The hebbian learning rule and associators

During the last few weeks we talked about a common way to calculate changes in connection strengths in a neural network, the so called “hebbian learning rule”, in which a change in the strength of a connection is a function of the pre – and postsynaptic neural activities.

If x_j is the output of the presynaptic neuron, x_i the output of the postsynaptic neuron, and w_{ij} the strength of the connection between them, and γ learning rate, the one form of a learning rule would be:

$$\Delta W_{ij}(t) = \gamma * x_j * x_i$$

A more general form of a hebbian learning rule would be:

$$\Delta W_{ij}(t) = F(x_j, x_i, \gamma, t, \theta)$$

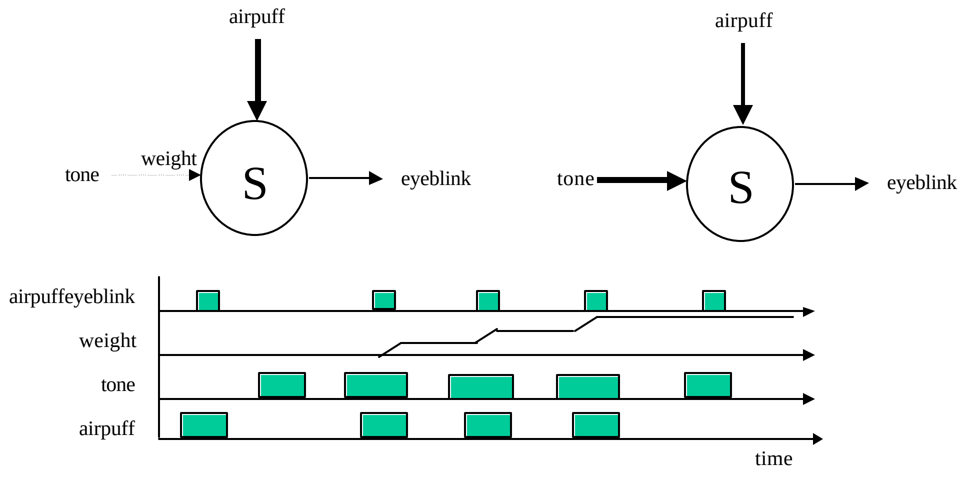
in which time and learning thresholds can be taken into account.

Time

We often say that the connection strength increases when the pre- and postsynaptic neurons are active “simultaneously”. Simultaneous is very relative depending on the system under consideration! For example, when synaptic plasticity is induced in a brain slice preparation, simultaneous can be as short as several ms. However, when an animal learns an association between a food taste and sickness, simultaneous can be as long as several hours!

Example we talked about in class

Classical conditioning can be modeled with a hebbian synapse. Consider an unconditioned stimulus (an air puff), an unconditioned response (eye blink), a conditioned stimulus (a tone) and a conditioned response (eye blink). Under normal conditions, an animal responds to an air puff with an eye blink. It does not respond to a tone with an eye blink. If the tone is paired with the air puff several times, then the animal acquires an association between the airpuff and the tone, and will now respond to the tone alone with an eye blink.
Consider a “black box” neural approach where one neuron receives input from the airpuff and from the tone. The neurons output represents the eye blink. The neuron is wired in such a way that at first, the airpuff but not the tone activates the neuron and produces an output. If we apply the airpuff input and the tone input together several times, then the neuron is active while the tone-input is active, and a hebbian learning rule will reinforce the strength of the connection between the tone and the neuron. This will lead to the fact that after a few trials, the tone alone will be able to activate the neuron. Neurons of this type can be recorded in the amygdala.



Application of the hebbian learning rule: the linear associator

Take a network of neurons which are organized in two distinct layers (called f and g). Neurons in layer f project to neurons in layer g but not vice versa (this would be called a "feedforward organization"). Each neuron in layers g and f is a linear unit, its output x is the sum of its inputs i (hence the name linear associator).

$x_i = I_i$ where xi is the neuron's output, Ii is the total input to that neuron. Remember that this means that a neuron's output can take on arbitrary values which represent mean activation values or firing rates (but not individual action potentials).

The strength of the connection form presynaptic neuron f_j to postsynpatic neuron g_i is given by w_{ij}. *Reminder: w_{ij} connection from neuron j to neuron i.* The activation of each neuron in the layer g is given by is sum of weighted inputs from the neurons in layer f.

