Elements on J. Hopfield Networks and G. Hinton's Boltzmann Machines

Jean-Eric Campagne
Université Paris-Saclay, CNRS/IN2P3, IJCLab, 91405 Orsay, France
jean-eric.campagne@ijclab.in2p3.fr

Abstract

The 2024 Nobel Prize in Physics[1] was awarded to John J. Hopfield and Geoffrey E. Hinton for their pioneering work in the 1980s, which laid the foundation for the significant advancements that followed in the field of artificial machine learning. Their two contributions noted by the Nobel committee, namely *Hopfield networks* and *Boltzmann Machines*, are presented here along with some numerical illustrations[2].

***K*eywords** Hopfield network · Boltzmann Machine · Machine Learning

1 Introduction

A selection of neural architectures is shown in Figure 1. This poster reveals increasingly complex neural networks as one explores its contents. We will focus exclusively on the architectures developed by John J. Hopfield (Hopfield, 1982) and Geoffrey E. Hinton (Hinton and Sejnowski, 1986) in the early 1980s. Since then, it is undeniable that deep neural network architectures have gained popularity due to their adaptability to various challenges and especially their superior performance, which has surpassed previous methods, particularly after the implementation of *convolutional operators* by Yann LeCun & Yoshua Bengio (Lecun and Bengio, 1995)[3] for automatic recognition of handwritten digits. Today, generative models[4] are the topic of much discussion not only among experts but also among the general public.

Let us state right away that these advances were made possible through collaboration across various disciplines and technologies. The *end-to-end* approach has also become a widely used leitmotif. However, understanding the exact nature of these architectures remains an area of mathematical research. Those seeking a deeper exploration of these topics may refer, for example, to the Collège de France lectures by Professor S. Mallat[5].

This document does not aim to be either a mathematical treatise or a course in statistical physics. References to more specialized articles and books are provided. In the following, we will first give a brief overview of the "building block" of an artificial neuron (Section 2), followed naturally by an introduction to Hopfield networks (Section 3) and Boltzmann Machines (Section 4).

[1] `https://www.nobelprize.org/prizes/physics/2024/press-release/`

[2] The companion site `https://github.com/jecampagne/demo-hopfield-rbm-networks` hosts `Python` scripts for reproducing the simulations on the `Google Colab` platform.

[3] See also `http://yann.lecun.com/exdb/lenet/index.html`. Note that Y. LeCun, Y. Bengio, and G. Hinton received the 2019 Turing Award `https://awards.acm.org/turing/award-recipients`

[4] For example, `Gemini` `https://gemini.google.com/`, `ChatGPT` `https://chatgpt.com/`, `Midjourney` `https://www.midjourney.com/`...

[5] `https://www.college-de-france.fr/fr/chaire/stephane-mallat-sciences-des-donnees-chaire-statutaire/events`. A companion website that provides lecture notes in both French and English, along with `Python` scripts illustrating the course, is also available at `https://github.com/jecampagne/cours_mallat_cdf`.

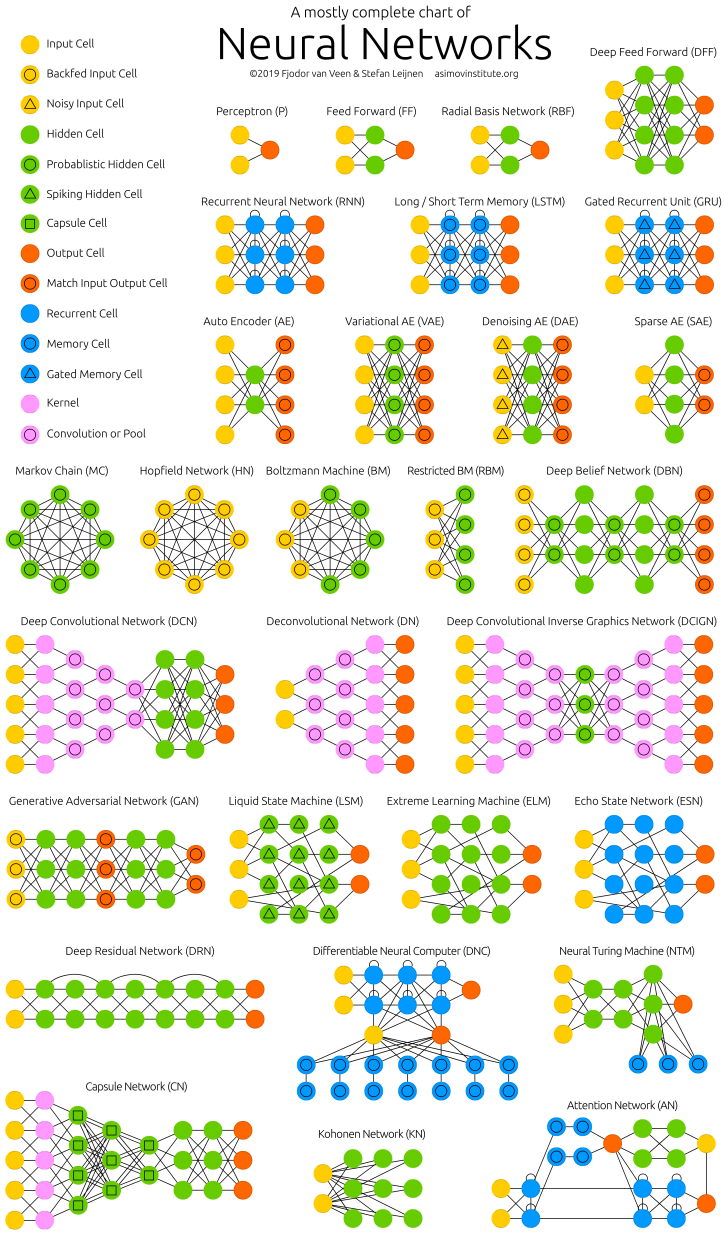

Figure 1: Different neural network architectures (`https://www.asimovinstitute.org/neural-network-zoo/`).

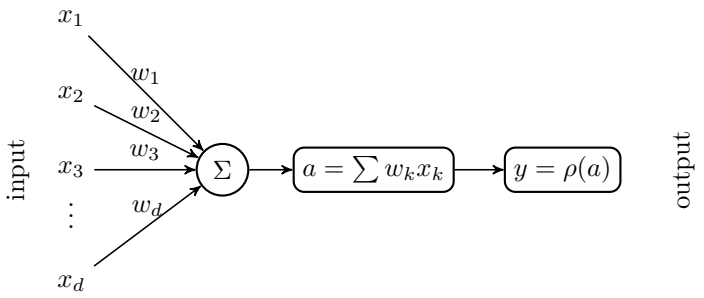

Figure 2: Simple schematic of a neuron's function, which from the inputs $\{x_i\}_{i\leq d}$ and a weight vector $\{w_i\}_{i\leq d}$ provides an output y. Generalizing to a neuron with multiple outputs is straightforward. The function ρ is a pointwise non-linearity.

2 The Basic Building Block: The Model of a Single Neuron

Let us consider the case of a single neuron studied by Frank Rosenblatt as early as 1957 (Rosenblatt, 1957):

1. Its *architecture* is shown in Figure 2; it has an *input* vector $x \in \mathbb{R}^d$, which it combines linearly with a vector of the same size $\boldsymbol{w}$. A constant may be added to this combination, or it can be included in the definition of $\boldsymbol{w}$ with $x(0) = 1$. The *activation* of the neuron is then defined as
$$a_i = \sum_j w_{ij} x_j \tag{1}$$

2. which is followed by an *update* procedure of the neuron using a non-linear function ρ, which can be either deterministic, e.g., sigmoid, tanh, ReLU, sine, or probabilistic, e.g., logistic. This *activation-update* (most often simply called activation) constitutes the *output* of the neuron:
$$y_i = \rho(a_i) \tag{2}$$

Now let us consider samples $\{x_i, y_i\}_{i\leq n}$ for which, for each input $x_i \in \mathbb{R}^d$, the output $y_i \in \{0, 1\}$ is known. One can pose the question as follows: is the probabilistic activation neuron, parameterized by
$$h_w(x) = P(y = 1|x; \boldsymbol{w}) = 1 - P(y = 0|x; \boldsymbol{w}), \tag{3}$$
capable of storing enough *information* to associate each y_i with its corresponding x_i from the sample set? This requires a kind of memory imprint in the values of the weights $\boldsymbol{w}$. To achieve this, we train it as a *binary classifier*.

The probability of y being 0 or 1 is given by
$$P(y|x; \boldsymbol{w}) = h_w(x)^y (1 - h_w(x))^{1-y}, \tag{4}$$
and we form the following likelihood, assuming that all samples are independent:
$$\mathcal{L}(\boldsymbol{w}) = \prod_{i=1}^{n} P(y_i|x_i; \boldsymbol{w}), \tag{5}$$
for which we seek a maximum to find the value of $\boldsymbol{w}$. To use minimization algorithms, we instead use $-\log \mathcal{L}(\boldsymbol{w})$, which we normalize by the number of samples n. Thus, we form the cost function (*loss*)
$$J(\boldsymbol{w}) = -\frac{1}{n} \sum_{i=1}^{n} \{y_i \log(h_w(x_i)) + (1 - y_i) \log(1 - h_w(x_i))\} \tag{6}$$

Now, we need to specify an expression (a model) for $h_w(x)$ (that is, for $P(y = 1|x; \boldsymbol{w})$). In principle, one might consider something like[6]
$$h_w(x) = \rho(\langle x, \boldsymbol{w} \rangle) = \frac{1}{2}\Big(1 + \mathrm{sgn}(\langle x, \boldsymbol{w} \rangle)\Big) \tag{7}$$

[6] the dot product is denoted by $\langle ., . \rangle$.

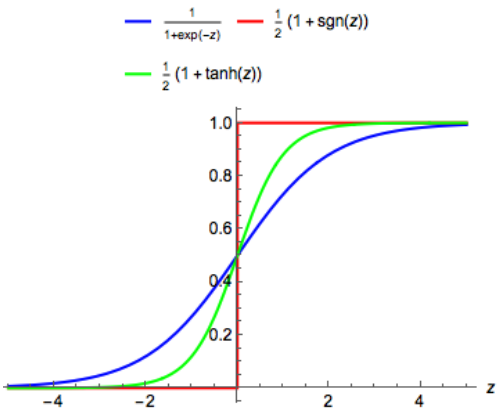

Figure 3: Various functions $h_w(z) = \rho(\langle x, \boldsymbol{w}\rangle)$ for performing likelihood minimization.

where $\mathrm{sgn}(x)$ is the "sign of x" function. Indeed, if we consider a separating surface between the two populations labeled "0" or "1," the vector $\boldsymbol{w}$ corresponds to the normal to this hyper-surface, so the sign is $+$ for the 1s and $-$ for the 0s.

However, this expression cannot be used as an argument of the logarithm; this is why David Cox introduced[7] the *logistic function* (or *sigmoid*) (Figure 3) as an activation function in this type of problem in 1958:

$$h_w(x) = \rho(\langle x, \boldsymbol{w}\rangle) = \frac{1}{1 + e^{-\langle x, \boldsymbol{w}\rangle}} \tag{8}$$

This formulation can be justified for our purposes within the framework of a Gibbs distribution (see Section 4). To proceed with *logistic regression*, a regularization term needs to be added to the cost function (Equation 6) to address potential overfitting. This is done by introducing a penalty, for example, using the L2 norm[8], meaning that $J(\boldsymbol{w})$ then becomes

$$J_r(\boldsymbol{w}) = -\frac{1}{n}\sum_{i=1}^{n}\Big\{y_i \log(h_w(x_i)) + (1 - y_i)\log(1 - h_w(x_i))\Big\} + \lambda\|\boldsymbol{w}\|_2^2 \tag{9}$$

Several observations can be made from here regarding potential generalizations, connections to Statistical Mechanics, and the perspective of the neuron as a memory rather than a classifier.

1. A first observation concerns the generalization of the previous expression[9] as

$$J_r(\boldsymbol{w}) = \frac{1}{2}\|\boldsymbol{w}\|_2^2 + C\sum_{i=1}^{n}\Big\{y_i \mathrm{cost}_1(\langle x_i, \boldsymbol{w}\rangle) + (1 - y_i)\mathrm{cost}_0(\langle x_i, \boldsymbol{w}\rangle)\Big\} \tag{10}$$

   The functions $\mathrm{cost}_1(z)$ for the logistic and tanh expressions are shown in Figure 4 in blue and green, respectively. The case of the red curve expresses a threshold effect, associated with the *hinge loss* related to the *margin* in "Support Vector Machine" (SVM) methods developed by Vladimir Vapnik and collaborators, initially in 1962-64 (Vapnik and Chervoneva, 1964)[10] and then in the 1990s (Cortes and Vapnik, 1995)

$$\mathrm{cost}_1^{hinge}(\langle x_i, \boldsymbol{w}\rangle) = \begin{cases} 1 - \langle x_i, \boldsymbol{w}\rangle & \langle x_i, \boldsymbol{w}\rangle < 1 \\ 0 & \langle x_i, \boldsymbol{w}\rangle \geq 1 \end{cases} \tag{11}$$

   The three functions (Figure 4) are upper bounds[11] for the "sign" function shown in black. They allow the problem to be made convex, which simplifies minimization.

