# Hopfield model (Part 2)

Instructor: Mark Kramer

# Remember ...

Consider

$$S_i = \text{sign}\left\{\sum_{j \neq i} \xi_i \xi_j S_j\right\} = \text{sign}\left\{\xi_i \boxed{\sum_{j \neq i} \xi_j S_j}\right\}$$

What is this?

Conclusions

$$\boxed{\sum_{j \neq i} \xi_j S_j} \approx \xi \cdot S$$

mean $\approx 0$ when S & $\xi$ are random (i.e., different patterns)

mean $> 0$ when S & $\xi$ are similar (i.e., nearby patterns)

*pseudo<u>orthogonality</u>*

# Today

Store and retrieve memories in the Hopfield model

# Model system

Consider neuron $i$ Use $S = 2\,\mathrm{V} - 1$

$$S_i = \begin{cases} +1 & \text{if neuron is “firing at maximum rate”} \\ -1 & \text{if neuron is “not firing”} \end{cases}$$

Dynamics

$$S_i = \mathrm{sign}\left\{\sum_{\mathrm{j} \neq \mathrm{i}} T_{ij} S_j\right\}$$

where we choose at random a neuron $i$ to update at each moment

# Model system

Consider neuron $i$ Use $S = 2\,\mathrm{V} - 1$

$$\mathrm{S}_i = \mathrm{sign}\left\{\sum_{\mathrm{j} \neq \mathrm{i}} T_{ij}\mathrm{S}_j\right\}$$

$T_{ij}$ = connection to neuron $i$ from neuron $j$

= 0 if not connected

Note: reciprocal connections are allowed

**Q**: What is $T_{ij}$? To store pattern $\xi$: use $T_{ij} = \xi_i \xi_j$ where $T_{ii} = 0$

# Matrix

We need to compute

$$S_i = \text{sign}\left\{\sum_{\mathrm{j} \neq \mathrm{i}} T_{ij} S_j\right\}$$

matrix

vector

More linear algebra ...

# Matrix

A rectangular array of numbers

$$W = \begin{pmatrix} w_{11} & \cdots & w_{1m} \\ \vdots & & \vdots \\ w_{n1} & \cdots & w_{nm} \end{pmatrix}$$

$n \times m$ matrix

in Python

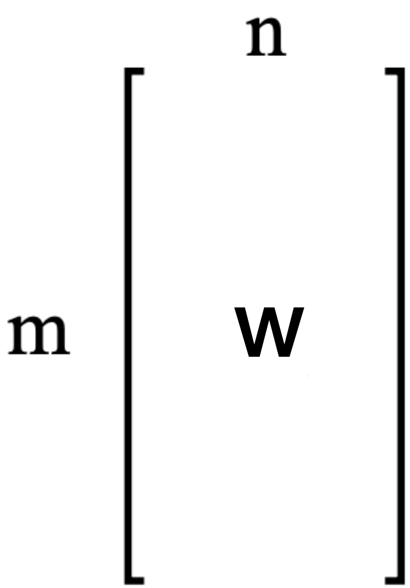
```
# make a 3 x 4 matrix
W = np.array([[1, 7, 3, 0], [2, -1, 2, -1], [1, 1, 1, 1]])
```

# Matrix

A matrix can be

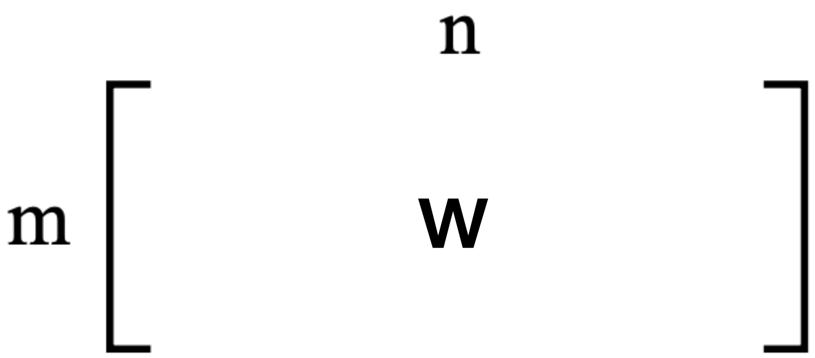
“tall and skinny”

