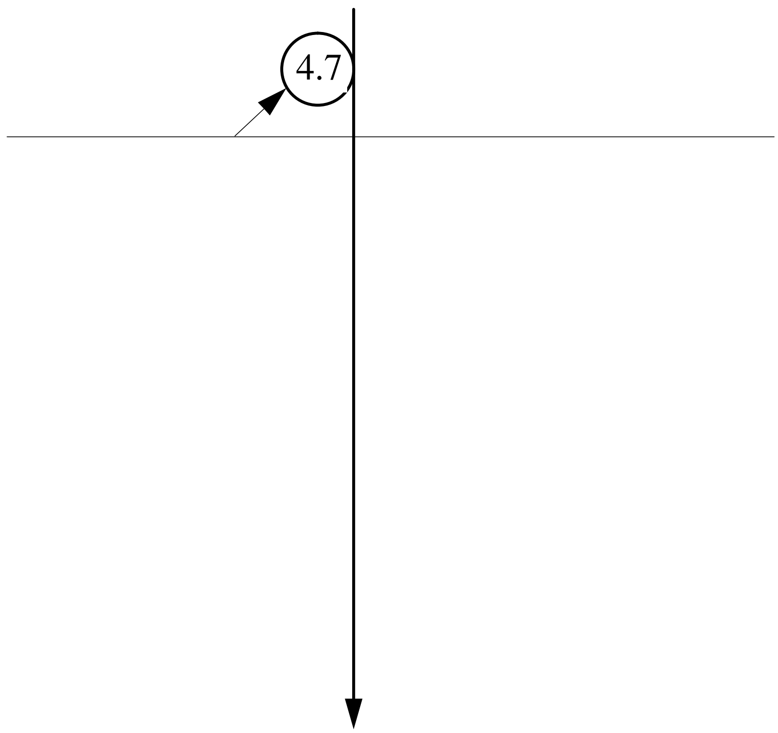
Vectors, Matrices, and Associative Memory

David S. Touretzky

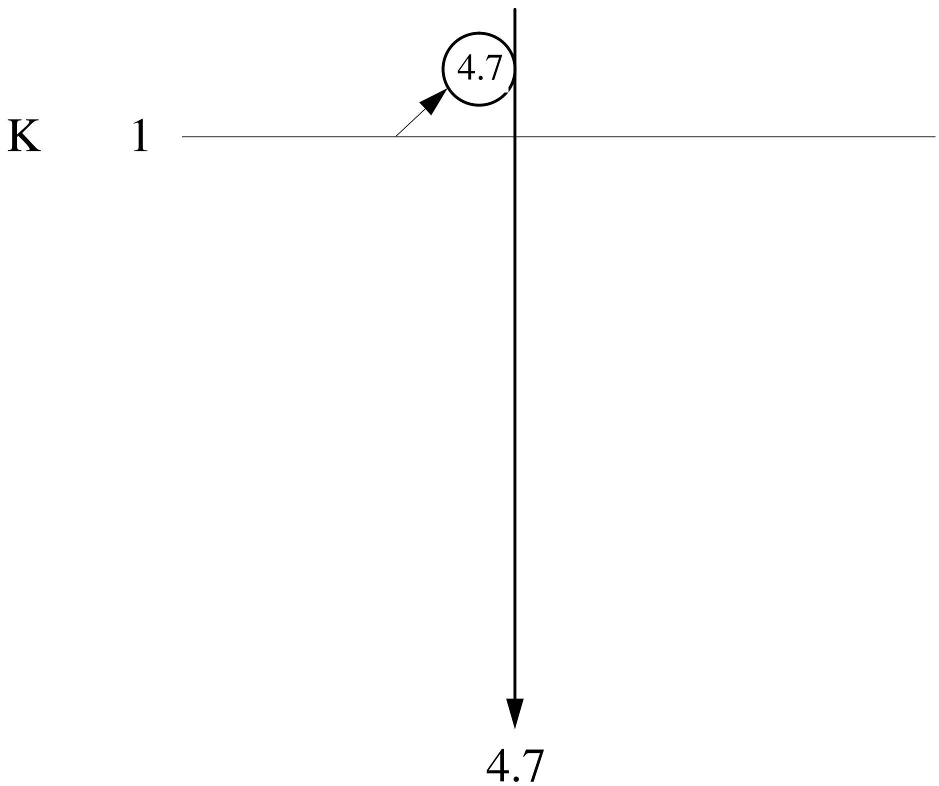
David S. Touretzky, Carnegie Mellon University