$x_i = \Sigma w_{ij}\ x_j$ where xj are the outputs from the presynaptic neurons in layer f and w_{ij} are the corresponding connection strength or synaptic weights.
The strength of each connection is calculated from the product of the pre- and postsynaptic activities scaled by a "learning rate" γ (which determines how fast connection weights change).
$\Delta w_{ij} = \gamma * g_i * f_j$. If you don't want to write down the individual equations for each neuron, you can write a vector notation:

Vectors are lists of numbers:

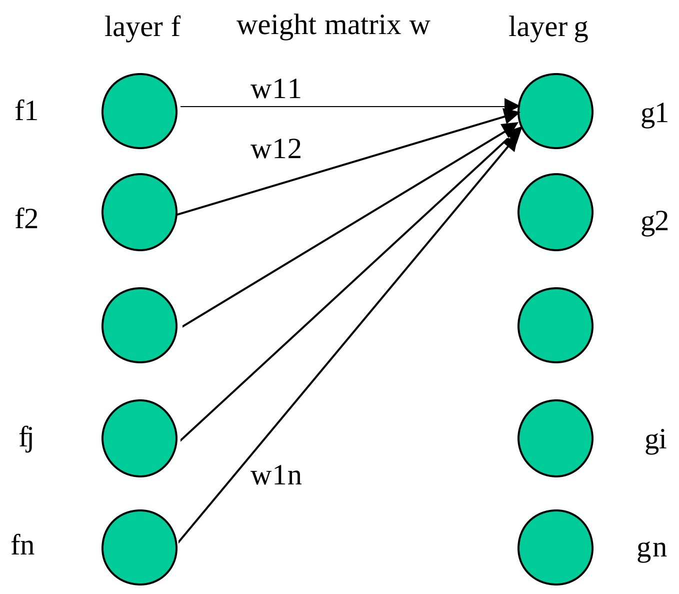
$$\mathbf{f} = \begin{bmatrix} f1 \\ f2 \\ fj \\ . \\ . \\ fn \end{bmatrix} \quad \text{and} \quad \mathbf{g} = \begin{bmatrix} g1 \\ g2 \\ gi \\ . \\ . \\ gn \end{bmatrix}$$

In order to multiply these two vectors, we have to "transpose" one of them:

$g^T = [g1, g2, .. gi, .. gn]$, then : $\mathbf{w} = \gamma\, \mathbf{g}_{\mathbf{f}}{}^{\mathbf{T}} = \begin{bmatrix} g1 \\ g2 \\ gj \\ . \\ .gn \end{bmatrix} [f1, f2,..fi...fn]$ this is called the "outer product" (as opposed to the "inner product" or "dot product, scalar product" ($a = \mathbf{g}^{\mathbf{T}}\mathbf{f}$)
and

$$w = \begin{bmatrix} w11 = g1f1 & w12 = g1f2\ \dots & w1n = g1fn \\ w21 = g2f1 & & \\ & & \\ wn1 = gnf1 & w2n = gnf2\ \dots & wnn = fngn \end{bmatrix}$$

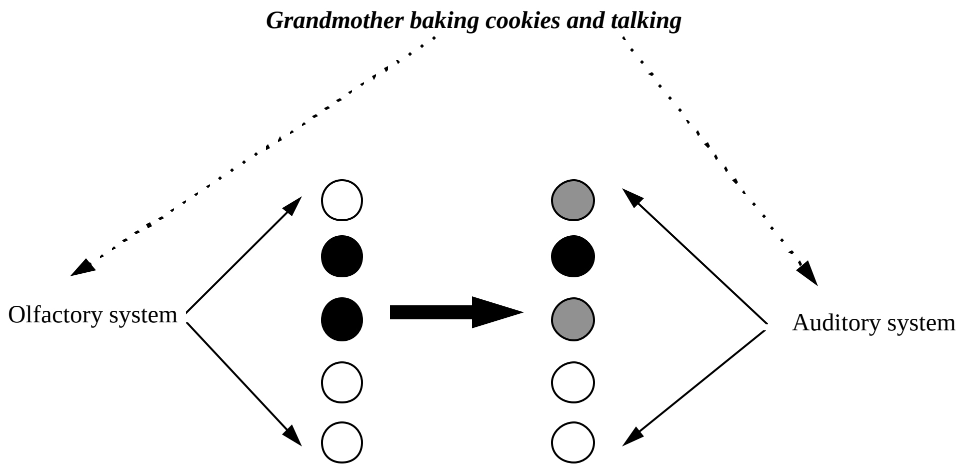
is the resulting matrix of synaptic weights.



g1 = x(f1) w11 + x(f2) w12 ... x(fj) w1j ... + x(fn) w1n
gi = x(f1) wi1 + x(f2) wi2 ... x(fj) wij ... + x(fn) win
etc ..

