Bocconi

Bachelor of Science in Economics,
Management and Computer Science

# Navigating Complex Energy Landscapes: Replicated Approach to the Binary Perceptron

**Bachelor of Science thesis by:**

Leonardo Tonelli

student ID no 3216378

**Advisor:**

Prof. Luca Saglietti

Academic Year 2024-2025

# Contents

# 1 Introduction

The fields of deep learning and artificial neural networks are experiencing an exponential rise of interest, largely motivated by their remarkable performance in many pattern recognition tasks (e.g. natural language processing [1], image recognition [2]). However, the underlying mechanisms governing these architectures remain largely **unexplained**, for the most part driven by intuitive heuristics. While this approach, coupled with increasing computational power, had proven effective, significant potential is lost due to the lack of profound understanding. This article aims to explore algorithms inspired by a **deep understanding** of the problem, illustrating the importance of problem exploration in designing effective new solutions.

The problem analyzed is the **Binary Perceptron**, the foundational component of deep learning architectures but with the added complexity of binary synaptic weights. While the Binary Perceptron itself may not be directly applied to contemporary problems, it serves as source of inspiration for more complex systems. The solution algorithms reviewed in this dissertation have, in fact, motivated applications in more advanced models, such as Deep Neural Networks.

The article draws inspiration from the work of **Baldassi et al.** [**3**], which will be frequently cited. Although their methods are carefully described, our implementation of their solutions will incorporate slight variations both to stimulate understanding and to offer a different perspective.

Since the solutions studied stem from statistical mechanics' description of the problem, we will clarify the connections between the seemingly disparate topics. From these insights, the intuition behind the methodology proposed by Baldassi et al. will become apparent, leading to two derived algorithmic solutions.

The primary objective of this article is to **communicate** a compelling approach to algorithm design, which is grounded in theoretical awareness of the problem. This text also aims to be **accessible** to readers with varying levels of expertise on the subject, thus extended derivations and more detailed information will be relegated in the Appendices. The complete implementations of the described solutions, along with the numerical experiments conducted, are available in the referenced code base [4].

# 2 Theoretical background

This chapter will clarify the connections between statistical physics and our optimization problem, introduce the Binary Perceptron model, analyze its complexity and lastly review the theoretical insights that inspired the new methodology applied in this article.

## 2.1 Statistical mechanics and computer science

Statistical Mechanics is a subfield of physics that specifically studies the behavior of microscopic entities by applying probability theory and statistical methods. The inherent randomness of the drastically small and the practical intractability of very large systems define the **necessity** of stochastic descriptions of microscopic dynamics. From the probabilistic descriptions of micro-states (i.e., the set of possible configurations of atoms in a system), statistical mechanics derives the equilibrium properties and macroscopic behavior of the system, explaining phenomena such as temperature, pressure, and entropy. Tools and frameworks from this approach to complexity have been applied across a wide range of disciplines, inspiring a variety of methods that have resulted in impactful, high-value applications.

In the field of computer science, these tools have been proven useful in multiple settings, most importantly in **combinatorial optimization** and machine learning. Equilibrium statistical physics finds fertile ground in these topics, as the foundational structure of such problems can often be reformulated as high-dimensional probabilistic systems, much like physical systems in statistical mechanics. Moreover, shared mathematical structures simplifies the transfer of knowledge between the disciplines. Many computational problems can indeed be characterized by configurations, cost functions, and constraints that closely resemble states, energies, and interactions found in physical systems. A central analogy, other than mathematical abstraction, lies in their **shared minimization principle**: in computational applications a cost function is usually to be minimized to find optimal configurations (e.g. optimization, machine learning, neural networks etc.), while physical systems evolve toward minimizing free energy by probabilistically exploring microscopic states. Focusing more narrowly on the problem discussed in this article, the connection between combinatorial optimization problems and physics is reviewed briefly in the following section.