A Simple Memory

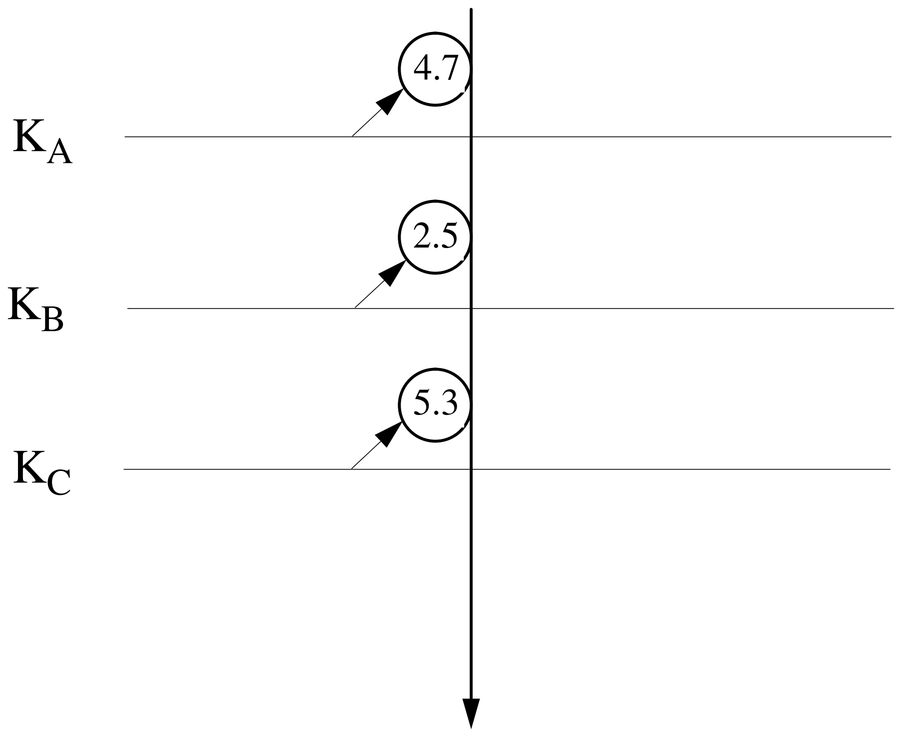
K



A Simple Memory

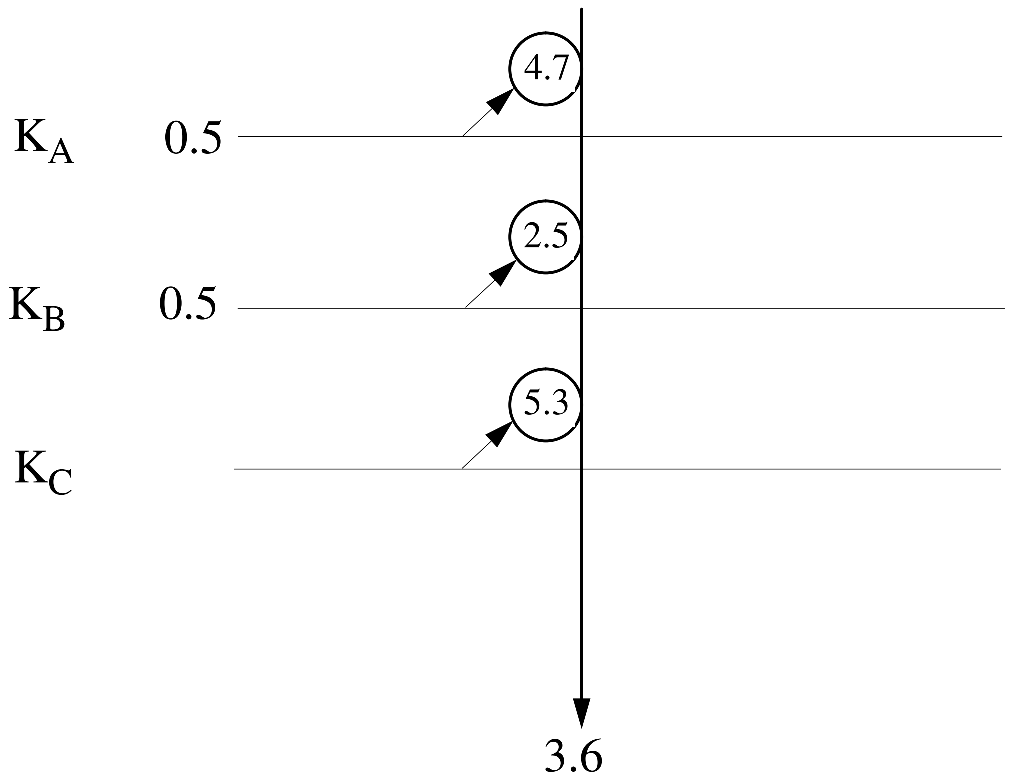


A Simple Memory



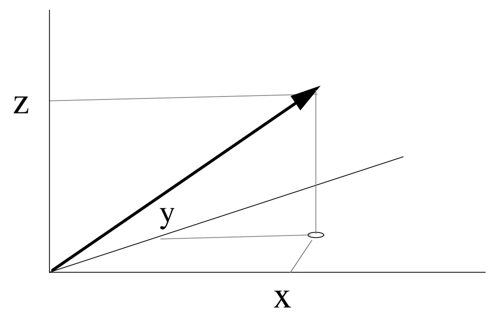
Each input line activates a particular memory.