# rows > # columns




“short and fat”

# rows < # columns



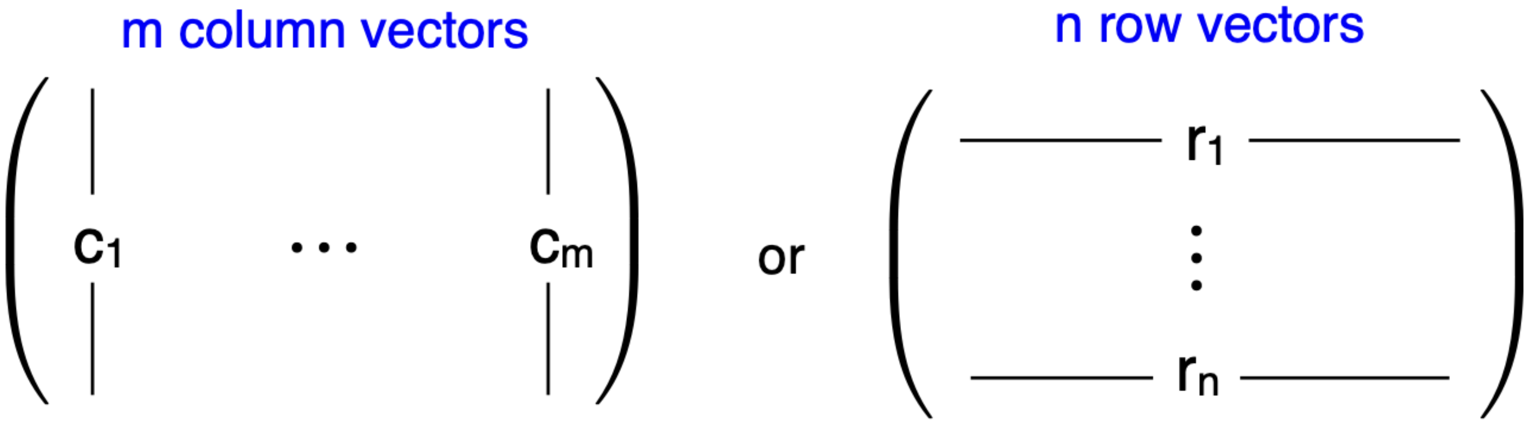
# Matrix

A rectangular array of numbers

$$W = \begin{pmatrix} w_{11} & \cdots & w_{1m} \\ \vdots & & \vdots \\ w_{n1} & \cdots & w_{nm} \end{pmatrix}$$

$n \times m$ matrix

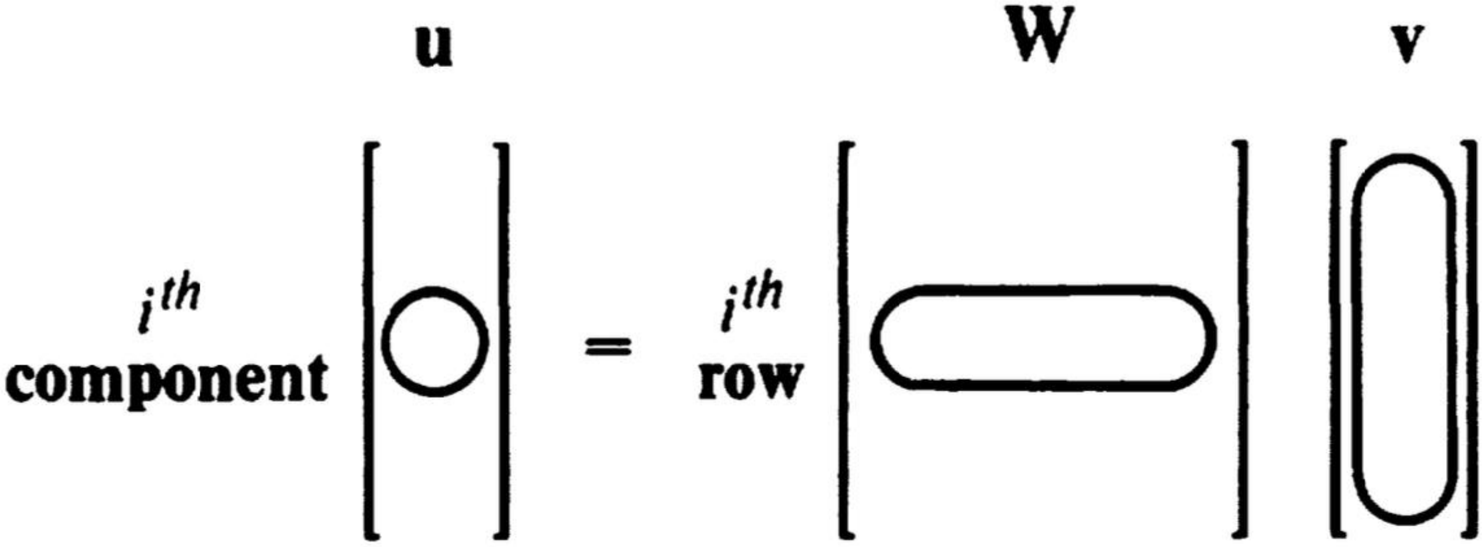
can think of it as:



# Matrix multiplication

$$\vec{u} = W\vec{v}$$

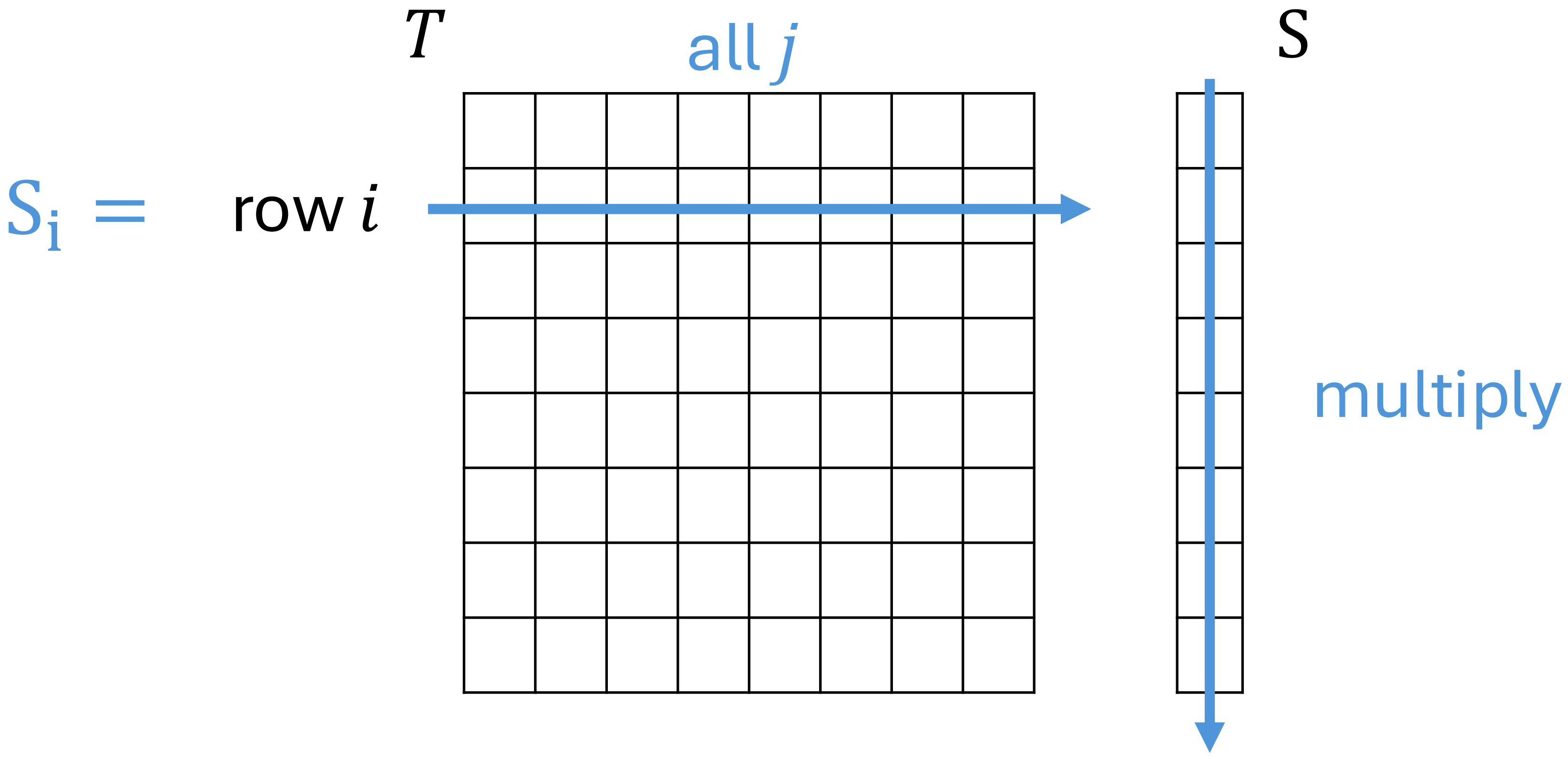
One perspective: dot product with each row:



