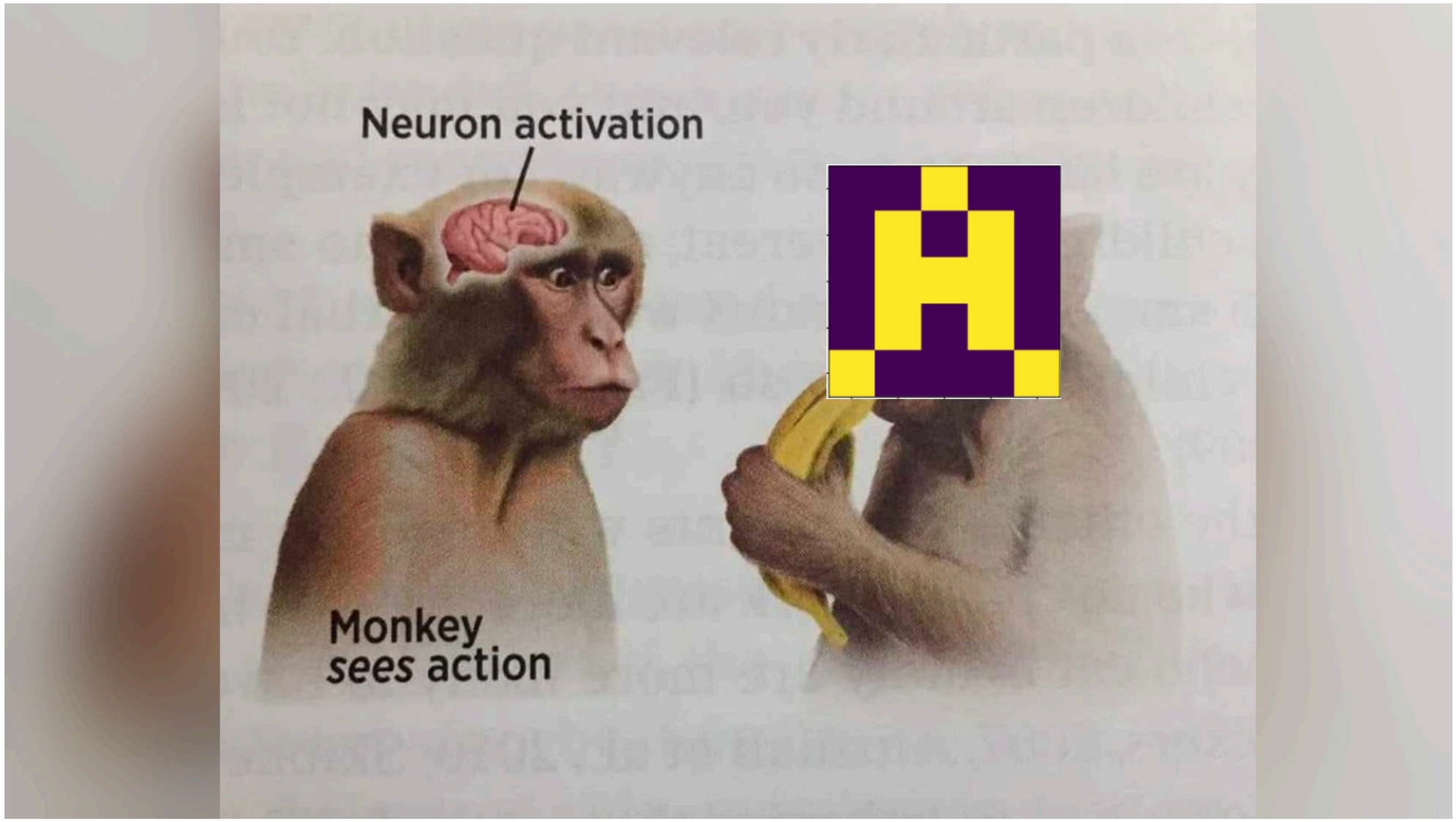
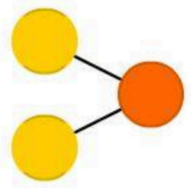
Homework question:
Implement a neural network to *distinguish** low-quality characters