[7] Incidentally, D. Cox considered Joseph Berkson (1899-1982) to be the inventor of *logistic regression* in 1955. Similarly, one could also mention biologist Raymond Pearl (1879-1940), who used it in the 1920s to describe the growth curve of a population over time (logistic curve).

[8] Note: The L2 norm is defined as $\|\boldsymbol{x}\|_2 = \sqrt{\sum_{j=1}^{d}|x_i|^2}$.

[9] In the literature, λ is often factored out of the $\{\}$, and $1/(2\lambda) = C$ is used instead.

[10] Originally written in Russian, this paper was later translated into English.

[11] Note: The logistic/tanh cost functions are scaled by $1/\log(2)$ here, which does not impact minimization.

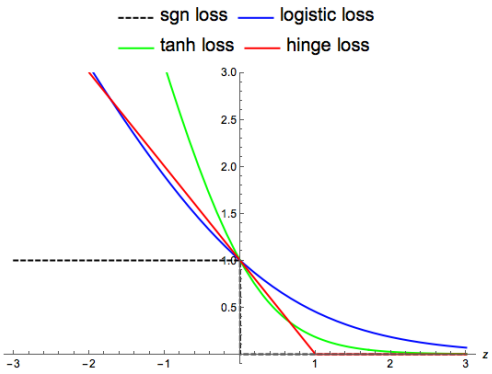

Figure 4: Candidate risk/cost functions.

In the case of minimizing $J_r(\boldsymbol{w})$, to interpret the rationale behind choosing cost_1^{hinge}, let's consider the case where $C \gg 1$. The problem then becomes one of minimizing

$$\min_{\boldsymbol{w}} \left(\frac{1}{2} \|\boldsymbol{w}\|^2 \right) \tag{12}$$

under the *constraints* that cancel out the term in C as follows:

$$\begin{cases} \langle x_i, \boldsymbol{w} \rangle \geq 1 & \text{if } y_i = 1 \\ \langle x_i, \boldsymbol{w} \rangle \leq -1 & \text{if } y_i = 0 \end{cases} \tag{13}$$

This reproduces the margin criterion of Support Vector Machines in the case $C \gg 1$.

2. A second remark concerns the probabilistic interpretation of the function $J_r(\boldsymbol{w})$ (Equation 9) that we minimize to derive the optimal vector $\boldsymbol{w}$. We have rewritten it as (Equation 6):

$$J_r(\boldsymbol{w}) = J(\boldsymbol{w}) + \lambda R(\boldsymbol{w}) \tag{14}$$

Now, $J(\boldsymbol{w})$ is the log-likelihood

$$P(\{y_i\}|\{x_i\}, \boldsymbol{w}) = \frac{1}{Z_J(\boldsymbol{w})} \exp\{-J(\boldsymbol{w})\} \tag{15}$$

Thus, the regularization part $\lambda R(\boldsymbol{w})$ can be interpreted as the logarithm of a *prior*:

$$P(\boldsymbol{w}|\lambda) = \frac{1}{Z_R(\lambda)} \exp\{-\lambda R(\boldsymbol{w})\} \tag{16}$$

Note that in the case of L2 regularization, this *prior* corresponds to a Gaussian distribution with $\sigma_w^2 = 1/\lambda$, so the normalization $Z_R(\lambda)$ equals $(2\pi/\lambda)^{d/2}$ (recall: $\boldsymbol{w}$ is of dimension d). Therefore, we can construct the posterior probability of the vector $\boldsymbol{w}$ given the training set and the value of λ. It follows that

$$P(\boldsymbol{w}|D_{tr}) = \{x_i, y_i\}, \lambda) = \frac{P(\{y_i\}|\{x_i\}, \boldsymbol{w}) P(\boldsymbol{w}|\lambda)}{P(\{y_i\}, \lambda)} = \frac{1}{Z_{J_r}(\boldsymbol{w})} \exp\{-J_r(\boldsymbol{w})\} \tag{17}$$

In this context, the optimum $\boldsymbol{w}^*$ is the value that is the most probable of $P(\boldsymbol{w}|D_{tr})$, which translates to

$$\boldsymbol{w}^* = \underset{\boldsymbol{w}}{\text{argmax}}\ P(\boldsymbol{w}|D_{tr}) = \underset{\boldsymbol{w}}{\text{argmin}}\ (-\log P(\boldsymbol{w}|D_{tr})) = \underset{\boldsymbol{w}}{\text{argmin}}\ J_r(\boldsymbol{w}), \tag{18}$$

where $\boldsymbol{w}^*$ is also the value of $\boldsymbol{w}$ that minimizes the regularized cost function in the deterministic domain. The dual perspective on the study framework allows us to establish a bridge between *Ridge Regression* and *Bayesian Ridge Regression* in the literature (see for example Bishop, 2006). Note the similarity of $P(\boldsymbol{w}|D_{tr})$ to the Gibbs function where $-J_r(\boldsymbol{w})$ would be considered as an energy to minimize; we will return to this point regarding Boltzmann Machines (Section 4).

Finally, in Bayesian terms, if a new value x_{new} is presented at the input of the neuron, it will produce the output $y_{new} = h_{w^*}(x_{new})$ in a deterministic manner, while the probability distribution of y_{new} conditioned on the new value, the training set, and the value of the hyper-parameter λ is as follows:

$$P(y_{new}|x_{new}, D_{tr}, \lambda) = \int P(y_{new}|x_{new}, \boldsymbol{w}, \lambda) P(\boldsymbol{w}|D_{tr}, \lambda)\ \mathrm{d}\boldsymbol{w} \tag{19}$$

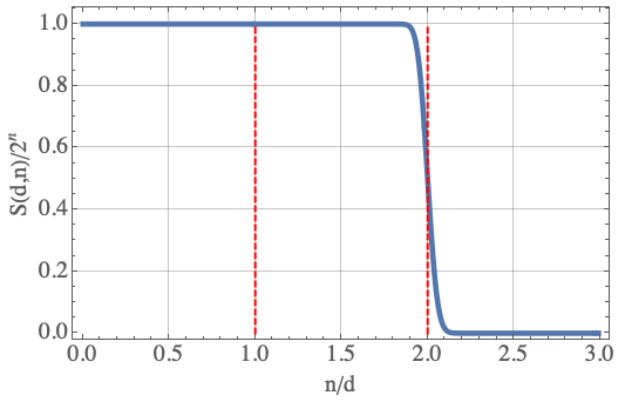

Figure 5: Function $S(d,n)/2^n$ as a function of n/d, for $d = 1000$. Inspired by Figure 40.9 from the book by MacKay (2003).

The integral can be computed using a Monte Carlo technique as soon as the dimensionality becomes high.

3. A third remark concerns the perspective of considering the neuron as a *memory* from which one would be able to reconstruct the training set D_{tr} composed of n distinct inputs. The question that arises is: *what is the capacity of such a neuron?* All in all, since $\boldsymbol{w} \in \mathbb{R}^d$, one would expect the neuron to have an infinite storage capacity. However, it is important to note that no one has direct access to the weights $\boldsymbol{w}$, the internal constituents of the neuron: interaction occurs solely through the input $\boldsymbol{x}$. Therefore, with n values of $\boldsymbol{x}$ available to us, and the neuron providing n bits in output (that is, for each input $\boldsymbol{x}_i$, the output is $y_i \in \{0,1\}$), the formulation of the question becomes: *can we reproduce all these bits without error* (2^n configurations)? Or rather, *what is the maximum number n of inputs that allows for no errors*? The neuron initially behaves according to the threshold function of Equation 7, which can be simplified by keeping only the "sign" function, meaning the response is then ± 1, which can be recoded as $\{0,1\}$. Let's assume there is no bias for now[12]. The question being posed is to ask *how many distinct functions of this type can be simulated using a vector $\boldsymbol{w}$ in d dimensions, with n inputs $\boldsymbol{x}$*. We denote this number as $S(d,n)$. The result is as follows (Cover (1965) and see also MacKay (2003)):

$$S(d,n) = \begin{cases} 2^n & d \geq n \\ 2\sum_{i=0}^{d-1} \binom{n-1}{i} & d \leq n \end{cases} \tag{20}$$

It is noteworthy that the maximum number of binary functions generated with n inputs (2^n) is the value of $S(d,n)$ as long as $n \leq d$.

What can we say when $n = d$? This particular value defines the maximum number of data points that can be labeled with bits without error. It can also be described as the number of samples that the neuron can classify in the worst-case scenario. This is referred to as the *Vapnik-Chervonenkis dimension*[13] (d_{vc}). We conclude that equipped with a binary activation function, its dimension d_{vc} is equal to d. The search for the value (or bound) of the VC dimension of different architectures and activation functions remains a current topic of interest (Bartlett and Maass, 2003; Petersen and Sepliarskaia, 2024).

What does the function $S(d,n)$ tell us for $n > d$ and large d? That is, beyond $d = d_{vc}$, is the capacity of the neuron really degraded? In fact, this is not the case, as can be seen in Figure 5: the larger d becomes, the closer the ratio $S(d,n)/2^n \approx 1$ until the value $n_{max} = 2d$ is reached, where a sharp drop occurs (this is somewhat an effect of overcapacity). Thus, we can label (0/1) a number of samples up to 2 times the size of the input. The maximum number of bits that can be stored (storage capacity) in this linear threshold neuron is therefore equal to $C(1,d) = 2d \times d = 2d^2$.

[12] Note that we can always treat $d \to d+1$ at the end if we include this bias.

[13] From Vladimir N. Vapnik and Alexey Chervonenkis, who introduced this concept in the 1970s.

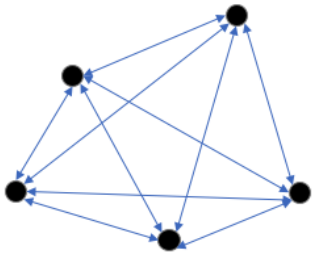

Figure 6: Hopfield network: $w_{ij} = w_{ji}$ (bidirectional arrow between two nodes), $w_{ii} = 0$ (no arrow looping back on the same node).

The generalization for a polynomial threshold function of degree p and for large values of d gives the following formula for storage capacity (Cover, 1965):

$$C(p,d) = 2d\binom{p+d}{p} \approx 2\frac{d^{p+1}}{p!} \tag{21}$$

Having established this foundational block that represents a single neuron, let us now look at multi-neuron architectures developed by J.J. Hopfield and G.E. Hinton.

3 Hopfield networks

An architecture popularized by John J. Hopfield in 1982 (Hopfield, 1982) corresponds to the interconnection of neurons according to the following principles:

1. The connections between two neurons (Figure 6) are *symmetric* and *bidirectional*, meaning that the action of neuron j on neuron i is not differentiated ($w(j \to i) = w_{ij}$) from the reverse action, therefore, $w_{ij} = w_{ji}$, and there is *no self-coupling*, hence $w_{ii} = 0$. The state of a neuron is denoted as x_i. If a bias is to be included in the network, a fictitious neuron x_0 with a permanent state of 1 is introduced. All outputs of the neurons are the inputs to the other neurons and vice versa, creating a network with N neurons. There are thus $N(N-1)/2$ different weights.

2. The activation function $\rho(x)$ can be, for example, a threshold function like $\mathrm{sgn}(x)$ that gives a binary value, or a function tanh that yields a continuous real variable. Thus, the state of a neuron is calculated in the same manner as for a single neuron. For instance, considering neuron i, one proceeds by a weighted sum of its inputs, followed by its activation[14]

$$a_i(\boldsymbol{x}) = \sum_{j \neq i} w_{ij} x_j \qquad\qquad x_i' = \rho(a_i) \tag{22}$$

   and x_i' becomes the new value of the neuron's state.

   However, it is important to specify how the network is updated after establishing an initial state for all neurons. In a sense, time is discretized, and we need to specify *the evolution equations of the network*. We can consider two categories of updates:

   - *synchronous mode*: all values $\{x_i(t+1)\}_{i \leq N}$ are calculated simultaneously for each neuron i using the set $\{x_i(t)\}_{i \leq N}$;
   - *asynchronous mode*: to calculate $\{x_i(t+1)\}_{i \leq N}$, we go through the N neurons either in a *fixed sequential order* or *randomly*, updating one neuron at a time. Thus, let us consider that we first

[14] nb. the expression for a_i can include a bias b_i, which can be integrated into the sum by redefining the weights $w_{i0} = b_i$ and adding a fixed node $x_0 = 1$.

examine neuron 1, then neuron 2, so that

$$x_1(t+1) = \rho\left(\sum_{j>1} w_{ij}x_j(t)\right)$$
$$\text{then} \quad x_2(t+1) = \rho\left(w_{12}x_1(t+1) + \sum_{j>2} w_{ij}x_j(t)\right) \tag{23}$$

and so forth.

