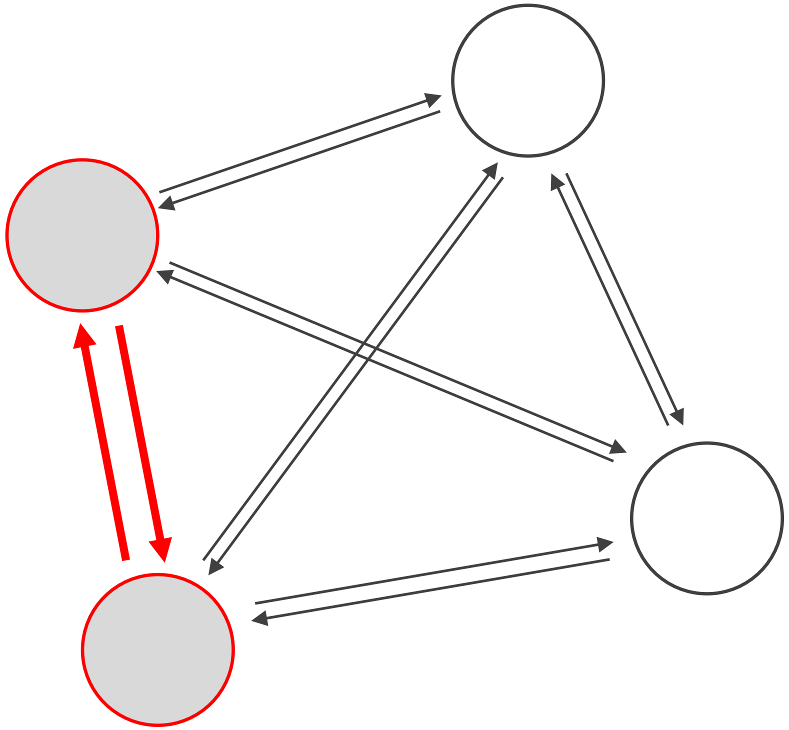
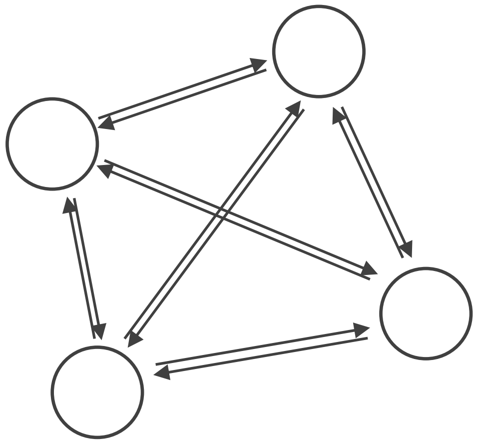
Hopfield networks



Hopfield networks

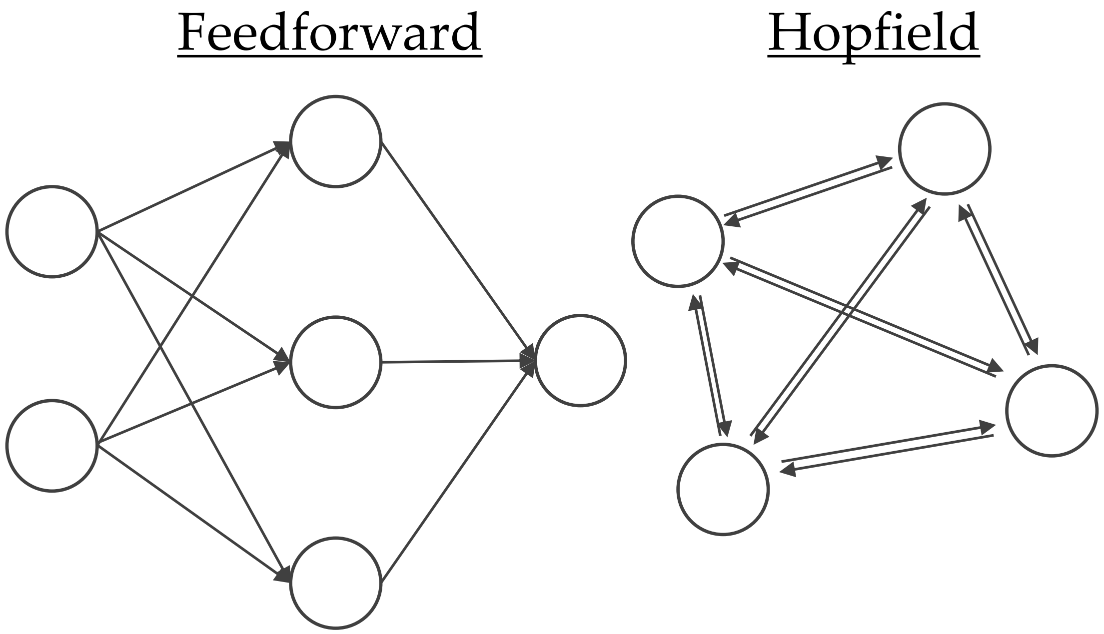
- Another type of neural network comprising n nodes
- Every node is connected to every other node
- The connection between each two nodes is bidirectional
  - The forward information flow both from node x_i to x_j and x_j to x_i
  - Similarly, backpropagation both from node x_i to x_j and x_j to x_i
  - Cycles in the information propagation
- They were early neural net models for learning memories
- Specifically, implemented with the Hebbian rule for 'associative learning'