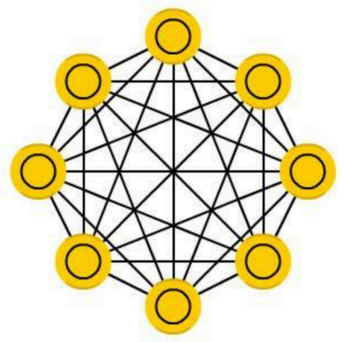
MCB111 Section week 8
Nicolas Gort Freitas
10/28/2022

Implementation options

Perceptron (P)



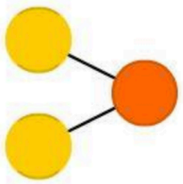
Hopfield Network (HN)



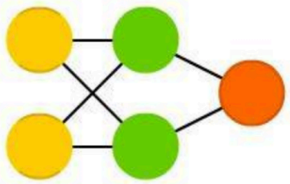
- Classification using a one-neuron network (i.e. a perceptron)
- Reconstruction using a Hopfield network

Hopfield networks are recurrent networks that don't use backpropagation to learn

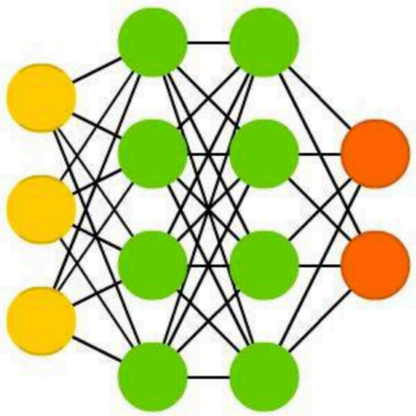
Perceptron (P)



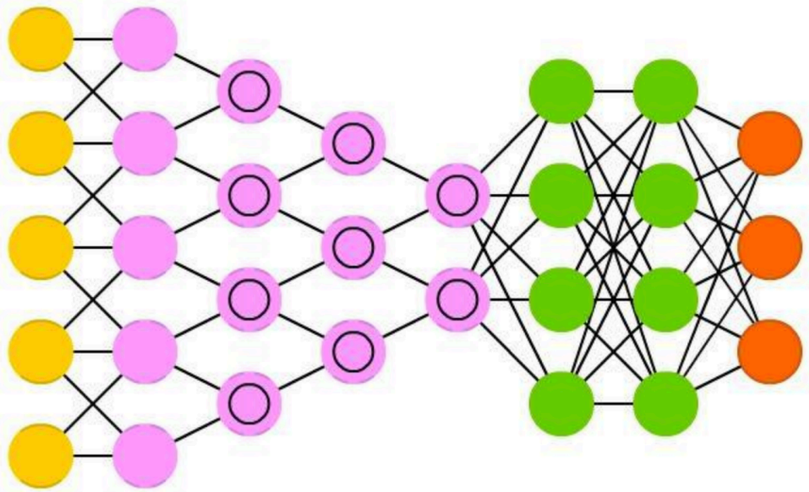
Feed Forward (FF)



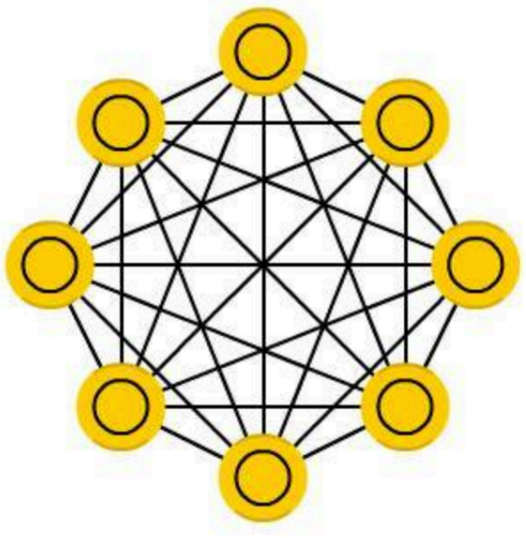
Deep Feed Forward (DFF)