For the following numerical applications, we use the random asynchronous method.

The early work of J. Hopfield reported in 1982 focuses on the emergence of collective properties in artificial neural networks based on neurobiology. This work rekindled interest in artificial neural networks, which had been neglected following the book by Marvin Minsky and Seymour Papert from 1969 (Minsky and Papert, 1969).

For example, one might ask the question: how to store $\boldsymbol{x} = \{x_i\}_{i\leq N}$ bits (coded as ± 1) in a stable manner? That is, what are the conditions for $\boldsymbol{x}$ to be invariant under the action of one iteration of evolution:

$$\forall i \leq N, \quad \rho(a_i(\boldsymbol{x})) = x_i \tag{24}$$

In summary, *we are seeking fixed points* of the system's evolution transformation.

By analogy, one can conceive of the evolution of the Hopfield network as a relaxation of a physical system towards a state of local energy minimum:

$$E(x) = -\frac{1}{2}\sum_{i,j} w_{ij}x_i x_j \tag{25}$$

(Note that the $1/2$ accounts for the symmetry of w_{ij}.) However, we want to be able to store P patterns $(\{x_i^{(p)}\}_{i\leq N, p\leq P})$, that is to say, to find *stable states* that can be reproduced from slightly different states. These patterns are thus *attractors* of the evolution transformation. The image in terms of the energy function landscape is to create attractive basins in the shape of funnels that are deep and as wide as possible.

In fact, Hopfield networks have *memorization* properties that can be described as astonishing and continue to inspire research and applications (see, for example, Ramsauer et al., 2020; Wen et al., 2009). In addition to memorizing patterns, one can mention pattern recognition, restoration of degraded images, combinatorial optimization (e.g., the traveling salesman problem), information retrieval in databases, optimal control of complex systems (e.g., traffic management), and in neuroscience, the functioning of the brain, as well as in statistical physics, the study of spin glasses, etc. Without being exhaustive in this short article, we will therefore highlight some properties of these networks, particularly concerning memorization.

We therefore need to be able to find the weights w_{ij} from the P patterns given *a priori*. This is a supervised learning problem.

3.1 Hebb learning method

The first learning method that was used is attributed to Donald Hebb, who proposed in 1949 (Hebb, 1949) a law of increasing synaptic efficiency between neurons during repeated and persistent stimulation[15]. This law, in the context of Hopfield networks, is expressed as[16]

$$w_{ij} = \frac{1}{N}\sum_{p=1}^{P} x_i^{(p)}x_j^{(p)} \tag{26}$$

where N is the number of neurons in the network, P is the number of patterns, and satisfying $w_{ii} = 0$ for all $i \leq P$. Note that if we consider a first pattern $x^{(1)}$ of dimension $N \times 1$, then the weight matrix can be expressed in the form

$$\boldsymbol{w}^{(1)} = \frac{1}{N}\left(x^{(1)}x^{(1)T} - I_{N\times N}\right) \tag{27}$$

[15] nb. Long-term synaptic persistence in the hippocampus of rabbits was evidenced around the 1970s.
[16] nb. The normalization $1/N$ is not important in itself for optimization.

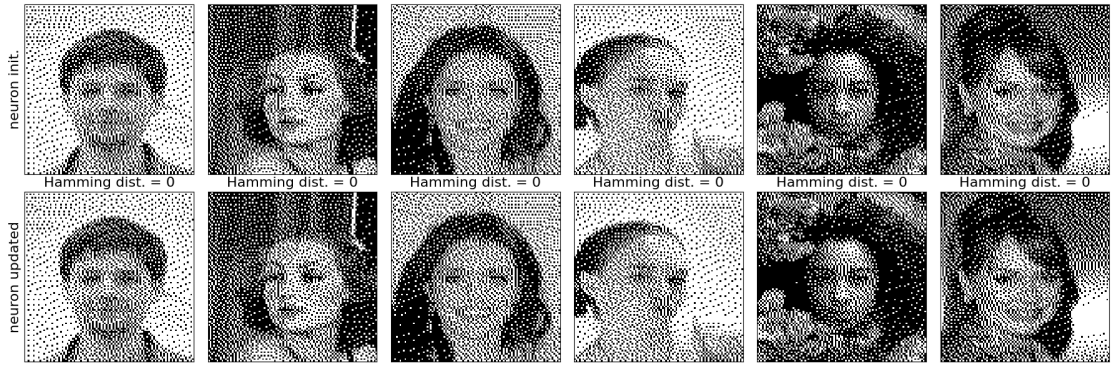

Figure 7: Top: Binarized images (± 1 code white/black) of 100×100 pixels used to compute the weights by Hebb's method for a Hopfield network of 10,000 neurons. Bottom: After a random asynchronous update of the neurons, it can be observed that the 8 patterns remain stable. The Hamming distance counts the number of differing pixels between the images in the same column.

and the energy of a state x is given by

$$E(x) = -\frac{1}{2} x^T \boldsymbol{w}^{(1)} x = -\frac{1}{2N} (x^T x^{(1)} x^{(1)T} x - x^T x) \tag{28}$$

Now, $x^T x = \|x\|^2 = N$ because the components of the vectors are equal to ± 1, thus

$$E(x) = -\frac{1}{2N} |\langle x, x^{(1)} \rangle|^2 + \frac{1}{2} \tag{29}$$

Thus, the energy is minimized for $x = x^{(1)}$, where $x^{(1)}$ is a stable state and its energy is equal to $-(N-1)/2$. The question that arises is to know how many patterns can be stored in such a way that they are stable minima of the energy?

Memorization: robustness. If we design a network of $N = 100^2$ neurons, we can store, by fixing the weights according to Hebb's law (Equation 26), for example, 8 binarized images of 100×100 pixels of faces from the "CelebA" dataset (Liu et al., 2015) (Figure 7). It is quite remarkable that we can cancel 50% of the weights w_{ij} (disconnect relationships between neurons) while keeping the weight matrix symmetric, and still, the 8 stored patterns remain stable.

Now, let's keep the initial Hopfield network whose weights are calculated from the 8 binarized images that are stable patterns. We take one of these 8 images and randomly nullify 99.7% of its pixels, initializing the neurons of the network with this highly degraded version. We then proceed with a single iteration of updating the states of the neurons. The result is shown in Figure 8: we can clearly see the original image again. Another robustness test is presented in Figure 9. Starting from the same initial image (stable pattern), we randomly flip the state of 48.0% of its pixels. We then proceed, as before, with a single iteration of updating the states of the network: once again, the original image is retrieved.

These numerical tests perfectly illustrate the resilience of such a memorization device and the attractor power of the memorized patterns.

Memorization: Capacity. However, as soon as we attempt to add new patterns beyond the initial 8, the number of patterns that become unstable, meaning they are not fixed points of the neuron state update operation, also increases as illustrated in Figure 10 for the network used previously. Around 30 patterns, all become unstable. However, if we repeat this exercise with random patterns (Rademacher distribution), the network shows its first instability after approximately ≈ 400 stored patterns (Figure 11). Thus, a capacity limit is noted, although it is more significant for these random patterns than for faces. The main difference lies in the existence of spatial correlations at different scales concerning faces, while they are absent in random patterns. This effect has been studied in the literature (see for example Löwe, 1998; De Marzo and Iannelli, 2023). However, it should be noted that many properties of Hopfield networks are estimated from random patterns.

We can attempt to estimate the maximum number P_{max} of patterns that can be stored by requiring that the probability of flipping — after one iteration and having initialized the network with any stored pattern — of

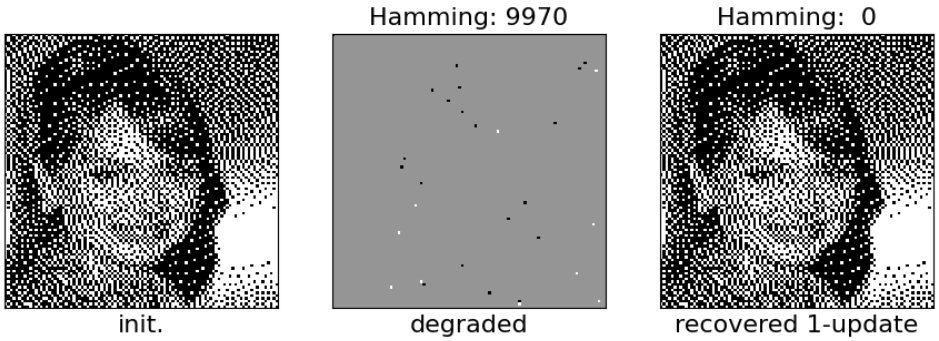

Figure 8: On the left: original image. In the center: degraded image after randomly setting 99.7% of the pixels to zero. This image serves as input to the Hopfield network whose weights were calculated using Hebb's method (Figure 7). On the right: state of the neurons after one iteration of network evolution: the unaltered pattern image is recovered, illustrating the "attractor" power of these stored patterns.

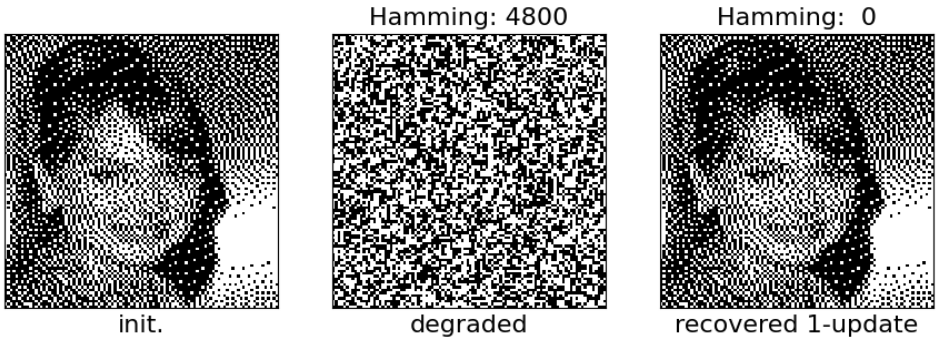

Figure 9: Same caption as Figure 8, except this time 48% of the state of the neurons was set to zero before their update.

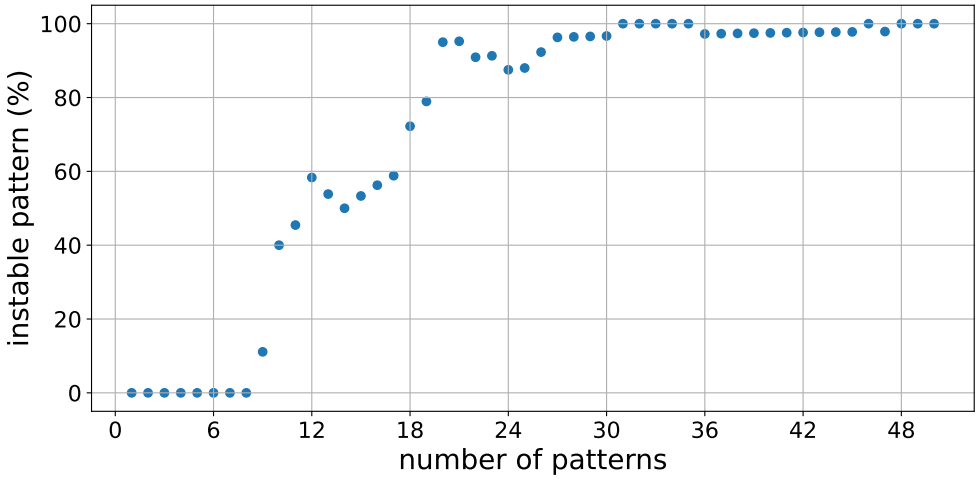

Figure 10: Fraction of unstable patterns (faces) as a function of the number of patterns attempted to be integrated into the network of 10,000 neurons used for Figures 7 to 9. From 8 patterns onwards, it is observed that the network increasingly lacks the capacity to memorize them, meaning there are more and more unstable patterns after an iteration of neuron state updates.