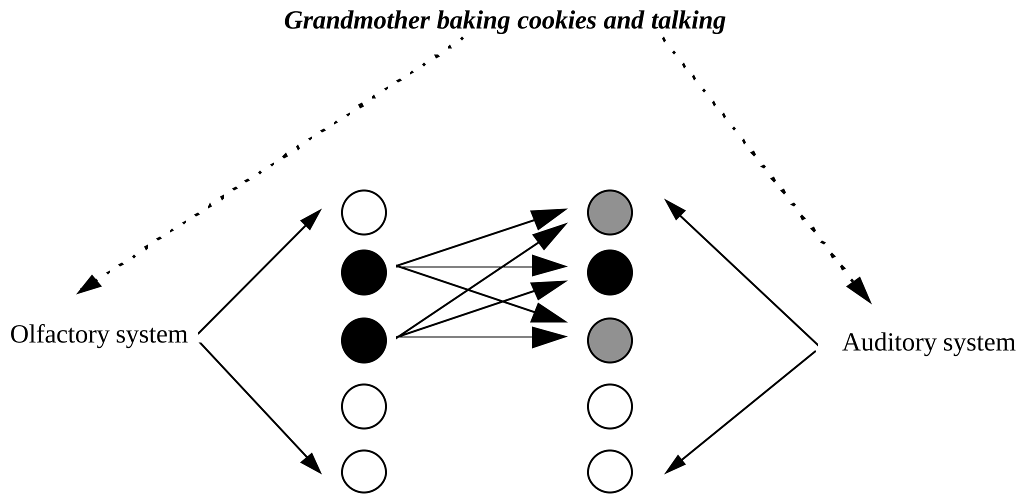
The linear associator stores associations between a pattern of neural activations in the input layer f and a pattern of activations in the output layer g. Once the associations have been stored in the connection weights between layer f and layer g, the pattern in layer g can be “recalled” by presentation of the input pattern in layer f.

Lets look at a familiar example, that of your grandmother. Lets assume that when you were a child, your grandmother would often bake a specific type of cookie, and that you learned to associate the smell of those cookies baking with the sound of your grandmother's voice. So, we have one layer of neurons receiving inputs from your auditory system (layer g), and one layer of neurons receiving input from your olfactory system (layer f). Neurons in layer g have receptive fields that are arranged such that some of them will respond more or less every time you smell the specific cookies, and neurons in layer g have receptive fields that can respond to your grandmothers voice.

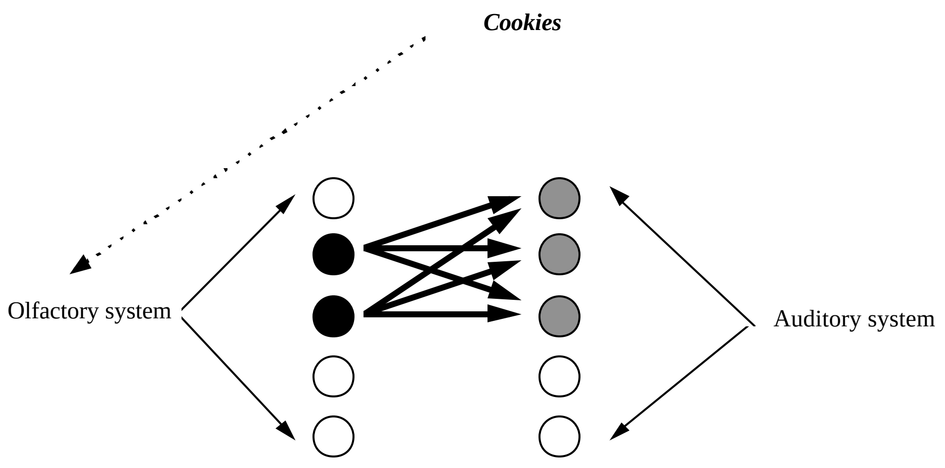


At first., smelling cookies will activate some neurons in layer g, but will have no or little effect on the neurons in layer f because the synaptic weights between neurons in layer g and f are weak or zero.

