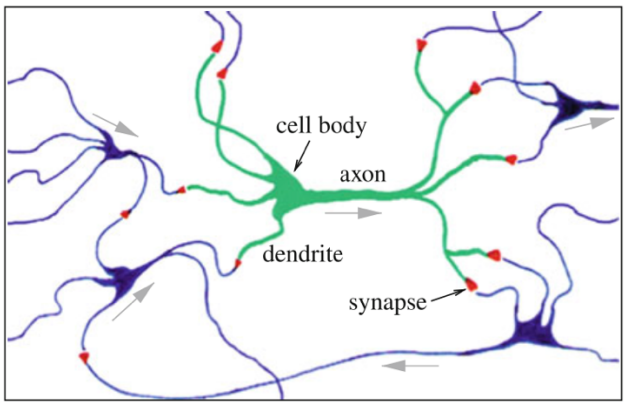
Information Technology for Digital Humanities
Lecture 8
October 16th 2024
Machine Learning

Neural Networks

Neural networks are networks of nerve cells found in the brains of humans and animals. The human brain contains approximately 100 billion nerve cells. We owe our intelligence and our ability to learn various motor and intellectual skills to the complex relay mechanisms and adaptability of the brain. For many centuries, biologists, psychologists, and physicians have tried to understand how the brain works. Around 1900, a revolutionary insight emerged: these tiny physical building blocks of the brain—nerve cells and their connections—are responsible for awareness, associations, thoughts, consciousness, and the ability to learn.
The first major step toward neural networks in AI was made in 1943 by McCulloch and Pitts, in a paper titled "A Logic Calculus of the Ideas Immanent in Nervous Activity." They were the first to present a mathematical model of the neuron as a basic switching element of the brain. This paper laid the foundations for the development of artificial neural networks and, consequently, for this important branch of AI.
We can think of the field of neural network modeling and simulation as the "bionic" branch of AI. (Bionics refers to the discipline that aims to convert discoveries in living nature into innovative technology.) Nearly all areas of AI attempt to recreate cognitive processes, such as logic or probabilistic reasoning. However, the tools used for modeling—namely mathematics, programming languages, and digital computers—have very little in common with the human brain. With artificial neural networks, the approach is different. Drawing from our understanding of the function of natural neural networks, we aim to model, simulate, and even reconstruct them in hardware form. Every researcher in this field faces the fascinating and exciting challenge of comparing their results with human performance.

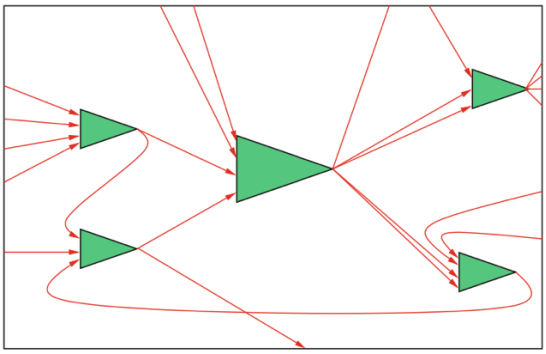


Structure and Function of Neurons

Each of the approximately 100 billion neurons in a human brain has, as shown in a simplified illustration below, the following structure and function. In addition to the cell body, a neuron has an axon, which can form local connections with other neurons through their dendrites. However, the axon can also grow up to a meter in length, taking the form of a nerve fiber that extends throughout the body.
The neuron's cell body can store small electrical charges, similar to a capacitor or a battery. This memory is charged by incoming electrical impulses from other neurons. The stronger

the electrical impulse, the higher the voltage. If the voltage exceeds a certain threshold, the neuron activates, meaning it releases its stored energy by sending a spike along the axon and synapses. This electrical current splits and reaches many other neurons through the synapses, where the same process is repeated.

This leads to the question of how the neural network is structured. Each of the approximately 10^{11} neurons in the brain is connected to about 1,000 to 10,000 other neurons, resulting in a total of more than 10^{14} connections. Considering that this vast number of delicate connections is made up of soft, three-dimensional tissue, and that experiments on the human brain are not easy to conduct, it becomes clear why we do not have a detailed circuit diagram of the brain. Given the brain's immense size and complexity, it is likely that we will never fully understand its complete circuit layout.