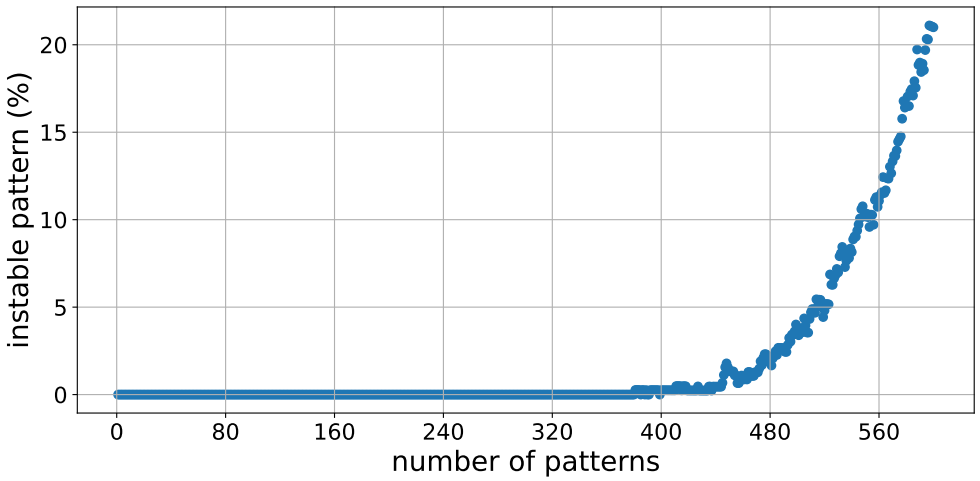

Figure 11: Same legend as Figure 10, but this time the patterns are generated randomly following the Rademacher distribution (binomial distribution with parameter $p = 0.5$ with values ± 1).

any bit of any pattern is less than a small ε. We generate the patterns randomly with N bits. Let $\boldsymbol{x}^{(p)}$ be a pattern among the P patterns stored at this step; we initialize the states of the N neurons with this pattern and proceed to one iteration. For any neuron i in the network, we have:

$$a_i = \sum_{j \neq i} w_{ij} x_j^{(p)} \tag{30}$$

However, the weights w_{ij} are obtained from Hebb's rule, and we can isolate the contribution of pattern p:

$$w_{ij} = \frac{1}{N} x_i^{(p)} x_j^{(p)} + \frac{1}{N} \sum_{k \neq p} x_i^{(k)} x_j^{(k)} \tag{31}$$

Thus,

$$a_i = \underbrace{\frac{1}{N} \sum_{j \neq i} x_i^{(p)} (x_j^{(p)})^2}_{T_1} + \underbrace{\frac{1}{N} \sum_{j \neq i} \sum_{k \neq p} x_i^{(k)} x_j^{(k)} x_j^{(p)}}_{T_2} \tag{32}$$

The first term simplifies by noting that $(x_j^{(p)})^2 = 1$, so we have:

$$T_1 = \frac{N-1}{N} x_i^{(p)} \approx x_i^{(p)} \tag{33}$$

It is noteworthy that if the second term (T_2) were absent, neuron i would not change state. Thus, the tendency to change state comes from this second term. The product of three independent random numbers following a Rademacher distribution also follows a Rademacher distribution. The partial sum of n numbers generated by a Rademacher distribution is simply a random walk on $\mathbb{Z}$, and thus asymptotically follows a normal distribution $\mathcal{N}(0, \sigma = \sqrt{n})$. Therefore, a_i follows a normal distribution (if N and P are large)[17]:

$$a_i \sim \mathcal{N}\left(x_i, \sigma = \sqrt{\frac{P}{N}}\right) \tag{34}$$

Let's assume $x_i = 1$ (the case $x_i = -1$ is identical), then the probability that it becomes negative and induces a flip of the neuron to the value -1 is given by:

$$P(a_i < 0) = \int_{-\infty}^{0} \exp\left(-\frac{1}{2}\left(\frac{x-1}{\sigma}\right)^2\right) \frac{dx}{\sigma\sqrt{2\pi}} = \frac{1}{2}\mathrm{erfc}\left(\sqrt{\frac{N}{2P}}\right) \tag{35}$$

[17]Note that $T_2 \approx z/N$ with $z \sim \mathcal{N}(0, \sigma = \sqrt{NP})$, hence $T_2 \sim \mathcal{N}(0, \sigma = \sqrt{P/N})$.

This expresses that if we set the probability of a particular neuron flipping during the first iteration to 1%, then the number of patterns must approximately correspond to about 18% of the number of neurons in the network ($P \approx 0.18N$). When we try to include more patterns, collective effects cause several neurons to flip.

We can now use an asymptotic form of the probability[18] that at least one neuron among P patterns flips to equal ε, which sets the value of the maximum number of admissible patterns P_{max}:

$$\frac{P_{max}}{N} \approx \frac{1}{4\log N + 2\log(1/\varepsilon) + 3\log(P_{max}/N)} \tag{36}$$

For $\varepsilon = 1\%$ and $N = 10^4$, we obtain $P_{max} \approx 0.03N \approx 300$, which we indeed observe in our numerical experiments[19]. Therefore, we confirm that the network memorizes many more randomized patterns than structured patterns like faces. It is shown that the patterns that are best suited to be stored must be independent and identically distributed according to the Rademacher distribution.

Asymptotically, the maximum number of stable randomized patterns in a network of N neurons (according to our definition) is (see Ramsauer et al., 2020, and references therein):

$$P_{max} \approx O\left(\frac{N}{\log N}\right) \tag{37}$$

A complete study of discrete Hopfield networks raises questions, for example, about the existence of stable states with a few bits differing from the desired patterns *(spin glass states)*, or the existence of additional stable states, the existence of stable states resulting from linear combinations of the desired patterns, etc. This highlights a classification of states according to the value of P/N. It is shown (see for example Crisanti et al., 1986; Hertz, 2018) that if we accept a bit of error, there is a rather abrupt threshold for $P \approx 0.138N$ beyond which the network fails to memorize. Since each pattern has N bits and there are about $N^2/2$ weights, we can say that *a "classical" Hopfield network stores on the order of* 0.28 *bits/weight.*

3.2 Perceptron Learning

The previous section is partly based on Hebbian learning. However, let us consider a neuron in isolation. It has $N-1$ inputs and 1 output with an activation function, and its task is as follows: given a pattern $x^{(k)}$ that we want to fix, if the $N-1$ inputs of the neuron are the true values $\{x_j^{(k)}\}_{j\neq i, j\leq N}$, then we would like the output of the neuron to be $x_i^{(k)}$. Thus, for each neuron, it has perfectly defined inputs and a target that can serve for supervised learning. It is then natural to use a slightly modified cost function of a perceptron (Section 2):

$$J_r(w) = -\sum_{i=1}^{N}\sum_{k=1}^{P}\left\{y_i^{(k)}\log\Big(h_w(x_i^{(k)})\Big) + (1-y_i^{(k)})\log\Big(1-h_w(x_i^{(k)})\Big)\right\} + \frac{\alpha}{2}\|w\|^2 \tag{38}$$

where

$$y_i^{(k)} = \begin{cases} 1 & \text{if} \quad x_i^{(k)} = +1 \\ 0 & \text{if} \quad x_i^{(k)} = -1 \end{cases} \tag{39}$$

and

$$h_w(x_i^{(k)}) = \frac{1}{1+e^{-\beta\sum_{j=1}^{N} w_{ij}x_j^{(k)}}} \tag{40}$$

where we have introduced a parameter β that recalls the factor $1/kT$ from Statistical Physics, which regulates the slope at the origin of the sigmoid. To minimize the cost function J_r, we use a gradient descent method with a learning rate (ℓ_r). Thus, for a minimization step, we have the following evolution equation in the case of the sigmoid[20]:

[18] $\operatorname{erfc}(x) \underset{x\to\infty}{\approx} e^{-x^2}/(x\sqrt{x})$.

[19] The approximation (Equation 36) gives $P_{max} = 283$, while the "exact" solution yields $P_{max} = 298$.

[20] The term involving α is often called "weight decay" in the literature (Bishop, 2006)

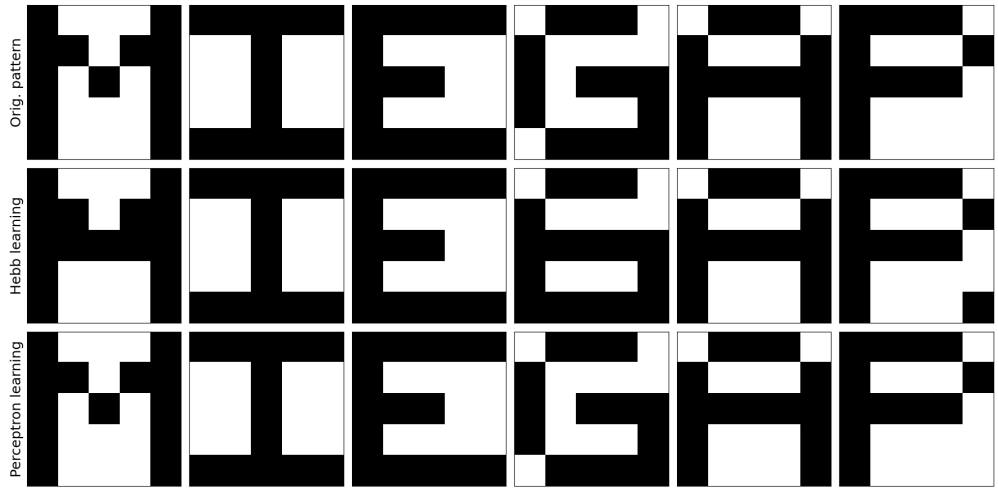

Figure 12: Illustration of the superior capacity of a Hopfield network with 25 neurons using "perceptron-style" learning compared to Hebbian learning. Top: the 6 patterns we want to fix. Middle: result after Hebbian learning, where only 3 recorded patterns are 100% stable. Bottom: result after "perceptron-style" learning with $\beta = 1.4$, a learning rate of 0.01, and a weight decay of 0.01. Figure inspired by the book of MacKay (2003).

$$
\begin{aligned}
w(t+1) &= w(t) - \ell_r \nabla_w J_r(w) \\
\nabla_w J_r(w) &= -\beta \sum_{i,k} x_i^{(k)} (y_i^{(k)} - h_w(x_i^{(k)})) + \alpha w
\end{aligned}
\tag{41}
$$

From a practical point of view, note that we can absorb the term β into the definitions of ℓ_r and α.

A simple test is illustrated in Figure 12 with a small network of 25 neurons that memorizes 5×5 patterns, which cannot memorize more than 4 patterns by the classical Hebbian method. However, by minimizing J_r, we can get it to memorize 6 patterns, resulting in a significant improvement. As for the network with $N = 10,000$ neurons used previously, numerical experimentation shows that it can store just over 3,000 random patterns of size 100×100, which is a factor of 10 improvement compared to Hebbian learning. Thus, it is clear that the capacity of networks can be increased by changing the method of weight learning.

In the literature, there are other learning methods that also explore the effects of biases, self-coupling terms, and even non-symmetric connections (Tolmachev and Manton, 2020).

3.3 Other activation functions

In the previous section, the Hebbian weight learning method was changed to a "perceptron-style" version; however, the activation function of neuron i remains the original one, namely

$$
a_i(\boldsymbol{x}) = \rho \left(\sum_j w_{ij} x_j \right) \tag{42}
$$

where $\rho(x)$ is essentially the "sign" function or differentiable forms, to provide the dynamics of the system once the weights are optimized. The so-called "modern" versions of Hopfield networks clearly utilize different activation functions, which grants them enhanced capabilities.

For example, in 2016, Dmitry Krotov and John J. Hopfield (Krotov and Hopfield, 2016) studied a new type of associative memory, termed *"dense associative memory"* (or DAM), using the following function

$$
a_i(\boldsymbol{x}) = \operatorname{sgn} \left(\sum_{p=1}^{P} \left[F(x_i^{(p)} + \sum_{j \neq i} x_j^{(p)} x_j)] - F((-1) x_i^{(p)} + \sum_{j \neq i} x_j^{(p)} x_j) \right] \right) \tag{43}
$$

where $\{\boldsymbol{x}^{(p)}\}_{p\leq P}$ are the patterns we want to store, and F is a smooth function from $\mathbb{R}$ to $\mathbb{R}$. Notably, if $F(x) = x^2$, then

$$a_i(\boldsymbol{x}) = \text{sgn}\left[\sum_{p=1}^{P}\sum_{j\neq i} x_i^{(p)} x_j^{(p)} x_j\right] \tag{44}$$

which is simply the classical version and does not change the properties. The question is: what happens if we now use a polynomial form $F(x) = x^n$? We find the following result:

(Krotov & Hopfield (2016)) *In the case where the activation function in Equation 43 is governed by a function $F(x) = x^n$, considering patterns whose bits follow a Rademacher distribution, we have the following results concerning the probability P_{error} that a single neuron is unstable (with K and N large):*

$$P_{error} \approx \sqrt{\frac{(2n-3)!!}{2\pi}\frac{P}{N^{n-1}}}\exp\left(\frac{N^{n-1}}{2P(2n-3)!!}\right) \tag{45}$$

and the maximum number of stored patterns P_{max} can be expressed as $P_{max} = \alpha_n(P_{error})N^{n-1}$. Notably, if $P_{error} = 0.5\%$ and $n = 2$, we find $P_{max} \approx 0.14N$ for the standard network, and if $P_{error} < 1/N$, which corresponds to a virtually error-free memory, then

$$P_{max} \approx \frac{1}{2(2n-3)!!}\frac{N^{n-1}}{\log N} \tag{46}$$

This bound is to be compared with the classical formula obtained (Equation 37). For instance, if $N = 10^4$ and $n = 3$, then $P_{max} \approx 200\ N$, demonstrating the potential improvement in storage capacity.