Hopfield



Hopfield vs feedforward networks

- Feedforward networks have connections that make up for acyclic graphs
- Feedback networks are networks that are not feedforward
- Hopfield networks:
  - Fully connected feedback networks
  - Symmetric weights, no self-connections
  - Associative (Hebbian) learning
- No separation of hidden vs visible
  - Neurons (nodes) update themselves
  - Based on all other neurons



[Information Theory, Inference, and Learning Algorithms, D. MacKey]

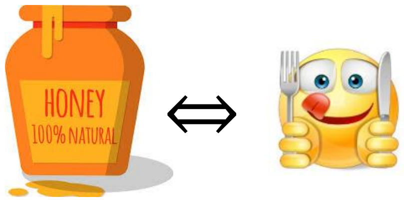
Hebbian learning

- Positively correlated neurons reinforce each other's weights

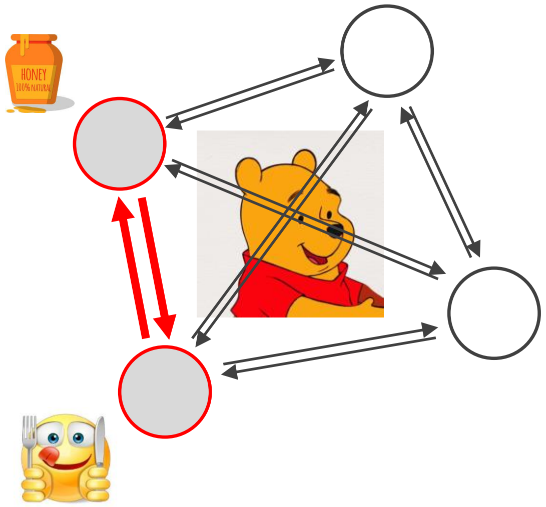
$$\frac{dw_{ij}}{dt} \propto \text{correlation}\,(x_i, x_j)$$

- Associative memories ⇔ No supervision ⇔ Pattern completion

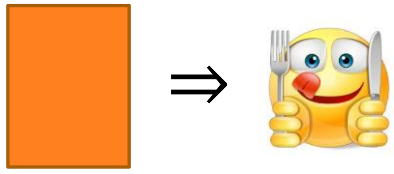
Time t



Winnie the Hopfield



Time $t+1$



Hopfield network

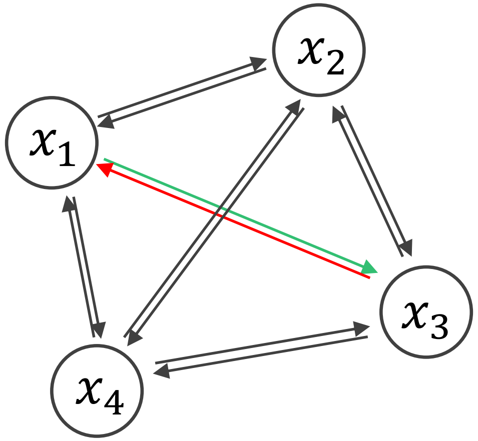
- Binary Hopfield defines neuron states given neuron activation a

$$x_i = h(a_i) = \begin{cases} 1 & a_i \geq 0 \\ -1 & a_i < 0 \end{cases}$$

- Continuous Hopfield defines neuron states given neuron activation a

$$x_i = \tanh(a_i)$$

- Note the feedback connection!
  - Neuron x_1 influences x_3, but x_3 influences x_1 back
- Who influences whom first?
  - Either synchronous updates: $a_i = \sum_j w_{ij} x_j$
  - Or asynchronous updates: one neuron at a time (fixed or random order)