## 2.2 Statistical physics and combinatorial problems

Combinatorial problems are characterized by astronomically large configuration spaces, where solutions' structure (where they cluster, how many exist, how "far apart" they are) is extremely difficult to describe. Statistical physics provides powerful tools to understand this structure, largely through spin glass theory—a central topic in the field—since the seminal work of Parisi and Mézard in 1985 [5]. Spin glasses themselves raise fundamentally combinatorial problems: finding low-energy configurations corresponds to solving complex optimization tasks. Because of this deep connection, the mathematical methods developed to study spin glasses, such as the replica and cavity methods [6], have been incredibly useful in **solution landscape descriptions**. These methods focus on analyzing random instances of combinatorial problems and studying their typical properties as the size of the problem increases. This average-case approach is particularly valuable because, in many combinatorial problems, worst-case instances are extremely rare in practice and often don't reflect the **typical behavior**. Statistical physics provides a way to bypass worst-case intractability by revealing the "typical structure" and complexity of solutions in large random configurations. The new algorithms studied in this review are directly inspired by this "typical" analysis of the solution landscape in the binary perceptron, as we will see in the following sections.

The parallelism between the two disciplines extends beyond the mere use of tools from one in the other. Additional similarities became evident when it was found that a specific combinatorial problem (k-SAT) "when promoted to its random version, exhibits a behavior which is strongly reminiscent of a statistical-mechanics phase transition" [7]. In other words, there were discovered drastic changes in satisfiability of the problem when gradually changing the specified difficulty of the problem. For example in random 3-SAT, there's a sharp transition around a clause-to-variable ratio of $\approx 4.26$. Below this, instances are likely satisfiable and easy; above this, they are mostly unsatisfiable—and instances near the threshold are computationally hardest. This similarity (which has been studied extensively in [8, 9, 10] and discovered for other problems such as TSP [11] and IPP [12]) found critical impact in the design of new algorithms, prediction of difficulty and the study of the solution landscape.

After an introduction to the Perceptron model, Section 2.4 will address a crucial transition phase in the Binary Perceptron problem.

## 2.3 Binary Perceptron model

The Perceptron model, introduced in 1958 [13], is the simplest type of neural network and it's a foundational idea for deep learning architectures. It is a computational model inspired by artificial neuron [14], where the activation of the neuron is determined by a weighted sum of its inputs (stimuli to synapses), followed by the application of a threshold-based function to produce a binary decision (if the neuron should react to stimuli or not). In mathematical perspective, the perceptron **maps** multidimensional input features to binary outputs, classifying data entries in a category (1 or -1). The simple framework can be described by the Diagram 1, which we can take as reference to explain both components and mechanisms of the model.

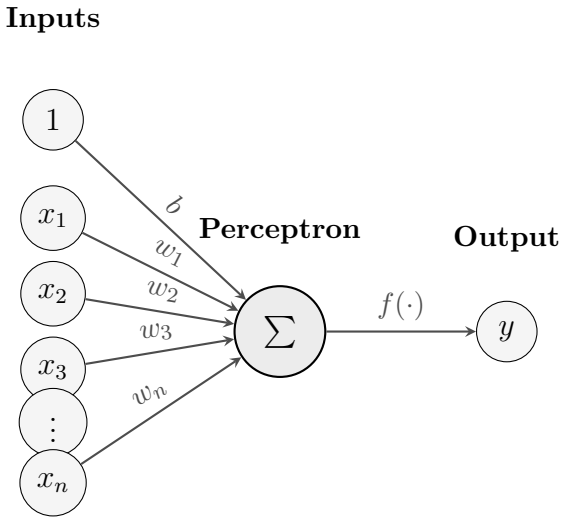



Figure 1: Perceptron Model Architecture

The architecture takes a real-valued vector $\mathbf{x} = [1, x_1, ..., x_N]$ as inputs (features of a data point), where each entry $x_i$ is multiplied by a specific weight $w_i$ and the sum of the results is passed as input to a function $f(\cdot)$ called activation function. The result $y$ can be then summarized by the following equation

$$y = f(b + x_1 w_1 + x_2 w_2 + ... + x_n w_n) = f(\mathbf{x}^T \mathbf{w} + b) \tag{1}$$

The parameter $b$ is called bias, and its contribution to the output is independent to the single data point vector $x$, this improves the flexibility of the classifier in learning the decision boundary. Although useful, in our binary case, no bias element was introduced. The activation function instead brings the continuous input to a binary value, and in its

simplest specification, is the step function $H(\cdot)$ where

$$H(x) = \begin{cases} -1, & x < 0 \\ 1, & x \geq 0 \end{cases} \tag{2}$$

Once specified correctly, this simple model classifies multi-attribute entities **solely** based on the weights parameters $\mathbf{w}$. Given a set of correctly classified entities, the problem becomes one of constraint satisfaction: how to adjust the weights $\mathbf{w}$ so that all the provided patterns are correctly classified.