Adaptive Structure of the Brain and the Role of Synapses

From today's perspective, it is no longer worthwhile to attempt to create a complete circuit diagram of the brain, because its structure is adaptive. The brain changes and adjusts based on individual activities and environmental influences. Synapses, which form the connections between neurons, play a central role in this process. At the point where two neurons connect, it's as if two wires meet. However, these two conductors are not perfectly connected; there is a small gap between them that electrons cannot directly cross.
This gap is filled with chemicals called neurotransmitters. These neurotransmitters can become ionized when a voltage is applied, allowing them to carry a charge across the gap. The conductivity of this space depends on many parameters, such as the concentration and chemical composition of the neurotransmitter. It's important to understand that the brain's functioning is highly sensitive to changes in these synaptic connections—this is evident, for example, in the effects of alcohol or other drugs.
How does learning occur in a neural network like this? Interestingly, it's not the active units—the neurons themselves—that are adaptive, but rather the connections between them, the synapses. Specifically, the conductivity of these synapses can change. We know that a synapse becomes stronger the more electrical current it carries. A stronger synapse means it has higher conductivity. Synapses that are frequently used gain increasing importance (or *weight*). On the other hand, synapses that are rarely used or inactive see their conductivity diminish over time, which can even lead to their death.

Asynchronous Processing and Parallelism in the Brain

All neurons in the brain operate asynchronously and in parallel, but at a much lower speed compared to a computer. A neural impulse lasts about one millisecond, which is also the time it takes for ions to cross the synaptic gap. As a result, a neuron's clock frequency is less than

one kilohertz, making it about 10^6 times slower than modern computers. However, this disadvantage is more than offset in many complex cognitive tasks, such as image recognition, due to the brain's high degree of parallel processing across its neural network.

The brain connects with the external world through sensory neurons, such as those in the retina of the eyes, or through nerve cells with very long axons that extend from the brain to the muscles, enabling actions like moving a leg.
However, it remains unclear how the principles we've discussed enable intelligent behavior. Like many researchers in the field of neuroscience, we will try to provide some insights through simulations using simple mathematical models. This will help us understand how cognitive tasks, such as pattern recognition, become possible.

The Mathematical Model

First, we replace the continuous time axis with a discrete time scale. Neuron *i* performs the following calculation in a single time step. The "loading" of the activation potential is achieved simply by summing the weighted output values (x_1, ..., x_n) from all incoming connections, using the following formula:

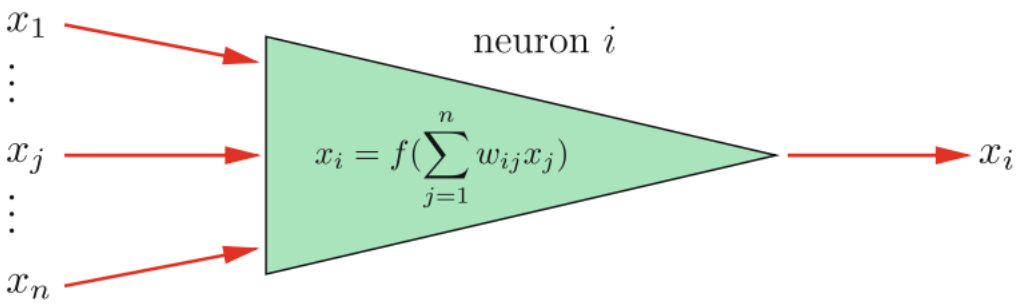
$$\sum_{j=1}^{n} w_{ij} x_j .$$

Next, an activation function *f* is applied, and the result is:

$$x_i = f\left(\sum_{j=1}^{n} w_{ij} x_j\right)$$

The result is then transmitted to neighboring neurons as output.

In the following figure, this type of modeled neuron is shown.



The structure of a formal neuron, which applies the activation function f to the weighted sum of all inputs.

There are several options for the activation function.
One is the threshold function (Heaviside step function, named after Oliver Heaviside, 1850-1925, an English mathematician, physicist, and electrical engineer):

$$H_\Theta(x) = \begin{cases} 0 & \text{if } x < \Theta, \\ 1 & \text{else.} \end{cases}$$

The neuron then calculates its output as follows:

$$x_i = \begin{cases} 0 & \text{if } \sum_{j=1}^{n} w_{ij} x_j < \Theta, \\ 1 & \text{else.} \end{cases}$$