A Simple Memory



Mixtures of memories are also possible.

Memories As Vectors



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

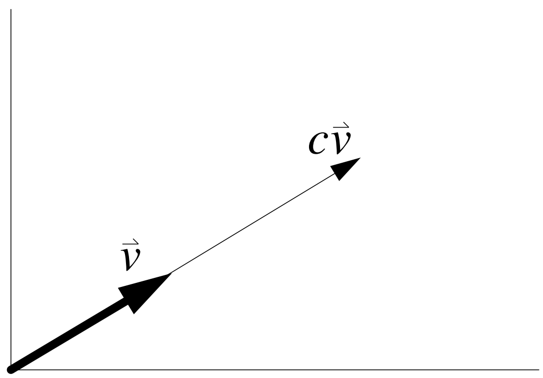
Basis unit vectors:

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

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

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

Length of a Vector

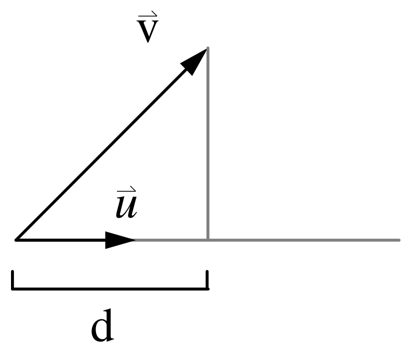


Let $\|\vec{v}\|$ = length of $\vec{v}$.

Then $\|c\vec{v}\| = c \cdot \|\vec{v}\|$.

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

Dot Product: Geometric Definition



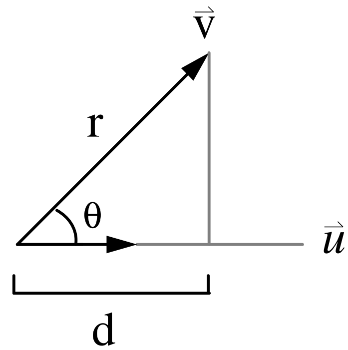
Let $\vec{v}$ be a vector and $\vec{u}$ a unit vector.

Two axioms for dot product:

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

$$c\vec{v}_1 \bullet \vec{v}_2 = c(\vec{v}_1 \bullet \vec{v}_2) = \vec{v}_1 \bullet c\vec{v}_2$$

Dot Product: Geometric Definition

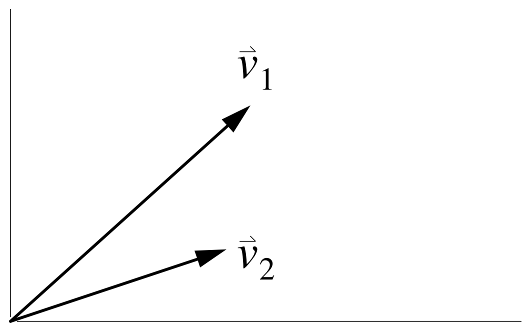


$$d = r \cdot \cos\theta$$

$$r = \|\vec{v}\|$$

$$\vec{v} \bullet \vec{u} = \|\vec{v}\| \cdot \cos\theta$$

Dot Product of Two Vectors



$$\vec{v}_1 \bullet \vec{v}_2 = \left\| \vec{v}_1 \right\| \cdot \left\| \vec{v}_2 \right\| \cdot \cos\theta$$

Proof:

Dot Product: Algebraic Definition

$$\vec{v} = \langle v_1, v_2 \rangle$$

$$\vec{w} = \langle w_1, w_2 \rangle$$

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

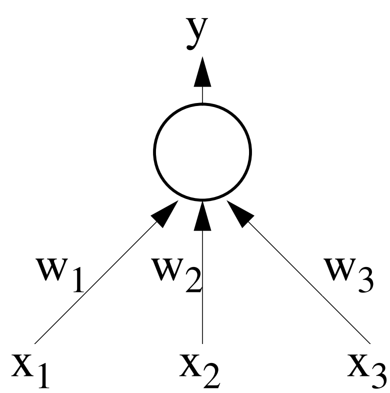
But also:

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

How can we relate these two definitions?