When you experience your grandmother's voice and the cookie smell at the same time, the synapses or connections between layer f (responding to olfactory stimuli) and layer g (responding to auditory stimuli) are changed via the hebbian learning rule.
$\Delta w_{ij} = \gamma * g_i * f_j$. The learning rule will affect only connections between neurons that are both active. Lets assume neurons depicted in black and gray are activated above threshold, and neurons depicted in white are inactive. The following synapses will undergo "learning":

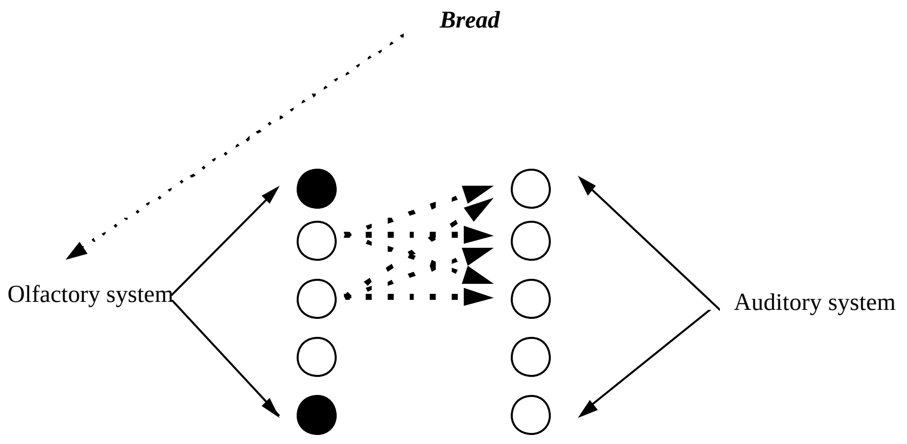


Assume that the outputs x of active neurons are 1, the learning rate is 1.0, and the initial synaptic weights are 0.0. Then, for all synapses connecting to active neurons,
$\Delta w_{ij} = \gamma * g_i * f_j$. = 1.0*1.0*1.0 = 1.0, and the resulting synaptic weights are 1.0.
Subsequently, if you smell the specific cookies your grandmother used to bake, the smell will "evoke" the sound of her voice by reactivating the neurons that were initially responding to the voice. The system has formed an "association" between the smell and the voice.



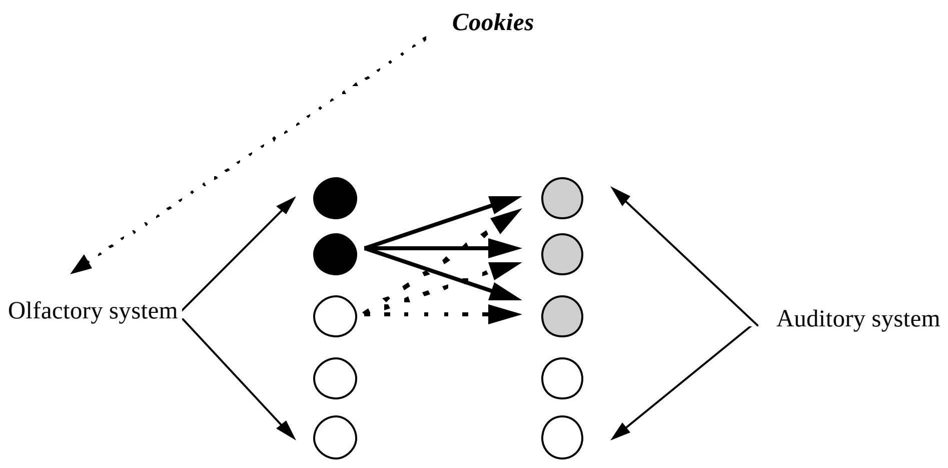
$$x(g_i) = \Sigma\, w_{ij} * x(f_j)$$

If you smelled a very different smell from the one you learned to associate with your grandmother's cookies, the voice would not be evoked.