Let's introduce this satisfaction problem mathematically. If we have a set of $K$ correctly classified points, we shall distinguish the correct classification $y$ from the perceptron prediction $\hat{y}$. Hence the vectorized version of Equation 1 is

$$\hat{\mathbf{y}} = H(\mathbf{X}\mathbf{w}), \tag{3}$$

where the function $H(\cdot)$ applied element-wise $H(\mathbf{y}) = [H(y_1), \ldots, H(y_K)]$, $X \in \mathbb{R}^{K \times N}$, $\hat{\mathbf{y}} \in \mathbb{R}^K$ and $\mathbf{w} \in \mathbb{R}^N$.

Therefore for each of its own instance, given a set of $K$ random input patterns $\{\mathbf{x}^j\}_{j=1}^{j=K}$, where each of them has a corresponding correct output $y^j \in \{+1, -1\}$, we want to find the optimal weight vector $\mathbf{w}^*$ such that the network output is equal to the desired value $y^j$ for all $j$. Borrowing notation from [15], the condition can be rewritten as

$$\sum_{j=1}^{K} \Theta(-y^j \hat{y}^j) = \sum_{j=1}^{K} \Theta(-y^j H(\mathbf{x}_j^T \mathbf{w} + b)) = 0, \tag{4}$$

where $\Theta(x) = 1$ if $x > 0$ and 0 otherwise.

The problem specification considered in this article is restricted to the complex case of a Binary Perceptron, where the synapses considered are restricted to **binary** values ($w_i \in \{+1, -1\}, \forall i \in \{1, ..., N\}$). Additionally, to make the problem computationally more accessible, the random inputs of the perceptron are considered to be binary vectors, so that $x_i \in \{+1, -1\}, \forall i \in \{1, ..., N\}$ (unlike the $x_i^j \overset{\text{i.i.d.}}{\sim} \mathcal{N}(0, 1)$ used in [16]).

The same optimization problem can be framed back into biological perspective, where the dilemma is how the synapses can adjust optimally to learn a specific neuron activation patterns. In other words how a neuron learns whether it should be active or not given a

certain input stimuli. This is still an open problem, but solution in the artificial setting are inspiring new insightful ideas in biology [17].

## 2.4 Solvability

Before addressing the solvability of the Binary Perceptron problem, it's important to define the **constraint density** $\alpha = \frac{K}{N}$, a key ratio used to benchmark the difficulty of Constrained Satisfaction Problems (CSPs). This number quantifies the number of constraints $K$ relative to the number of variables $N$. A higher constraint density indicates a higher number of constraints with respect to variables, making the problem more difficult. The converse is true for low values of $\alpha$. In the Binary Perceptron case, the constraints $K$ are the patterns that the neuron needs to learn to classify, while the amount of variables $N$ is the size of the weight vector $\mathbf{w}$. This ratio is also used to describe the **capacity** of a perceptron (or neural network), representing the maximum number of patterns per synapse that can be correctly classified (or "stored") by the most effective learning rule.

### 2.4.1 Continuous v. Discrete Perceptron

In its continuous version, the Perceptron model can learn the correct configuration $\mathbf{w}^*$ by applying the Perceptron algorithm [13], that updates the weights gradually using the update rule

$$(w_i)^{t+1} \to (w_i)^t + \eta(y^j - \hat{y}^j)x_i^j, \tag{5}$$

for all weights $i$, one input vector $j$ at a time, parametrized by the learning rate $\eta$. This algorithm proved to be very effective for simple linearly separable patterns, and served as the starting point for more complex learning procedures. However, this algorithm cannot be applied directly to our discrete case, where learning requires some additional considerations.

The discrete constraint on the weights pose a radical **increase in complexity** on our problem. The solution space configurations are drastically reduced and the binary nature of the values limits the solutions exploration. An evident proof of this complexity increase is the differing upper bounds for the maximum storage capacity of continuous versus discrete architectures. E. Gardner [18] calculated the maximum capacity of a standard continuous perceptron to be approximately $\alpha_c \approx 2$. On the other hand, Mézard's

calculations [19], employing statistical physics methods [20], predicted a maximum capacity of $\alpha_c \approx 0.83$. This reduction means that for a fixed size $N = 100$, we would expect to learn a maximum of $\approx 200$ patterns in the continuous case, while only $\approx 83$ in the discrete case. Then the maximum number of learnable patterns per synapsis is **more than halved** when transitioning from a continuous to a discrete weighted system. The maximum capacity for both problems represents effectively a transition phase in physical terms. Before that threshold the problem is solvable while on the other edge the solution any solution vanishes. These analysis are crucial when assessing the difficulty of the problem a priori.