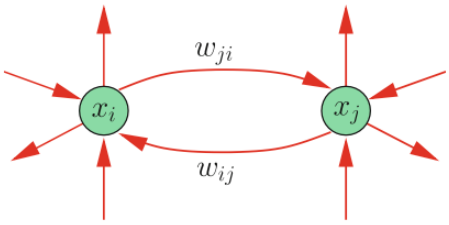
The activation of a neuron can only take on the values of 0 or 1.
Modeling learning is essential for neural networks theory. As mentioned earlier, one way learning can occur is by strengthening a synapse based on how many electrical impulses it must transmit. This principle was proposed by Donald Hebb (1904-1985, a Canadian neuropsychologist) in 1949 and is known as Hebb's rule:
If a connection w_{ij} exists between neuron j and neuron i, and repeated signals are sent from neuron j to neuron i, resulting in both neurons being active simultaneously, the weight w_{ij} is strengthened.
A possible formula for the change in weight Δw_{ij} is:

$$\Delta w_{ij} = \eta x_i x_j$$

where the η factor (called "learning rate") determines the impact of learning steps.
There are many variations of this rule, which result in different types of networks or learning algorithms.

Hopfield Network

Looking at Hebb's rule, we see that for neurons whose outputs are 0 and 1, the weights can only increase over time. It is not possible for a neuron to weaken or even die according to this rule. This can be modeled, for example, by introducing a decay constant that weakens an unused weight over time.
This issue is addressed differently in the model presented by Hopfield in 1982. It uses binary neurons with two values: -1 for inactive and 1 for active. Using Hebb's rule, a positive contribution to the weight occurs whenever two neurons are active simultaneously. However, if only one of the two neurons is active and the other is inactive, Δw_{ij} is negative, that is, the weight between the two neurons decreases.
Hopfield networks are based on this idea: the patterns to be learned are stored in the form of weight configurations among the neurons. To retrieve a stored pattern, it is enough to provide a similar pattern. The system will then find the closest matching saved pattern. A classic application of this approach is handwriting recognition.
In the learning phase of a Hopfield network, N encoded binary patterns, stored in vectors (sequences of bits of fixed size) q^1, ..., q^N, must be learned. Each component $q_r^j \in \{-1; 1\}$ of vector q^j represents a pixel of a pattern. For vectors constituted by n pixels, a neural network of n neurons will be used, that is, one neuron for each pixel. Each neuron is connected to all other neurons, and the connections between two neurons have the same weight in both directions ($w_{ij} = w_{ji}$) but no neuron is connected to itself ($w_{jj} = 0$).



Connections in a Hopfield network

It is possible to learn N patterns by simply calculating all the weights using the formula:

$$w_{ij} = \frac{1}{N} \sum_{k=1}^{N} q_i^k q_j^k.$$

This formula shows an interesting relationship with Hebb's rule. Each pattern in which pixels i and j have the same value contributes positively to the weight w_{ij}. Every other pattern gives a negative contribution. Since each pixel corresponds to a neuron, the weights between neurons that have the same value simultaneously are reinforced here.
Once all the patterns have been stored, the network can be used for pattern recognition. We provide the network with a new pattern **x** and update the activations of all neurons asynchronously according to the rule:

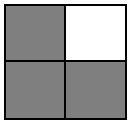
$$x_i = \begin{cases} -1 & \text{if } \sum_{\substack{j=1 \\ j \neq i}}^{n} w_{ij} x_j < 0, \\ 1 & \text{else} \end{cases}$$

until the network becomes stable, that is, until the activations no longer change. This way of operating is expressed by the following algorithm:

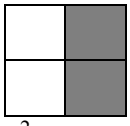
HOPFIELDASSOCIATOR($\boldsymbol{q}$)
Initialize all neurons: $\boldsymbol{x} = \boldsymbol{q}$
Repeat
  i = Random(1, n)
  Update neuron i according to
Until $\boldsymbol{x}$ converges
Return ($\boldsymbol{x}$)

Exercise

Train a Hopfield network with the following patterns:



q^1



q^2

Each pixel corresponds to a neuron in the network, so we will have a network with 4 neurons, each connected to all the others but not to itself. Therefore, we will have $4^2 - 4 = 12$ connections (synapses) to which we associate 12 weights.
We represent the "active" pixels (in gray) with the value +1 and the "inactive" pixels (in white) with -1, and we represent the 2 patterns in vector form, reading the values row by row, from top to bottom, from left to right:
$q^1 = [1, -1, 1, 1]$
$q^2 = [-1, 1, -1, 1]$.
Let's use the formula for calculating the weights of the connections. The basic idea is that the connections between neurons with the same value contribute positively, while those between neurons with different values contribute negatively.