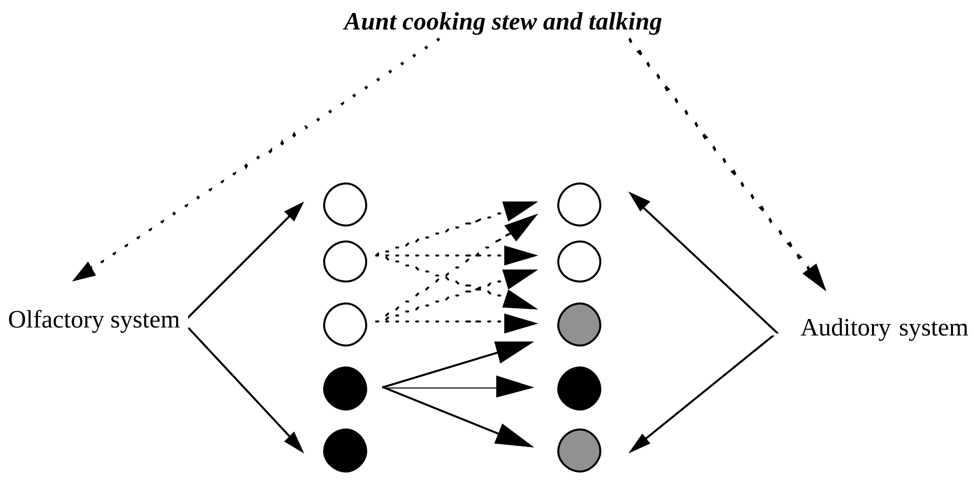
$$x(g_i) = \Sigma\, w_{ij} * x(f_j)$$

But if you smelled a smell that was only slightly different (or overlapping) with the cookie smell, the voice might be evoked to a lesser degree!

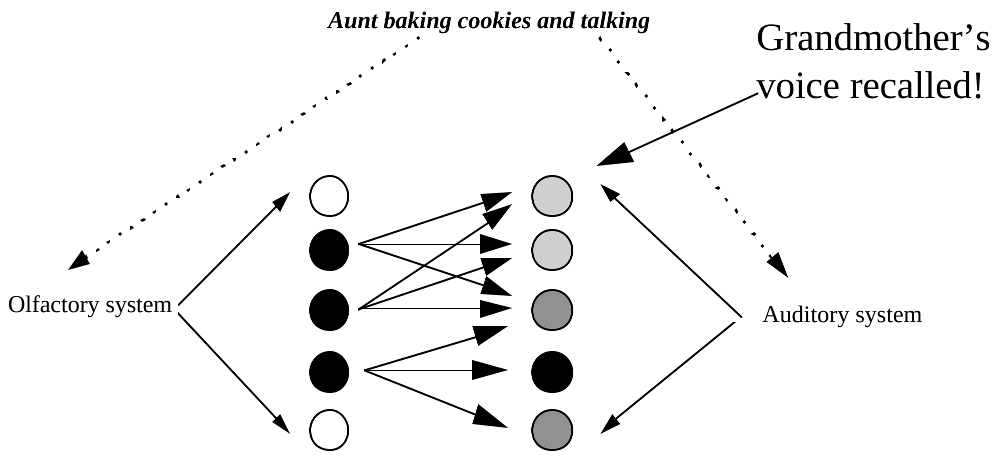


$$x(g_i) = \Sigma\, w_{ij} * x(f_j)$$

If you had an aunt that was cooking delicious stews and talking while she was cooking, you might form a second association between a smell and a voice!



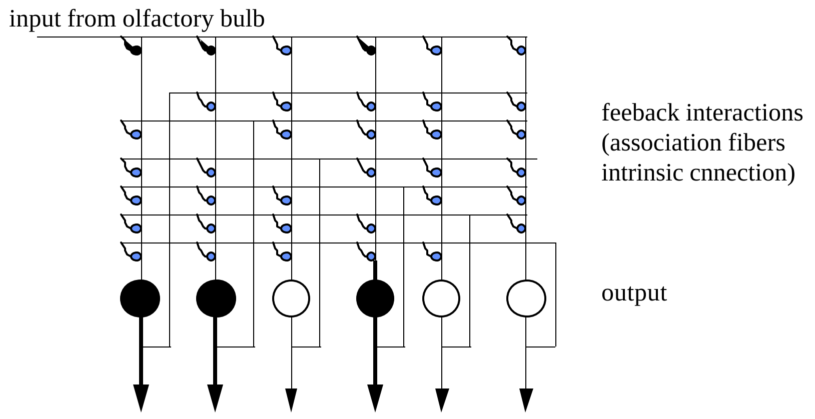
But if your aunt was baking cookies very similar to those your grandmother was baking and your brain was really functioning as a linear associator, you may become confused as to whose voice is whose!