Dot Products And Neurons

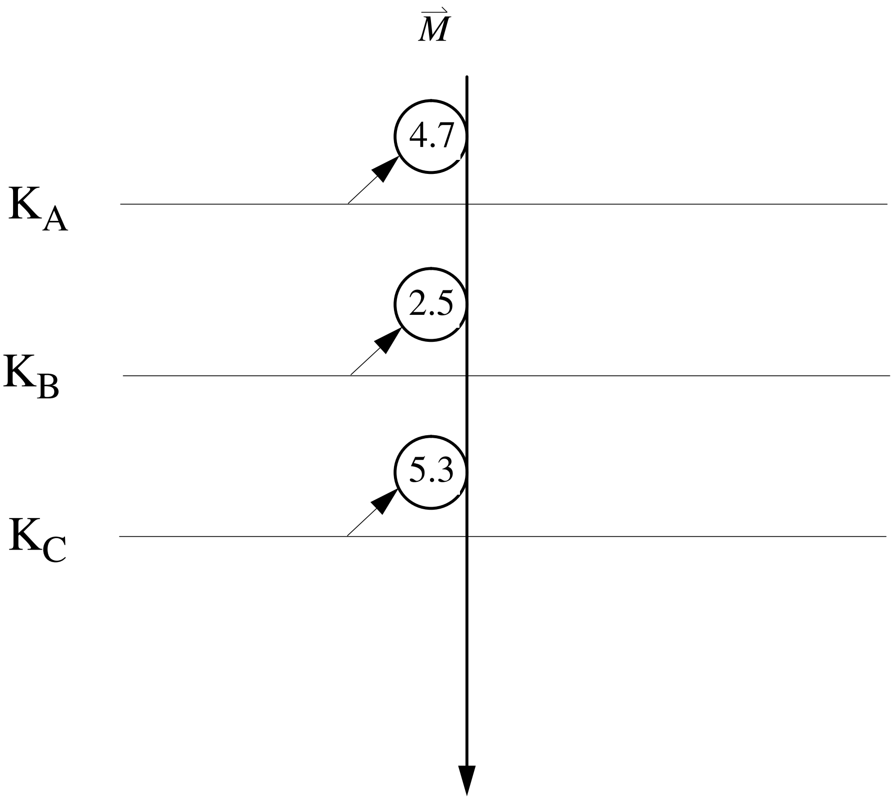
- A neuron that linearly sums its inputs is computing a dot product.



$$y = \vec{x} \bullet \vec{w}$$

- The output y will be largest when the input vector is in the same direction as the weight vector.

Associative Retrieval as Dot Product



Retrieving memory K_A is equivalent to computing $\vec{K}_A \bullet \vec{M}$.

This works for mixtures of memories as well.

Orthogonal Keys

The key vectors are mutually orthogonal.

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

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

$$\theta = \arccos 0 = 90^o$$

We don't have to use vectors of form <...0,1,0...>.

Any set of mutually orthogonal unit vectors will do.

Example: rotate K_A, K_B, K_C by 45 degrees about the x axis, then 30 degrees about the z axis.

$$J_A = \langle\ 0.87,\ \ 0.49,\ \ 0\ \rangle$$
$$J_B = \langle -0.35,\ \ 0.61,\ \ 0.71 \rangle$$
$$J_C = \langle\ 0.35, -0.61,\ \ 0.71 \rangle$$

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

$$J_A \bullet J_B = (0.87) \bullet (-0.35) + (0.49) \bullet (.61) + 0 \bullet (0.71) = 0$$