$$w_{ij} = \frac{1}{N} \sum_{k=1}^{N} q_i^k q_j^k .$$

Let's create the weight matrix W, keeping in mind that Hopfield networks do not allow neurons to be connected to themselves ($w_{ii} = 0$) and that the connections are symmetric ($w_{ij} = w_{ji}$). Remembering that:
$q^1 = [1, -1, 1, 1]$
$q^2 = [-1, 1, -1, 1]$
we have that
$w_{11} = 0$
$w_{12} = w_{21} = (q^1_1 * q^1_2 + q^2_1 * q^2_2)/2 = (-1 -1)/2 = -1$
$w_{13} = w_{31} = (q^1_1 * q^1_3 + q^2_1 * q^2_3)/2 = (1 +1)/2 = 1$
$w_{14} = w_{41} = (q^1_1 * q^1_4 + q^2_1 * q^2_4)/2 = (1 -1)/2 = 0$
$w_{22} = 0$
$w_{23} = w_{32} = (q^1_2 * q^1_3 + q^2_2 * q^2_3)/2 = (-1 -1)/2 = -1$
$w_{24} = w_{42} = (q^1_2 * q^1_4 + q^2_4 * q^2_2)/2 = (-1 +1)/2 = 0$
$w_{33} = 0$
$w_{34} = w_{43} = (q^1_3 * q^1_4 + q^2_4 * q^2_3)/2 = (1 -1)/2 = 0$
$w_{44} = 0$

0	-1	1	0
-1	0	-1	0
1	-1	0	0
0	0	0	0

Now the network has "learned" the 2 patterns q^1 and q^2. The acquired "information" is not in the values assumed by the neurons but in the weights. The neurons can be set to new input values. The network responds to these values and modifies them according to the rule:

$$x_i = \begin{cases} -1 & \text{if } \sum_{\substack{j=1 \\ j \neq i}}^{n} w_{ij} x_j < 0, \\ 1 & \text{else} \end{cases}$$

Let's see what happens when we input a value x that was previously learned by the network. ($x = q^1$).

$x = [\,1, -1, 1, 1]$

$x_{1new} = \text{sgn}(w_{12}*x_2 + w_{13}*x_3 + w_{14}*x_4) = \text{sgn}(-1*-1 + 1*1 + 0*1) = \text{sgn}(2) = 1$
$x_{2new} = \text{sgn}(w_{21}*x_1 + w_{23}*x_3 + w_{24}*x_4) = \text{sgn}(-1*1 + -1*1 + 0*1) = \text{sgn}(-2) = -1$
$x_{3new} = \text{sgn}(w_{31}*x_1 + w_{32}*x_2 + w_{34}*x_4) = \text{sgn}(1*1 + -1*-1 + 0*1) = \text{sgn}(2) = 1$
$x_{4new} = \text{sgn}(w_{41}*x_1 + w_{42}*x_2 + w_{43}*x_3) = \text{sgn}(0*1 + 0*-1 + 0*1) = \text{sgn}(0) = 1$

$x_{new} = [\,1, -1, 1, 1]$

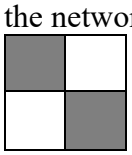
The new values of the neurons are identical to those given as input: the configuration is stable. Metaphorically speaking, the network has "recognized" a pattern it knows.
The same goes for $x = q^2$.

$x = [\,-1, 1, -1, 1]$

$x_{1new} = \text{sgn}(w_{12}*x_2 + w_{13}*x_3 + w_{14}*x_4) = \text{sgn}(-1*1 + 1*-1 + 0*1) = \text{sgn}(-2) = -1$
$x_{2new} = \text{sgn}(w_{21}*x_1 + w_{23}*x_3 + w_{24}*x_4) = \text{sgn}(-1*-1 + -1*-1 + 0*1) = \text{sgn}(2) = 1$
$x_{3new} = \text{sgn}(w_{31}*x_1 + w_{32}*x_2 + w_{34}*x_4) = \text{sgn}(1*-1 + -1*1 + 0*1) = \text{sgn}(-2) = -1$
$x_{4new} = \text{sgn}(w_{41}*x_1 + w_{42}*x_2 + w_{43}*x_3) = \text{sgn}(0*-1 + 0*1 + 0*-1) = \text{sgn}(0) = 1$