Deep Convolutional Network (DCN)



Hopfield Network (HN)



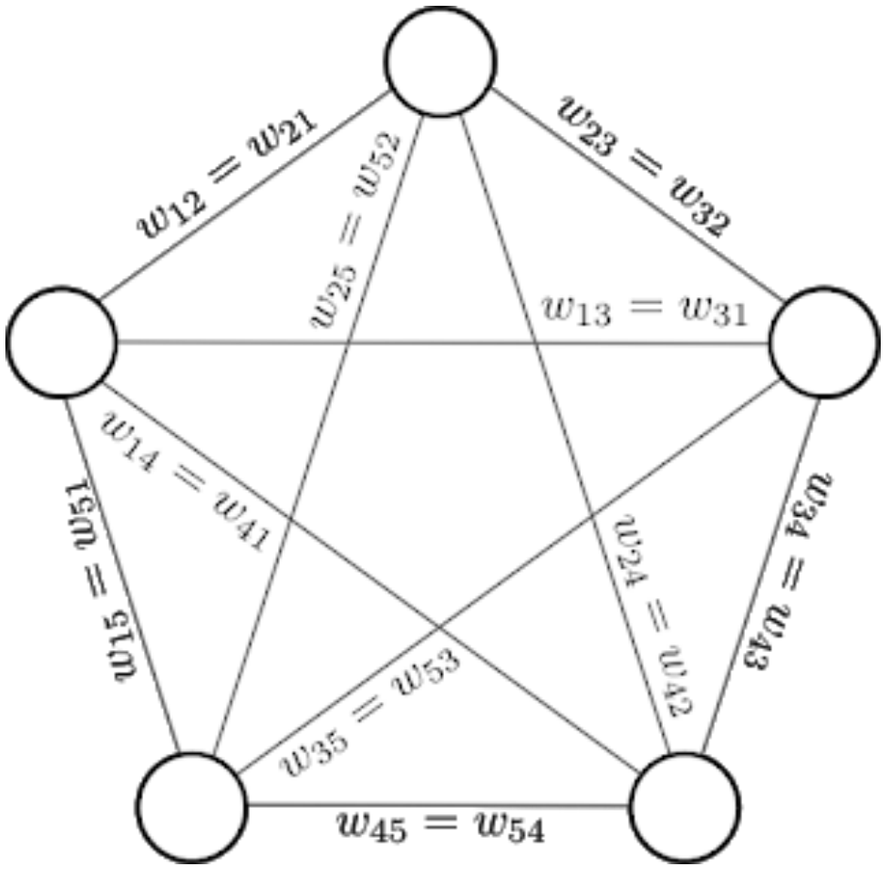
Recurrent networks include cycles in their topology

Some, like LSTMs, do apply backpropagation

Hopfield networks, on the other hand uses something

Source: asimovinstitute.org

Hopfield networks are fully connected graphs with symmetrical weights



The weights are usually packed together in a matrix
With N rows and N columns, where N is the number of neurons

