot \vec{J}_C\right) \cdot m_C$$

1 **0** **0**

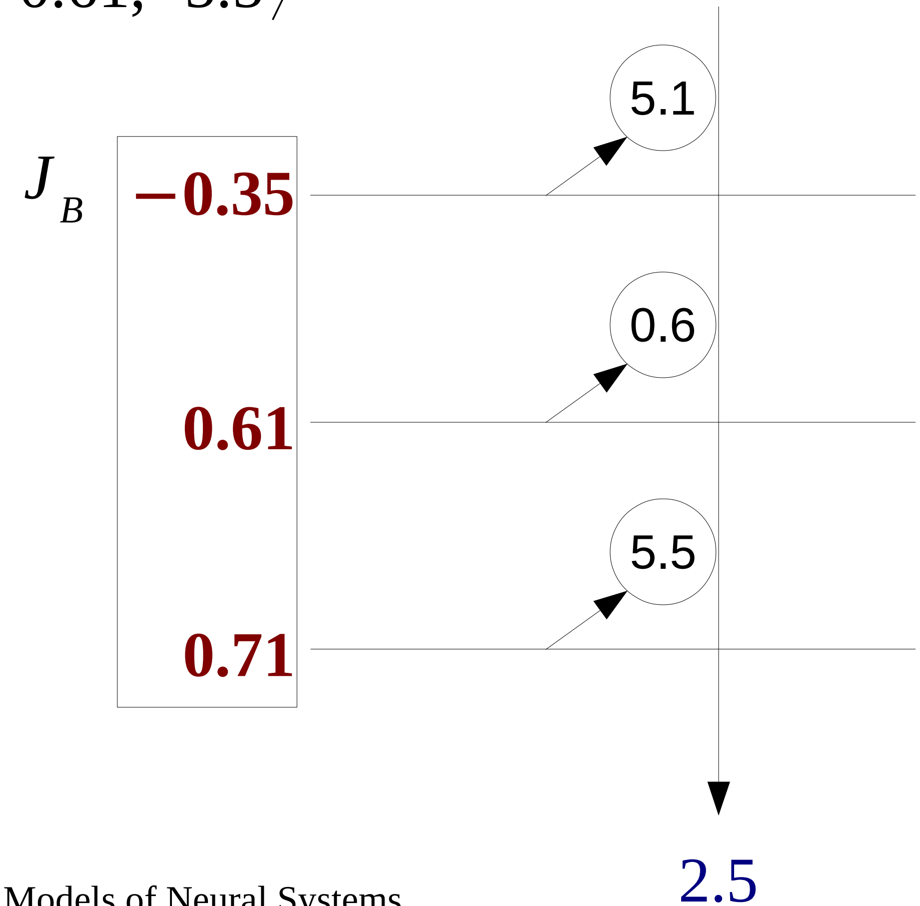
Setting the Weights

$m_A = 4.7 \qquad J_A = < \quad 0.87, \quad 0.49, \quad 0 \quad >$

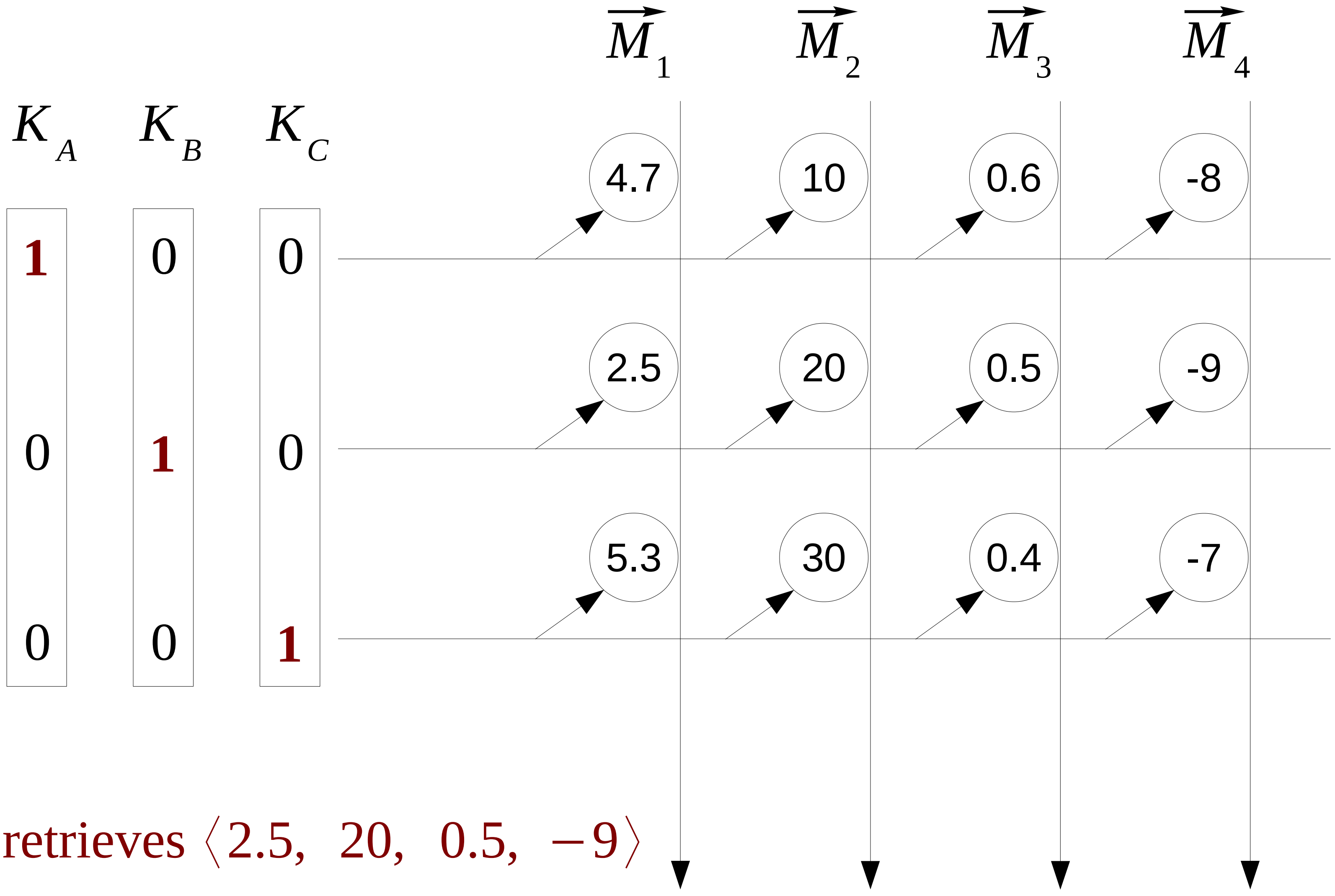
$m_B = 2.5 \qquad J_B = < \quad -0.35, \quad 0.61, \quad 0.71 \quad >$

$m_C = 5.3 \qquad J_C = < \quad 0.35, \quad -0.61, \quad 0.71 \quad >$

$$\vec{M} = \sum_k m_k \vec{J}_k = \langle 5.1,\ 0.61,\ 5.5 \rangle$$



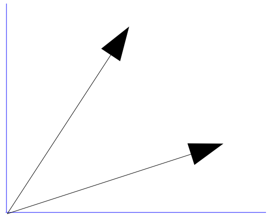
Storing Vectors: Each Stored Component Is A Separate Memory



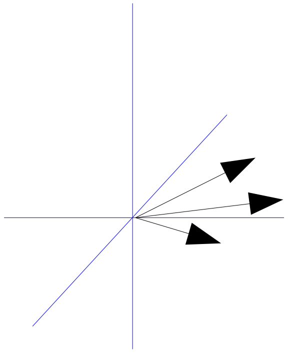
K_B retrieves $\langle 2.5,\ 20,\ 0.5,\ -9 \rangle$

Linear Independence

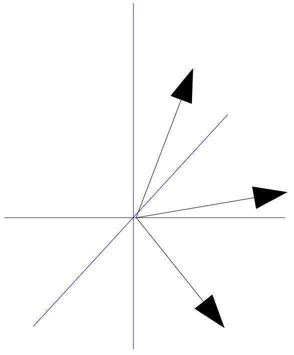
- A set of vectors is *linearly independent* if no element can be constructed as a linear combination of the others.
- In a system with *n* dimensions, there can be at most *n* linearly independent vectors.
- Any set of *n* linearly independent vectors constitutes a basis set for the space, from which any other vector can be constructed.



