Assessing the limits of a temperature dependent Hopfield neural network with Hebbian learning.

MANUEL ALMAGRO RIVAS

Physics degree. Universidad de Granada.

M. Almagro: malmriv@correo.ugr.es

Compiled October 19, 2024

We implement a Hopfield neural network with Hebbian learning and Markov chain Monte Carlo methods. Three different variables that affect the network (temperature, number of stored patterns and data corruption) are studied. The maximum number of patterns is estimated at $P = (0.137 \pm 0.005)N$. The Traveling salesman problem is approached using a Hopfield network in order to showcase how these can be used for optimisation purposes. **Keywords:** ***artificial neural networks, Hopfield networks, content-addressable memory, pattern recognition, MCMC, computational optimisation.***

CONTENTS

1. INTRODUCTION.

Initially belonging to the realm of "automata studies", *nerve nets* (as they were called) were little more than mathematical objects with almost no possible practical applications. Nevertheless, neural networks have found their way into countless areas (especially with the advent of *deep learning* in the last few decades). In this work, we will focus on the Hopfield neural network (HNN), developed in the decade of the 1970-80. After a first paper describing their behaviour (Hopfield, 1982), some other studies were carried out in order to investigate the similarities between these artificial networks and biological networks of neurons (Hopfield, 1984). The results showed certain similarities, which prove HNNs interesting even if major simplifications are taken in their implementation. Hopfield networks can be represented as non-directed graphs (since every synapse is symmetrical). If the correct learning rules are implemented, HNNs show associative memory, and more specifically, *content-addressable* memory; a certain input will result in the network retrieving the desired pattern, even if the input has been heavily corrupted and there is no precise reference to the pattern to be retrieved. In this work we will implement Hebbian learning rules, and therefore this will apply. The difference with *address-based* memory (used in standard storage devices) is that these can only retrieve information if its exact location in the drive is known. We will show that the evolution of a HNN can be modeled as a Markov chain relying on Monte Carlo methods. We will specifically use the Metropolis-Hastings algorithm, developed by Metropolis et al. (1953) and later popularised by Hastings (1970). The behaviour of the network under non-optimal conditions will be studied. This includes studying the effects of data corruption, temperature and memory overload.

2. THEORETICAL BACKGROUND.

A. Connections with other branches of science.

Certain physical models from the realm of Statistical Mechanics are (mathematically) very closely related to Hopfield networks. Ising based models, where a discrete-valued lattice is used to represent the spins of the constituents in an ordered structure, share a common mathematical formulation with HNN. Specifically, the Hamiltonian of a square, $N \times N$ spin glass lattice is given by:

$$\mathcal{H}(s) = -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\left(s(i,j)\cdot\sum_{neighbours} s(i_v,j_v)\right) \quad (1)$$

We will see that the Hamiltonian of a Hopfield network (see Equation 5) is very similar to that of a spin glass model. It will become apparent (after introducing the necessary formalism) that methods commonly applied in these types of simulations can be successfully implemented to work with a HNN. Before going into implementation-related details, we need to characterise how neurons and are connected to each other in a Hopfield network. We will do this through the concept of *weights*). The impact each neuron carries on the whole network (which can be modeled using the concept of *bias terms*) will also be discussed here.

B. Hebbian learning.

The underlying assumption about the behaviour of individual neurons in a HNN can be best summarised under the so called Hebbian postulate: "*neurons that fire together, wire together*". Effectively, this implies that for any two neurons, there must exist some type of link between them. Furthermore, this link (the *synapse*, in biological terms) must be somehow strengthened when two neurons act together frequently, for they must be part of some sort of "neural circuit" dedicated to a certain function. (The notion that neurons that participate in the same type of activity frequently act together is well known to modern-day researchers; in fact, entire regions of the brain are known to become active when key stimuli appear). We will also assume that a particular neuron cannot connect with itself. This somewhat fuzzy idea will be translated into the concept of *weights*. For a system with N neurons, there must exist a total of $\frac{N(N-1)}{2}$ distinct connections between them. These connections are not directed: they just convey information about the strength of the synapse between any pair of neurons. Therefore, a neural network can be seen computationally as a vector with N elements storing binary digits. The weight matrix can be seen as a $N \times N$ symmetrical matrix with a null main diagonal ($\omega_{ii} = 0$) due to self-interaction being forbidden. The weights can be specified in numerous ways. In this work we will follow the Hebbian postulates (Hebb, 1949); for a pattern $\Xi = \{\xi_i\}_{i=1,..,N}$ (that is, the binary representation of the data we want to store), we will define the weight matrix like follows:

$$\omega_{ij} = \frac{1}{a(1-a)N}(\xi_i - a)(\xi_j - a) \quad (2)$$

(Where $a = \frac{1}{N}\sum_i^N \xi_i$). If the network is to store numerous patterns Ξ_μ ($\mu = 1, ..., P$), we will need to define the weight matrix over all of them. The weights matrix is then computed as:

$$\omega_{ij} = \sum_{\mu=1}^{P} \frac{1}{a^\mu(1-a^\mu)N}(\xi_i^\mu - a^\mu)(\xi_j^\mu - a^\mu) \quad (3)$$

(It is important to keep in mind that a must be computed separately for every pattern, not over all of them; hence the notation a^μ). A pattern is said to be a *strong attractor* when the associated weights matrix results in input data correctly converging to said pattern. All of this, of course, does not reveal anything about the limits of the network itself. Since any given matrix stored in floating-point representation cannot carry infinite information, there must be a number of patterns P that makes the network start behaving erratically. This is not due to computational limits (such as the upper bound imposed by using a matrix 8-byte entries): it is an intrinsic property of HNNs. This upper bound of patterns is usually expressed in terms of the constant α_c, defined as $\alpha_c = P/N$. We will give a more precise definition of what acting erratically means, and also find an approximation of this value. Furthermore, we will lay out the notion that the weights matrix somehow encodes the so-called *cost function* to be optimised, therefore enabling us to solve more sophisticated problems using a HNN.