Fortified by this result, a question arises: what happens when $n \to \infty$ or even when the function is $F = e^x$? Mete Demircigil and collaborators showed the following theorem in 2017 (Demircigil et al., 2017):

(Demircigil et al, (2017)) *Consider the case where the activation function in Equation 43 is governed by the function $F(x) = e^x$. Let $0 < \alpha < \log(2)/2$, and let $\{\boldsymbol{x}^{(p)}\}_{p\leq P}$ such that $P = e^{\alpha N} + 1$ (where N is the number of neurons in the network), with i.i.d Rademacher patterns, and $\rho \in [0, 1/2[$. Then for all $p \leq P$ and for all $\tilde{\boldsymbol{x}}^{(p)} \in \mathcal{S}(\boldsymbol{x}^{(p)}, \rho N)$, the Hamming sphere centered on $\boldsymbol{x}^{(p)}$ with a supposed integer radius ρN, we have*

$$\mathbb{P}(\exists q\ \exists j,\ \textit{such that } a_j(\tilde{\boldsymbol{x}}^{(p)}) \neq \boldsymbol{x}_j^{(p)}) \to 0 \tag{47}$$

if α is chosen such that

$$\alpha < \frac{I(1-2\rho)}{2} \tag{48}$$

with[a]

$$I(x) = \frac{1}{2}\{(1+x)\log(1+x) + (1-x)\log(1-x)\} \tag{49}$$

[a] This is recognized as being proportional to the Shannon entropy of a Bernoulli process with probability x for one of its values. $I(1) = \log(2)$, $I(0) = 0$.

The *Hamming sphere*, named after Richard W. Hamming, is based on the eponymous distance that we used to calculate the difference in the number of bits between two patterns (Section 3). This notion generalizes to any kind of alphabet. It is indeed a distance, and it plays a role in coding theory. In this case, Hopfield networks "correct" patterns that differ slightly from a stable pattern.

The theorem implies that for large values of N, almost surely all patterns are fixed points of the network's dynamics, and the maximum number of patterns that can be stored is on the order of ($\alpha < I(1)/2$)

$$P_{max} \approx 2^{N/2} \tag{50}$$

which is a considerable increase compared to the classical Hopfield model. For example, with the network we used to store binarized images of size 100×100, $N = 10^4$ and thus $P_{max} \approx 1.4\ 10^{1505}$. In comparison, the estimated number of protons in the Universe is approximately 10^{80}, which becomes a very modest number. However, in exchange for this colossal number of stable states, the radius of the Hamming sphere centered on each pattern tends towards 0, making them very localized attractors.

Incidentally, the proof of the theorem relies on results from Large Deviations Theory introduced in 1966 by S. R. S. Varadhan (Varadhan, 2008). It concerns the asymptotic behavior of tails of distributions. In particular, regarding n i.i.d variables $(X_i)_{i\leq n}$ following the Rademacher distribution, if we form their sum $S_n = \sum_i X_i$, then

$$\forall x \in]0,1[, \quad \mathbb{P}(S_n \geq nx) \leq e^{-nI(x)} \tag{51}$$

with $I(x)$ being the function given in the theorem. Such inequalities quantify the deviation of random variables from a fixed value (e.g., their expectation) and are referred to as *"concentration inequalities"*.

3.4 Continuous Hopfield Networks: Analogy with the Ising Model

One aspect that quickly emerged after the introduction of discrete Hopfield networks is the fact that the function "sgn" (sign) is not differentiable. This issue can be resolved by allowing the neuron to output a real number in the interval $[-1, 1]$, rather than just the values ± 1. We have already touched on this point by using the function $h_w(x)$ (Figure 3) during the "perceptron-style" learning (Section 3.2). In the literature, these are referred to as *continuous Hopfield networks*, a notion introduced by J.J. Hopfield in 1984 (Hopfield, 1984). In particular, there is a natural link with the Ising model[21]

It was in 1974 that William A. Little demonstrated the emergence of persistent states related to long-range effects in the Ising model (Little, 1974). In fact, the study of spin glasses gained popularity through their connections with neural networks after the work of J.J. Hopfield. In this context, the non-linear activation function is given by

$$x_i = \tanh(\beta a_i) = \rho_\beta(a_i) = \rho(\beta a_i) \tag{52}$$

with $\beta = 1/T$ $(k = 1)$ where T can be viewed as the temperature of the system. As $T \to 0$, the activation function approaches the $\text{sgn}(x)$ function of the previous networks. The dynamics of the states of the neurons in the system is modeled by a differential equation

$$\frac{dx_i}{dt} = -x_i(t) + \rho_\beta(a_i(t)) \tag{53}$$

When the system reaches a stable point for all neurons $(\forall i,\ dx_i/dt = 0)$, then $x_i(t) = \rho_\beta(\sum_j w_{ij}x_i(t)) = \rho_\beta(a_i(t))$. It turns out that this dynamic system possesses a Lyapunov function (provided that the weight matrix w_{ij} is symmetric) whose expression is (see, for example, Šíma and Orponen, 2003)

$$E = -\frac{1}{2}\sum_{ij} w_{ij}x_i x_j + \frac{1}{\beta}\sum_i \int_0^{x_i} \rho^{-1}(x)\ dx \tag{54}$$

$$= -\frac{1}{2}\sum_{ij} w_{ij}x_i x_j + \frac{1}{\beta}\sum_i I(x_i) \tag{55}$$

where $I(x)$ is the function from Theorem 2.

In fact, the activation and asynchronous state update relations of the neurons are entirely equivalent to those obtained in the variational method of independent spins applied to the Ising model, and the expression above is nothing other than the Gibbs free energy (Jain et al., 2018).

We will not pursue the numerous studies of continuous Hopfield networks in fields such as statistical physics or for technical applications and optimization problems. The paper by H. Ramsauer et al., already mentioned (Ramsauer et al., 2020), utilizes a new activation function that is none other than the *attention mechanism* of the transformer `BERT` (Devlin, 2018; Kenton and Toutanova, 2019). This enables the design of new layers of deep neural networks that (sic) *"allow for new approaches to deep learning, beyond fully connected, convolutional, or recurrent networks, and offer mechanisms for grouping, memory, association, and attention."* Thus, we see that J.J. Hopfield's networks have developed an entire disciplinary field.

We will now turn to networks that constitute a continuation of these developments by directly implementing a Gibbs distribution.

[21] In brief, the "Ising model" in its original version is a model of interacting spins introduced by Wilhelm Lenz in 1920 and solved in 1D only by his student, Ernest Ising, who was able to show the absence of phase transition. The exact solution in 2D by Lars Onsager in 1944 allowed, first, to prove the existence of a phase transition, for example that of (para-ferro) magnetism, using methods from Statistical Physics; second, to understand why there could be none in 1D; and third, to open new approaches for studying critical exponents in the 1970s.

4 Boltzmann Machines

In Section 3.4, we discussed the version of Hopfield networks where the function $x_i = \tanh(a_i)$ is used to update the state of a neuron (x_i) following its activation $a_i = \sum_j w_{ij} x_j$. This type of model has a conception reminiscent of the Ising model[22] where one seeks to minimize the energy of a system of *interdependent* spins[23]

$$E(\boldsymbol{x}) = -\frac{1}{2}\boldsymbol{x}^T \boldsymbol{W}\boldsymbol{x} = -\frac{1}{2}\sum_{i,j} w_{ij} x_i x_j \tag{56}$$

The Hopfield network allows for the implementation of an approximate solution optimizing where the spins are *independent*. However, it is legitimate to ask whether there is a way to solve the original problem using a neural network that directly satisfies the Boltzmann-Gibbs distribution[24]

$$P(\boldsymbol{x}|\boldsymbol{W}) = \frac{1}{Z(\boldsymbol{W})} e^{-E(\boldsymbol{x};\boldsymbol{W})} = \frac{1}{Z(\boldsymbol{W})} \exp\left\{\frac{1}{2}\boldsymbol{x}^T \boldsymbol{W}\boldsymbol{x}\right\} \tag{57}$$

with the partition function formally written as

$$Z(\boldsymbol{W}) = \sum_{\boldsymbol{x}} \exp\left\{\frac{1}{2}\boldsymbol{x}^T \boldsymbol{W}\boldsymbol{x}\right\} \tag{58}$$

where the sum over $\boldsymbol{x}$ considers all possible bit configurations.

To this end, a new type of neural architecture was introduced by D. Ackley, G. Hinton, and T. Sejnowski in 1985 (Ackley et al., 1985), albeit in a context dealing with parallel architectures. Note that the authors also respond to the article by Minsky & Papert from 1969 (Minsky and Papert, 1969) and motivated the amended re-edition of 1988 (Minsky and Papert, 1988).

There are different variants of these architectures; however, they are primarily interconnected neural networks for which:

- the activation of a neuron i is identical to that of a Hopfield network

$$a_i(\boldsymbol{x}) = \sum_j w_{ij} x_j \tag{59}$$

  with generally $w_{ii} = 0$ and $w_{ij} = w_{ji}$, and recall that we can always include biases w_{i0} through a fixed activity neuron $x_0 = 1$;

- the update of the neuron is stochastic, and for binary states 0/1, we have the following probabilities[25]

$$P(x_i = +1) = \frac{1}{1 + e^{-a_i}} \qquad P(x_i = 0) = 1 - P(x_i = +1) \tag{60}$$

Note that the difference in energy required to switch from state +1 to state 0 for neuron i is given on one hand by

$$\Delta E_i = E(x_i = 0) - E(x_i = +1) = a_i \tag{61}$$

and on the other hand by the Gibbs distribution

$$\Delta E_i = \log\left(\frac{p_+}{1 - p_+}\right) \qquad \text{with} \quad p_+ = P(x_i = +1) \tag{62}$$

which reaffirms the logistic function (Equation 60) that thus directly derives from the Gibbs distribution.

If we have a training set represented as independent and identically distributed patterns with N bits, as in the Hopfield network, $\mathcal{D}_{tr} = \{\boldsymbol{x}^{(p)}\}_{p \leq P}$, we can use Bayesian formulation to obtain the *posterior* distribution of the weights to optimize them. We have then

$$P(\boldsymbol{W}|\mathcal{D}_{tr}) = \frac{P(\mathcal{D}_{tr}|\boldsymbol{W}) \times P(\boldsymbol{W})}{P(\mathcal{D}_{tr})} \tag{63}$$

[22] See note 21

[23] In the Ising model, it is customary to denote the weights w_{ij} as J_{ij}, interpreted as interactions between spins i and j, which are noted as σ_i instead of x_i.

[24] In most cases, the factor β is omitted.

[25] Note: We adopt an "on/off" state model for a neuron represented by 1-bit ($x_i \in \{0, 1\}$). If we had $x_i = \pm 1$, there would be a factor of 2 in the weight assigned to a_i.

where $P(\boldsymbol{W})$ is a *prior* on the weights and $P(\mathcal{D}_{tr}|\boldsymbol{W})$ is the likelihood that we seek to maximize. It involves the partition function $Z(\boldsymbol{W})$ according to

$$\log P(\mathcal{D}_{tr}|\boldsymbol{W}) = \log \prod_{p=1}^{P} P(\boldsymbol{x}^{(p)}|\boldsymbol{W}) = \sum_{p} \left(\frac{1}{2} \boldsymbol{x}^{(p)T} \boldsymbol{W} \boldsymbol{x}^{(p)} - \log Z(\boldsymbol{W}) \right) \tag{64}$$

Now, since the matrix $\boldsymbol{W}$ is symmetric, it can be easily shown that[26]

$$\frac{\partial \log Z(\boldsymbol{W})}{\partial w_{ij}} = \sum_{\boldsymbol{x}} x_i x_j P(\boldsymbol{x}|\boldsymbol{W}) = \mathbb{E}_{\boldsymbol{x} \sim P(\boldsymbol{x}|\boldsymbol{W})}[x_i x_j] = \langle x_i x_j \rangle_{P(\boldsymbol{x}|\boldsymbol{W})} \tag{65}$$

Thus, the gradient of the log-likelihood can be written as

$$\frac{1}{P} \frac{\partial \log P(\mathcal{D}_{tr}|\boldsymbol{W})}{\partial w_{ij}} = \frac{1}{P} \sum_{p} \left(\boldsymbol{x}_i^{(p)} \boldsymbol{x}_j^{(p)} - \langle x_i x_j \rangle_{P(\boldsymbol{x}|\boldsymbol{W})} \right) = \langle x_i x_j \rangle_{\mathcal{D}_{tr}} - \langle x_i x_j \rangle_{P(\boldsymbol{x}|\boldsymbol{W})} \tag{66}$$

where we have noted

$$\langle x_i x_j \rangle_{\mathcal{D}_{tr}} = \frac{1}{P} \sum_{p} \boldsymbol{x}_i^{(p)} \boldsymbol{x}_j^{(p)} \tag{67}$$

The weight update by gradient ascent thus takes the following form

$$w_{ij}^{t+1} = w_{ij}^{t} + \ell_r \left(\langle x_i x_j \rangle_{\mathcal{D}_{tr}} - \langle x_i x_j \rangle_{P(\boldsymbol{x}|\boldsymbol{W})} \right) \tag{68}$$

The gradient of the log-likelihood is the difference between the correlation $\langle x_i x_j \rangle$ estimated from the training set and that estimated from the model currently being optimized. The first correlation is similar to Hebb's method and corresponds exactly to what is done after one iteration with a *prior* $W = 0$. However, subsequently, the second term becomes significant, and its estimation is done via Monte Carlo to sample the Gibbs probability density (Equation 57). Note that the gradient of the log-likelihood becomes zero when the correlation observed in the model coincides with that observed in the training set.