Aunt baking cookies and talking
Grandmother's voice recalled!
Olfactory system
Auditory system

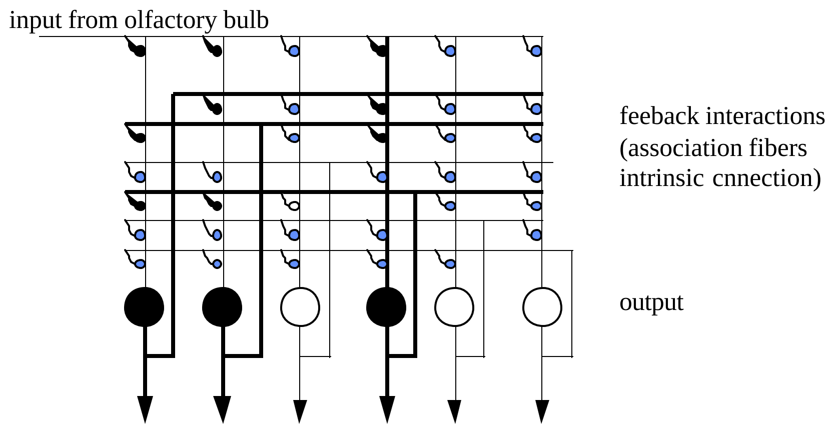
Examples will be calculated in class!

Auto-associative memory

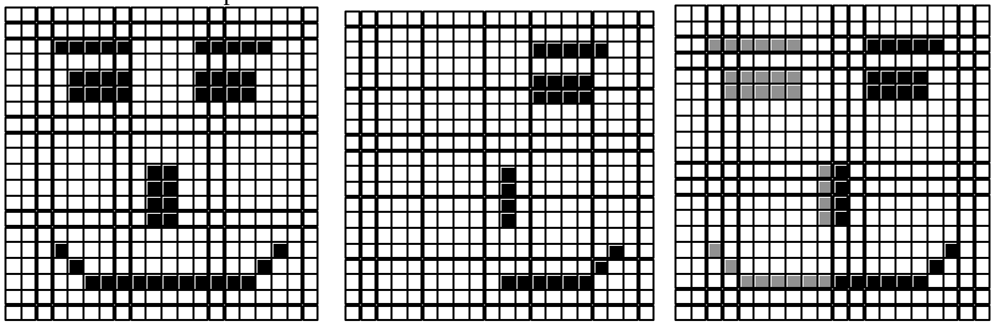
Next, we will further explore associative memory circuits and look at a different type of associative memory circuits called auto-associative memory. In an auto associative memory, different parts of a same pattern are associated with each other (as opposed to an associative memory network, in which two patterns are associated with each other).
Imagine that you smell a certain smell which activates a number of neurons in your olfactory cortex to various degrees:



Now imagine that excitatory synapses exist between all pairs of neurons in the cortical network (fully connected). If all of these synapses obey a hebbian learning rule, than the weights of all synapses between neurons that are simultaneously active will be increased.


input from olfactory bulb
feeback interactions (association fibers intrinsic cnnection)
output

Now if you reactivate the pattern only partially (you smell a similar or a weaker smell), because of the existing strong synapses, the original pattern will be reactivated.
An illustrative example is that of a face:



Once the pattern has been stored by creating strong excitatory connections between all pixels (neurons) that are active (black), a partial presentation of the pattern will lead the recall of the initial pattern.
A particularly important class of neural networks which fulfill associative memory functions are so called "Hopfield networks", which have a recurrent structure and the development of which is inspired by statistical physics. They share the following features:

- Nonlinear computing units (or neurons)
- Symmetric synaptic connections ($w_{ij} = w_{ji}$)
- No connections of a neuron on itself ($w_{ii} = 0$)
- Abundant use of feedback (usually, all neurons are connected to all others)

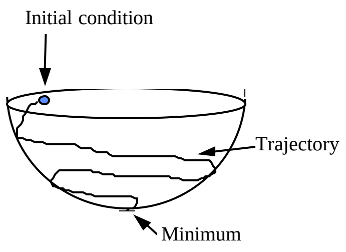
(Feedback means that a neurons sends a synapse to a neuron it also receives a signal from, so that there is a closed loop).
Most often (for a network with N units, where x_i is the activation value of unit i, w_{ij} the weight of the connection between unit j and unit i, and E an energy function):