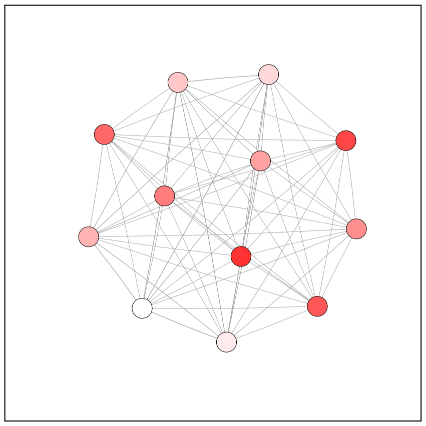

Fig. 1. Visual representation of a Hopfield network. Synapses are non-directed (they conform a *feedback network* as opposed to a *feedforward* network) and each neuron is directly connected to the rest of the network and has a particular bias (represented by color here).

As a side note: self-interaction has been found in real neurons, where they are called *autapses*. They can also be implemented in a HNN (Gosti et al., 2019) if the weights are defined so that self-interaction does not result in destructive behaviour. The resulting network can store more patterns, with a trade-off between capacity and accuracy.

C. Bias term.

Biological neurons do not fire whenever their environment changes. There is an associated electric potential called *threshold potential*, which needs to be exceeded for neurons to start reacting. In a HNN, neurons have an associated bias term, which is computed just once before any pattern retrieval has taken place. It is real-valued and static. The bias term of the i-th neuron, θ_i, will be given by:

$$\theta_i = \frac{1}{2}\sum_j \omega_{ij} \quad (4)$$

That is, the bias will be proportional to the sum of the weights from all the synapses connecting the i-th neuron with the rest of the network. A neuron that participates heavily in most patterns will have a higher bias term, which will translate in a more noticeable reduction in the energy function once that neuron's state changes, as we shall soon prove.

D. Defining the Hamiltonian.

There are multiple ways to define a Hamiltonian for such system. The definition we are to use incorporates both the weights and the bias terms:

$$\mathcal{H}(s) = -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\omega_{ij}s_i s_j + \sum_i s_i\theta_i \tag{5}$$

Equation 5 can be easily compared with Eq. 1. The role of neighbouring nodes in a spin-glass model has been replaced by the action of the weights. Furthermore, we have added a new term (the bias), but the overall expression remains very similar. Our objective is to lower the total energy of the system as the network reaches a state increasingly similar to one of the patterns that have been stored in it. The Hamiltonian describes the energy associated with the so-called *state space*. Throughout the state space lay the *attractors*, points where the energy of the state is a local minima. Some of these represent learned patterns (and their negatives); some attractors represent certain undesirable states known as *spurious states* which do not actually represent any pattern in the learning set. We will explore these concepts briefly. With the Hamiltonian in place, we still need to define an algorithm in order to automate the evolution of the system.

E. The Metropolis-Hastings algorithm.

Our network will evolve one neuron at a time, and each change could potentially increase or lower the energy of the network depending on the current state. The probability of many consecutive changes made at random lowering $\mathcal{H}(s)$ is small, because each attractor is one out of 2^N possible configurations. This points us in the direction of constructing a Markov chain —a statistical model used when an event's (usually small) probability of happening depends solely on the current state of the system—. This would allow us to take controlled steps towards a more stable state, instead of updating the network at random expecting improvements. But constructing Markov chain only implies that the choice of whether to update a neuron or not will be *guided*. The choice of *which* neurons should be updated is best left to randomness, in order to produce a more natural (but still statistically sound) evolution of the system. Thus, we will generate a Markov chain using Monte Carlo methods. These techniques are known as *MCMC methods*, for short. They have proven fruitful in spin glass models research. Let us introduce the necessary formalism. The probability of a given system going from a state X_1 to a state X_N is given by:

$$P_N(X_1, ..., X_N) = P(X_1)\cdot T(X_1 \to X_2)\cdot(...)\cdot T(X_{N-1}\to X_N) \tag{6}$$

Where $P(X_i)$ is the probability of the state X_i taking place and $T(X_i \to X_j)$ is the probability of a transition between states X_i and X_j happening. If we consider the probability of a state X taking place at a time t, $g(X,t)$, we can conclude:

$$g(X, t+1) = \sum_{X'} g(X', t)\cdot T(X' \to X) = \tag{7}$$

$$\sum_{X'\neq X} g(X', t)T(X' \to X) + g(X,t)\cdot T(X \to X) \tag{8}$$

In Eq. 8 we are merely recognising that it is possible that state X is already taking place, and therefore we must take it out of the sum. If $T(X \to X) = 1$ (the probability of the current state happening must be 1 by definition), we get:

$$g(X, t+1) = g(X,t) + \sum_{X'\neq X}\Big(g(X',t)T(X'\to X) - g(X,t)T(X\to X')\Big) \tag{9}$$

We have expressed the previous sum considering that the probability of reaching state X' implies *getting there*, $T(X' \to X)$, and *not going back*, $T(X \to X')$. Equation 9 is known as the **master equation**. For a state X to be stationary, it would need to verify $g(X,t) = g(X,t+1) := g(X)$. In other words, the sum in Eq. 9 should need to be null, which implies:

$$\boxed{g(X',t)T(X'\to X) = g(X,t)T(X\to X')} \tag{10}$$

That is, the probability of the system switching back and forth between state X and any other close state X' should be identical. Therefore the system constantly oscillates around state X by undergoing very small fluctuations. Equation 10 is known as the **detailed balance equation**. From this, we can infer:

$$T(X' \to X) = \frac{g(X,t)}{g(X',t)}T(X \to X') \tag{11}$$

The quotient in Eq. 11 is key in understanding Markov chains as a useful technique. It is a quotient between probabilities. This implies that any probabilistic analysis involved from now on does not require us to know exactly which function $f(X,t)$ directs the evolution of the system, but **only a function that is proportional to** $f(X,t)$. In spin glass models, such function could take the form of $e^{\beta\mathcal{H}(X)}$. Therefore, the quotient in Eq. 11 takes the form:

$$T(X' \to X) = e^{-\beta(\mathcal{H}(X)-\mathcal{H}(X'))}T(X \to X') \tag{12}$$

We still do not know which form $T(X \to X')$ would take. We can use the equation suggested by Metropolis et al. (1953), which is:

$$T(X \to X') = \min\left(1, e^{-\frac{\beta}{\mathcal{T}}(E(X')-E(X))}\right) \tag{13}$$

Where $\mathcal{T}$ is the temperature. The parameter β affects the shape of the function; for higher β values, the function resembles a discrete step. In this work we will implement $\beta = 1$ (represented by the dark blue line in Figure 2).