$$\mathbf{W} \in \mathbb{R}^{N \times N}$$

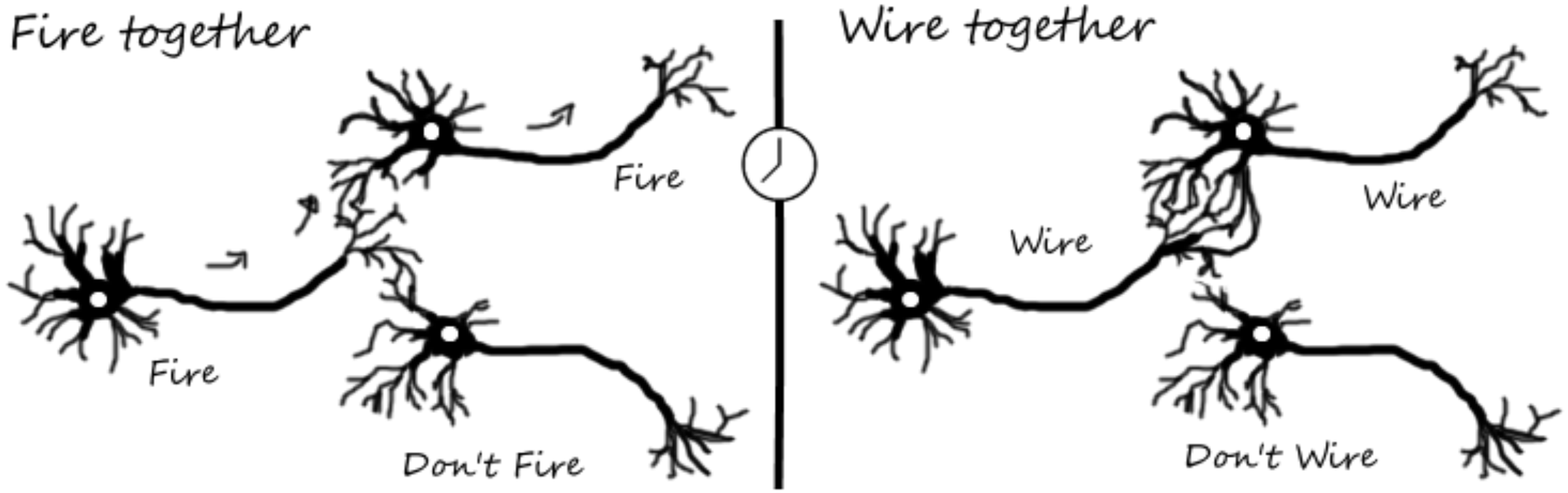
Self-edges have weight 0 by definition.

$$w_{n,n} = 0$$

Weights are symmetrical for any pair of neurons m, n:

$$w_{m,n} = w_{n,m}$$

Hopfield networks are modeled after associative memory



This principle, called Hebbian learning, postulates that it is most efficient for neurons to strengthen synapses among neurons that tend to fire in the same situations.

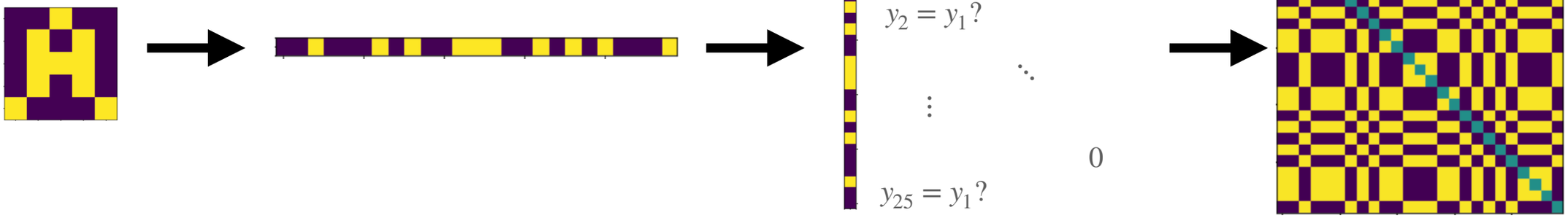
Hopfield networks interpret training observations in terms of their pairwise relationships