Hopfield memory

- Network updates $x_i \in \{-1, 1\}$ till convergence to a stable state
  - Recurrent inference cycles
  - Not 'single propagation'
- Stable means x_i does not flip states no more

(a)

moscow------russia
lima----------peru
london-----england
tokyo--------japan
edinburgh-scotland
ottawa------canada
oslo--------norway
stockholm---sweden
paris-------france

(b)
moscow---:::::::: ⟹ moscow------russia
::::::::::--canada ⟹ ottawa------canada

(c)
otowa-------canada ⟹ ottawa------canada
egindurrh-sxotland ⟹ edinburgh-scotland

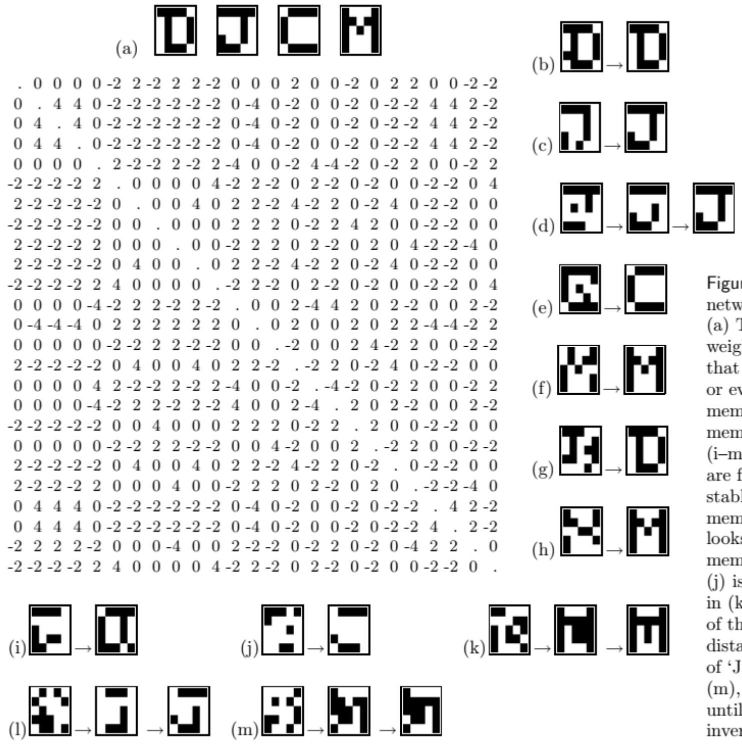


Figure 42.3. Binary Hopfield network storing four memories. (a) The four memories, and the weight matrix. (b–h) Initial states that differ by one, two, three, four, or even five bits from a desired memory are restored to that memory in one or two iterations. (i–m) Some initial conditions that are far from the memories lead to stable states other than the four memories; in (i), the stable state looks like a mixture of two memories, 'D' and 'J'; stable state (j) is like a mixture of 'J' and 'C'; in (k), we find a corrupted version of the 'M' memory (two bits distant); in (l) a corrupted version of 'J' (four bits distant) and in (m), a state which looks spurious until we recognize that it is the inverse of the stable state (l).

Energy function

- Hopfield networks minimize the quadratic energy function

$$f_\theta(\boldsymbol{x}) = \sum_{i,j} w_{ij} x_i x_j + \sum_i b_i x_i$$

- Lyapunov functions are functions that
  - Decreases under the dynamical evolution of the system
  - Bounded below
- Lyapunov functions converge to fixed points
- The Hopfield energy is a Lyapunov function
  - Provided asynchronous updates
  - Provided symmetric weights