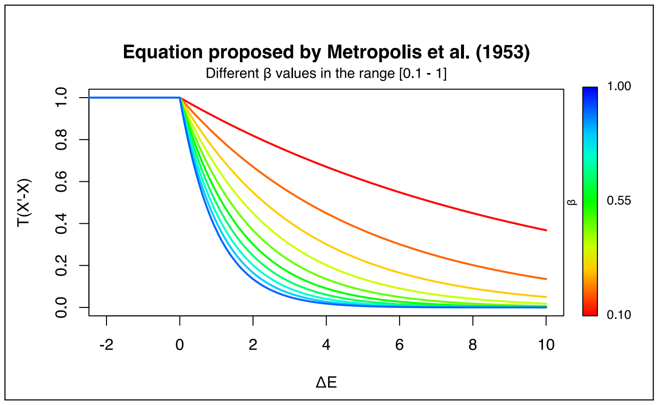

Fig. 2. Equation 13 plotted for different values of β.

If a change in the system results in energy decreasing, $e^{-\frac{\beta \Delta E}{T}} > 1$ and the function will always return the minimum value, 1. If the energy increases, $e^{-\frac{\beta \Delta E}{T}} < 1$ and the function will return the value computed by the exponential function. The algorithm suggested by Metropolis et. al would be:

Algorithm 1. Metropolis algorithm

```
1: procedure METROPOLIS
2:     $i \leftarrow \mathcal{U}(0, N)$                          ▷ go to random neuron
3:     $p \leftarrow \min(1, \exp(-\frac{\beta \Delta E}{T}))$    ▷ probability of switch
4:     $\xi \leftarrow \mathcal{U}(0, 1)$                        ▷ random value in [0,1]
5:     if $\xi < p$ then
6:         switch i-th neuron
7:     go to 2
```

We are comparing the probability p of a neuron changing its state against a random value from a uniform distribution $\xi \in \mathcal{U}[0, 1]$. The value of p depends, of course, on how a given change affects the global energy of the system. Note that when Algorithm 1 is applied N times, every neuron has had an equal chance of changing its value. We call every stack of N iterations a Monte Carlo iteration.

3. OBJECTIVES.

We will implement a HNN following the aforementioned specifications. We will select a size of $N = 1024$ neurons, since this would allow us to store data corresponding to 32 px × 32 px images. The parameter in the Metropolis equation will be fixed as $\beta = 1$, as was mentioned before. The patterns will be both non-random (when they are few) and random (when we need to test the limits of the network, thus needing a lot of patterns). Several different variables will be studied.

1. **Anomalies in pattern retrieval**. There are two main anomalies frequently associated with HNN: negative pattern retrieval, and spurious pattern retrieval.
2. **Energy profile**. We will check that the energy of the network decreases until it has reached a minimum. We will do this for several patterns, but our analysis will remain qualitative for the most part.
3. **Data corruption role in pattern retrieval**. We will study how the amount of noise applied to a pattern affects the ability of the network to retrieve it from memory.
4. **Temperature role in pattern retrieval**.
5. **Memory overload role in pattern retrieval**. We will test the maximum number of patterns the network can store without acting erratically. This is, we will check the percentage of patterns recalled with at least a 75% accuracy for different numbers of patterns. The ratio $\alpha_c = P/N$ will be computed.
6. **Going further**. A HNN can be used to solve (or, at least, approximate) a different problem where optimisation is key: the Traveling salesman problem. The basic procedure is outlined.

In order to do this, we need to define a metric that will allow us to measure quantitatively the correspondence between any state s and a given pattern $\Xi_\mu = \{\xi_i\}_{i=1,..,N}$. This metric, which is known as the *overlap parameter*, is given by:

$$m^\mu(s) = \sum_{\mu=1}^{P} \frac{1}{a^\mu(1-a^\mu)N}(\xi_i^\mu - a)(s_i - a) \quad \mathbf{(14)}$$

Its domain is $[-1, 1]$. A value of $m^\mu(s) = 1$ means that the network has retrieved the pattern μ, while a value of $m^\mu(s) = -1$ implies that the network has retrieved the *negative* (see Figure 4).

4. IMPLEMENTATION.

The implementation of a HNN is straightforward. Nevertheless, there are two problems that should be overcome before engaging in a thorough study. These concern the efficiency of computing changes in energy, and the conversion between image files and binary data.

A. Computing ΔE efficiently.

The Hamiltonian of a HNN (Eq. 5) implies that, in order to compute $\Delta\mathcal{H}$, one needs to reevaluate the whole network N^2 times/iteration. Some algebraic manipulation can take time complexity down to $\mathcal{O}(N)$. In order to achieve this, suppose that the $i-$th neuron changes its state from $s_i = 0 \rightarrow s_i = 1$. This means:

$$\Delta\mathcal{H}_i(s) = \Big(-\frac{1}{2}\sum_{j=1}^{N}\omega_{ij}s_j + \frac{1}{2}\sum_{j=1}^{N}\omega_{ij} \Big) - 0 = \frac{1}{2}\sum_{j=1}^{N}\omega_{ij}(1-s_j) \quad \mathbf{(15)}$$

The opposite change ($s_i = 1 \rightarrow s_i = 0$) gives:

$$\Delta\mathcal{H}_i(s) = \frac{1}{2}\sum_{j=1}^{N}\omega_{ij}(s_j - 1) = -\frac{1}{2}\sum_{j=1}^{N}\omega_{ij}(1-s_j) \quad \mathbf{(16)}$$

Since Eq. 15 and Eq. 16 only differ by a minus sign, we can establish that, for any $s_i \rightarrow s_i'$ change:

$$\boxed{\Delta\mathcal{H}(s_i \rightarrow s_i') = \frac{s_i' - s_i}{2}\sum_{j=1}^{N}\omega_{ij}(1-s_j)} \quad \mathbf{(17)}$$

Equation 17 implies that we only need to run through the network once in order to know $\Delta\mathcal{H}$.