Training Pattern

Flattened image

Weights computed by pairwise comparisons

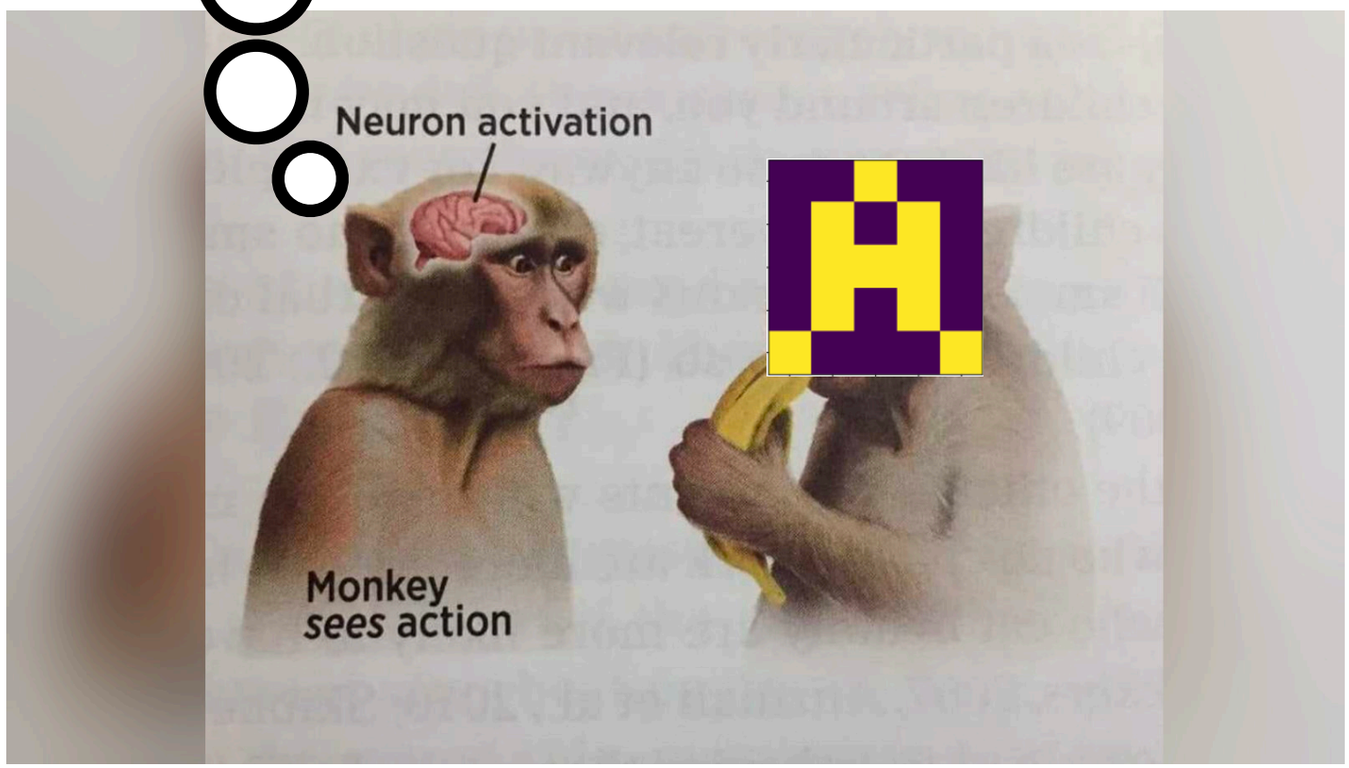
Weight matrix associated with training pattern



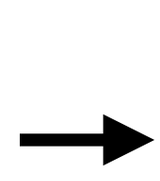
$\mathbf{y} \in \{-1,1\}^N$

$w_{m,n} = y_m \cdot y_n$

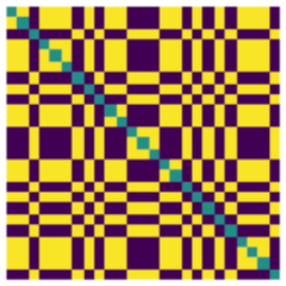
$\mathbf{W} \in \mathbb{R}^{N \times N}$