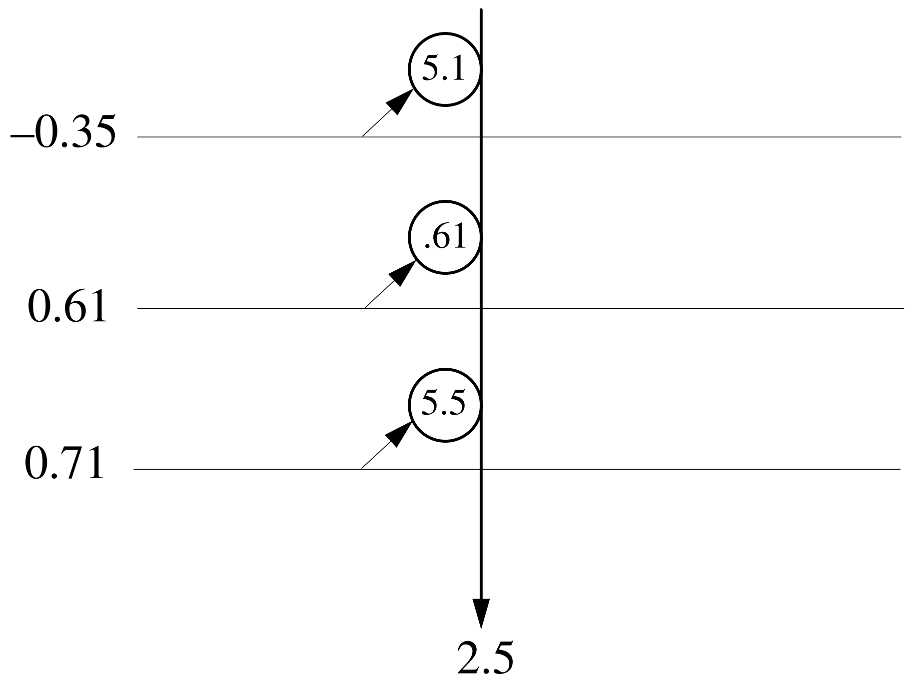
Setting the Weights

$$W_i = \left(\sum_j M_j \cdot \vec{K}_j\right)_i$$

$$M_1 = 4.7 \qquad \vec{K}_1 = \langle 0.87, 0.49, 0 \rangle$$

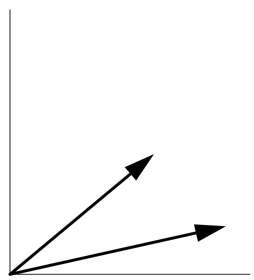
$$M_2 = 2.5 \qquad \vec{K}_2 = \langle -0.35, 0.61, 0.71 \rangle$$

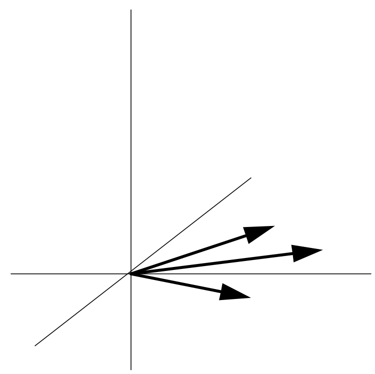
$$M_3 = 5.3 \qquad \vec{K}_3 = \langle 0.35, -0.61, 0.71 \rangle$$



Linear Independence

- 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 constituted.





Linear Independence is Enough

- The key vectors do not have to be orthogonal for the associative memory to work correctly.
- All that is required is linear independence.
- However, since $K_A \bullet K_B$ is not 0, we cannot set the weights as simply as above. Matrix inversion is one solution.

$$\underline{K} = \langle \vec{K}_1, \vec{K}_2, \vec{K}_3 \rangle$$

$$\vec{M} = \langle M_1, M_2, M_3 \rangle$$

$$\vec{W} = \underline{K}^{-1} \cdot \vec{M}$$

- There is also an iterative algorithm (the Widrow-Hoff or LMS algorithm) for incrementally adjusting a weight matrix.

The LMS Algorithm

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

- Guaranteed to converge to a solution if the key vectors are linearly independent.
- This is the way perceptrons are trained.
- Why can't it solve XOR?
- Keys for XOR problem aren't linearly independent!

0	0	0
0	1	1
1	0	1
1	1	0

High Dimensional Systems

- In typical uses of associative memories, the key vectors have a large number of dimensions.
- Computing matrix inverses is time consuming, so don't bother.
- If the vectors are sparse, they will be nearly orthogonal.
- How can we check?

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

- 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, 0,0,0> is 76 degrees.
- Because the keys are not orthogonal, there will be "noise" in the memory.
- The result of a memory retrieval will be a mixture of two stored memories.