B. Converting images $\leftrightarrow$ binary data.

In order to convert images to binary data, we need to use lossless storage formats (like BMP).

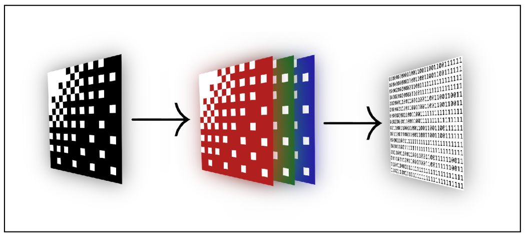

Fig. 3. The R/G/B channels can be accessed independently, and turned into binary data.

By saving images as 8 bits, 3 channels (red/green/blue) files, each pixel will be stored as a numerical vector with three entries ranging from 0 to $2^8 - 1 = 255$. (This decision is not arbitrary. Formats like JPG apply compression filters based on Fourier analysis in order to discard lower-frequency colors, and thus any given image cannot be accessed and read pixel by pixel). Conversion to a binary string can be achieved by accessing a single channel, normalising every entry by 255, and rounding to the nearest integer in the case of grayscale images (either 0 or 1). Converting binary digits to images is straightforward, since it can be done by using plotting software of one's choice. In this work, we will use *R 4.0.0* along with the *bmp* package in order to access BMP files. Minimal working examples of the code written for our purposes is available in the section of Online resources. This process allows us to manipulate large quantities of images at once, which will be necessary.

C. What do we mean by data corruption?

Data corruption has not been defined yet. From now on, X% data corruption means that the original pattern has been modified so that each pixel has an X% chance of changing its current state. This means that 0% data corruption leaves the pattern unaltered, 50% corresponds to plain noise with no information retained whatsoever, and 100% corresponds to the negative version of the pattern.

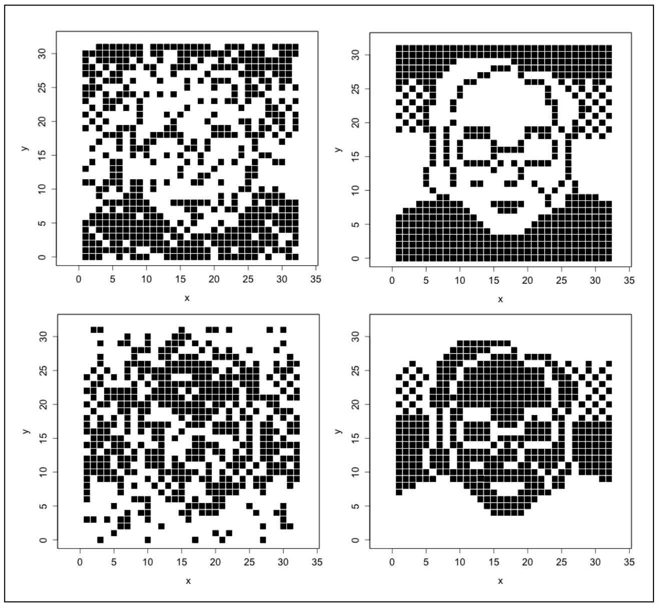

Fig. 4. Top left: pattern corresponding to "asimov.bmp" corrupted at 20%. **Top right:** retrieved pattern. **Bottom left:** same pattern, corrupted at 80%. **Bottom right:** retrieved *negative.*

5. OBJECTIVES.

A. Anomalies in pattern retrieval.

A.1. Negative patterns.

Getting the negative of a pattern (see Figure 4) is a common occurrence in Hopfield networks. This happens because both a pattern and its negative represent minima in the state space due to signs cancelling out in Eq. 5. This could potentially be a problem. But we have defined the overlap parameter, which allows us to check whether the retrieved information corresponds to the original pattern ($m^\mu(s) = 1$) or its negative ($m^\mu(s) = -1$). Exploring the evolution of the energy both during pattern retrieval and negative retrieval reveals that both processes show the same energy profile. The energy diagram corresponding to the patterns in Figure 4 can be seen in Figure 5. The slight difference in energies after convergence has been achieved is due to initial states being different; Figure 5 only shows change in energy, not absolute energy.

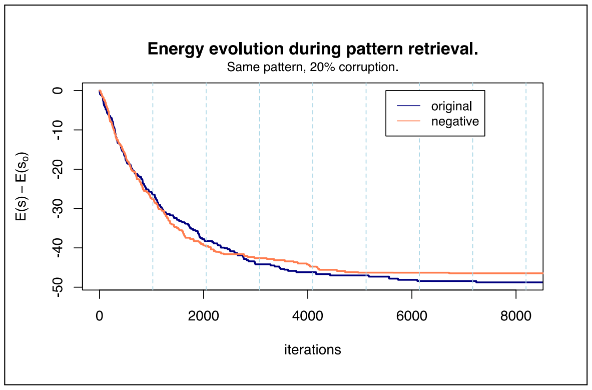

Fig. 5. ΔE during pattern and negative retrieval. Vertical lines represent each Monte Carlo iteration.

A.2. Spurious states.

Spurious states are those that correspond to local minima in the state space, but do not match any pattern learned by the network. They are, commonly, superpositions of an odd number of learned patterns. Their mathematical description has proven difficult, but some approximations can be made. If one supposes the patterns V_i ($i = 1, 2, ..., P$) in the learning set to be mutually orthogonal (in the sense of vector orthogonality: $V_i^T V_j = \mathcal{C}\delta_{ij}$, where δ_{ij} is the Kronecker delta function and $\mathcal{C}$ a constant), then an interesting result can be proven (Akra, 1988). For small values of P, the prevalence of spurious patterns can be checked computationally. But for values of P larger than $P = 7$ (which is, by far, the most interesting case) the number of minima $\mathcal{M}$ in the state space corresponding to valid configurations can be bounded by:

$$\frac{3^P}{2} < \mathcal{M} < \frac{2^{P^2}}{P!} \qquad \textbf{(18)}$$

While the number of spurious patterns, $\mathcal{S}$, is given by $\mathcal{S} = \mathcal{M} - P$. Thus, when many patterns are stored ($P \approx 0.136N$), the state space is filled with minima corresponding to spurious patterns. And, since in the study developed by Akra (1988) the size of the network does not seem to partake in this relationship, there is little that can be done to solve this problem. Making the network bigger can result in minima getting further apart, but they will not decrease in number and it is unclear if the convergence region for those minima would also increase as a result. This is an important limitation of Hopfield networks, and also the reason why they cannot store as many patterns as it might seem, as we will briefly explore.