Learning algorithm

```
w = x' * x ;          # initialize the weights using Hebb rule

for l = 1:L           # loop L times

        for i=1:I                 #
          w(i,i) = 0 ;            #  ensure the self-weights are zero.
        end                       #

        a  = x * w      ;     # compute all activations
        y  = sigmoid(a) ;     # compute all outputs
        e  = t - y      ;     # compute all errors
        gw = x' * e     ;     # compute the gradients
        gw = gw + gw'   ;     # symmetrize gradients

        w  = w + eta * ( gw - alpha * w ) ;  # make step

endfor
```

Continuous-time continuous Hopfield network

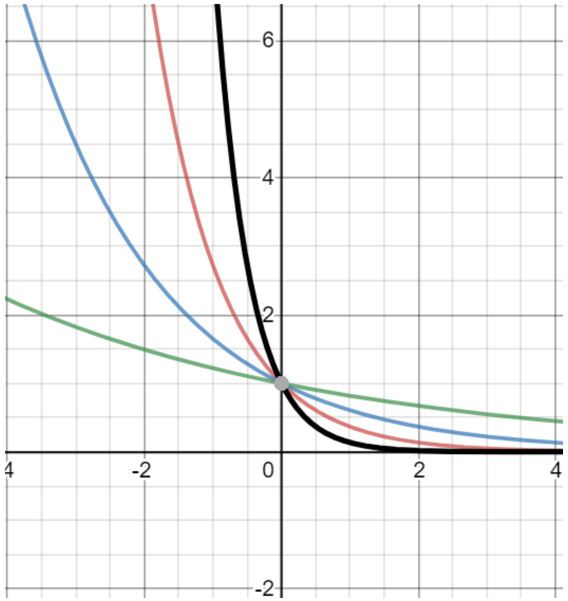
- We can replace the state variables with continuous-time variables
- At time t we compute instantaneous activations

$$a_i(t) = \sum_j w_{ij} x_j(t)$$

- The neuron response is governed by a differential equation

$$\frac{d}{dt} x_i(t) = -\frac{1}{\tau}\left(x_i(t) - h(a_i)\right)$$

- For steady a_i the neuron response goes to stable state



Hopfield networks for optimization problems

- Optimize function under constraints
- The stable states will be the optimal solution
- Weights must ensure *valid* and *optimal* solutions

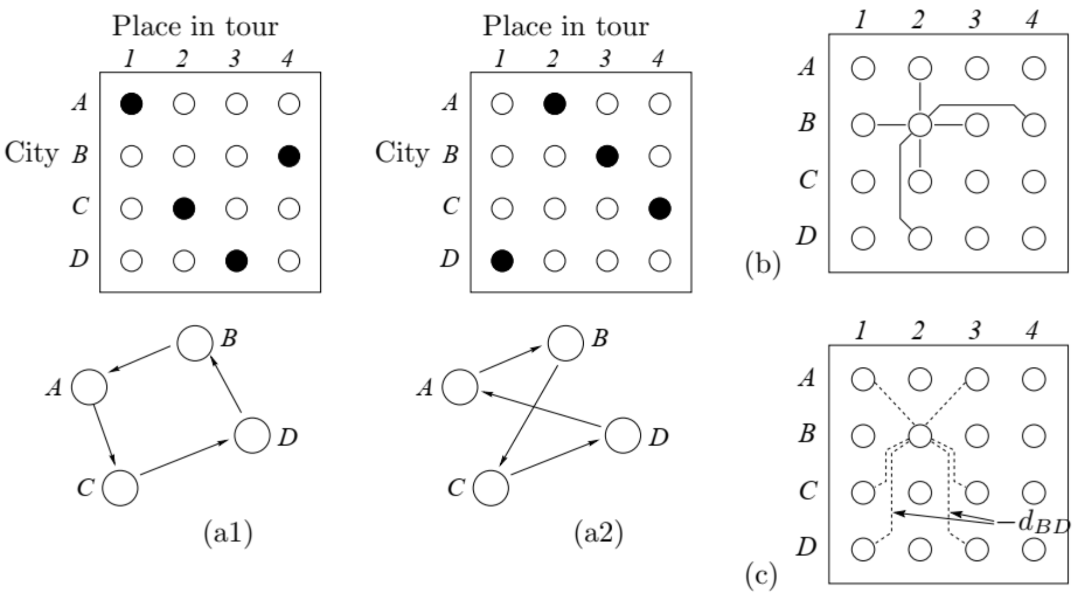
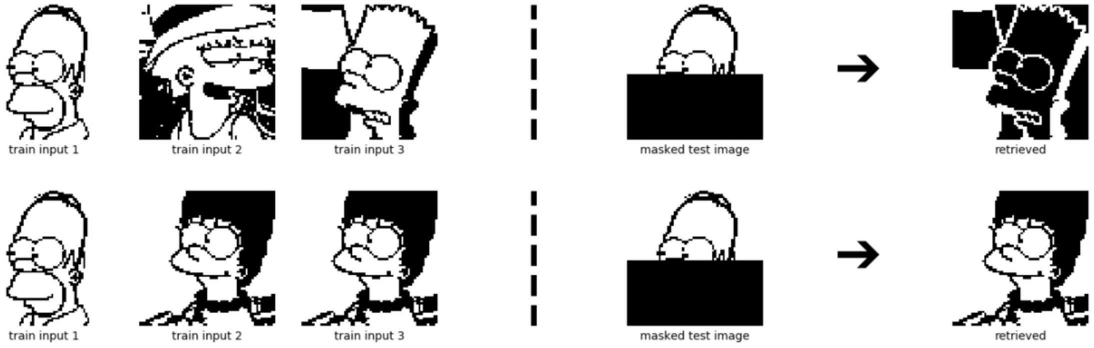


