CONTENT-ADDRESSABLE MEMORIES

* Common memories: retrieve data by providing the address of the memory location where the data is stored.
* *Content-addressable* memories (CAMs, also called *associative* memories): retrieve data based on part of the data itself. Two types:
  + *Autoassociative* memories: part of the pattern to be retrieved is given as input. Example: *Hopfield* memories.
  + *Heteroassociative* memories: one pattern is retrieved as function of another. Example: *Kanerva* memories.
* The principle for the implementation of CAMs is to use the equilibrium points of nonlinear dynamical systems. They are built from interconnected artificial neurons.
* The equilibrium points are also called *attractors*. The area of the state space around an attractor is a *basin of attraction.*

HOPFIELD CAM MODEL (1)

* There are N neurons that are fully interconnected (connections between any pairs of neurons). The output values are given by: x_i, $i = 1, \ldots, N$.
* The output values are discrete: $x_i = 1 \vee x_i = -1$. All outputs are collected in the vector $\boldsymbol{x}$.
* The outputs are computed from: $x_i(t+1) = g\left(\sum_{j=1}^{N} w_{ij} x_j(t)\right)$. Note that in this type of neural network, the weights are fixed and the state (neuron outputs) evolve in time.
* The weights are symmetric: $w_{ij} = w_{ji}$.

HOPFIELD CAM MODEL (2)

* The patterns to be stored in the memory are the vectors $\boldsymbol{x}^p$, $p = 1, \ldots, Q$.
* The weights are chosen as: $w_{ij} = \sum_{p=1}^{Q} x_i^p x_j^p$. This is the *Hebbian learning rule.*
* Note that the weights are the elements of the correlation matrix of the patterns: $W = \sum_{p=1}^{Q} \boldsymbol{x}^p (\boldsymbol{x}^p)^T$.

HOPFIELD CAM PROPERTIES

* States close to one of the patterns to be solved evolve to the that particular pattern (if some conditions are respected).
* States corresponding to the patterns are not the only stable states (equilibrium points). An undesired stable state is called a *spurious state.*

LYAPUNOV STABILITY OF THE HOPFIELD CAM MODEL (1)

* Consider the following Lyapunov function: $V(\boldsymbol{x}) = -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} w_{ij}x_ix_j$.
* Because x_ix_i is always positive: $V(\boldsymbol{x}) = C - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1,j\neq i}^{N} w_{ij}x_ix_j$.
* Suppose some x_k changes value from time t to time $t+1$ and all other outputs remain the same. The change in the Lyapunov function:

$$\Delta V = V(\boldsymbol{x}(t+1)) - V(\boldsymbol{x}(t))$$

$$\Delta V = -\sum_{l=1,l\neq k}^{N} w_{kl}x_k(t+1)x_l(t) + \sum_{l=1,l\neq k}^{N} w_{kl}x_k(t)x_l(t)$$

LYAPUNOV STABILITY OF THE HOPFIELD CAM MODEL (2)

* $\Delta V = 0$ for $x_k(t+1) = x_k(t)$.
* If $x_k(t+1) = -x_k(t)$:

$$\Delta V = 2\sum_{l=1,l\neq k}^{N} w_{kl}x_k(t)x_l(t) = 2x_k(t)\sum_{l=1}^{N} w_{kl}x_l(t) - 2w_{kk}.$$

* Both terms are negative (in the first term, the summation should have a different sign than $x_k(t)$ due to the assumption; the second term is negative as all w_{kk} are positive due to the Hebbian learning rule), proving that $\Delta V \leq 0$ always holds which means that the system always converges to an equilibrium point.