B. Energy profile.

Initially, $P = 10$ patterns were loaded onto a HNN ($N = 32^2$). These patterns were not random, but represented different 32×32 black & white images recognizable by a human observer, like faces, plants, symbols and objects. The network received each pattern after a 20% corruption filter was applied to it as input. The energy change of the net was monitored.

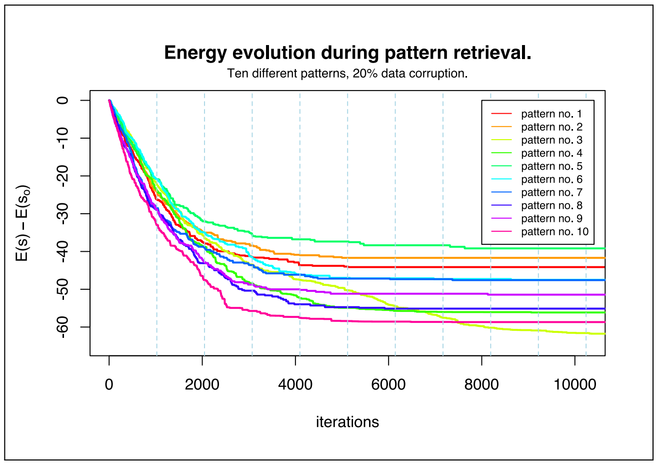

Fig. 6. ΔE monitored for 10 different patterns. Vertical lines represent each Monte Carlo iteration.

Adequate convergence was human-checked. There is not much one can inquire about Figure 6, except for the fact that different patterns seem to display different values of ΔE even though every iteration starts with the same amount of data corruption. Finding the reason behind these differences implies understanding what makes a strong attractor in the state space. An initial guess could be made looking at the definition of the weights matrix. Weights are computed attending to each pattern's deviation from its own "average bit", a^{μ} (see Eq. 3). Consequently, one could hope to find some kind of correlation between how much any given pattern deviates from the most "frequent pattern", and how strong it is as an attractor. (We will reveal in advance that this does not seem to be a good framing of the problem). In order to study this, the *global average pattern* defined. It was computed by calculating the average of each individual pixel across all patterns. That is:

$$\bar{s}_{ij} = \frac{1}{P}\sum_{\mu=1}^{P}\left(\frac{1}{N}\sum_{i,j} s_{ij}^{\mu}\right) \qquad \textbf{(19)}$$

Two magnitudes were computed afterwards:

1. The percentage by which every individual pattern deviates from the global average pattern.
2. The mutual information between every individual pattern and the global average pattern.

(The mutual information $I(X;Y)$ between two sets X and Y is defined as $I(X;Y) := H(X) - H(X|Y)$, where $H(X)$ is the Shannon's entropy of the set X and $H(X|Y)$ is the joint entropy of both sets; this was computed using a standard R function). Both of these magnitudes were plotted against ΔE, showing no correlation at all (see Figure 7). Therefore, how "different" a state is from the average pattern is not a good predictor of ΔE. Looking at Eq. 17 it is evident that a larger absolute value of ω_{ij} would contribute to a steeper ΔE profile; so we might look into the relationship that patterns hold with their weights. Whatever relationship might exist, it is clear that is must be non-trivial. We know that for $a = 1$ and $a = 0$ (totally blank patterns) we will find a division-by-zero singularity. But for any value of a in between, we can perform a simulation. We generated 512 random patterns, taking care that each one holds an a value incrementally greater than the last in order to cover the full interval $]0,1[$. Then, weights were computed for each pattern, and the resulting weight matrix was averaged. Results can be seen in Figure 8.

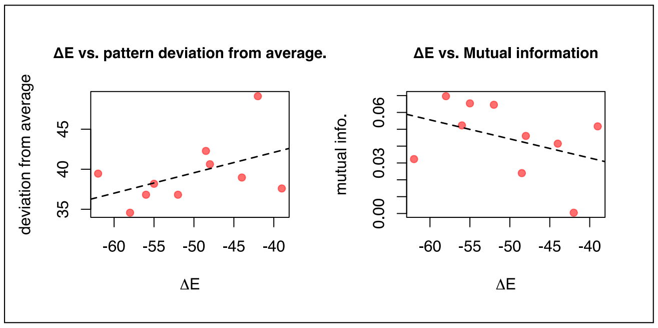

Fig. 7. Two different measures of how "unique" a pattern is were plotted against ΔE with no correlation showing.

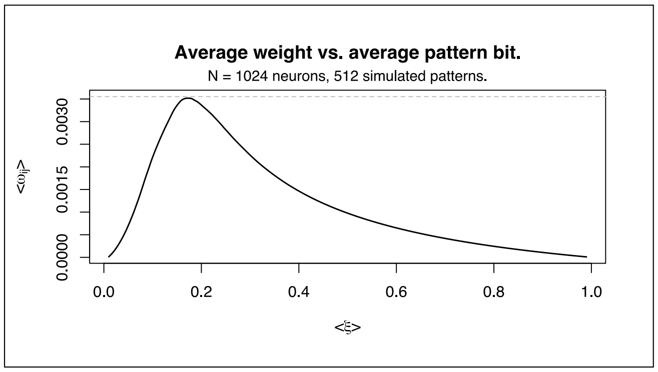

Fig. 8. Relationship between $a = \langle\xi\rangle$ and the average value of the weights matrix

.

Patterns with an $a_{max} \approx 0.18$ generate the highest ω_{ij} values on average. Therefore, a further study that enables us to take more variables into account (like N, the cumulative effect of memorising more patterns, etc.) could be useful in understanding precisely how to determine which patterns are going to make for stronger attractors. That lays outside the scope of this work.