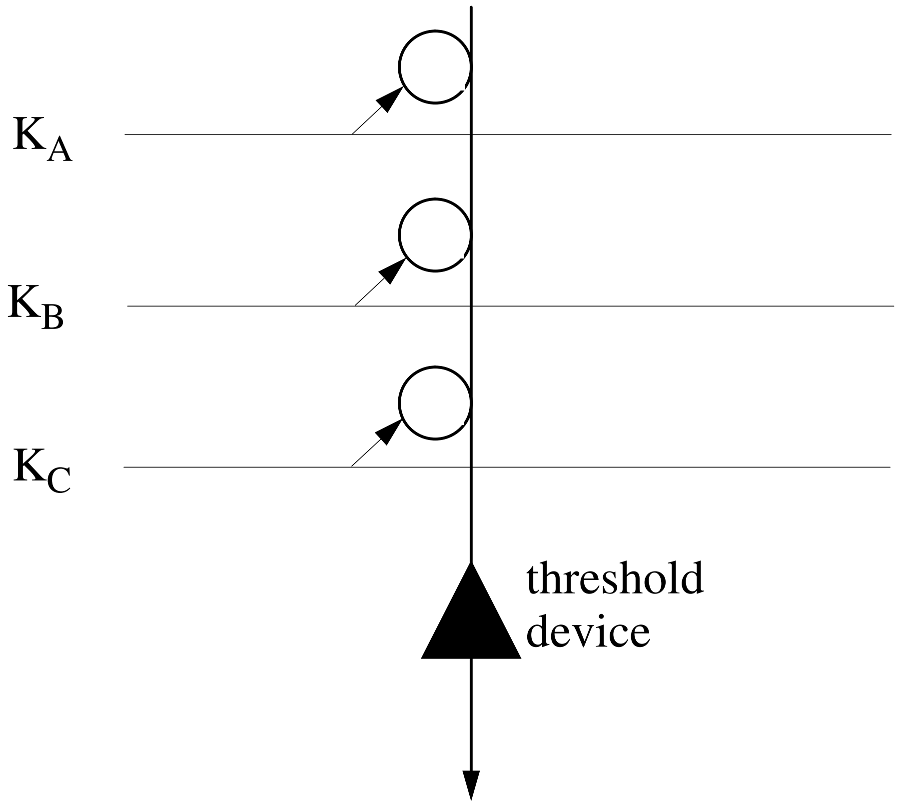
Binary Complements

- 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 ways the two vectors are equivalent.
- The equivalence does not hold in matrix associators.
- Angle between <1,0,0,0> and <0,1,0,0> = 90^{o}.
- Angle between <0,1,1,1> and <1,0,1,1> = 48.2^{o}.

Eliminating Noise

- In general, we cannot eliminate noise in the memory.
- However, if we apply some constraints on the stored memory values, then noise can be reduced.
- Example: assume that 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 value might be retrieved as 0.1 or –0.2.
- Use a nonlinear unit with a threshold of 0.5.

Thresholding for Noise Reduction



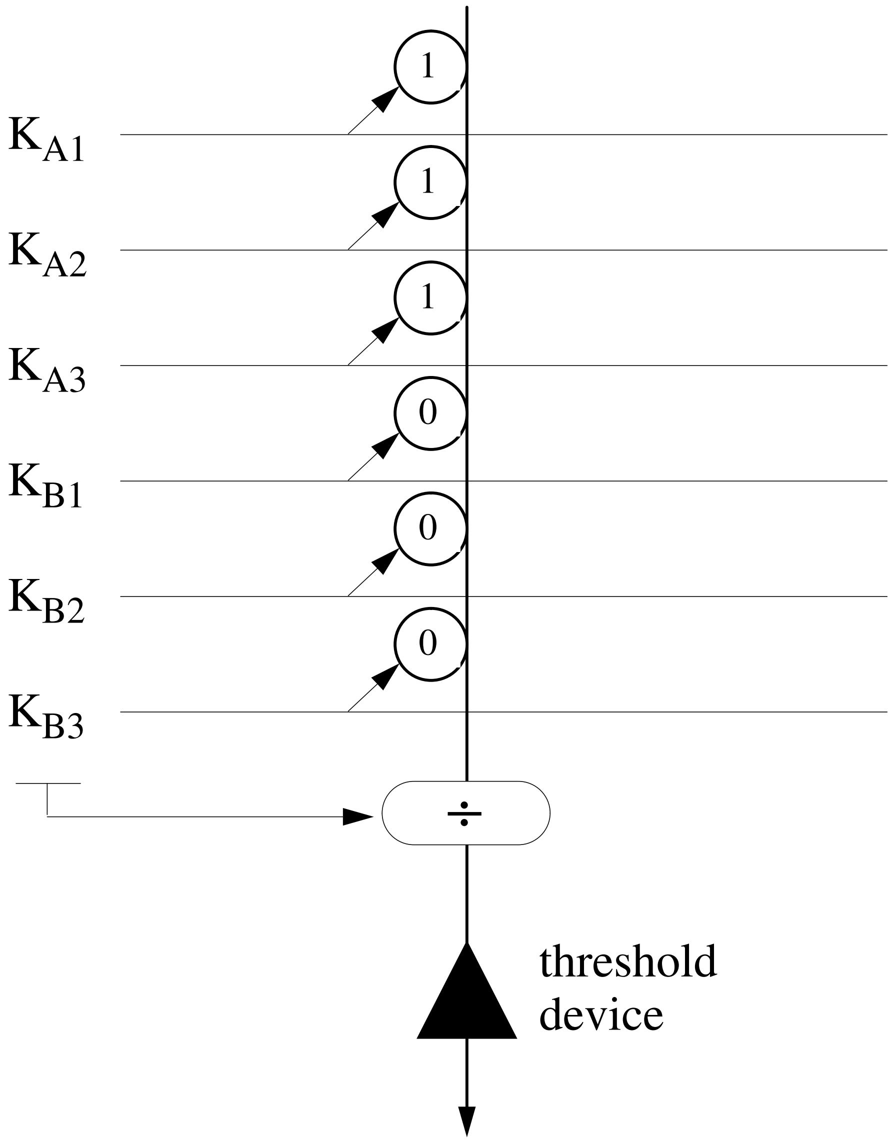
Partial Keys

- Suppose we use some redundancy in our keys:

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

- Each W_i value will be scaled by 1/3.
- Results should be the same, for normal keys.
- How about a partial key: <1,0,0, 0,0,0> ?
- If we apply the normal threshold of 0.5, the answer will always be zero.
- Solution: *normalize* the input to the threshold unit by dividing by the length of the key.