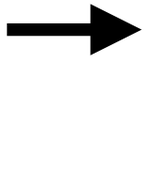


Training on multiple patterns

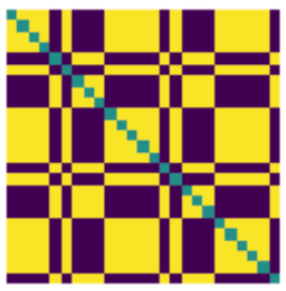


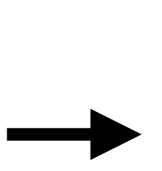
W trained on “A”



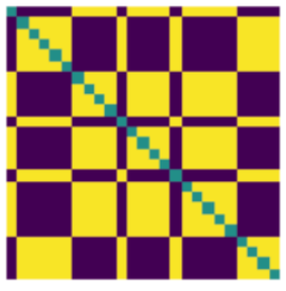


W trained on “B”





W trained on “C”



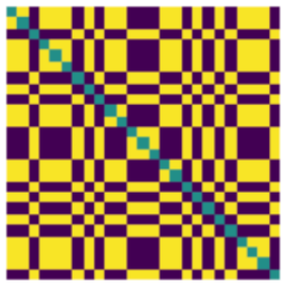
Training on multiple patterns

Simply add up the results from each observation!

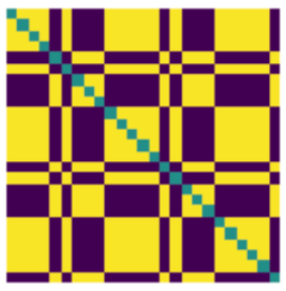
$$w_{m,n} = \sum_{i}^{K} y_m^{(i)} \cdot y_n^{(i)}$$

Here m and n are features of each flattened observation,
K is the number of observations

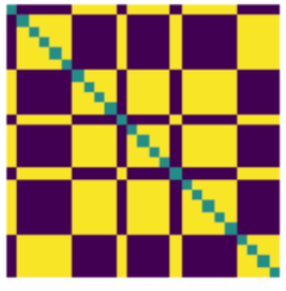
W trained on “A”



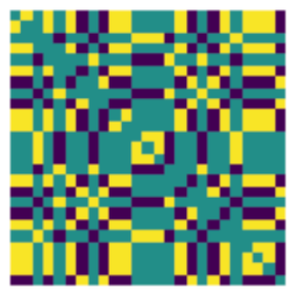
W trained on “B”



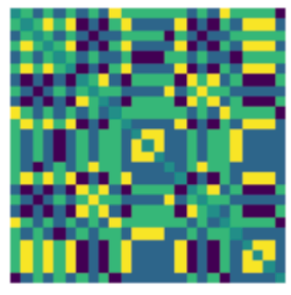
W trained on “C”