C. Data corruption role in pattern retrieval.

A conservative amount of data corruption of 20% has been applied whenever necessary in former sections. Nevertheless, it could prove interesting to see how much a given pattern can be corrupted before the network has trouble recalling it correctly. In order to gather data, we set up a 32×32 HNN with $P = 100$ random patterns. For each essay (represented with a point in Figure 9), 50 patterns were picked at random from the set of 100 available patterns. A selected percentage of data corruption was applied, and the network received these fifty corrupted patterns as input. The output was evaluated using the overlap parameter introduced in Eq. 14, and any retrieval with an overlap greater than +0.75 was deemed successful. The essay was repeated eleven times, in increments of +5% data corruption starting from 0% and up to 50% (where all information has been lost). Figure

9 shows an interesting result. At low levels of data corruption, the network shows subpar performance: it was not able to fully retrieve the correct pattern every time, even though the initial deviation from it was minimal. Upon close inspection, one can see the network was not struggling to recall the correct pattern μ ($m^{\mu} < +0.75$), but simply recalled an incorrect option μ' (with $m^{\mu'} > +0.75$). Nevertheless, data corruption greater than approximately 19% results in the opposite behaviour. When much of the initial pattern has been corrupted, the HNN starts making more right guesses than a linear relationship would predict. For patterns with 40% data corruption about 80% of the original information has been lost. Nevertheless, the HNN is able to retrieve the correct pattern about 48% of the time. (The approximate percentages are due to the smooth fit having an associated uncertainty that is not negligible). But entropy always wins, and no information can be meaningfully extracted from a completely random pattern. At 50% data corruption as defined in this work, the only way a certain input results in the "correct" output is pure chance. This is what we see in Figure 9. Therefore, one can conclude that HNNs as described here are not very susceptible to data corruption, and are therefore a robust means for pattern storage.

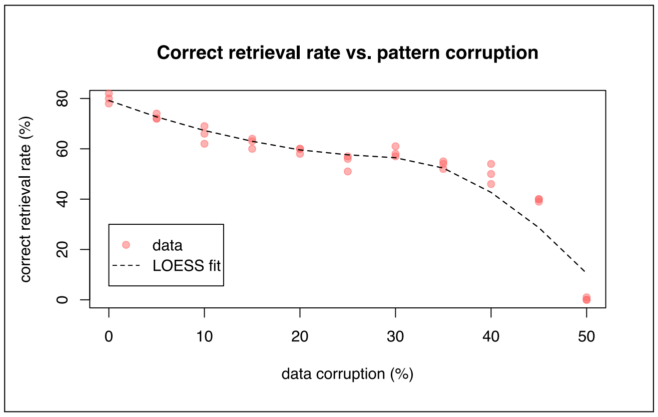

Fig. 9. Relationship between data corruption and success retrieval

D. Temperature role in pattern retrieval.

The function proposed by Metropolis et al. (Eq. 13) includes temperature as a variable. It can be shown that, for $\mathcal{T} = 0$, a change that reduces the energy of the system ($\Delta E < 0$) will result in $T(X \to X') = 1$, and therefore said transition will always take place. A change that increases the energy of the system ($\Delta E < 0$) at $\mathcal{T} = 0$ will, on the other hand, never take place. But it could be interesting to see the effects of different temperatures. The HNN was again loaded with $P = 10$ patterns, and received any of those patterns with 20% data corruption for each essay. Fifty essays were conducted, each one with a temperature increasingly higher in the interval $T \in [0, 0.3]$ K. The overlap with the correct pattern was monitored in each essay, and the results were plotted (Figure 10) using color-coded lines.

The trend is clear. For $\mathcal{T} = 0$, the network reaches the desired configuration soon. But as temperature raises, overlap decreases steadily. Plotting only the overlap after 10 MC iterations as a function of temperature, a linear relationship becomes more evident (Figure 11). The linear fit agrees well with the data ($R^2 = 0.9573$). An important remark should be made: even though

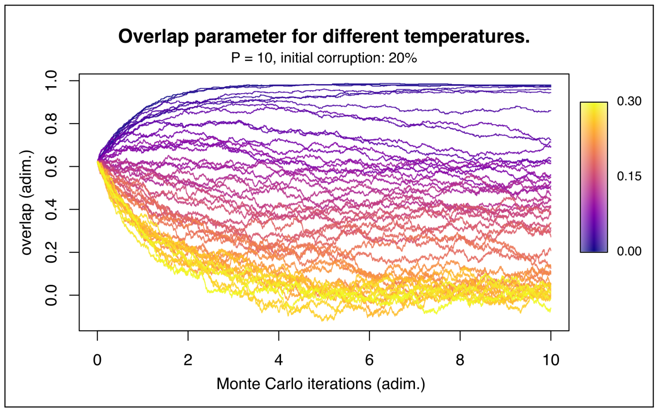

Fig. 10. Overlap parameter vs. Monte Carlo iterations. Fifty essays with $P = 10$ and 20% data corruption were carried out. Warmer colors correspond to higher temperatures.

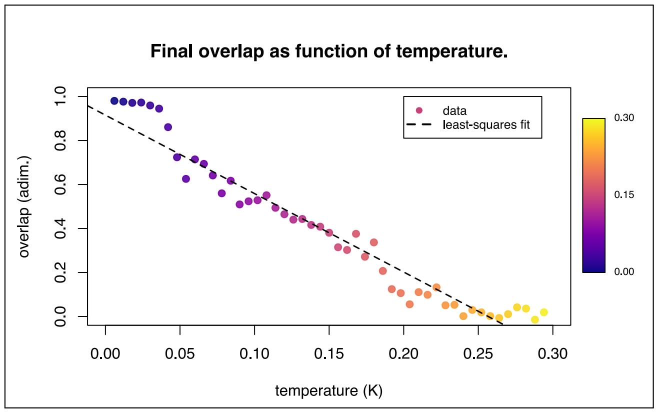

Fig. 11. Final overlap vs. $\mathcal{T}$. Least-squares fit: $m(\mathcal{T}) = (-3.57 \pm 0.12)\mathcal{T} + (0.93 \pm 0.03)$ and $n = 50$, with $R^2 = 0.9573$.