$x_{new} = [\,-1, 1, -1, 1] = x = q^2$

Now let's see what happens when we input a configuration different from those learned by the network. For example, the following configuration:



$x = [1, -1, -1, 1]$

Let's calculate the output (this time with a more visual notation, including the weight matrix):

0	-1	1	0
-1	0	-1	0
1	-1	0	0
0	0	0	0

Computation of x_{1new}:

sgn(0*1+	-1*-1+	1*-1+	0*1)=1
-1	0	-1	0
1	-1	0	0
0	0	0	0

Computation of the other components of x_{new}:

sgn(0*1+	-1*-1+	1*-1+	0*1)=1
sgn(-1*1+	0*-1+	-1*-1+	0*1)=1
sgn(1*1+	-1*-1+	0*-1+	0*1)=1
sgn(0*1+	0*-1+	0*-1+	0*1)=1

Watch out: x_{new} ([1,1,1,1]) is different from x. The network is not stable, that is, it hasn't converged yet. We need to go on with the computation and check how the neurons are behaving.

Let's compute x_{new2}:

sgn(0*1+	-1*1+	1*1+	0*1)= 1
sgn(-1*1+	0*1+	-1*1+	0*1)= -1
sgn(1*1+	-1*1+	0*1+	0*1)= 1
sgn(0*1+	0*1+	0*1+	0*1)= 1

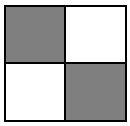
We obtain x_{new2} = [1, -1, 1, 1]. Again, an output different from the input, hence we need to do another round of computation.

Let's compute x_{new3}:

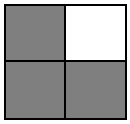
sgn(0*1+	-1*-1+	1*1+	0*1)= 1
sgn(-1*1+	0*-1+	-1*1+	0*1)= -1
sgn(1*1+	-1*-1+	0*1+	0*1)= 1
sgn(0*1+	0*-1+	0*1+	0*1)= 1

This time x_{new3} = x_{new2}, this means the network has reached a stable configuration (further computation would give this same result).

Notice that $x_{new3} = x_{new2} = q^1$, which means that the following happens.
Given this in input



the network, after some computation steps, converges to:



This behavior can be interpreted as the network performing a correction. An input arrives that resembles q^1 but with one "corrupted" bit, and the network is still able to recognize the correct configuration.

Let's now see what happens if we input the following pattern:

x = [-1,1,1,-1]

Let's compute x_{new}:

sgn(0*-1+ -1*1+ 1*1+ 0*-1)=1

sgn(-1*-1+ 0*1+ -1*1+ 0*-1)=1

sgn(1*-1+ -1*1+ 0*1+ 0*-1)=-1

sgn(0*-1+ 0*1+ 0*1+ 0*-1)=1

x_{new}([1,1,-1,1]) is different from x([-1,1,1,-1]).
The network is not stable, hence we repeat the computation with x_{new} as input:

sgn(0*1+ -1*1+ 1*-1+ 0*1)=-1

sgn(-1*1+ 0*1+ -1*-1+ 0*1)=1

sgn(1*1+ -1*1+ 0*-1+ 0*1)=1

sgn(0*1+ 0*1+ 0*-1+ 0*1)=1

x_{new2} ([-1,1,1,1]) is different from x_{new}, hence we compute further:

sgn(0*-1+ -1*1+ 1*1+ 0*1)=1

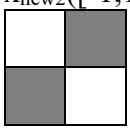
sgn(-1*-1+ 0*1+ -1*1+ 0*1)=1

sgn(1*-1+ -1*1+ 0*1+ 0*1)=-1

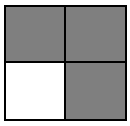
sgn(0*-1+ 0*1+ 0*1+ 0*1)=1

We obtain x_{new3}=[1,1,-1,1], which is the same as x_{new}.

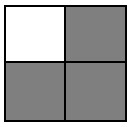
This means that the network will continue to oscillate between $x_{new}([1,1,-1,1])$ and $x_{new2}([-1,1,1,1])$, that is, giving in input