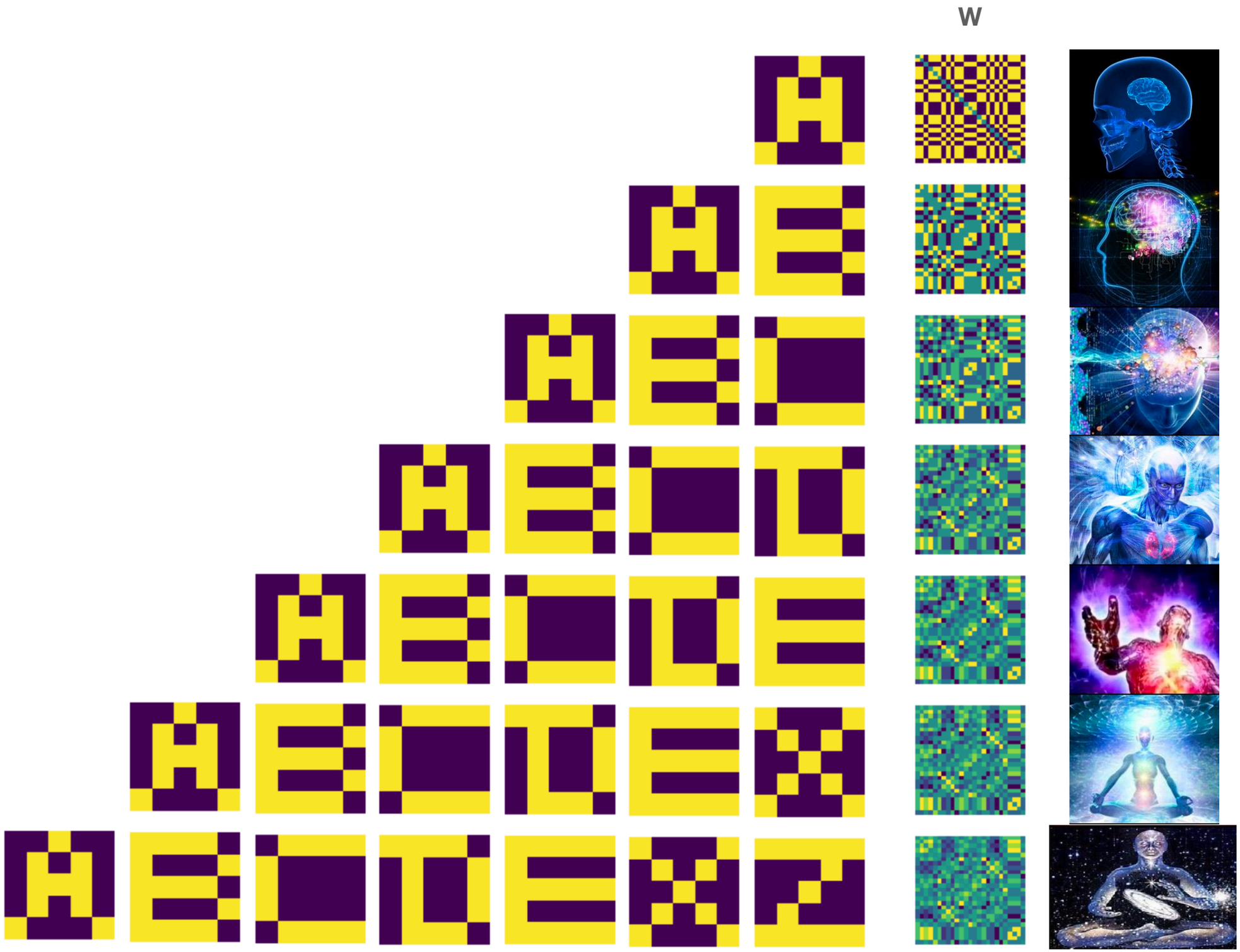
W trained on “A” and “B”



W trained on “A”, “B”, and “C”