HNNs have been proven very resistant to data corruption, the tolerance for the effects of temperature is very low. This might seem like an unimportant finding, but the role of unpredictable fluctuations should be taken into consideration, especially in bigger networks. Metastable states are common in spin glass models, and the probability of a metastable state giving way to a completely stable state is very small. Setting a temperature that is low but non-null could help overocome these problems. In order to account for temperature-induced noise, a possible solution to retrieve the desired pattern could consist of two distinct steps; one where the same input is given to the network several times, and a second one where the various outputs are averaged to reconstruct the real pattern (so that any remaining noise averages out of the pattern).

E. Memory overload role in pattern retrieval.

As mentioned earlier, a finite netowork cannot store infinite information. Therefore, there must exist an upper bound for P. The relationship of P with the size of the network is thought to be linear. This ratio is frequently symbolised by $\alpha_c = P/N$, and many estimations are available online. Nevertheless, not many authors make their methodology explicit, and instead use vague terms like "well retained patterns". We would like our methods to be as clear and quantitative as possible in order to allow others to be able to be able to replicate this finding. In our case, we started with a 32×32 HNN as specified before. An initial estimation showed that the network did not have problems retrieving patterns when P is in the order of dozens. After setting $P = 200$ random patterns, the network showed a more irregular behaviour. Therefore, the upper bound in our study is set as $P_{max} = 200$. Up to 200 random patterns were therefore generated, and 40 HNNs were prepared. Each network stored five patterns more than its predecessor, starting at $P_1 = 5$ and finishing at $P_40 = 200$. For each essay (represented with a single point in Figure 12), an already trained network with a fixed P_i was given as input 50 random patterns after applying a 10% data corruption filter to them. The networks were left to evolve for 4000 single-step iterations. The criteria for considering each output as "correct" was that the corresponding overlap parameter be greater than $+0.75$. (Of course, networks with $P_i < 50$ had to converge towards the same patter more than once; but the 10% data corruption filter was applied every time, so they started from different states every time). For each essay consisting of 50 attempts of recovering the right pattern, the success percentage was computed. The results can be seen in Figure 12.

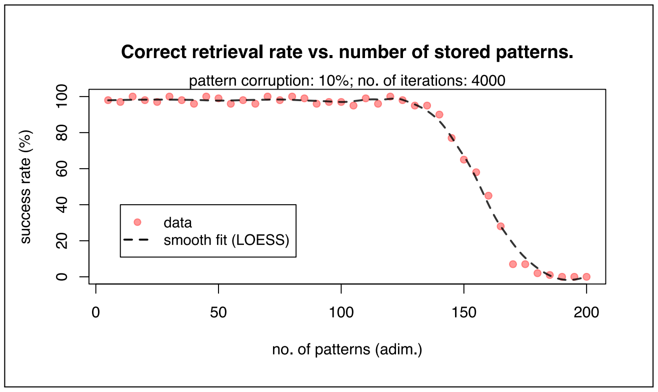

Fig. 12. Effect of different values of P in the network's ability to retrieve patterns.

The network is able to retrieve the correct pattern for low values of P almost every time. At some point between $P = 100$ and $P = 150$, the network's efficacy to retrieve the correct pattern decreases rapidly. By $P = 200$, the network no longer has any ability to recall patterns correctly. In order to best determine the cut-off P after which the network's efficacy starts decreasing, the discrete derivative of the success rate vs. P was computed in absolute terms. Figure 13 shows the absolute value of the derivative. The first value that breaks the linear tendency is found at $P = 140$. Since the essays were carried in increments of $\Delta P = 5$, we get an approximation of $\alpha_c = \frac{140 \pm 5}{32^2} = 0.137 \pm 0.005$. The network can store a number of patterns equivalent to (13.7 ± 0.5) % of its size N. This is consistent with the findings of Hopfield himself ($\alpha_c = 0.15$, in Hopfield, 1982) as well as more recent ($\alpha_c \approx 0.14$ by Folli et al., 2017) and precise ($\alpha_c = 0.138$, by Anderson, 1995) estimations. This limit can seem low when considering the network itself (which can be in any of 2^N different states) and the weights matrix (which requires allocating N^2 entries in memory, from which $\frac{N(N-1)}{2}$ would be distinct). The explanation is that a HNN does not store raw patterns, but turns them into *attractors* in the state space, each of which has a certain convergence domain. The trade-off is that, even if less patterns can be stored, convergence is guaranteed even if some data has been corrupted. But when P goes beyond $0.14N$, there are too many attractors in the pattern space, and therefore convergence towards the correct pattern is not guaranteed. Knowing the value of α_c is useful because the size of the HNN can be selected depending on how many patterns are to be memorised.

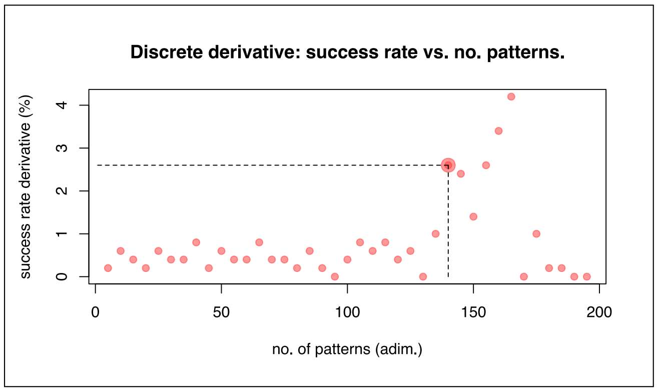

Fig. 13. Discrete derivative of the data shown in Figure 12. The first value that breaks the constant trend is located at $P = 140$. The absolute value has been taken because only a break in the trend was necessary.