We remark that this maximum likelihood method can be formulated differently. By requiring that the model distribution ($P(\boldsymbol{x}|\boldsymbol{W})$) approaches the empirical distribution of the training set samples, $P(\boldsymbol{x}|\mathcal{D}_{tr}) = \frac{1}{P} \sum_p \delta(\boldsymbol{x} - \boldsymbol{x}^{(p)})$, we minimize the Kullback-Leibler divergence[27] denoted D_{KL}:

$$\begin{aligned} D_{KL}(P(\boldsymbol{x}|\mathcal{D}_{tr})||P(\boldsymbol{x}|\boldsymbol{W})) &= \mathbb{E}_{\boldsymbol{x} \sim P(\boldsymbol{x}|\mathcal{D}_{tr})} \left[\log \frac{P(\boldsymbol{x}|\mathcal{D}_{tr})}{P(\boldsymbol{x}|\boldsymbol{W})} \right] \\ &= \mathbb{E}_{\boldsymbol{x} \sim P(\boldsymbol{x}|\mathcal{D}_{tr})}[\log P(\boldsymbol{x}|\mathcal{D}_{tr})] - \mathbb{E}_{\boldsymbol{x} \sim P(\boldsymbol{x}|\mathcal{D}_{tr})}[\log P(\boldsymbol{x}|\boldsymbol{W})] \\ &= -\frac{1}{P} \sum_{p} \left(\frac{1}{2} \boldsymbol{x}^{(p)T} \boldsymbol{W} \boldsymbol{x}^{(p)} - \log Z(\boldsymbol{W}) \right) + C \end{aligned} \tag{69}$$

with C a constant independent of $\boldsymbol{W}$ that does not intervene in the optimization of this divergence, which yields the same weights $\boldsymbol{W}$ as the optimization of the likelihood.

4.1 Introduction of Hidden Neurons

The development of Boltzmann machines would not represent a major (r)evolution compared to Hopfield networks if one were to limit oneself to the description of two-point statistics in the data (i.e., $\langle x_i x_j \rangle$). Knowledge of this statistic is only sufficient for Gaussian processes. But what about the statistics needed to describe images? The major contribution of G. Hinton et al. was the use of hidden neurons to describe higher-order correlations (Ackley et al., 1985).

The activation of hidden neurons is identical to that of the neurons we have used so far. The only difference is that only the "non-hidden" (or *visible*) neurons interact with the "outside world." In particular, when we provide a training set for the *visible* neurons ($\boldsymbol{x}$), $\mathcal{D}_{tr} = \{\boldsymbol{v}^{(p)}\}_{p \leq P}$, the states of the hidden neurons are not specified; we denote them as $\boldsymbol{h}$. For simplification, the complete state of the network is given by $\boldsymbol{z} = (\boldsymbol{v}, \boldsymbol{h})$.

[26] Note: We denote $\langle x \rangle_{p(x)}$ as the average of the random variable x according to its probability distribution $p(x)$.

[27] See, for example, Jean-Eric Campagne. Notes and comments regarding the lectures by S. Mallat at the Collège de France (2024). Master's Course. Learning and generation by random sampling, `https://hal.science/hal-04550771v2`

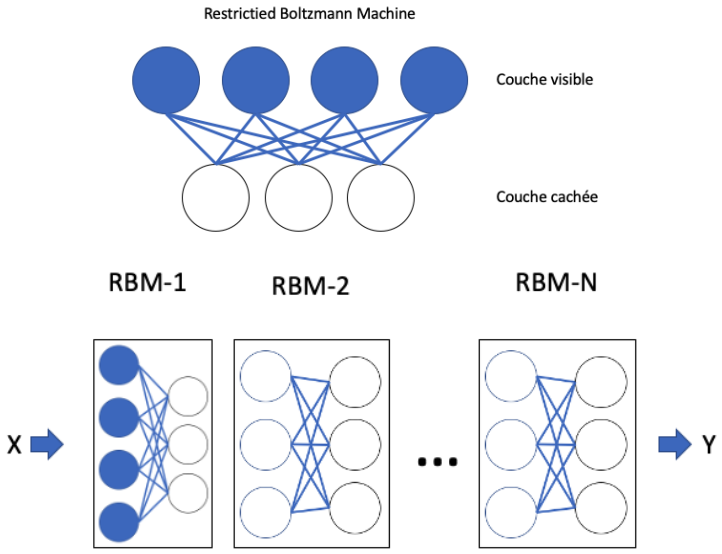

Figure 13: Top: Illustration of a Boltzmann machine with 1 hidden layer where only connections between "visible" neurons and "hidden" neurons are considered. This is called a Restricted Boltzmann Machine (RBM). Bottom: A collection of RBMs trained one after the other. This is a progressive learning technique for "large" neural networks using tanh activation functions.

To optimize the weights, we first need an expression for the likelihood, particularly for a pattern $\boldsymbol{v}^{(p)}$, we have

$$P(\boldsymbol{v}^{(p)}|\boldsymbol{W}) = \sum_{\boldsymbol{h}} P(\boldsymbol{v}^{(p)}, \boldsymbol{h}|\boldsymbol{W}) = \frac{\sum_{\boldsymbol{h}} \exp\{\frac{1}{2}\boldsymbol{z}^T \boldsymbol{W} \boldsymbol{z}\}}{\sum_{\boldsymbol{v},\boldsymbol{h}} \exp\{\frac{1}{2}\boldsymbol{z}^T \boldsymbol{W} \boldsymbol{z}\}} \tag{70}$$

The gradient with respect to w_{ij} of the global log-likelihood is calculated as before, yielding the following expression instead of Equation 66:

$$\frac{\partial \log P(\mathcal{D}_{tr}|\boldsymbol{W})}{\partial w_{ij}} = \sum_{p} \left(\langle z_i z_j \rangle_{P(\boldsymbol{h}|\boldsymbol{v}^{(p)}, \boldsymbol{W})} - \langle z_i z_j \rangle_{P(\boldsymbol{v},\boldsymbol{h}|\boldsymbol{W})} \right) \tag{71}$$

We find the two terms rearranged, with the first fixing the states $\boldsymbol{x}$ to the values of the training set while leaving the hidden states "free," and the second term where the states of all the neurons stem from the joint distribution of the current model.

The learning process for Boltzmann machines is resource-intensive; thus, we distinguish between two types of Boltzmann machines: the RBMs ("R" for *restricted*) (Figure 13, top), which only consider connections between "visible" neurons and "hidden" neurons, and the DBMs ("D" for *Deep*), which are a sequence of RBMs (Figure 13, bottom).

Historically, DBMs played an important role following RBMs in the modern resurgence of neural networks after G. Hinton, S. Osindero, and Y. Teh published a fast training algorithm in 2006 (Hinton et al., 2006) resembling the back-propagation algorithm (see e.g. Rumelhart et al., 1986; Lecun, 1988), which enabled the development of deep architectures. We will limit ourselves to a brief study of RBMs.

4.2 Restricted Boltzmann Machine

In the conditions of an RBM, the energy takes the form

$$E(\boldsymbol{v}, \boldsymbol{h}, \boldsymbol{W}) = -\boldsymbol{v}^T \boldsymbol{W} \boldsymbol{h} \tag{72}$$

where $\boldsymbol{W}$ is a real matrix[28], of dimension $N \times H$ (with H hidden neurons, and we keep the notation N for the number of "visible" neurons[29]). As can be seen, the energy simplifies significantly as there are no longer the quadratic terms xx and hh.

[28]Note: The matrix is symmetric and has a zero diagonal, concerning the set of neurons $\boldsymbol{z}$. However, in the expression mentioned, $\boldsymbol{W}$ only pertains to the interactions between the two types of neurons.

[29]Note: The vectors of type "h" (resp. "v") are of dimension $1 \times H$ (resp. $1 \times N$)

Minimizing $-\log P(\mathcal{D}_{tr}|\boldsymbol{W})$ using gradient descent gives an update step for the weight w_{ij} written according to the traditional method (Equation 69):

$$w_{ij}^{t+1} = w_{ij}^{t} + \ell_r \left(\langle h_i v_j \rangle_{P(\boldsymbol{h}|\boldsymbol{v}^{(p)}, \boldsymbol{W})} - \langle h_i v_j \rangle_{P(\boldsymbol{v},\boldsymbol{h}|\boldsymbol{W})} \right) \tag{73}$$

As previously mentioned, costly Monte Carlo techniques are required to estimate the second term. G. Hinton then developed an alternative he named *contrastive divergence* (Hinton, 2002). The idea is to use a Markov chain that starts with the training data and stops after n steps to evolve the distribution $P(\boldsymbol{v}, \boldsymbol{h}|\boldsymbol{W}^n)$ towards the empirical distribution. In the case of RBMs, a good approximation for updating the weights has been developed and can be simply expressed as

$$w_{ij}^{t+1} = w_{ij}^{t} + \ell_r \left(\langle h_i v_j \rangle_{P(\boldsymbol{h}|\boldsymbol{v}^{(p)}, \boldsymbol{W})} - \langle h_i v_j \rangle_{P(\boldsymbol{v},\boldsymbol{h}|\boldsymbol{W}^n)} \right) \tag{74}$$

Furthermore, in most cases, a single step ($n = 1$) is sufficient, which we denote as CD-1. We hope that such a formulation for updating the weights will provide the correct gradient direction. The study of convergence conditions has been elaborated subsequently (see, for example, Sutskever and Tieleman, 2010).

The optimization algorithm using the CD-1 method can be outlined as follows (see, for example, Hinton, 2012):

1. At step t, let the training batch be $\mathcal{D}_{tr} = \{\boldsymbol{v}^{(p)}\}_{p \leq P}$ where each entry is of dimension $1 \times N$.
2. Draw a batch of $\{\boldsymbol{h}\}_{\leq P}$ (each of dimension $1 \times H$) according to $P(h|\{\boldsymbol{v}^{(p)}\}, \boldsymbol{W}^t)$;
3. Compute $\delta W_+ = {\boldsymbol{v}^{(p)}}^T \boldsymbol{h}$;
4. Draw a batch of "fake" samples $\{\tilde{\boldsymbol{v}}\}_{\leq P}$ according to $P(v|\{\boldsymbol{h}\}, \boldsymbol{W}^t)$;
5. Draw a batch of new $\{\tilde{\boldsymbol{h}}\}_{\leq P}$ according to $P(h|\{\tilde{\boldsymbol{v}}\}, \boldsymbol{W}^t)$;
6. Compute $\delta W_- = \tilde{\boldsymbol{v}}^T \tilde{\boldsymbol{h}}$;
7. Update the weights $\boldsymbol{W}^{t+1} = \boldsymbol{W}^t + \ell_r(\delta W_+ - \delta W_-)$ and loop back to step 1 until the desired number of iterations (*epochs*) is reached.

Note that if one wants an approximation of order CD-n, it suffices to loop n times over operations 4 and 5, changing $\boldsymbol{h}$ to $\tilde{\boldsymbol{h}}$ at each iteration.