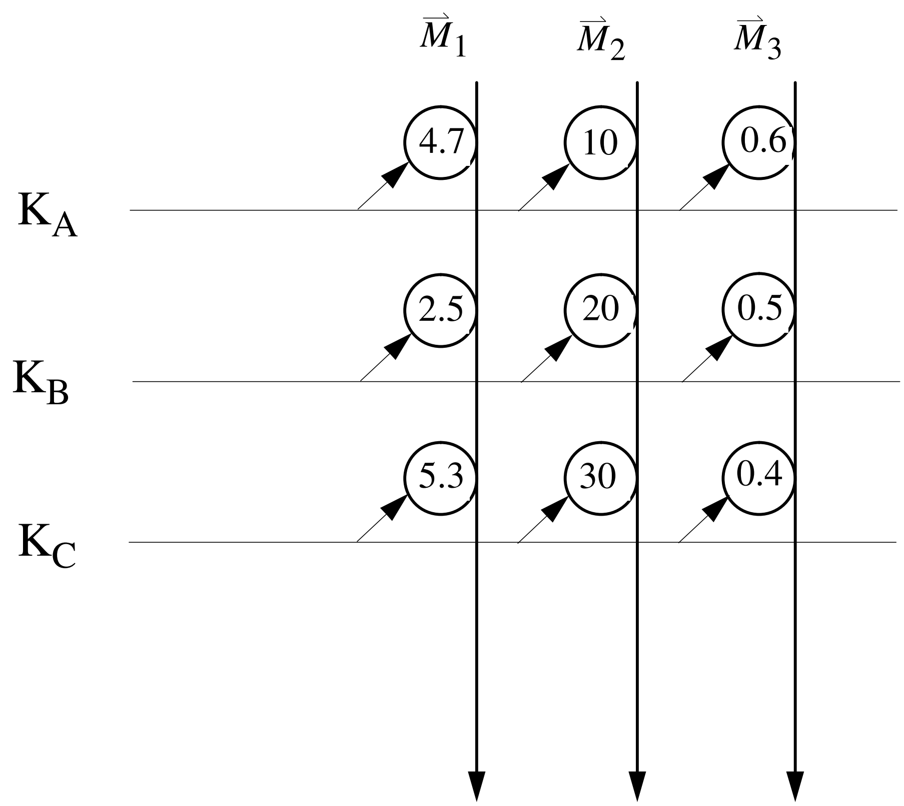
Scaling for Partial Keys



Storing Entire Vectors

To memorize vectors of length *n*, use a vector of *n* associative memories.

Memory K_A is the vector <4.7, 10, 0.6>.



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 vectors:

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

or a row matrix of column vectors:

$$[\vec{u} \qquad \vec{v} \qquad \vec{w}]$$

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

Matrix and Vector Transpose

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

$$\vec{u} = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix}$$

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

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

Weight Matrix by Outer Product

Let $\vec{u}, \vec{v}, \vec{w}$ be an orthonormal set (mutually orthogonal unit vectors).

Let $\underline{W} = \vec{u}\,(\vec{u}^T) + \vec{v}\,(\vec{v}^T) + \vec{w}\,(\vec{w}^T)$.

$$\begin{aligned}
\underline{W} &= [\; u_1\vec{u} + v_1\vec{v} + w_1\vec{w} \quad u_2\vec{u} + v_2\vec{v} + w_2\vec{w} \quad u_3\vec{u} + v_3\vec{v} + w_3\vec{w} \;] \\
\underline{W}\vec{u} &= [\; (u_1\vec{u}) \bullet \vec{u} \quad (u_2\vec{u}) \bullet \vec{u} \quad (u_3\vec{u}) \bullet \vec{u} \;] \\
&= [\; u_1\,(\vec{u} \bullet \vec{u}) \quad u_2\,(\vec{u} \bullet \vec{u}) \quad u_3\,(\vec{u} \bullet \vec{u}) \;] \\
&= [\; u_1 \quad u_2 \quad u_3 \;] \\
&= \vec{u}
\end{aligned}$$

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

Eigenvectors

- Suppose $\vec{u}, \vec{v}, \vec{w}$ are not really unit vectors.
- This means $(\vec{u} \bullet \vec{u}) \neq 1$.
- Then $\underline{W}\vec{u} = \lambda\vec{u}$.
- λ is an *eigenvalue* of the matrix $\underline{W}$.
- $\vec{u}$ is an *eigenvector* of the matrix.

- We can think of a matrix as a set of associative memories.
- The eigenvectors are the keys.
- They are the directions along which, for a given input, the memory will produce the largest output.
- The eigenvalues indicate how much a key is "stretched" by multiplication by the matrix.