F. Other applications.

So far we have only used Hopfield networks as pattern storage devices. Even if this is their main use, one should not lose sight of the core idea underlying their behaviour: a given energy function is minimised in successive steps. The methods used here (and Markov chain Monte Carlo methods in general) do not impose restrictions on the energy function to minimise. Therefore, $\mathcal{H}(s)$ can be defined as the energy associated with any system whose configuration we want to make more *stable*. Numerous problems are susceptible to be approached via Hopfield networks, but the Traveling salesman problem (TSP) is especially attractive. It was already approached by Hopfield & Tank (1985), although their approach was different to the one we will introduce now. This section is based on the alternative solution provided by Rojas (1996), although the notation has been changed for consistency. The problem is sufficiently easy to explain and visualise, but the correct solution(s) are tremendously hard to find by brute force. Given a set of N cities and the distances between them, what is the optimal route to follow (in terms of distance) if one wishes to visit every location just once? There are $\frac{(N-1)!}{2}$ possible combinations. Therefore, a set of 50 cities implies checking 3×10^{62} routes, which would optimistically take 9×10^{32} times the current age of the universe in a domestic computer. A HNN approaches this problem by minimizing a correctly defined energy function. (For large N values, results are not guaranteed to be optimal, but it is still a large improvement over brute-forcing the solution). Suppose a set of N cities S_i, $i = 1, 2, ..., n$. Any given legal route (every city is visited exactly once) can be represented by a binary table:

$$\begin{array}{c|cccc} & \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{4} \\ S_1 & 1 & 0 & 0 & 0 \\ S_2 & 0 & 1 & 0 & 0 \\ S_3 & 0 & 0 & 1 & 0 \\ S_4 & 0 & 0 & 0 & 1 \end{array} \quad \textbf{(20)}$$

That is, the first city to be visited is S_1, the second city is S_2, and so on. Note that, for a route to be allowed, there should be just one non-zero element per row and column. In any other case, the same city would be visited more than once (two non-zeros per row) or two cities would be visited at once (two non-zeros per column). Therefore, we need to define $\mathcal{H}(s)$ so that this situation is favored. If the elements in Table 20 are to be named x_{ij}, then a way of assigning lower energies to situations with just one non-zero element per column could be:

$$\mathcal{H}_{col} = \sum_{j=1}^{n} \left(\sum_{i=1}^{n} x_{ij} - 1 \right)^2 \quad \textbf{(21)}$$

Likewise, assigning lower energies to situations with just one non-zero element per row could be:

$$\mathcal{H}_{row} = \sum_{i=1}^{n} \left(\sum_{j=1}^{n} x_{ij} - 1 \right)^2 \quad \textbf{(22)}$$

Any route would be legal by this point, but not optimal. This can be accounted for by implementing a matrix (d_{ij}) in order to represent the distance separating pairs of cities. The energy associated with a travel between cities would be:

$$\mathcal{H}_{travel} = \frac{1}{2} \sum_{i,j,k}^{n} d_{ij} x_{i,k} x_{j,k+1} \quad \textbf{(23)}$$

Where i and j refer to cities and k is the position of a city in the list of destinations. Only the case where $x_{i,k},\ x_{j,k+1} > 0$ results in non-zero addition. The complete energy function to minimise is given by:

$$\mathcal{H} = \mathcal{H}_{travel} + \frac{\gamma}{2}(\mathcal{H}_{col} + \mathcal{H}_{row}) \quad \textbf{(24)}$$

Where γ is an free parameter that accounts for the relative importance of validity and optimal distance. We will not go into more detail because we will not implement this network. We only want to showcase how a HNN could be used for a purpose other than pattern recognition. Choosing an adequate energy function allows for activities such as finding optimal positions in a lattice, minimizing distance in a route, scanning pictures for patterns and even handling air traffic control efficiently (Kumar, 2016).

6. SOME CONCLUSIONS.

Certain conclusions can be extracted from this work.

1. **Neural networks are inherently limited in their storage capacity**. If storage is a constraint, this should be taken into consideration.
2. **Temperature plays a negative role in pattern retrieval**, but there are good reasons to suspect a good implementation could be beneficial for very large networks prone to metastable states.
3. HNNs **are adequate for highly corrupted input data**. For small amounts of noise they still function well, but their performance seems to be suboptimal.
4. A good choice for $\mathcal{H}$ can make **HNNs really versatile**, as demonstrated by their capacity to solve problems of diverse nature, such as the TSP.

We hope this work provided some insight into the underlying principles, limitations and applications of Hopfield neural networks.

7. ONLINE RESOURCES.

A generic version of the Fortran code written during the elaboration of this work can be found in the author's GitHub profile, along with some of the R scripts used in various sections.

8. BIBLIOGRAPHY.

1. Hopfield, J., 1982. *Neural networks and physical systems with emergent collective computational abilities*. **Proceedings of the National Academy of Sciences**, 79(8), pp.2554-2558. doi: 10.1073/pnas.79.8.2554
2. Hopfield, 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), pp.3088-3092. doi: 10.1073/pnas.81.10.3088
3. Hebb, D., 1949. *Organization Of Behavior: A Neuropsychological Theory*. Mahwah, N.J.: **L. Erlbaum Associates, Inc**.
4. Gosti, G., Folli, V., Leonetti, M. and Ruocco, G., 2019. *Beyond the Maximum Storage Capacity Limit in Hopfield Recurrent Neural Networks*. **Entropy**, 21(8), p.726.
5. Metropolis, N., Rosenbluth, A., Rosenbluth, M., Teller, A. and Teller, E., 1953. *Equation of State Calculations by Fast Computing Machines*. **The Journal of Chemical Physics**, 21(6), pp.1087-1092. doi: 10.1063/1.1699114
6. Hastings, WK (1970). "Monte Carlo Sampling Methods Using Markov Chains and Their Applications" (PDF). Biometrika. 57: 97–109. doi: 10.1093/biomet/57.1.97
7. Folli, V., Leonetti, M. and Ruocco, G., 2017. *On the Maximum Storage Capacity of the Hopfield Model*. **Frontiers in Computational Neuroscience**, 10. doi: 10.3389/fncom.2016.00144
8. Kumar, K., 2016. *Hopfield Neural Networks for Aircrafts' En-route Sectoring: KRISHAN-HOPES*. **IJCSN**, 5(1), pp. 149-156.
9. Akra, M., 1988. *On The Analysis of The Hopfield Network: A Geometric Approach*. **Massachusetts Institute of Technology**. Available online.
10. Anderson, J., 1995. *An Introduction To Neural Networks*. 3rd ed. Cambridge, Mass.: **MIT Press**, p. 412.
11. Rojas, R., 1996. *Neural Networks: A Systematic Introduction*. Berlin: **Springer**, pp. 360-365.
12. Hopfield, J.J., Tank, D.W. *"Neural" computation of decisions in optimization problems*. **Biol. Cybern.** 52, 141–152 (1985). doi: 10.1007/BF00339943