Linearly independent



Not linearly independent (all 3 vectors lie in the x-y plane)



Linearly independent

Linear Independence Is Enough

- Key vectors do not have to be orthogonal for an associative memory to work correctly.
- All that is required is linear independence.
- However, since $\vec{K}_A \cdot \vec{K}_B \neq 0$ we cannot set the weights as simply as we did previously.
- Matrix inversion is one solution:

$$\overline{K} = \langle \vec{K}_A,\ \vec{K}_B,\ \vec{K}_C \rangle$$

$$\vec{m} = \langle m_A,\ m_B,\ m_C \rangle$$

$$\vec{M} = \vec{m} \cdot \left(\overline{K}\right)^{-1}$$

- Another approach is an iterative algorithm: Widrow-Hoff.

The Widrow-Hoff Algorithm

1. Let initial weights $\vec{M}_0 = 0$.
2. Randomly choose a pair $m_i, \vec{K}_i$ from the training set.
3. Compute actual output value $a = \vec{M}_t \cdot \vec{K}_i$.
4. Measure the error: $e = m_i - a$.
5. Adjust the weights: $\vec{M}_{(t+1)} = \vec{M}_t + \eta \cdot e \cdot \vec{K}_i$
6. Return to step 2.

- Guaranteed to converge to a solution if the key vectors are linearly independent.
- This is the way simple, one layer neural nets are trained.
- Also called the LMS (Least Mean Squares) algorithm.
- Identical to the CMAC training algorithm (Albus).

High Dimensional Systems

- In typical uses of associative memories, the key vectors have many components (large # of dimensions).
- Computing matrix inverses is time consuming, so don't bother. Just assume orthogonality.
- If the vectors are <u>sparse</u>, they will be <u>nearly</u> orthogonal.
- How can we check?

$$\theta = \arccos \frac{\vec{v} \cdot \vec{w}}{\|\vec{v}\| \cdot \|\vec{w}\|}$$

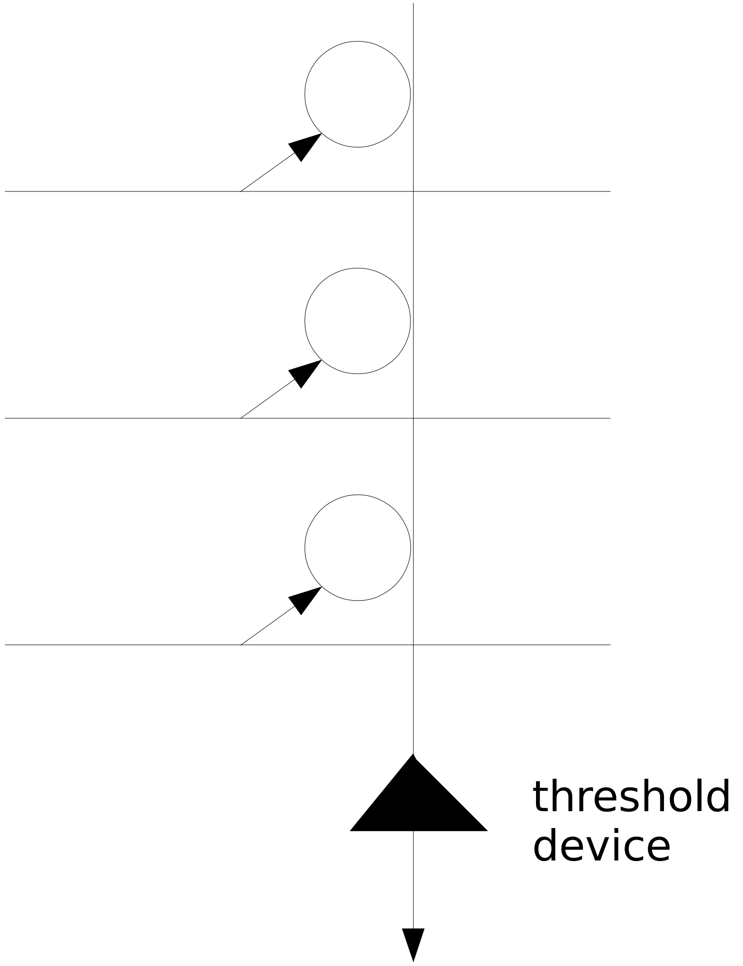
- Angle between <1,1,1, 1, 0,0,0, 0,0,0, 0,0,0>
  <0,0,0, 1, 1,1,1, 0,0,0, 0,00> is 76°.
- Because the keys aren't orthogonal, there will be interference resulting in “noise” in the memory.
  - Memory retrievals can produce a mixture of memories.

Eliminating Noise

- Noise occurs when:
  - Keys are linearly independent but not strictly orthogonal.
  - We're not using LMS to find optimal weights, but instead relying on the keys being *nearly* orthogonal.

- If we apply some constraints on the stored memory values, the noise can be reduced.
- Example: assume the stored values are binary: 0 or 1.
- With noise, a stored 1 value might be retrieved as 0.9 or 1.3. A stored 0 might come back as 0.1 or –0.2.
- Solution: use a binary output unit with a threshold of 0.5.

Thresholding for Noise Reduction



Partial Keys

- Suppose we use sparse, nearly orthogonal binary keys to store binary vectors:

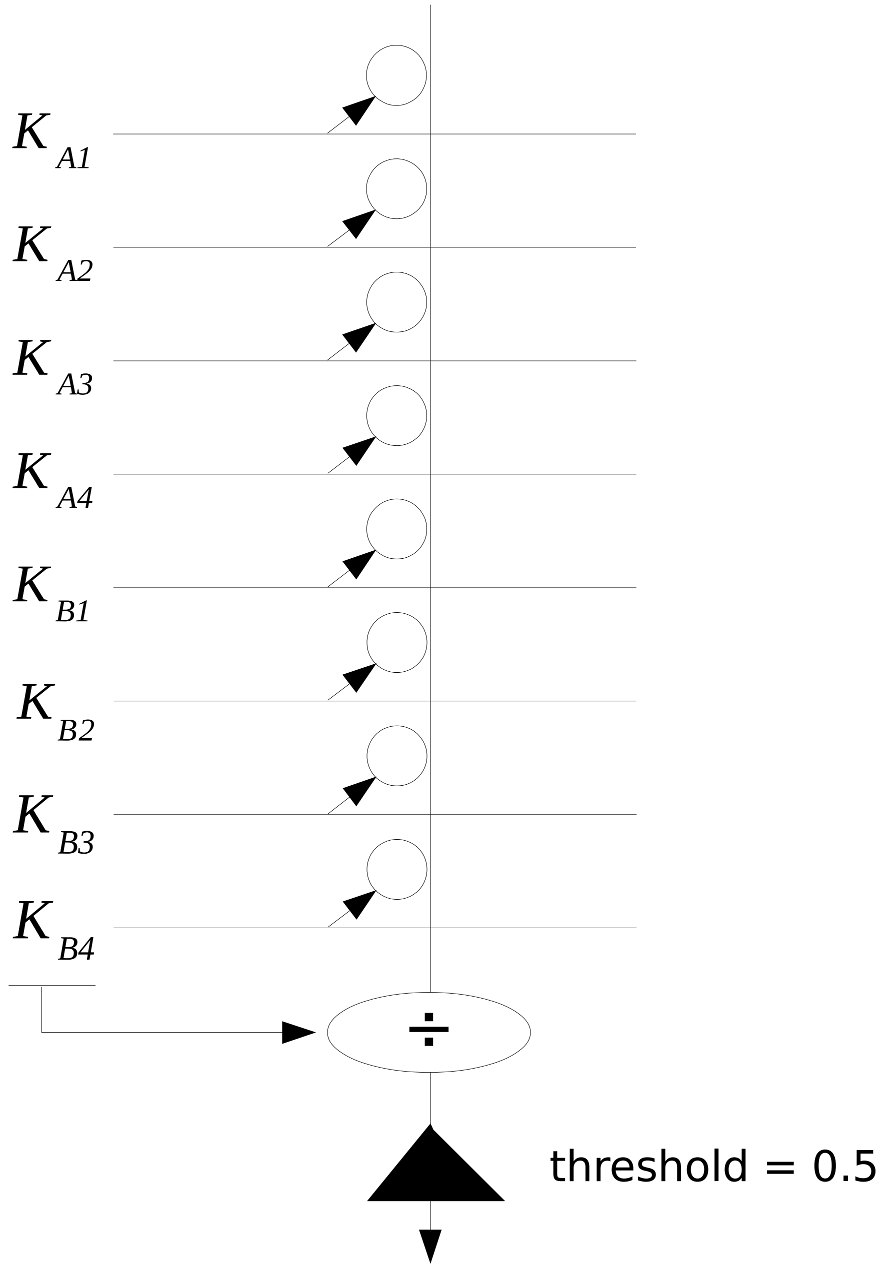
K_A = <1,1,1,1,0,0,0,0> K_B = <0,0,0,0,1,1,1,1>

- It should be possible to retrieve a pattern based on a partial key: <1,0,1,1,0,0,0,0>

- The threshold must be adjusted accordingly.