STORAGE CAPACITY (1)

* Take a stable state in a Hopfield memory equal to one of the patterns to be stored:

$$\boldsymbol{x}(t+1) = \boldsymbol{x}(t) = \boldsymbol{x}^p.$$

* So:

$$\boldsymbol{x}^p = g(W\boldsymbol{x}^p).$$

* The ith neuron obeys:

$$x_i^p = g\left[\sum_{j=1}^{N} w_{ij}x_j^p\right] = g\left[h_i^p\right].$$

* h_i^p is called the *induced local field*. Making use of the Hebbian learning rule:

$$h_i^p = \sum_{j=1}^{N}\sum_{q=1}^{Q} x_i^q x_j^q x_j^p.$$

* By taking apart the terms for which $p = q$ and making use of $x_j^p x_j^p = 1$ one gets:

$$h_i^p = Nx_i^p + \sum_{j=1}^{N}\sum_{q=1,q\neq p}^{Q} x_i^q x_j^q x_j^p.$$

* The output of the ith neuron will be stable as long as the absolute value of the second term is less than N.

STORAGE CAPACITY (2)

* The second term of the equation, the *noise* o, can be written as:

$$o = \sum_{q=1,q\neq p}^{Q} x_i^q \sum_{j=1}^{N} x_j^q x_j^p = \sum_{q=1,q\neq p}^{Q} x_i^q(\boldsymbol{x}^p \cdot \boldsymbol{x}^q).$$

* So, if all the patterns to be stored were orthogonal, the second term would always be zero. But the patterns don't need to be orthogonal.
* Suppose that each of the patterns are random. Then, the average value for each component is 0 (the average of 1 and -1) and the variance is 1.
* The random variable O associated to the entire second term will have $\mu = 0$ and $\sigma^2 = (Q-1)N$. Assuming that $Q \gg 1$, the variance is: QN.

STORAGE CAPACITY (3)

* The distribution of the random variable O is binomial, but can be approximated by a Gaussian with density function $\frac{1}{\sqrt{2\pi NQ}} e^{-\frac{o^2}{2NQ}}$.
* So, the probability that the second term becomes larger than the first one is:

$$P_{error} = P(O > N \mid x_i^p = -1) = P(O < -N \mid x_i^p = 1)$$

$$P_{error} = \frac{1}{\sqrt{2\pi NQ}} \int_N^{\infty} e^{-\frac{o^2}{2NQ}} do = \frac{1}{\sqrt{\pi}} \int_{\sqrt{\frac{N}{2Q}}}^{\infty} e^{-r^2} dr$$

$$P_{error} = \frac{1}{2}\left[1 - \operatorname{erf}\left(\sqrt{\frac{N}{2Q}}\right)\right], \text{ with } \operatorname{erf}(z) = \frac{2}{\sqrt{\pi}} \int_0^z e^{-y^2} dy.$$

STORAGE CAPACITY (4)

* The error function for large z can be approximated by:

$$\operatorname{erf}(z) = \frac{2}{\sqrt{\pi}} \int_0^z e^{-y^2} dy \simeq 1 - \frac{e^{-z^2}}{\sqrt{\pi}\, z}$$

* This means:

$$P_{error} = \frac{1}{2}\left[1 - \operatorname{erf}\left(\sqrt{\frac{N}{2Q}}\right)\right] \simeq \sqrt{\frac{N}{2Q\pi}}\, e^{-\frac{N}{2Q}}$$

* The probability for an entire pattern to be stable:

$$[1 - P_{error}]^N \simeq \left[1 - \sqrt{\frac{N}{2Q\pi}}\, e^{-\frac{N}{2Q}}\right]^N \simeq 1 - N\sqrt{\frac{N}{2Q\pi}}\, e^{-\frac{N}{2Q}}.$$

* The second term remains bounded by requiring: $-\frac{N}{2Q} \le \ln\frac{1}{N}$.

STORAGE CAPACITY (5)

* The last condition implies: $Q \le \frac{N}{2 \ln N}$.
* Using a similar reasoning and approximating NQ by N^2 one finds that all patterns in the memory are likely to be stable for: $Q \le \frac{N}{4 \ln N}$.

For more information consult:

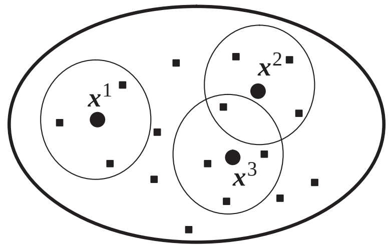
[1] Haykin, S., Neural Networks, A Comprehensive Foundation, Prentice Hall International, Upper Saddle River, New Jersey, Second Edition, (1999).