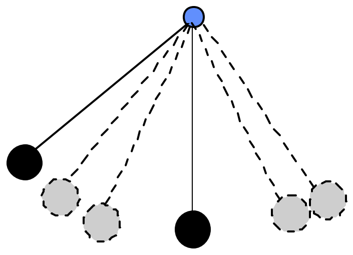
$$v_i = \sum_{j=1}^{N} w_{ij} * x_j \quad \text{and} \quad x_i = \{\frac{+1 \text{ if } \; v_i > 0}{-1 \text{ if } \; v_i < 0} \;[\text{if } \; v_i = 0 \text{ unit } j \text{ remains in its previous state}]$$

$$w_{ij} = \frac{1}{N} x_i * x_j \quad \text{if } i \neq j \qquad w_{ij} = 0 \text{ if } i = j$$

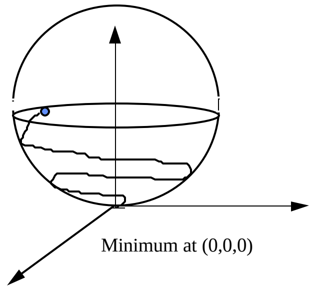
$$E = -\frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} w_{ij} * v_i * v_j$$

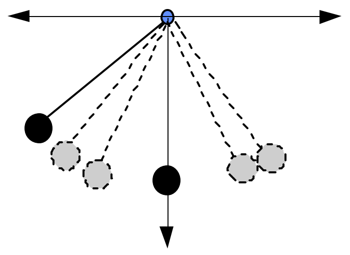
Among others, the Hopfield network is a recurrent network that embodies the storing of information in a dynamically stable configuration. Hopfields idea was that of locating each pattern that was to be stored at the bottom of the valley of an energy landscape, and then permitting a dynamical procedure to minimize the energy of the network in such a way that the valley becomes a basin of attraction.
Imagine that you place a marble on the wall of a container. It may role a round for a while (trajectory), but no matter where you throw it in (initial condition), will always end up on the bottom (minimum).



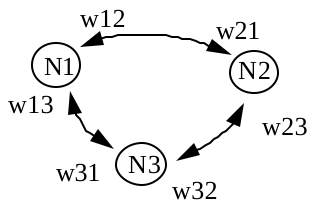


Similarly, if you have a pendulum, you can start it from any given location, it will always eventually end up in the same location no matter where you start it from. What changes from trial to trial is how long it takes to settle down.
In a neural network with binary threshold neurons (or McCulloch Pitts neurons) and symmetric weights, one can calculate the synaptic weights in such a way as to create a "minimum" into which the networks activity will settle. In the example above, the initial position, trajectories and the minimum are given by locations in space; in a neural network these would be characterized by the "activation pattern" or the vector representing the activation pattern of the neurons in the network. In physical examples, the minimum is defined by the physical dimensions as well as by the law of gravity (the marble will roll to the lowest possible location, the pendulum will stop at the location in which lateral forces on it are zero); in a neural network the minimum will be defined by the synaptic weights between the neurons.





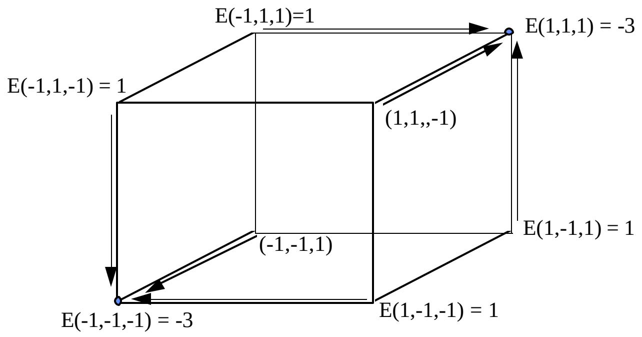
The idea is that in a network of binary threshold units (or McCulloch Pitts units) with symmetric weights (i.e. the weight of the connection from unit i to unit j is the same than that from unit j to unit i), there is a certain pattern of activation of these units that is very stable. For example, if all connection weights were to be +1, then the pattern for which all units have the value +1 is very stable. Once the units all have acquired that value, nothing will change anymore!