- Solution: *normalize* the input to the threshold unit by dividing by the length of the key provided.

Scaling for Partial Keys



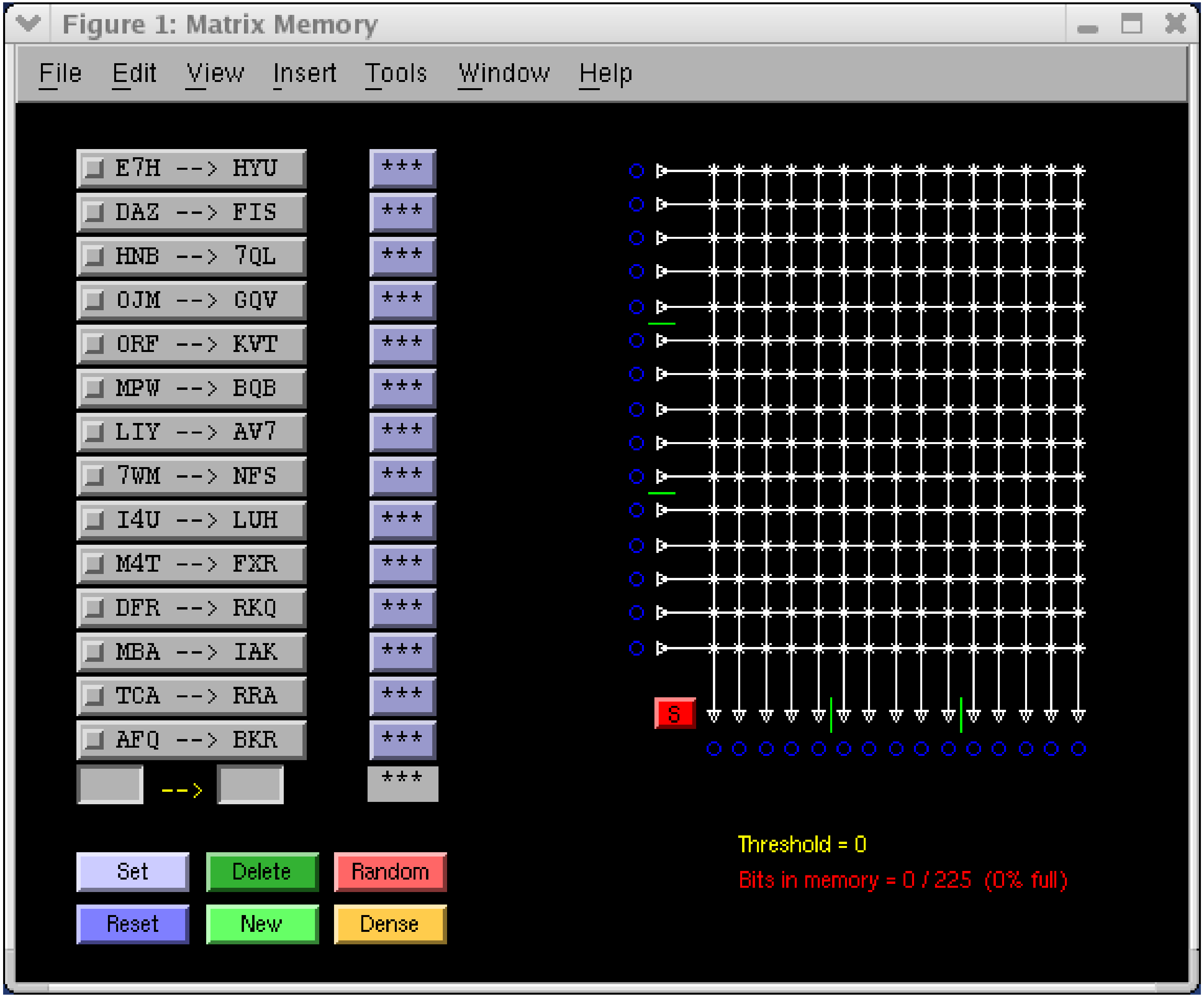
Warning About Binary Complements

- The binary complement of <1,0,0,0> is <0,1,1,1>. The binary complement of <0,1,0,0> is <1,0,1,1>.

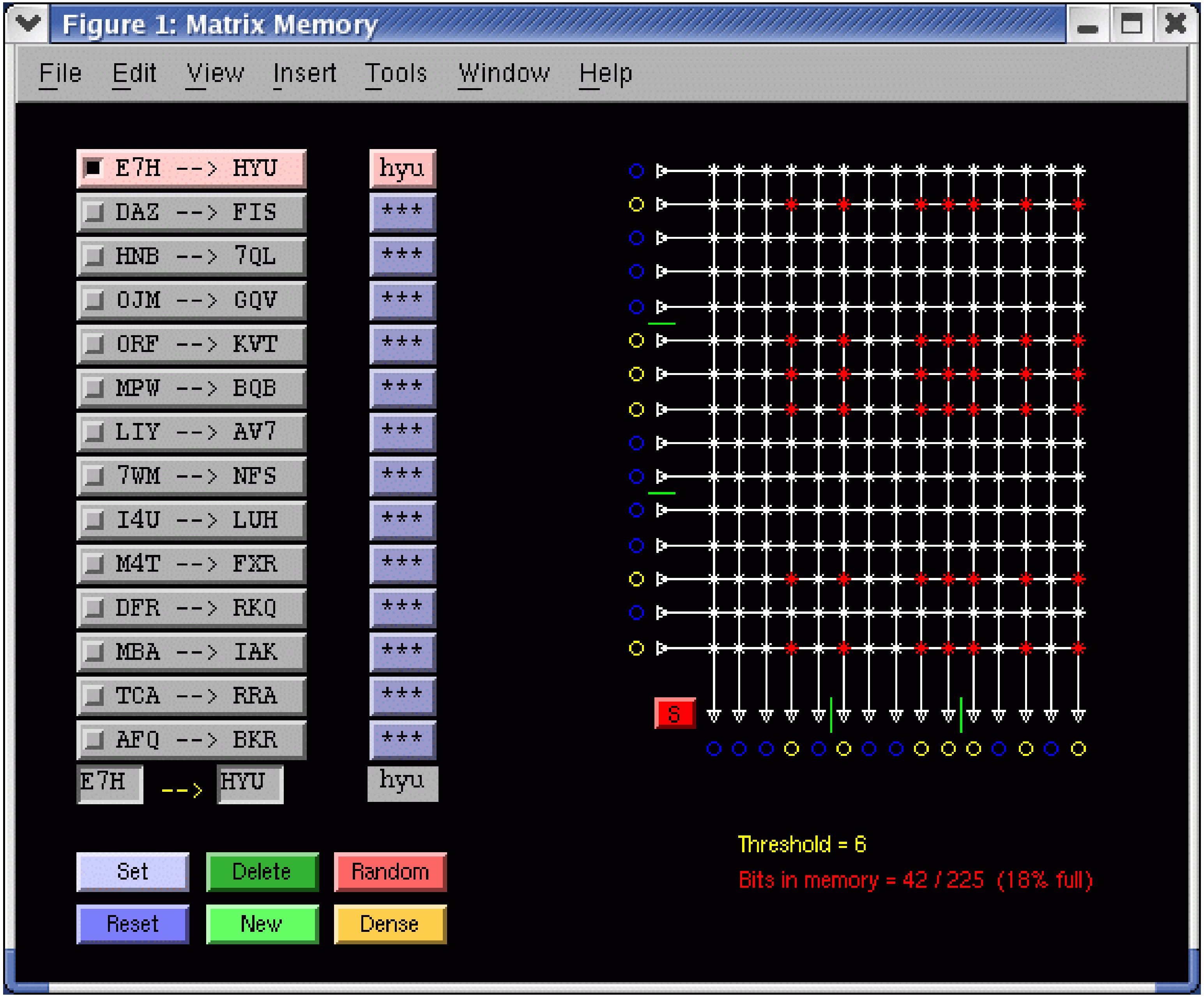
- In some respects, a bit string and its complement are equivalent, but this is not true for vector properties.

- If two binary vectors are orthogonal, their binary complements will not be:
  - Angle between <1,0,0,0> and <0,1,0,0> is 90°.
    - Angle between <0,1,1,1> and <1,0,1,1> is 48.2°.