[2] Hassoun, M.H. (Ed.), "Associative Neural Memories, Theory and Implementation", Oxford University Press, New York, (1993).

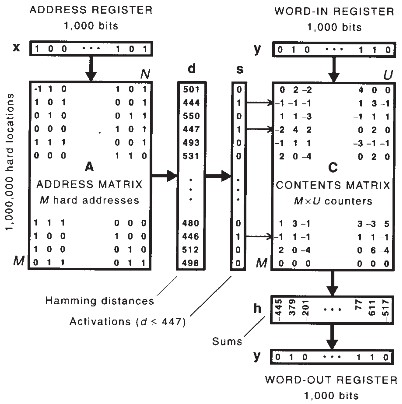
[3] Amit, D.J., "Modeling Brain Function, The World of Attractor Neural Networks", Cambridge University Press, Cambridge, (1989).

KANERVA MEMORIES: PRINCIPLES

* Also called *sparse distributed memory.*
* It is heteroassociative; it is meant to store pairs of patterns $(\boldsymbol{x}^p, \boldsymbol{y}^p)$, $p = 1, \ldots, Q$.
* The patterns $\boldsymbol{y}^p$ are not stored in a single location of the address space corresponding to $\boldsymbol{x}^p$, but distributed among multiple locations "close" to $\boldsymbol{x}^p$. It is not even necessary that $\boldsymbol{x}^p$ is itself a member of the address space.
* The patterns $\boldsymbol{y}^p$ are recovered by adding and thresholding the data stored at the locations close to $\boldsymbol{x}^p$.



KANERVA MEMORIES: VISUALIZATION



VECTOR DESCRIPTION (1)

* The vectors $\boldsymbol{x}^p$ have dimension N. The vector elements are binary valued but are encoded by -1 and 1 instead of the usual 0 and 1.
* The address space of the memory consists of M locations, $M \ll 2^N$. Every address $\boldsymbol{m}^k$, $k = 1, \dots, M$ can be represented by a vector of N elements encoded in the same way as the $\boldsymbol{x}^p$.
* Note that $\boldsymbol{m}^k \cdot \boldsymbol{x}^p$ is a measure for the distance between the two vectors (*not* the Hamming distance). $\boldsymbol{m}^k \cdot \boldsymbol{x}^p = N$ means that the two vectors have matching elements in all positions (their Hamming distance is zero); $\boldsymbol{m}^k \cdot \boldsymbol{x}^p = -N$ means that none of the elements match.

VECTOR DESCRIPTION (2)

* Collect all the row vectors $(\boldsymbol{m}^k)^T$ in a matrix A. Then $\boldsymbol{s}^p = \boldsymbol{\Theta}_D(A\boldsymbol{x}^p)$ is a column vector with elements 0 and 1. The threshold function $\boldsymbol{\Theta}_D$ returns 1 if its argument corresponds to a Hamming distance less than D (the threshold is applied to all vector elements).
* The contents of the memory are to be found in a matrix C (the vectors $\boldsymbol{y}^p$ also have components -1 and 1):

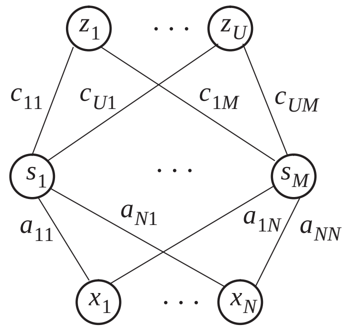
$$C = \sum_{p=1}^{Q} \boldsymbol{s}^p (\boldsymbol{y}^p)^T$$

* The retrieved data value $\boldsymbol{z}$ for a given input $\boldsymbol{x}$ is found from ($\boldsymbol{g}$ is a threshold function returning 1 if its argument is positive and 0 otherwise, applied to all vector elements):

$$\boldsymbol{z} = \boldsymbol{g}\left(C^T \boldsymbol{\Theta}_D(A\boldsymbol{x})\right).$$

NEURAL-NET IMPLEMENTATION

* From $\boldsymbol{z} = \boldsymbol{g}\left(C^T \boldsymbol{\Theta}_D(A\boldsymbol{x})\right)$ it can be seen that the Kanerva memory can be implemented by a two-layer feedforward (i.e. without feedback) neural network.
* The first layer transforms the input $\boldsymbol{x}$ to an intermediate signal $\boldsymbol{s}$ using weights from A and limiting function Θ_D.
* The second layer transforms the intermediate signal $\boldsymbol{s}$ to an output signal $\boldsymbol{z}$ using weights from C^T and limiting function g.