Associative Retrieval

- The ability to retrieve patterns based on partial keys is what associative memory is all about.
- Keys can also be “noisy.” The thresholding process will reduce the effects of the noise on the result.
- Example of a large scale associative memory: Kohonen’s experiments storing images of human faces.
- Any portion of a face is enough to recall the whole.

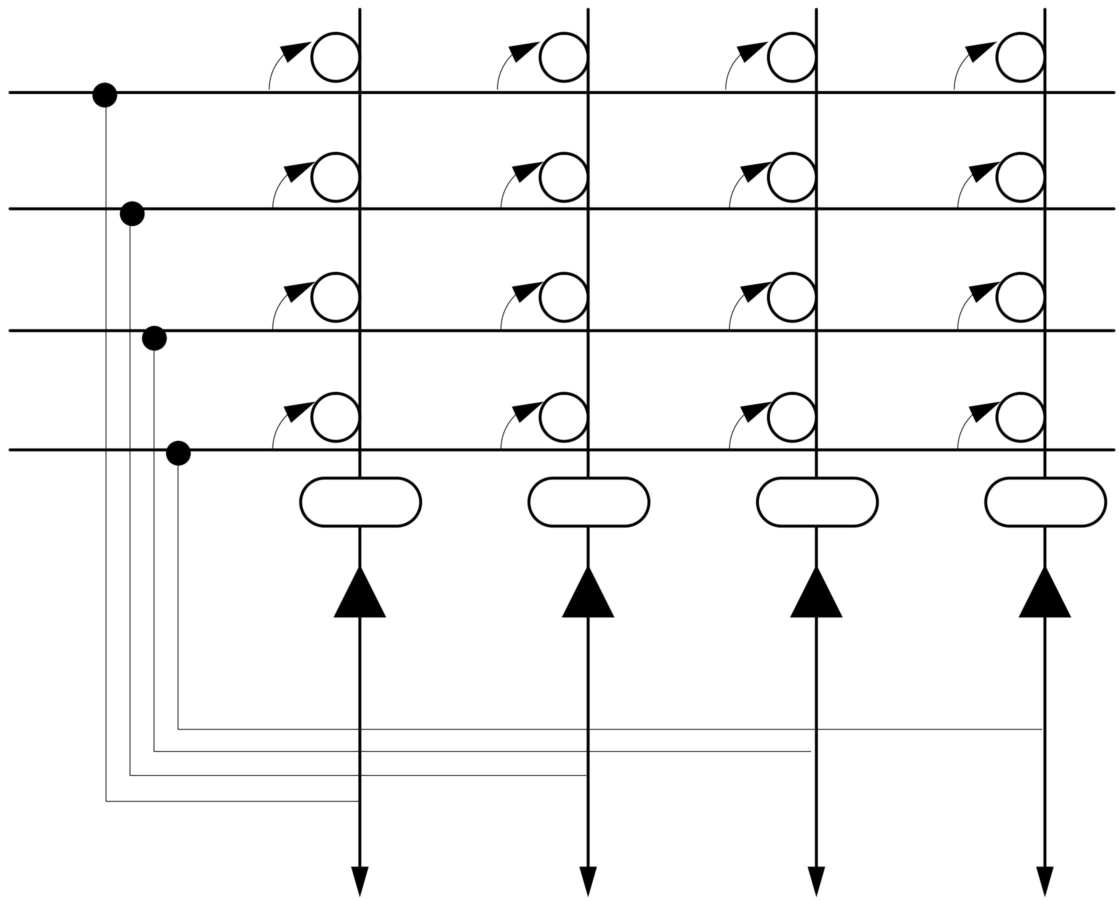
Auto-Associators

- Recall that the scalar associative memory generalizes to a memory for vectors. The weight vector W_i becomes a weight matrix W_{ij}.
- Combine keys and memory values into one large vector.

$$\vec{A} = \langle \vec{K}, \vec{M} \rangle$$

- Use this same vector for both the input (keys) and output (memory values) of the associative memory.
- Why do this?
- Suppose we start with a partial key K_0.
- Retrieval gives us an approximation A' = <K',M'>. Now we have a noisy estimate of M, called M'.
- We also have a new key, K', that is less noisy than K_0. Actually, all of <K',M'> can be used as the key.
- Feed <K',M'> back into the memory to get an even cleaner estimate A'' = <K'', M''>.

An Auto-Associator



Other Points to Note

- Recurrent connections are not required for associative memory.
- Kohonen discusses a case where one matrix associator feeds a second one. This is another way to get pattern cleanup.
- On the other hand, recurrence can be useful.
- Because associative memories perform mappings, they can be taught to produce sequences by feeding part of the output back to the input.
- A strict orthogonality requirement limits the capacity of the memory to the dimensionality of the input.
- Sparse patterns are nearly orthogonal, and in a high dimensional space they should produce little interference.
- Sparse patterns also seem more biologically plausible.