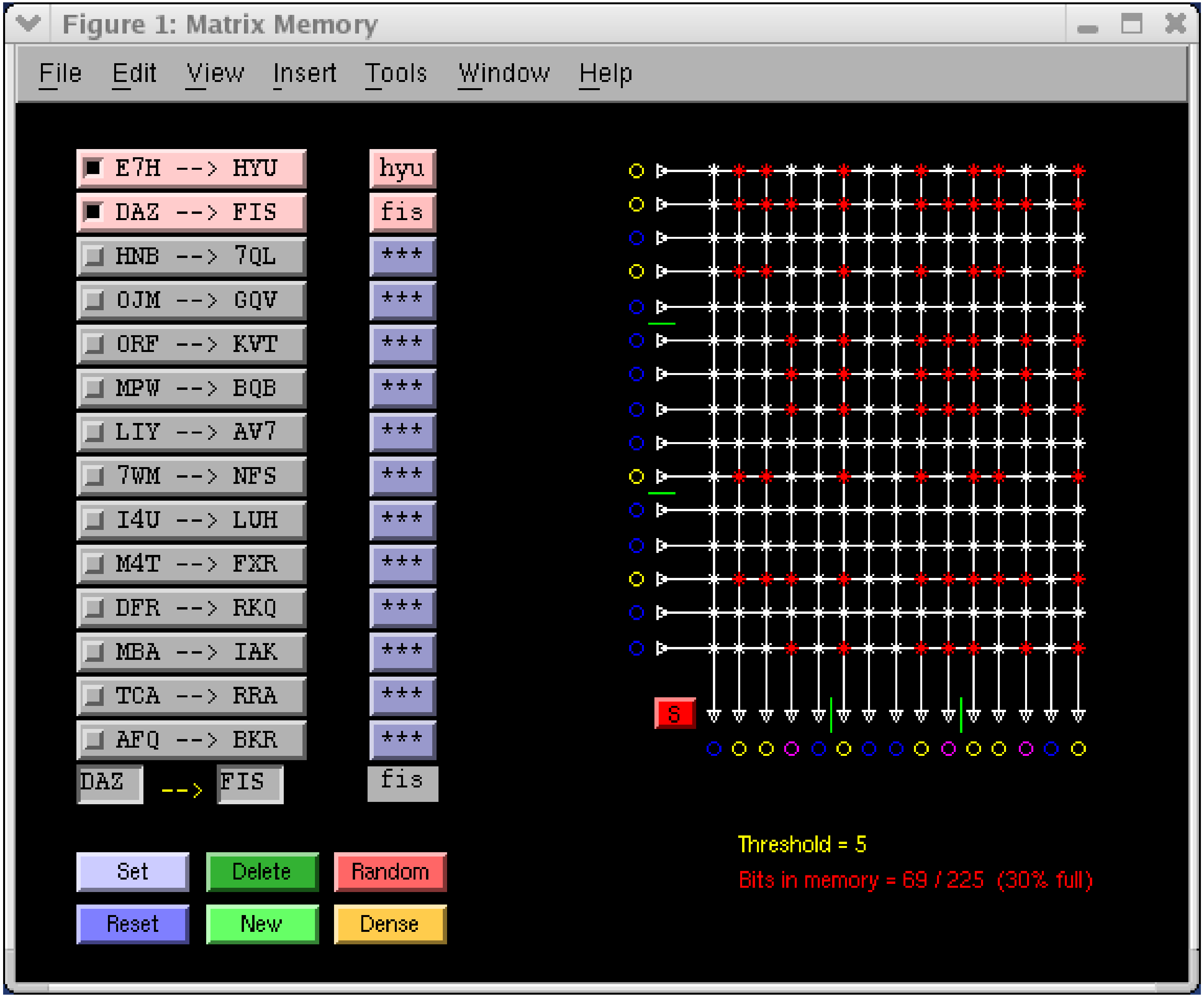
Matrix Memory Demo



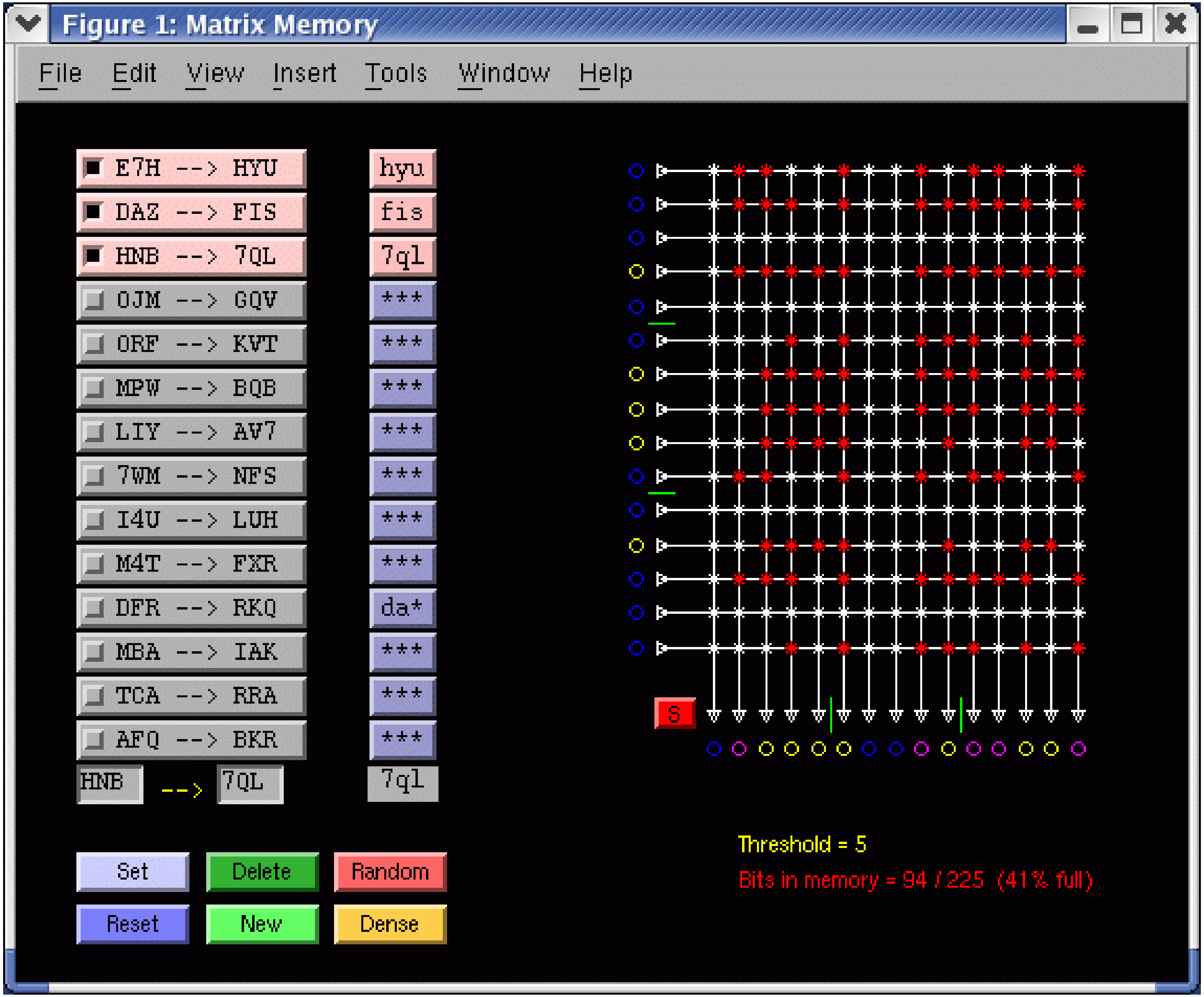
Matrix Memory Demo



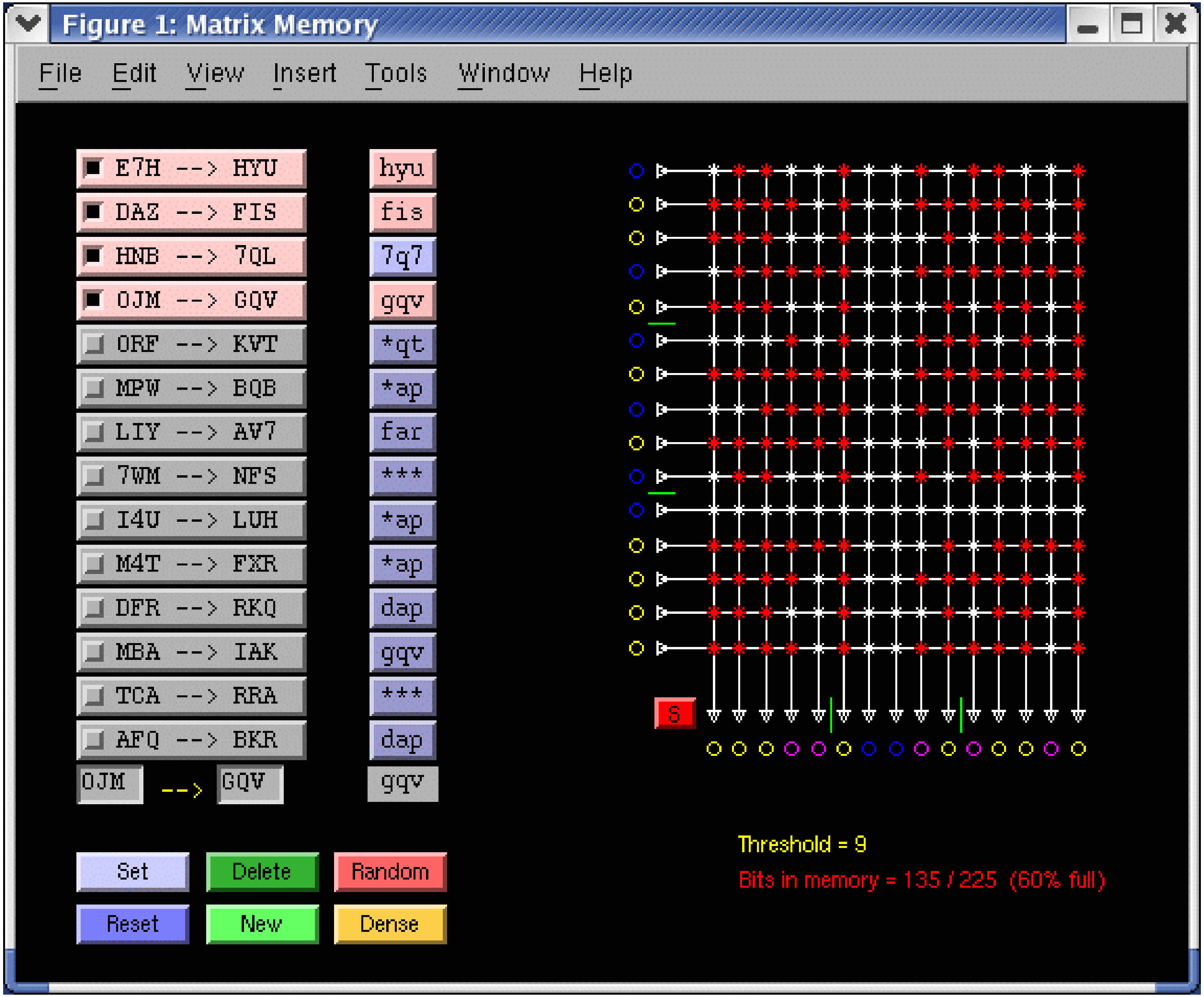
Matrix Memory Demo



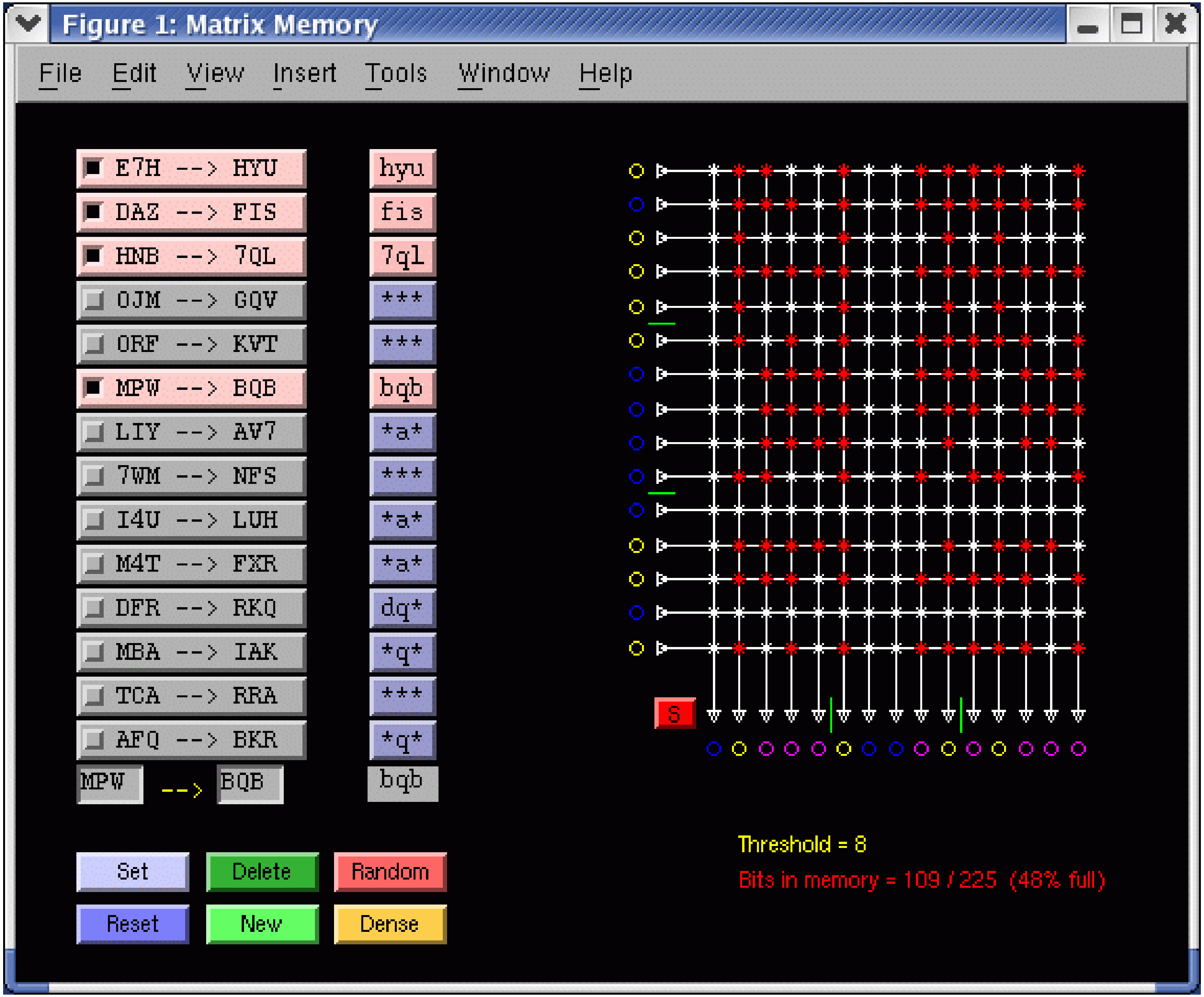
Matrix Memory Demo



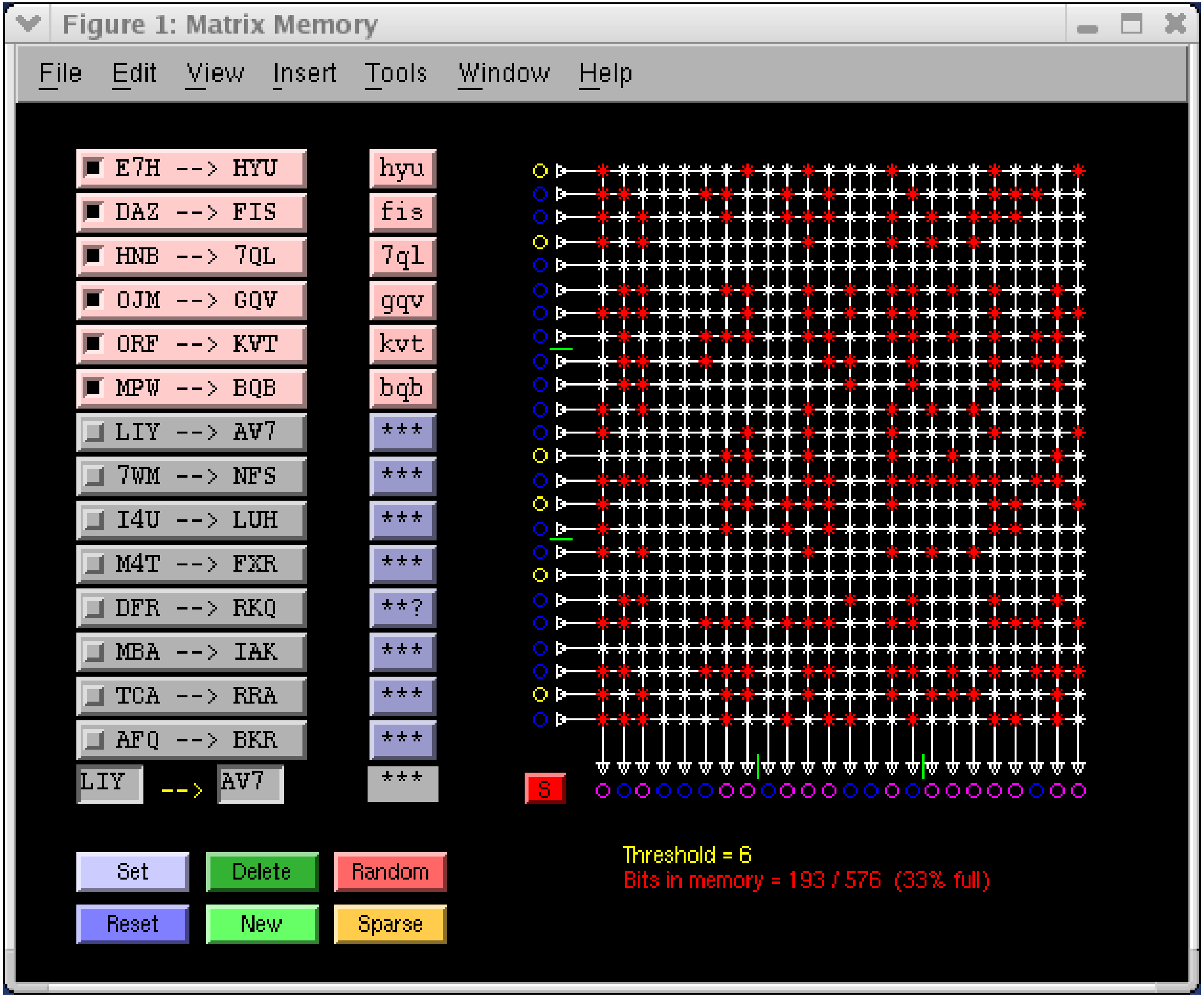
Matrix Memory Demo: Interference



Matrix Memory Demo

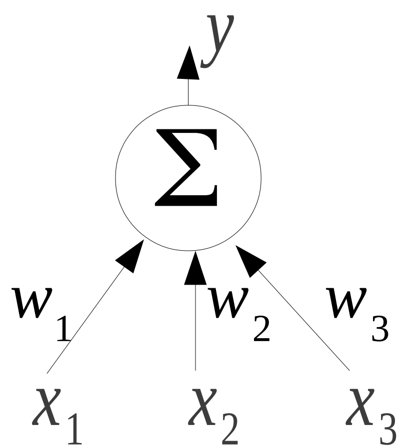


Matrix Memory Demo: Sparse Encoding



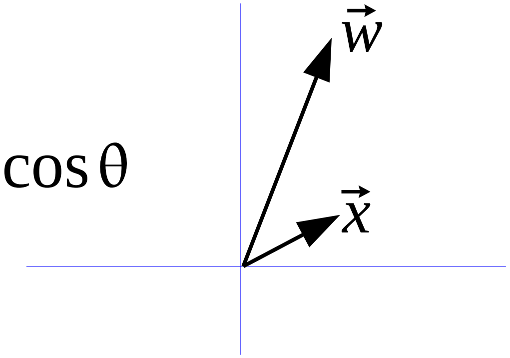
Dot Products and Neurons

- A neuron that linearly sums its inputs is computing a dot product of the input vector with the weight vector:



- The output **y** for a fixed magnitude input **x** will be largest when **x** is pointing in the same direction as the weight vector **w**.

$$y = \vec{x} \cdot \vec{w} = \|\vec{x}\| \; \|\vec{w}\| \; \cos\theta$$



Pattern Classification by Dot Product