So far we've covered how Hopfield networks can be trained…



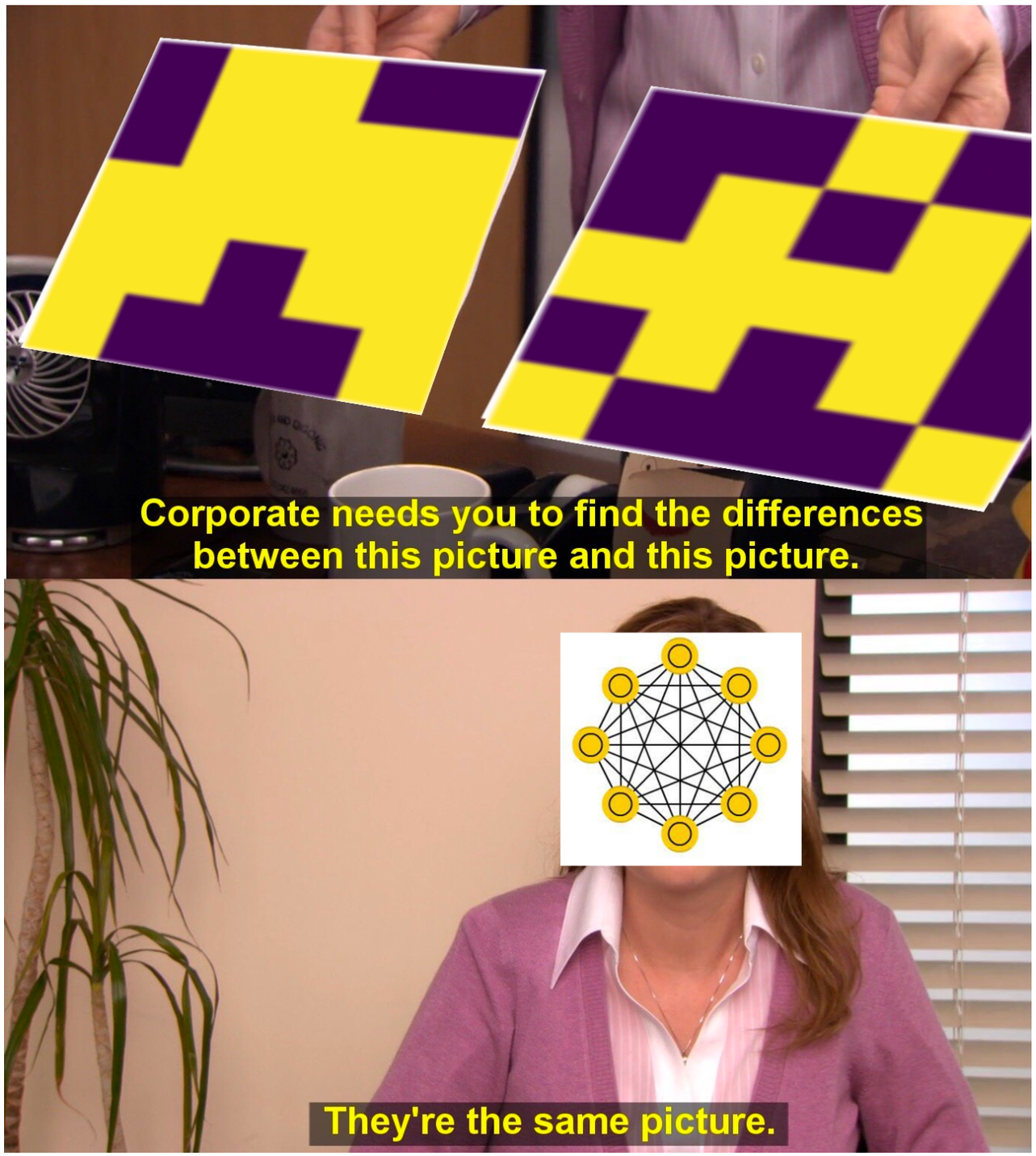
But how will they aid us in our task to recognize corrupted representations?

Image reconstruction is performed through iterative updates based on train weights

1. Update each element / pixel based on trained weights
2. Repeat until convergence
3. Check whether it matches one of the trained patterns?

$$\tilde{y}_n \leftarrow \begin{cases} +1 & \text{if } \sum_m w_{m,n} y_n \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

$$\mathbf{y} \rightarrow \tilde{\mathbf{y}}^{(1)} \rightarrow \ldots \rightarrow \tilde{\mathbf{y}}^{(\infty)}$$


Corporate needs you to find the differences
between this picture and this picture.
They're the same picture.

Final reflection questions

- Why does a Hopfield network reflect the principle of “fire together, wire together”?
- Many neuroscientists believe memories are in synapses rather than neurons. How would a Hopfield network relate to this model of memory?
- When updating our y’s, will we always converge to one of the elements in the training set?
- Can you unlearn a memory in a Hopfield network? How would you envision this happening?