COMBINATORIAL OPTIMIZATION WITH HOPFIELD NETWORKS (1)

* Consider the *Lyapunov* or *energy* function of Hopfield networks:
$$V(\boldsymbol{x}) = -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} w_{ij}x_ix_j.$$
* Minimization of this function by the Hopfield network was proved for neurons with output values -1 and 1 and the step limiting function. The energy is also minimized for neurons with a sigmoid limiting function and output values from 0 to 1 (with the symmetric weight constraint):
$$x_i = g(v_i);\ v_i = \sum_j w_{ij}x_j;\ g(v) = \frac{1}{1+e^{-\alpha v}}$$

COMBINATORIAL OPTIMIZATION WITH HOPFIELD NETWORKS (2)

* *Zero-one quadratic programming* problems are problems with Boolean variables x_i ($i = 1, \ldots, N$) and a quadratic cost function. A quadratic cost function looks like:
$$c(\boldsymbol{x}) = \sum_{x=1}^{N}\sum_{x=1}^{N} c_{ij}x_ix_j$$
* The cost function has exactly the same form as the energy function of a Hopfield network:
$$V(\boldsymbol{x}) = -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} w_{ij}x_ix_j$$
* So, any problem with a quadratic cost function is "solved" by constructing a Hopfield network and deriving the weights w_{ij} from the coefficients c_{ij}.
* The network will converge to a solution if the c_{ij} are symmetric. However, this solution will in general be a local optimum.

EXAMPLE: TRAVELING SALESPERSON PROBLEM (TSP)

* Definition: find the shortest *tour* that visit all cities in a given set of N cities exactly once; the distance between cities i and j is d_{ij} ($d_{ij} = d_{ji}$).
* Define Boolean variables x_{ia}, $i, a = 1, \ldots, N$. $x_{ia} = 1$ means that city i occurs at position a of the tour.
* A quadratic cost function for this problem is (the first term is the tour length; the second penalizes illegal solutions; α is a parameter):
$$c = \sum_{i=1}^{N}\sum_{j=1}^{N} d_{ij}\sum_{a=1}^{N} x_{ia}x_{i(a+1 \bmod N)} + \alpha\left[\sum_{i=1}^{N}\sum_{a=1}^{N}\sum_{b=1, b\neq a}^{N}(x_{ia}x_{ib} + x_{ai}x_{bi})\right]$$

PROBLEMS AND REMEDIES: POTTS NEURONS

* Finding a local optimum is not enough.
* Constraints are implicitly encoded in the cost function which means that there is no guarantee that they will be satisfied.
* *Potts neurons* tackle both problems:
  + The first problem is tackled by borrowing techniques from *simulated annealing* (in the context of neural nets the terms *mean field annealing* and *Boltzmann machines* are used).
  + The second problem is tackled by updating groups of neurons simultaneously.

POTTS UPDATING RULE

* Consider the induced local field v_{ia} of the neuron with output x_{ia} (as e.g. used in the TSP example). Instead of applying a limiting function that only depends on v_{ia}, use an updating rule that involves all v_{ia}, $a = 1, \ldots, M_i$ (in the TSP example, $M_i = N$ for all i).

$$x_{ia} = \frac{e^{\frac{-v_{ia}}{T}}}{\sum_{a=1}^{M_i} e^{\frac{-v_{ia}}{T}}}$$

* This rule guarantees that $\sum_{a=1}^{M_i} x_{ia} = 1$. It is hoped that one of the x_{ia} gets close to one and all others close to zero (it does not always work).
* The parameter T is the "temperature" and is gradually decreased.