From a practical standpoint, one can use an update of $\boldsymbol{W}$ that includes a *momentum* effect and L2 regularization (*weight decay*), and during training, one can monitor the evolution of the generation error $\sum_{i \leq P} \|\boldsymbol{v}^{(i)} - \tilde{\boldsymbol{v}}_i\|^2$. It is worth noting that a modern approach would aim to directly optimize this quadratic error. It remains to determine how to perform the various samplings at the different stages of the algorithm. This is where the decoupling between the "visible" neurons on one hand and the "hidden" neurons on the other simplifies the task. If we want to calculate the conditional probability of the "hidden" neurons given the "rest" (the "visible" neurons and the weights $\boldsymbol{W}$), we have:

$$P(\boldsymbol{h}|\boldsymbol{v}, \boldsymbol{W}) = \frac{e^{\boldsymbol{v}^T \boldsymbol{W} \boldsymbol{h}}}{\sum_{\boldsymbol{h}} e^{\boldsymbol{v}^T \boldsymbol{W} \boldsymbol{h}}} \tag{75}$$

Here, the notation $\sum_{\boldsymbol{h}}$ indicates that we sum over all configurations of the bits h_i:

$$\sum_{\boldsymbol{h}} := \sum_{h_1 \in \{0,1\}} \sum_{h_2 \in \{0,1\}} \cdots \sum_{h_H \in \{0,1\}} \tag{76}$$

Thus,

$$\sum_{\boldsymbol{h}} e^{\boldsymbol{v}^T \boldsymbol{W} \boldsymbol{h}} = \sum_{\boldsymbol{h}} \prod_{j=1}^{H} e^{\boldsymbol{v}^T \boldsymbol{W}_{\bullet_j} h_j} \tag{77}$$

$$= \prod_{j=1}^{H} \sum_{h_j \in \{0,1\}} e^{\boldsymbol{v}^T \boldsymbol{W}_{\bullet_j} h_j} = \prod_{j=1}^{H} \left(1 + e^{\boldsymbol{v}^T \boldsymbol{W}_{\bullet_j}}\right) \tag{78}$$

Therefore, we have:

$$P(\boldsymbol{h}|\boldsymbol{v}, \boldsymbol{W}) = \prod_{j=1}^{H} \frac{e^{\boldsymbol{v}^T \boldsymbol{W}_{\bullet_j} h_j}}{1 + e^{\boldsymbol{v}^T \boldsymbol{W}_{\bullet_j}}} = \prod_{j=1}^{H} P(h_j|\boldsymbol{v}, \boldsymbol{W}) \tag{79}$$

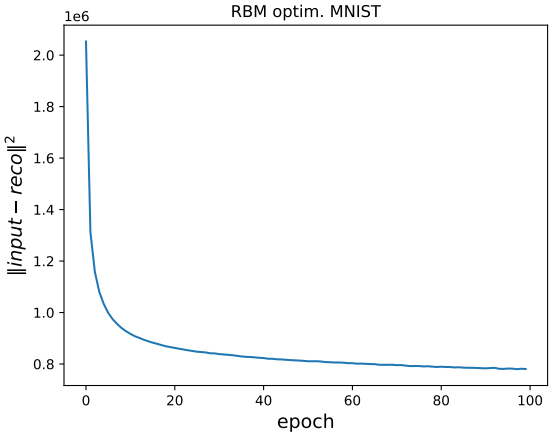

Figure 14: Evolution of $\sum_{i \leq P} \|\boldsymbol{v}^{(i)} - \tilde{\boldsymbol{v}}_i\|^2$ as a function of iteration over the training dataset (*epoch*).

We then obtain a factorization of the conditional probabilities, which simplifies the generation of the bits h_i. Moreover, ($\sigma(x)$ corresponds to the sigmoid function):

$$P(h_j = 1|\boldsymbol{v}, \boldsymbol{W}) = \sigma(\boldsymbol{v}^T \boldsymbol{W}_{\bullet j}) = 1 - P(h_j = 0|\boldsymbol{v}, \boldsymbol{W}) \tag{80}$$

Thus, it is sufficient to draw a number according to the uniform distribution on $[0, 1]$ and compare it to $\sigma(\boldsymbol{v}^T \boldsymbol{W}_{\bullet j})$ to declare the neuron h_i active (or not). The same applies to the visible neurons, where we have:

$$P(\boldsymbol{v}|\boldsymbol{h}, \boldsymbol{W}) = \prod_{i=1}^{N} P(v_i|\boldsymbol{h}, \boldsymbol{W}) \tag{81}$$

with

$$P(v_i = 1|\boldsymbol{h}, \boldsymbol{W}) = \sigma(\boldsymbol{W}_{i\bullet} \boldsymbol{h}) = 1 - P(v_i = 0|\boldsymbol{h}, \boldsymbol{W}) \tag{82}$$

Let us illustrate the minimization algorithm with an iconic training set.

4.3 Usage examples with the "MNIST" dataset

To illustrate some numerical aspects, we will use as "patterns" the digit images produced by Yann LeCun in 1998: these are small images representing manually written digits and labeled, of size 28×28 pixels, with values encoded in 255 levels of gray on $[0, 1]$ (60,000 for training and 10,000 for testing). This dataset of images (Modified National Institute of Standards and Technology, MNIST) has facilitated the development of many algorithms during various challenges[30] and remains a benchmark for educational approaches[31].

Optimization. Initially, the training set is used to optimize an RBM with $N = 28^2 = 784$ visible neurons and $H = 140$ hidden neurons[32]. We use batches of 64 images, and the algorithm described in the previous section is executed by iterating 100 times ("epochs") over the entire dataset. The CD-1 approximation is used. The evolution of the optimization is shown in Figure 14. Once the optimization is completed, we can then extract the elements of the matrix $\boldsymbol{W}^*$ according to 140 images of size 28×28 (Figure 15).

Linear Classification of Features. Now that the matrix $\boldsymbol{W}^*$ is fixed, we can use $\boldsymbol{h} = \sigma(\boldsymbol{v}\boldsymbol{W}^*)$ to obtain the *features* $\boldsymbol{h}$ for each image in the test set ($\boldsymbol{v}$). Then, from these vectors $\boldsymbol{h}$ (latent variables), we can reconstruct new digit images using $\boldsymbol{x} = \sigma(\boldsymbol{W}^* \boldsymbol{h}^T)$ and compare them to the original digit images. The quality of the reconstruction can be calculated using the L2 norm between the reconstructed image and the original image. The result is shown in Figure 16. Aside from a slight blurring effect, the reconstruction is quite faithful.

[30] `yann.lecun.com/exdb/mnist/`

[31] See other classic catalogs at `https://www.tensorflow.org/datasets/catalog/overview`.

[32] In this exercise, no biases are used, and the weights are initialized using a Gaussian distribution with mean 0 and standard deviation 0.1.

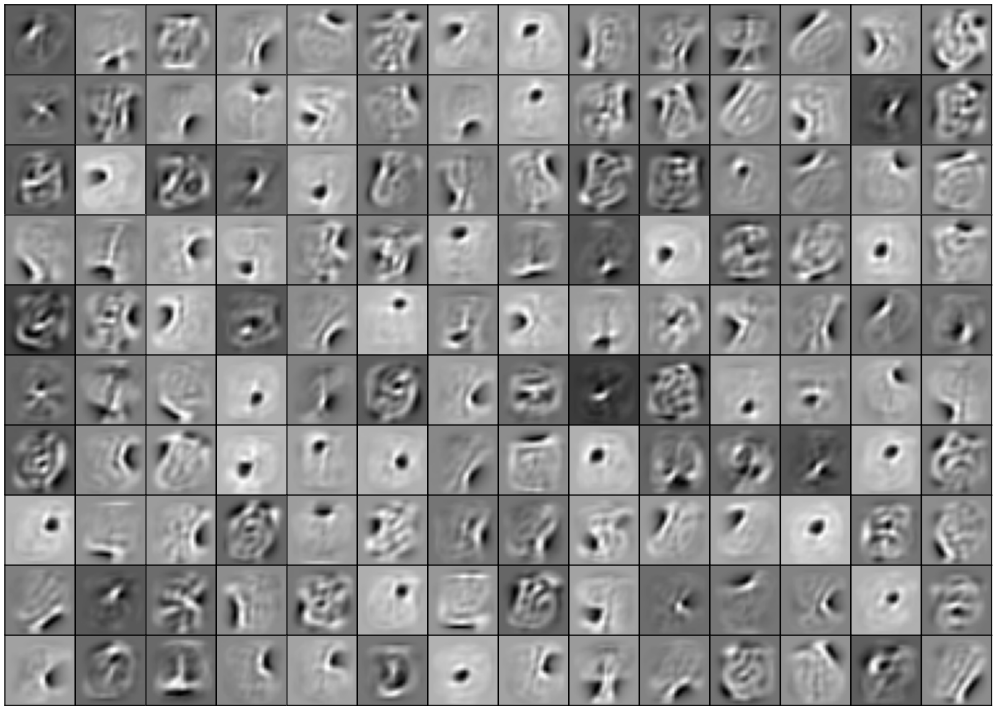

Figure 15: Collection of elements from the optimized matrix $\boldsymbol{W}^*$ (see text).

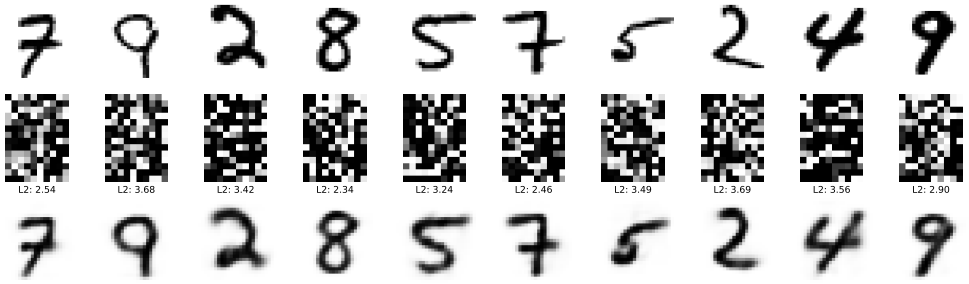

Figure 16: From digit images in the test set (1st row), once the weight matrix is optimized, we can obtain the representations (*features*) of each of them, namely the vectors $\boldsymbol{h}$, displayed as images of size 14×10 (2nd row). Then, from these representations, where the information is encoded and compressed, we can proceed to generate new digit images (3rd row). The L2 norm has been placed above these reconstructed images, comparing each time to the original test image (1st row of the same column).

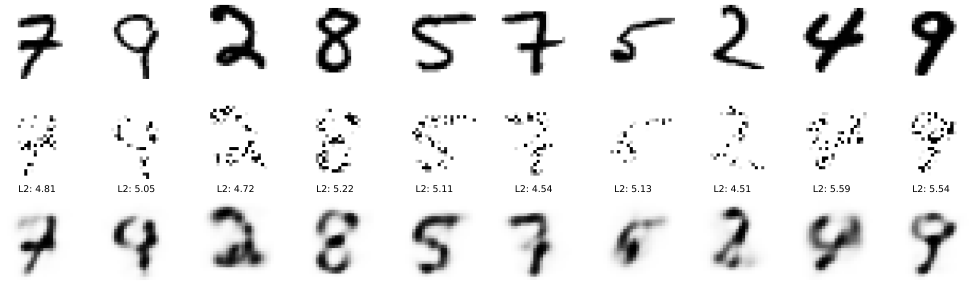

Figure 17: Examples of reconstruction by RBM of randomly deteriorated MNIST test images: at the top, the original; in the middle, the degraded version; at the bottom, the reconstructed version. The L2 norm between the noisy or reconstructed image and the original image is noted (same column).

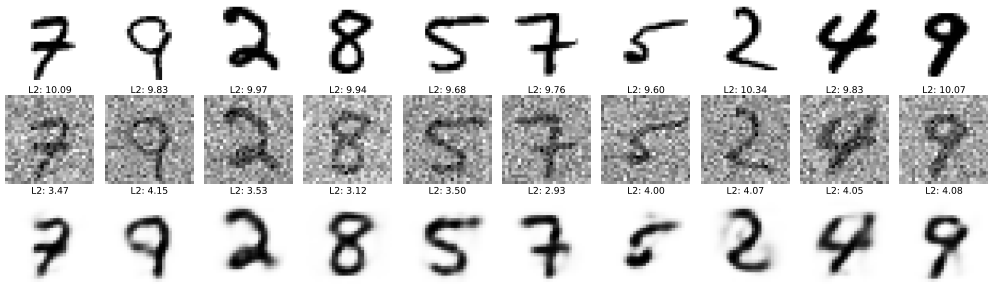

Figure 18: Examples of reconstruction of noisy images, which can be considered as denoising. The caption is similar to Figure 17.

Considering the training set of digit images, we can also associate their representations $\boldsymbol{h}^{(p)}$, and knowing the label of the digit $t^{(p)}$, we can form training pairs $(\boldsymbol{h}^{(p)}, t^{(p)})$ for a linear classifier[33] (Equation 9). Subsequently, once trained, this classifier can be evaluated on the MNIST test image set, which has also been passed through the RBM to extract the *features* as before. The classification score obtained is 94.5% for Top-1. If the images had been directly provided to the classifier, its score would have been around 90%, so the dimensionality reduction achieved by feature extraction through the RBM provides a substantial gain. Note that this score is no longer competitive compared to the performance of more refined methods that achieve 100% on this training set.

However, if linear classification is effective, this implies that in the $H = 140$ dimensional space, the average directions of the 10 digits (classes) are sufficiently well separated so that the fluctuations of the vectors of a given class do not generate "false" members of another class. That said, this *"feature extraction"* operation is general and produces the representation "$\Phi(x)$" of the data. The key is to know which architecture is suitable for the problem[34]. It should be noted that the optimization of the RBM is decoupled from its final use, the classification of digits; this is a different aspect from modern multi-layer architectures that merge both aspects: *feature extraction* and final classification.