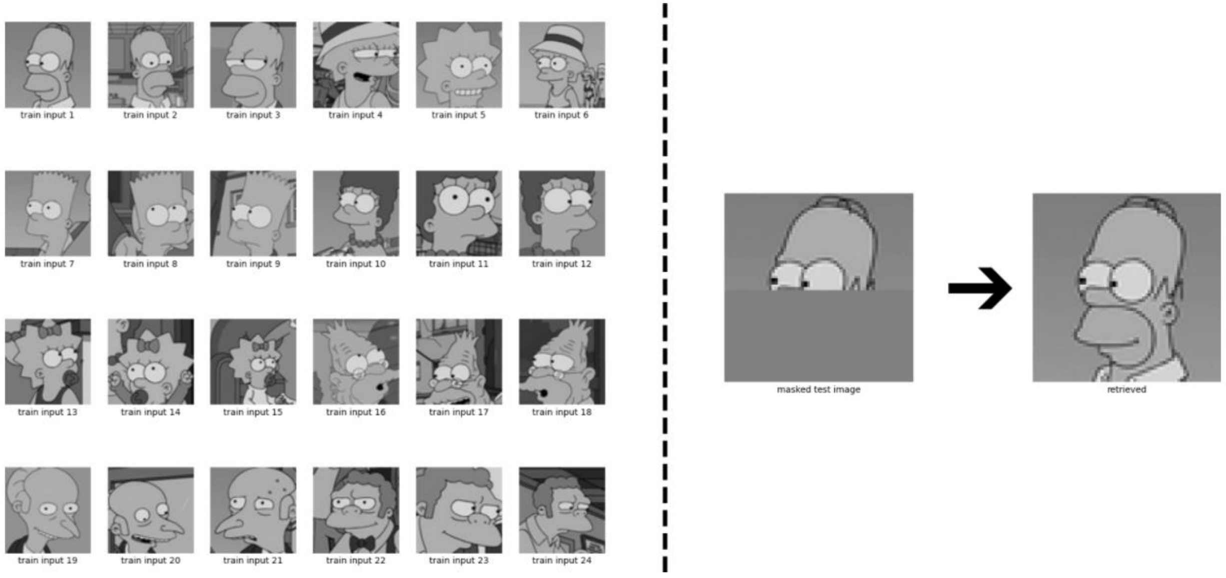
Figure 42.10. Hopfield network for solving a travelling salesman problem with $K = 4$ cities. (a1,2) Two solution states of the 16-neuron network, with activites represented by black = 1, white = 0; and the tours corresponding to these network states. (b) The negative weights between node $B2$ and other nodes; these weights enforce validity of a tour. (c) The negative weights that embody the distance objective function.

Hopfield networks is all you need

- Retrieving from stored memory patterns
- Update rule as in the attention mechanism in transformer networks

$$\boldsymbol{W} = \sum_{i=1}^{3} \boldsymbol{x}_i \boldsymbol{x}_i^T \,, \qquad \boldsymbol{x}_i \in \{-1, 1\}^d \,. \qquad (6)$$





Ramsauer et al., 2020