# Matrix multiplication

$$S_i = \text{sign}\left\{\sum_{j \neq i} T_{ij} S_j\right\}$$

A rectangular array of numbers



# Matrix-vector multiplication

another perspective: *linear combination of columns*

$$\underbrace{\begin{bmatrix} u_1 \\ \vdots \\ u_n \end{bmatrix}}_{\vec{\mathbf{u}}} = \underbrace{\begin{bmatrix} | & & | \\ \vec{c}_1 & \cdots & \vec{c}_m \\ | & & | \end{bmatrix}}_{\mathbf{W}} \underbrace{\begin{bmatrix} v_1 \\ \vdots \\ v_m \end{bmatrix}}_{\vec{\mathbf{v}}} = v_1 \cdot \begin{matrix} | \\ \vec{c}_1 \\ | \end{matrix} + v_2 \cdot \begin{matrix} | \\ \vec{c}_2 \\ | \end{matrix} + \ldots + v_m \cdot \begin{matrix} | \\ \vec{c}_m \\ | \end{matrix}$$

# Matrix-vector multiplication (by hand)

1) $\begin{bmatrix} 5 & 3 \\ 2 & 6 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix}$

2) $\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 3 \\ 11 \end{bmatrix}$

3) $\begin{bmatrix} -2 & 6 \\ 1 & -3 \\ -5 & 15 \end{bmatrix} \begin{bmatrix} 2 \\ 1 \end{bmatrix}$

4) $\begin{bmatrix} 1 & 3 & 7 \\ 2 & 4 & -1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 7 \end{bmatrix}$

5) $\begin{bmatrix} 3 & 0 & 0 \\ 0 & -2 & 0 \\ 0 & 0 & 5 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$

*diagonal matrix*

6) $\begin{bmatrix} 3 & 0 & 0 \\ 0 & -2 & 0 \\ 0 & 0 & 5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$

Q1: What do you notice about the relationship between the size of the matrix and the size of the vector?

Q2: what does multiplying by a diagonal matrix do to a vector?

# Matrix-vector multiplication

you will never (or rarely) need to do this by hand!

in python:

```
# make a 3 x 4 matrix
W = np.array([[1, 7, 3, 0], [2, -1, 2, -1], [1, 1, 1, 1]])

# make a 4 x 1 matrix (ie, a vector)
v = np.array([[1], [2], [-3], [0]])


# Compute W times v (matrix-vector product)
u = W @ v
```

note special symbol '@'
for matrix multiply!

Q: what size is u?

# Information storage

$$S_i = \text{sign}\left\{\sum_{j \neq i} T_{ij} S_j\right\}$$

Store the patterns in $T_{ij}$

To store pattern $\xi$: use $T_{ij} = \xi_i \xi_j$ where $T_{ii} = 0$

*"The stored state would always be stable under our processing algorithm"*

Let's try it.

*Python*

Linear Algebra: Matrices

Hopfield: Find the pattern