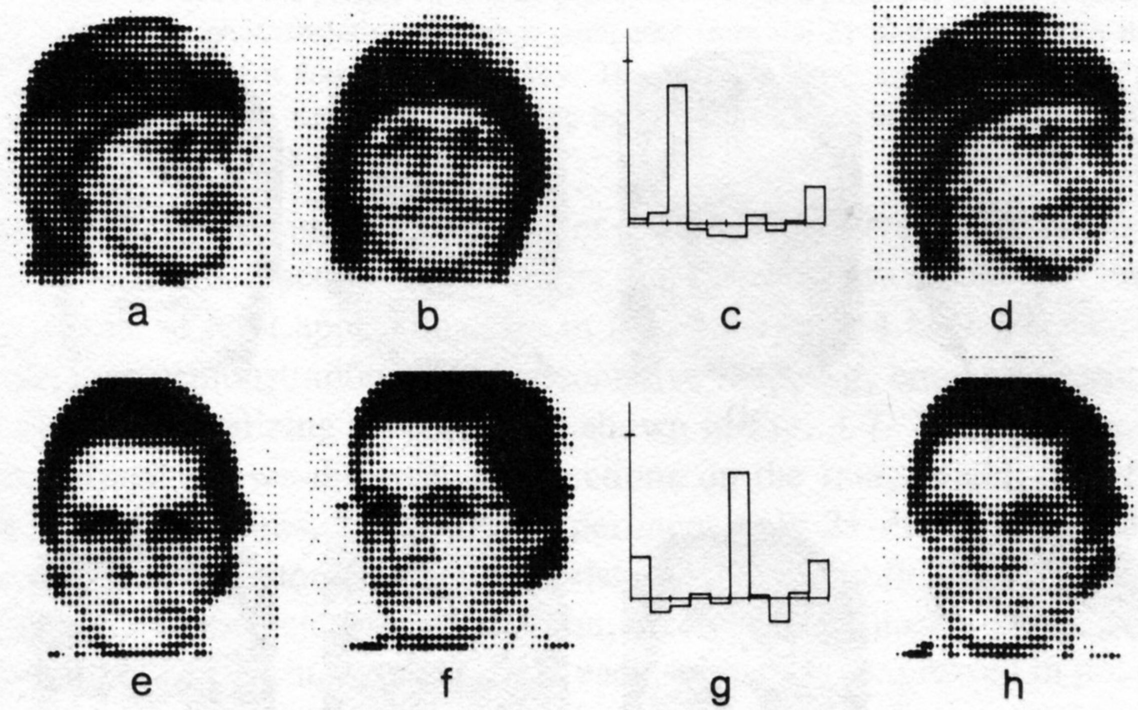
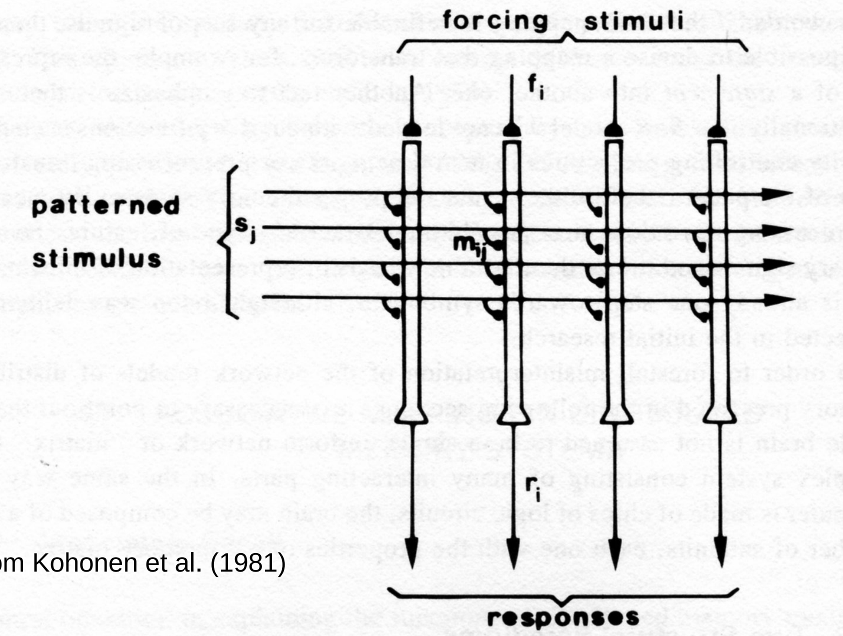


FIG. 4.5. Demonstration of classification by optimal associative mapping. Each of the 10 pattern classes employed consisted of pictures of one person photographed from five different angles, ranging from +45° to −45°. Image vectors with components consisting of discrete picture elements were used as pattern vectors; eight intensity levels were defined for each picture element. A distinct unit vector was associated with each person. Parts (a) and (b) show two prototypes from one pattern class (no. 3), and Parts (e) and (f) show two prototypes from another pattern class (no. 6). Part (d) shows a test image of the person in (a) and (b), taken from an angle not used among the prototypes. In the histogram of the recollection, (c), the position of the largest component correctly reveals the number of the class. Parts (g) and (h) repeat the same with another class.

From Kohonen et al. (1981)

Hetero-Associators

- Matrix memories are a simple example of associative memories.
- If the keys and stored memories are distinct, the architecture is called a *hetero-associator*.



From Kohonen et al. (1981)

FIG. 4.1. Associative network with a set of connected neurons shown schematically. s_j = elements of the stimulus pattern; r_i = elements of the response pattern; f_i = elements of the forcing stimulus pattern, which are absent during recall; m_{ij} = synaptic connections.

Hebbian Learning
Hetero-Associator

Auto-Associators

- If the keys and memories are identical, the architecture is called an *auto-associator*.
- Can retrieve a memory based on a noisy or incomplete fragment. The fragment serves as the "key".