Reconstruction of degraded images. Another use of an RBM, once the weights are known, is to reconstruct deteriorated images, as we did with Hopfield networks (Section 3). An example is shown in Figure 17. To do this, once an image is degraded, we proceed with one or more iterations of "generating a vector $\boldsymbol{h}$" followed by "generating a vector $\boldsymbol{v}$." In the end, the vector $\boldsymbol{v}$ produces a digit that closely resembles the undamaged digit. In a similar vein, we can use the RBM to denoise images, as illustrated in Figure 18. The results are quite satisfactory, even though they do not represent the state of the art.

4.4 Discussion

The RBM architecture has led to variants, such as those that allow for non-binary responses from neurons by choosing, for example, neurons with Gaussian responses. RBMs are often integrated into architectures for unsupervised learning problems. They are used in engineering, for instance, for feature extraction in the field of pattern recognition, in various recommendation systems, and for radar signal recognition in very noisy environments, etc. Improvements have emerged, such as the use of dropout to reduce overfitting (Roder et al., 2022). Finally, cascade architectures have been used, such as Deep Belief Networks (DBMs) by G. Hinton (Hinton, 2009). Interested readers can refer to the review by Zhang et al. (2018). It should be noted that RBMs have been used to reduce the complexity of training architectures like Autoencoders (Hinton and Salakhutdinov, 2006) (Figure 1), which are architectures aimed at reducing the dimensionality of the input, with the output constrained during training to minimize the quadratic error (for example) with respect to the input. We will not elaborate further on this subject, which opens up the vast field of *generative models*.

However, before concluding, it seems interesting to address a theme that extends beyond the scope of Boltzmann Machines. This concerns a defect that can be easily highlighted with the RBM optimized on the MNIST dataset. Simply translating the original image by 3 pixels can completely blur the classification, as shown in Figure 19. The translation has generated *adversarial examples* for the RBM architecture. This type

[33]For example, using logistic regression from the `scipy` library: `https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html`

[34]ibid. 5

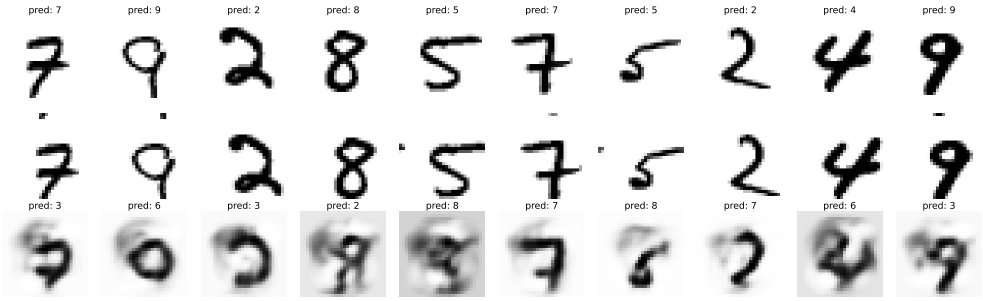

Figure 19: Examples of classification failure for images translated by 3 pixels in both directions. The first row shows the original image with the label obtained by the linear classifier based on representations from the RBM. The second row displays the images where the digits are shifted. The third row presents the images reconstructed by the RBM from the shifted images, along with the classification obtained from the features as well. Clearly, the rate of misclassifications significantly increases.

of defect has been revealed in the context of deep architectures (Szegedy et al., 2014).

A posteriori, in the face of this type of problem, several attitudes (neither exclusive nor exhaustive) can be considered, such as:

1. One can modify the way the model is trained. For instance, it would likely be possible to improve the model's robustness during the training phase by presenting it with translated images, or even slightly rotated or deformed ones. This involves a method of *"data augmentation"* that seeks to teach the model a certain variability of images concerning invariances of the response to transformations (global translation or diffeomorphism). This method has widely entered the practices of those who use neural networks on a daily basis.
2. A second approach involves reflecting on the *symmetries of the problem*: what are the transformations that leave the problem invariant? This is in this context that *convolutional neural networks* were developed based on ideas of reducing the number of weights through spatial invariances (Le Cun et al., 1989), which served as the starting point for the modern rise of neural networks. This is essentially what Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner wrote in 1998 (LeCun et al., 1998): *"[...] But the main flaw of unstructured networks for image or speech applications lies in the fact that they possess no intrinsic invariance with respect to translations or local distortions of the inputs. [...]"* These developments led to the elaboration of the "AlexNet" architecture in 2012 (Krizhevsky et al., 2012), which achieved spectacular results on the ImageNet dataset[35] and outperformed all previous top methods.

   This avenue invites reflection on the *operators* that must be implemented to construct the representation $\Phi(x)$ of the input x, which we refer to as *feature extraction*. This does not imply that other types of adversarial examples may not prove troublesome, but at least we push them into an increasingly restricted space. Here, mathematics attempts to regain control in order to understand the underlying mechanisms of neural networks, especially to explain the success of CNNs, while building a theoretical corpus that helps identify issues of instability[36].

5 Conclusion

The networks of J. J. Hopfield and the Boltzmann Machines of G. Hinton have been key components in developing architectures that subsequently led to the successes of deep convolutional networks and large generative models. These networks cover nearly all areas of signal processing, image, video, sound, speech, and all tasks related to text processing, inverse problems, etc. Notably, the 2024 Nobel Prize in Chemistry rewards Demis Hassabis and John Jumper for successfully utilizing the `Alphafold` model (Jumper et al., 2021) to predict the structure of almost all known proteins. The real question is which disciplinary field remains unaffected by these developments, as well as the applications in everyday life. However, as we have mentioned, not everything is entirely clear: why does it work so well? We still lack fundamental theorems.

[35] `https://www.image-net.org/`
[36] Ibid. 5

Mathematical research is active in this area across various fields, including statistics, probability, algorithms, stochastic processes, partial differential equations, and even algebraic geometry, among others. There is still much to discover following in the footsteps of these pioneers who have been honored this year.

References

Ackley, D. H., Hinton, G. E., and Sejnowski, T. J. (1985). A learning algorithm for boltzmann machines. *Cognitive science*, 9(1):147–169.

Bartlett, P. and Maass, W. (2003). *Vapnik-Chervonenkis dimension of neural nets*, pages 1188–1192. MIT Press, 2nd ed. edition.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*, volume 4 of *Information science and statistics*. Springer.

Cortes, C. and Vapnik, V. (1995). Support vector networks. *Machine Learning*, 20:273–297.

Cover, T. (1965). Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition. *IEEE Transactions on Electronic Computers*, EC–14:326–334.

Crisanti, A., Amit, D. J., and Gutfreund, H. (1986). Saturation level of the hopfield model for neural network. *Europhysics Letters*, 2(4):337.

De Marzo, G. and Iannelli, G. (2023). Effect of spatial correlations on hopfield neural network and dense associative memories. *Physica A: Statistical Mechanics and its Applications*, 612:128487.

Demircigil, M., Heusel, J., Löwe, M., Upgang, S., and Vermet, F. (2017). On a model of associative memory with huge storage capacity. *Journal of Statistical Physics*, 168:288–299.

Devlin, J. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Hebb, D. O. (1949). Organization of behavior. new york: Wiley. *J. Clin. Psychol*, 6(3):335–307.

Hertz, J. A. (2018). *Introduction to the theory of neural computation*. Crc Press.

Hinton, G. E. (2002). Training products of experts by minimizing contrastive divergence. *Neural Comput.*, 14(8):1771–1800.

Hinton, G. E. (2009). Deep belief networks. *Scholarpedia*, 4(5):5947. revision #63393.

Hinton, G. E. (2012). A practical guide to training restricted boltzmann machines. In *Proceedings of Neural Networks: Tricks of the Trade (2nd ed.)*, pages 599–619.

Hinton, G. E., Osindero, S., and Teh, Y.-W. (2006). A Fast Learning Algorithm for Deep Belief Nets. *Neural Computation*, 18(7):1527–1554.

Hinton, G. E. and Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507.

Hinton, G. E. and Sejnowski, T. J. (1986). Learning and relearning in Boltzmann machines. In Rumelhart, D. E. and Mcclelland, J. L., editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 1: Foundations*, pages 282–317. MIT Press, Cambridge, MA.

Hopfield, J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences of the United States of America*, 79:2554–8.

Hopfield, J. J. (1984). Neurons with graded response have collective computational properties like those of two-state neurons. *Proceedings of the National Academy of Sciences*, 81(10):3088–3092.

Jain, V., Koehler, F., and Mossel, E. (2018). The mean-field approximation: Information inequalities, algorithms, and complexity. In *Conference On Learning Theory*, pages 1326–1347. PMLR.

Jumper, J., Evans, R., Pritzel, A., Green, T., Figurnov, M., Ronneberger, O., Tunyasuvunakool, K., Bates, R., Žídek, A., Potapenko, A., Bridgland, A., Meyer, C., Kohl, S., Ballard, A., Cowie, A., Romera-Paredes, B., Nikolov, S., Jain, R., Adler, J., and Hassabis, D. (2021). Highly accurate protein structure prediction with alphafold. *Nature*, 596:1–11.

Kenton, J. D. M.-W. C. and Toutanova, L. K. (2019). Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of naacL-HLT*, volume 1, page 2. Minneapolis, Minnesota.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In Pereira, F., Burges, C. J. C., Bottou, L., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc.

Krotov, D. and Hopfield, J. J. (2016). Dense associative memory for pattern recognition. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, page 1180–1188, Red Hook, NY, USA. Curran Associates Inc.

Le Cun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1989). Handwritten digit recognition with a back-propagation network. In *Proceedings of the 2nd International Conference on Neural Information Processing Systems*, NIPS'89, page 396–404, Cambridge, MA, USA. MIT Press.

Lecun, Y. (1988). A theoretical framework for back-propagation. In Touretzky, D., Hinton, G., and Sejnowski, T., editors, *Proceedings of the 1988 Connectionist Models Summer School, CMU, Pittsburg, PA*, pages 21–28. Morgan Kaufmann.

Lecun, Y. and Bengio, Y. (1995). *Convolutional Networks for Images, Speech and Time Series*, pages 255–258. The MIT Press.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

Little, W. A. (1974). The existence of persistent states in the brain. *Mathematical biosciences*, 19(1-2):101–120.

Liu, Z., Luo, P., Wang, X., and Tang, X. (2015). Deep learning face attributes in the wild. In *Proceedings of the IEEE international conference on computer vision*, pages 3730–3738.

Löwe, M. (1998). On the storage capacity of hopfield models with correlated patterns. *The Annals of Applied Probability*, 8(4):1216–1250.

MacKay, D. J. C. (2003). *Information Theory, Inference, and Learning Algorithms*. Copyright Cambridge University Press.

Minsky, M. and Papert, S. (1969). *Perceptrons: An Introduction to Computational Geometry*. MIT Press, Cambridge, MA, USA.

Minsky, M. and Papert, S. (1988). *Perceptrons: An Introduction to Computational Geometry*. MIT Press, Cambridge, MA, expanded edition.

Petersen, P. C. and Sepliarskaia, A. (2024). Vc dimensions of group convolutional neural networks. *Neural Networks*, 169:462–474.

Ramsauer, H., Schäfl, B., Lehner, J., Seidl, P., Widrich, M., Gruber, L., Holzleitner, M., Pavlovic, M., Sandve, G. K., Greiff, V., Kreil, D. P., Kopp, M., Klambauer, G., Brandstetter, J., and Hochreiter, S. (2020). Hopfield networks is all you need. *CoRR*, abs/2008.02217.

Roder, M., de Rosa, G. H., de Albuquerque, V. H. C., Rossi, A. L. D., and Papa, J. P. (2022). Energy-based dropout in restricted boltzmann machines: Why not go random. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 6(2):276–286.

Rosenblatt, F. (1957). The perceptron - a perceiving and recognizing automaton. Technical Report 85-460-1, Cornell Aeronautical Laboratory, Ithaca, New York.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *nature*, 323(6088):533–536.

Šíma, J. and Orponen, P. (2003). Continuous-time symmetric hopfield nets are computationally universal. *Neural Computation*, 15(3):693–733.

Sutskever, I. and Tieleman, T. (2010). On the convergence properties of contrastive divergence. In Teh, Y. W. and Titterington, M., editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 789–795, Chia Laguna Resort, Sardinia, Italy. PMLR.

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I. J., and Fergus, R. (2014). Intriguing properties of neural networks. In Bengio, Y. and LeCun, Y., editors, *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*.

Tolmachev, P. and Manton, J. H. (2020). New insights on learning rules for hopfield networks: Memory and objective function minimisation. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE.

Vapnik, V. N. and Chervoneva, A. (1964). On class of perceptrons. *Automation and remote control*, 25(1):103.

Varadhan, S. R. S. (2008). Large deviations. *The Annals of Probability*, 36(2):397 – 419.

Wen, U.-P., Lan, K.-M., and Shih, H.-S. (2009). A review of Hopfield neural networks for solving mathematical programming problems. *European Journal of Operational Research*, 198(3):675–687.

Zhang, N., Ding, S., Zhang, J., and Xue, Y. (2018). An overview on restricted boltzmann machines. *Neurocomput.*, 275(C):1186–1199.