the network oscillates forever between

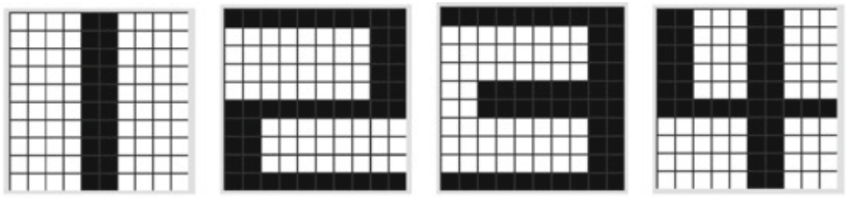


and

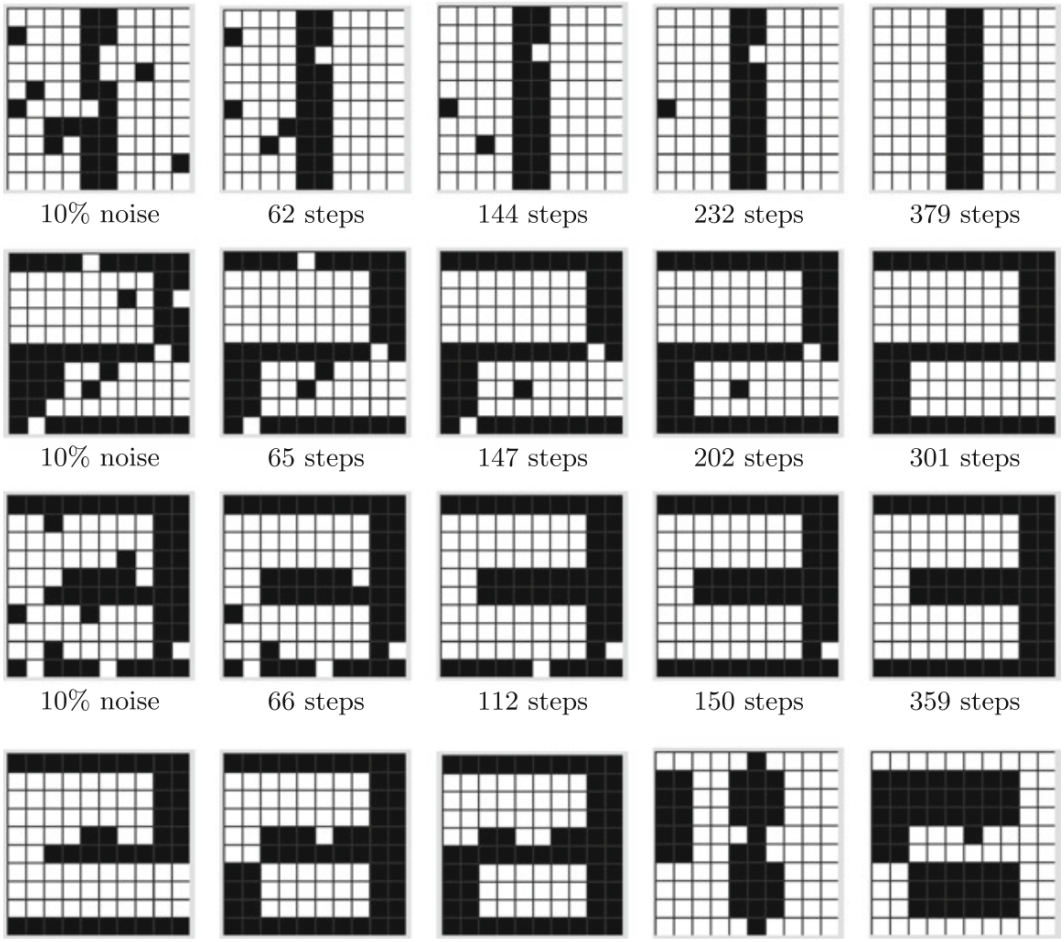


Example of Application to Character Recognition

First, the models for the digits 1, 2, 3, and 4 are used for training, that is, the weights are calculated. Then, we input a pattern with added noise and allow the Hopfield network dynamics to run until convergence.
In rows 2 to 4 of the figure below, five snapshots of the network's development during recognition are shown.
With 10% noise, all four learned models are reliably recognized.
With more than 20% noise, the algorithm often converges to other learned models or even to configurations that were not part of the training set.
This demonstrates the network's robustness to small amounts of noise, but also its limitations when the noise level is too high.



The four training examples learned by the network.



Several stable states of the network which were not learned.

Bibliography

D.E. Rumelhart, G.E. Hinton, and R.J.Williams, “Learning Internal Representations by Error Propagation” in D. Rumelhart and J. McClelland (eds) *Parallel Distributed Processing*, volume 1. MIT Press, 1986

M. A. Nielsen, “Neural Networks and Deep Learning”, Determination Press, 2015