v1, v2, v3: inputs to neuron 1, 2
x1, x2, x3: outputs of neuron 1,2
w12, w21 etc: connection strength
	w12=w13=w21=w23=w31=w32=1.0
	A.
	t=0 x1=x2=x3=1 initial condition
	t=1 v1=w12*x2+w13*x3 = 1; x1=1
	v2=w21*x1+w23*x3 = 1; x2=1
	v3=w31*x1+w32*x2 = 1; x3=1
	This will never change, it's a "stable state"
	B.
	t=0 x1=x2=1; x3=-1 initial condition
	t=1 v1=w12*x2+w13*x3 = 0; x1=1 (stays as is)
	v2=w21*x1+w23*x3 = 0; x2=1 (stays as is)
	v3=w31*x1+w32*x2 = 2; x3=1
	This will never change, network reached a stable state!

Each particular pattern is called a “state” of the network, and the space of all possible states is called the state space (sometimes also phase space). For example, the state space of a two-neuron network comprises four different states: (-1,-1);(-1,1);(1,-1); and (1,1). For each distribution of connection weights in such a network, there exist a number of stable states, which are also referred to as attractors. In the example above, there are several stable states which coexist with the same connection weights.

t=0 x1=-1=x2=-1; x3=1 initial condition
t=1 v1=w12*x2+w13*x3 = 0; x1=-1 (stays as is)
 v2=w21*x1+w23*x3 = 0; x2=-1 (stays as is)
 v3=w31*x1+w32*x2 = -2; x3=-1
t=2 v1=w12*x2+w13*x3 = -2; x1=-1
 v2=w21*x1+w23*x3 = -2; x2=-1
 v3=w31*x1+w32*x2 = -2; x3=-1

As one can see, there are at least 2 stable states in this particular network, each with a given "basin of attraction". Initial conditions with two neurons active, one inactive, will go to (1,1,1) and initial conditions with two neurons inactive and one active will go to (-1,-1,-1).
An attractor is a point in the state pace to which the activation values of the units want to go, i.e. the attractor has a basin of attraction around it. This means that if for example the point (1,-1) in a two-dimensional space is a stable point, then if we perturb the network (perturb means to add noise, or change a value), in such a way for example that we give the units the values (1.2,-0.8), then the attractor “attracts” those values back, and because of the dynamics of the network the values will become (1,-1) again. Each trajectory terminates at a stable point.



The stable states or attractors correspond to valleys of the energy function (the energy function is given by the connection matrix, and has a given value for each pattern of unit activations). The stable states or attractors correspond to valleys of the energy function (the energy function is given by the connection matrix, and has a given value for each pattern of unit activations). In the example above, the energy function $E = -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} w_{ij} * x_i * x_j$ (more or less equivalent to $E = -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} w_{ij} * v_i * v_j$) corresponds to:

E = -1/2 * (w12*v1*v2+w13*v1*v3+w21*v2*v1+w23*v2*v3+w31*v3*v1+w32*v3*v2)
if at t=0 x1=-1=x2=-1; x3=1 then E(t=0) = -1
and at t=1 x1=x2=x3=x4 = -1 then E(t=1) = -3 which is < E(t=0).
Similarly, if at t=0 x1=1=x2=-1; x3=1 then E(t=0) = 1 and at t=1
 v1=w12*x2+w13*x3 = 2; x1=1
 v2=w21*x1+w23*x3 = 2; x2=1
 v3=w31*x1+w32*x2 = 0; x3=1
E(t=1) = -0.5*3 = -1.5 < E(t=0)

Spurious states or spurious attractors represent stable states of the network that are different from the memories of the network which served to calculate the connection matrix. They arise mainly for two reasons: (1) the energy function is symmetric, such that its values remain unchanged when all the units take on the opposite values of the stored memory (all +1’s are replaced by –1s). As a consequence, if the memory (-1,1,-1) is a stable state, then it’s opposite, (1,-1,1) is also a stable state; (2) There is an attractor for every mixture of stored patterns.

Does it work well? Not very well. Not many patterns can be stored, and those should really be orthogonal to be stored perfectly.
Is it "realistic"? Not really. First, it necessitates neurons that make both inhibitory and excitatory connections and second it needs symmetric weights.

Can one use it as a model for more realistic associative memory circuits? Certainly! Many brain circuits provide the essential ingredients: extensive feedback connections between excitatory neurons as well as a hebbian-like learning rule (LTP).