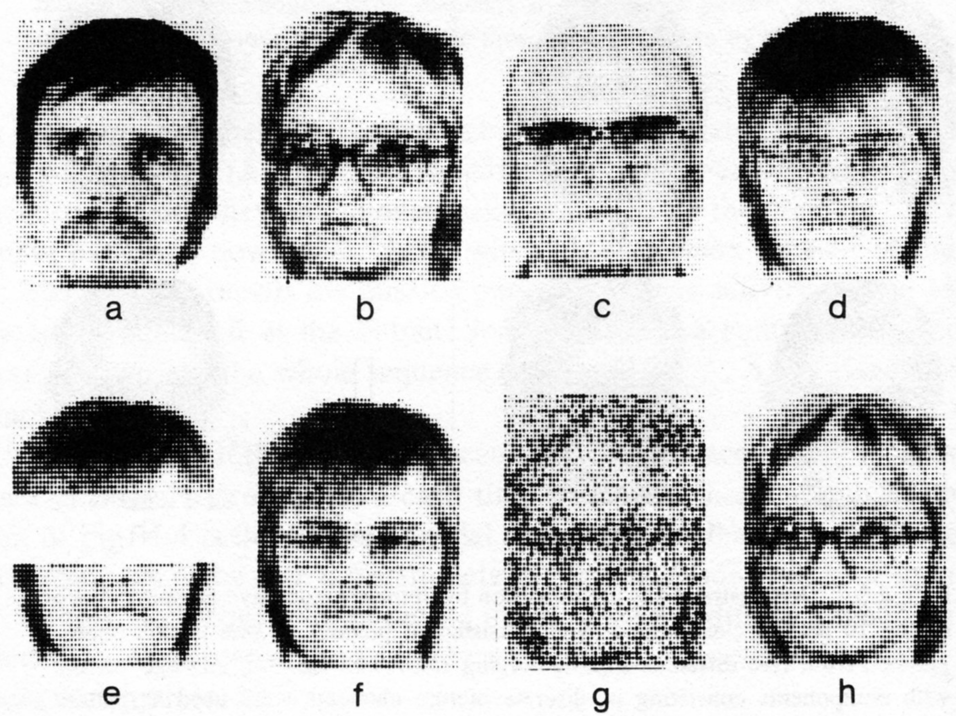
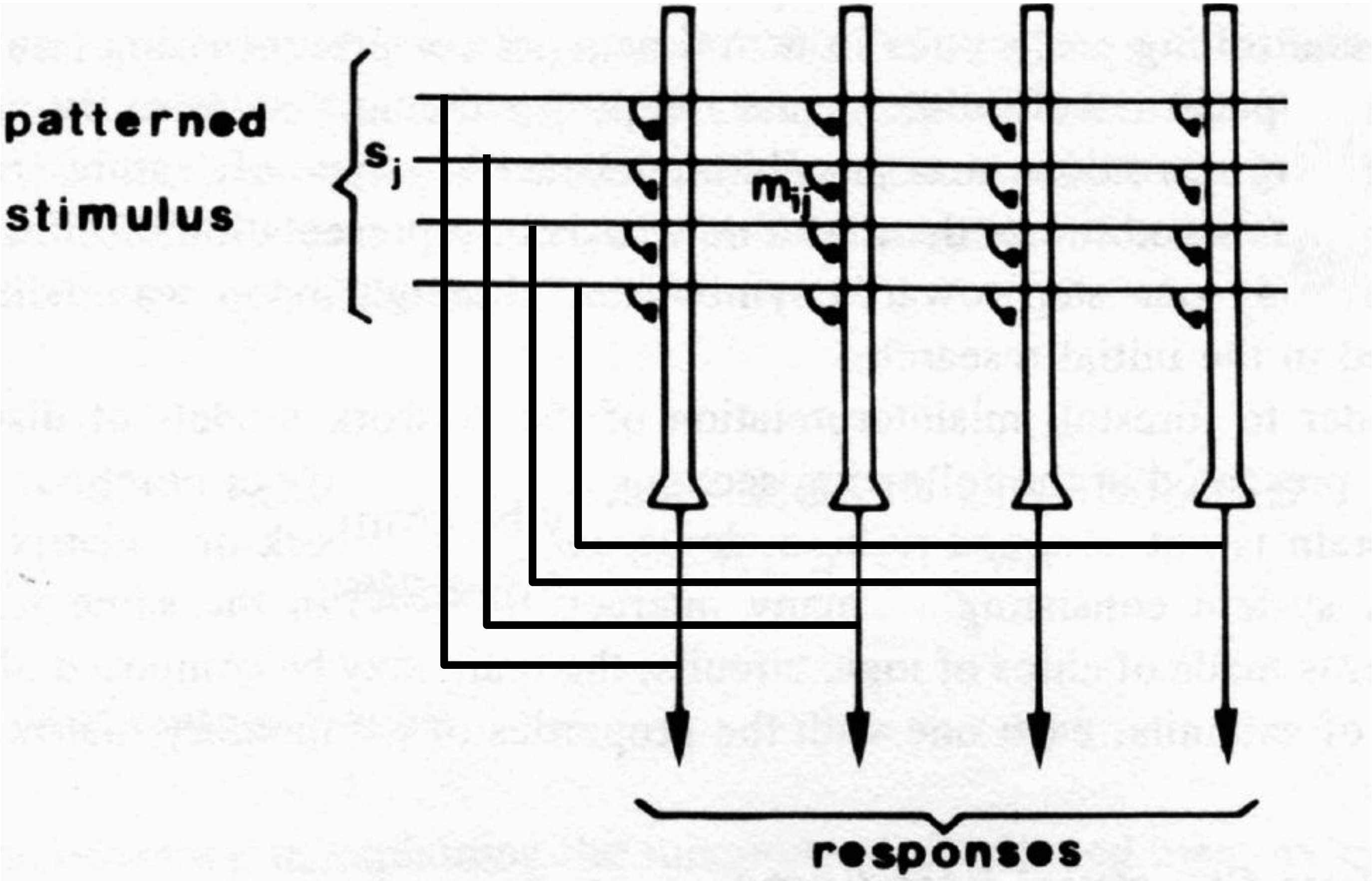


FIG. 4.6. Demonstration of autoassociative recall. Parts (a) through (d) show 4 of the 100 prototype images used to construct the autoassociative projector. When an incomplete or noisy version of a prototype, (e) and (g), respectively, served as the key pattern, the recollection resulting in the optimal autoassociative mapping is then shown to reconstruct the original appearance in (f) and (h), respectively.

From Kohonen et al. (1981)

Feedback in Auto-Associators

- Supply an initial noisy or partial key K_0.
- Result is a memory K_1 which can be used as a better key.
- Use K_1 to retrieve K_2, etc. A handful of cycles suffices.



Matrix and Vector Transpose

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}^T = \begin{bmatrix} a & d & g \\ b & e & h \\ c & f & i \end{bmatrix}$$

$$\vec{u} = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} \qquad \vec{u}^T = \begin{bmatrix} u_1 & u_2 & u_3 \end{bmatrix}$$

column vector

row vector

A Matrix is a Collection of Vectors

One way to view the matrix

$$\begin{bmatrix} u_1 & v_1 & w_1 \\ u_2 & v_2 & w_2 \\ u_3 & v_3 & w_3 \end{bmatrix}$$

is as a collection of three column vectors:

$$\begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} \quad \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} \quad \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix}$$

In other words, a row matrix of column vectors:

$$\begin{bmatrix} \vec{u} & \vec{v} & \vec{w} \end{bmatrix}$$

For many operations on vectors, there are equivalent operations on matrices that treat the matrix as a set of vectors.

Inner vs. Outer Product

Column vector $\vec{u}$ is $N \times 1$

Inner product: $(1 \times N) \times (N \times 1) \rightarrow 1 \times 1$

$$(\vec{u}^T)\vec{u} = u_1 \cdot u_1 + \ldots u_N \cdot u_N = \|\vec{u}\|^2$$

Outer product: $(N \times 1) \times (1 \times N) \rightarrow N \times N$

$$\vec{u}(\vec{u}^T) = \begin{bmatrix} u_1 u_1 & u_1 u_2 & u_1 u_3 \\ u_2 u_1 & u_2 u_2 & u_2 u_3 \\ u_3 u_1 & u_3 u_2 & u_3 u_3 \end{bmatrix} = \begin{bmatrix} u_1 \vec{u} & u_2 \vec{u} & u_3 \vec{u} \end{bmatrix}$$

Weights for an Auto-Associator

- How can we derive the auto-associator's weight matrix?
  - Assume the patterns are orthogonal
  - For each pattern, compute the outer product of the pattern with itself, giving a matrix.
  - Add up all these outer products to find the weight matrix.

$$\overline{M} = \sum_{\vec{p}} \vec{p}\,(\vec{p}^{\,T})$$

- Note: at most *n* patterns can be stored in such a memory, where *n* is the number of rows or columns in the weight matrix.
- Note: the input patterns are not unit vectors (see next slide), but we can compensate for that by using the division trick.

Weight Matrix by Outer Product

Let $\vec{u}, \vec{v}, \vec{w}$ be an orthonormal set.

Let $\overline{M} = \vec{u}(\vec{u}^T) + \vec{v}(\vec{v}^T) + \vec{w}(\vec{w}^T)$

$$\overline{M} = \left[u_1\vec{u}+v_1\vec{v}+w_1\vec{w} \quad u_2\vec{u}+v_2\vec{v}+w_2\vec{w} \quad u_3\vec{u}+v_3\vec{v}+w_3\vec{w} \right]$$

Therefore:

$$\begin{aligned} \overline{M}\vec{u} &= [\quad (u_1\vec{u})\cdot\vec{u} \quad (u_2\vec{u})\cdot\vec{u} \quad (u_3\vec{u})\cdot\vec{u} \quad] \\ &= [\quad u_1(\vec{u}\cdot\vec{u}) \quad u_2(\vec{u}\cdot\vec{u}) \quad u_3(\vec{u}\cdot\vec{u}) \quad] \\ &= [\quad u_1 \quad u_2 \quad u_3 \quad] \\ &= \vec{u} \end{aligned}$$

For orthogonal unit vectors, the outer product of the vector with itself is exactly the vector's contribution to the weight matrix.

Eigenvectors

Let $\overline{M}$ be any square matrix.

Then there exist unit vectors $\vec{u}$ such that $\overline{M}\vec{u} = \lambda\vec{u}$.

Each $\vec{u}$ is called an eigenvector of the matrix.

The corresponding λ is called an eigenvalue.

- We can think of any matrix as an auto-associative memory. The “keys” are the eigenvectors.
- Retrieval is by matrix-vector multiplication.
- The eigenvectors are the directions along which, for a unit vector input, the memory will produce the locally largest output.
- The eigenvalues indicate how much a key is “stretched” by multiplication by the matrix.

Other Ways to To Get Pattern Cleanup

- Recurrent connections are not required. Another approach is to cascade several associative memories.

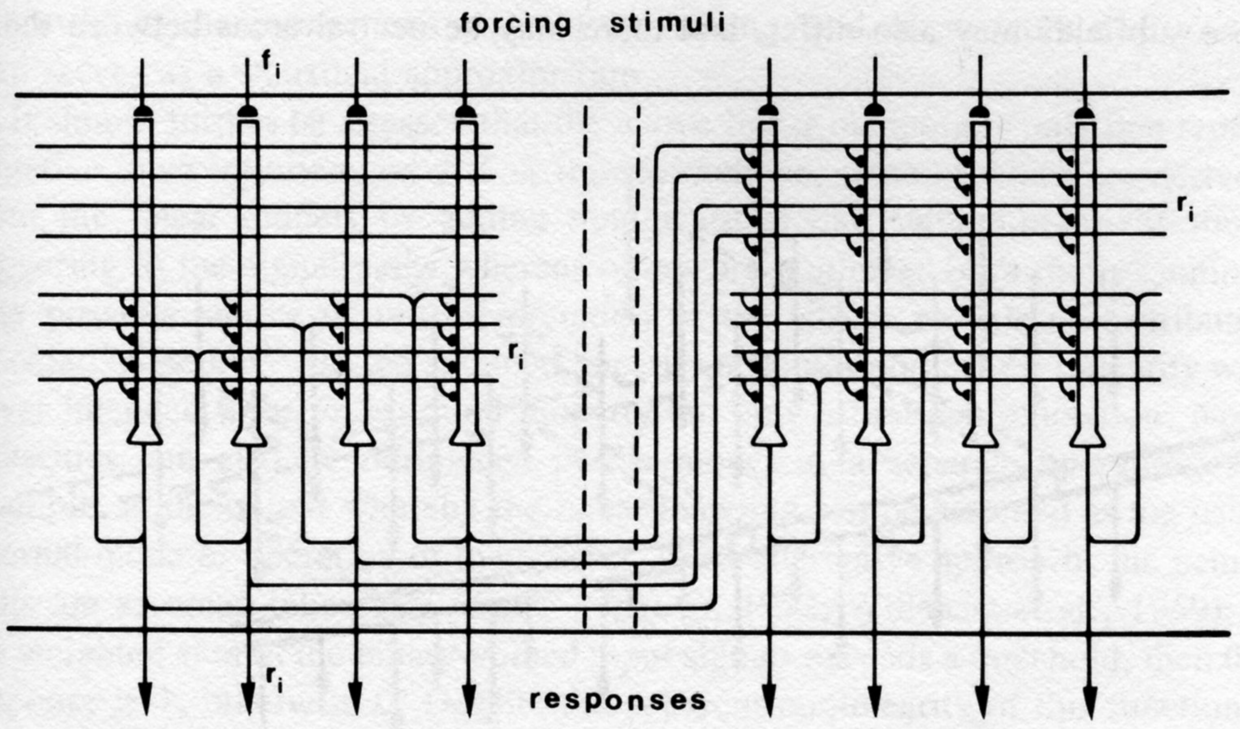


FIG. 4.2. The modular associative network with recurrent feedback. f_i = elements of the forcing or afferent stimulus pattern; r_i = elements of the response pattern that are fed back into the network at shorter and longer distances. The vertical dotted lines separate the two subsystems shown, and the horizontal solid lines represent the surfaces of the laminar network.

Retrieving Sequences

- Associative memories can be taught to produce sequences by feeding part of the output back to the input.

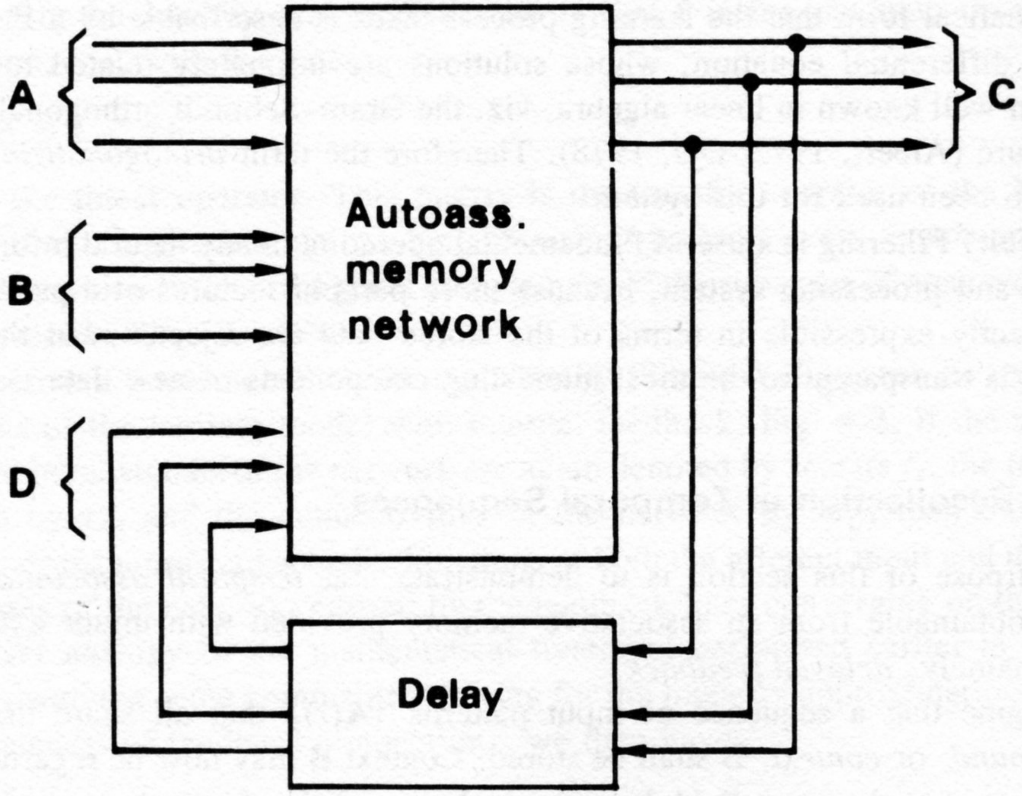


FIG. 4.4. A system for the associative recall of sequences. **A** = forcing input vector; **B** = constant background or context pattern; **C** = response pattern, the recollection from autoassociative memory; **D** = feedback pattern, equal to the response at a previous instant, with the time difference given by the delay.

Summary

- Orthogonal keys yield perfect memories via a simple outer product rule.
- Linearly independent keys yield perfect memories if matrix inverse or the Widrow-Hoff (LMS) algorithm is used to derive the weights.
- Sparse patterns in a high dimensional space are nearly orthogonal, and should produce little interference even using the simple outer product rule.
- Sparse patterns also seem more biologically plausible.