### 2.4.2 Binary Perceptron solution landscape

The Perceptron problem with binary synaptic weights is known to be intractable [21] in the worst case, categorized as an NP-Hard problem. Nevertheless, tools borrowed from statistical physics provided valuable descriptions of the solution landscape in the typical case. The equilibrium description is dominated by an exponential number of **local minima** [22] that trap with ease standard cost minimization strategies such as Monte Carlo algorithms. Furthermore, research has shown that the optimal weight configurations are geometrically isolated, making them hard to find with local search movements [23].
In contrast with the pessimistic view of the physicists, algorithms have been found to effectively find optimal solutions [24, 25, 26]. This discrepancy was later resolved by a study by Baldassi et al. [27], which introduced a new large deviation analysis that unveiled the existence of a different class of solutions **clustered** in dense regions, **accessible** to easy learning protocols. These dense regions, not only justified the existence of effective algorithms, but represented an appealing region with more generalizable solutions. When a learning algorithm finds a solution within a dense cluster, it implies that small perturbations to the weights would most probably still result in a correct classification. This **robustness** makes these solutions less susceptible to noise and more adaptable to unseen data, improving their practical utility.

To demonstrate the accessibility of those regions, the researchers proposed a new methodology designed to derive simple algorithms capable of finding optimal solutions by actively seeking these dense clusters. The next section will elaborate on the large deviation statistics introduced by Baldassi et al., and the intuition behind the new methodology.

## 2.5 Interacting Replicas approach

In the article [15], the researchers propose a new measure designed to "ignores isolated solutions and enhances the statistical weight of large, accessible regions of solutions", providing a starting point for deriving new algorithms. Before diving into the new measure, it is essential to have a deeper understanding of what a measure means in this context and how this statistical concept relates to our optimization problem.

### 2.5.1 Distribution over states

In statistical physics, the statistical ensemble is an abstraction representing the totality of states a system might occupy, along with a probability distribution over these configurations $\sigma$ (or $\mathbf{w}$ in the case of the Binary Perceptron). From this foundational framework, one can derive all the thermodynamic properties of the system.
Specifically, when the system is in thermal equilibrium (i.e., at a fixed temperature), the configurations form what is known as the canonical ensemble, and their probability distribution follows the **Boltzmann distribution**:

$$P(\sigma;\beta) = \frac{1}{Z(\beta)} e^{-\beta E(\sigma)} \tag{6}$$

Here, $E(\sigma)$ represents the energy of configuration $\sigma$, $\beta$ is the inverse temperature, and $Z(\beta)$ is a normalization constant known as the partition function.
This framework is not limited to physical systems, it has been successfully applied to a wide range of complex systems, provided a **suitable energy function** $E(\sigma)$ is defined. In these contexts, the statistical physics approach has yielded valuable insights into optimization problems, where the energy plays the role of a cost function to be minimized, and the configurations $\sigma$ represent the variables of the problem (e.g., weights).

This probability distribution over configurations is useful because **naturally prioritizes** configurations of lower energy (cost), with a concentration over those, parametrized by $\beta$. At high temperatures (small $\beta$), the exponential term $e^{-\beta E(\sigma)}$ is relatively flat, meaning that configurations with higher energy still have a significant probability. This corresponds to exploring a broader range of the solution space. As the temperature decreases ($\beta$ increases), the distribution becomes sharper, concentrating more probability on lower-energy configurations. Conversely, in the limit $\beta \to +\infty$ (approaching zero

temperature), the term $e^{-\beta E(\sigma)}$ becomes very large for configurations with the absolute minimum energy and infinitesimally small for all others. Consequently, the probability distribution becomes concentrated at the optimal configurations (global minima). In this limit, all optimal configurations are considered equally likely, and all suboptimal configurations have a probability approaching zero. Therefore, this probability distribution provides a direct link between the statistical properties of a system and the search for optimal solutions. It allows us to analyze the landscape of an optimization problem by studying the behavior of the system at different "temperatures", and it guides the system towards **desired low-cost states**. As we will see in Section 3.1, this perspective forms the theoretical background for an effective physics-inspired heuristic method for tackling discrete optimization problems.

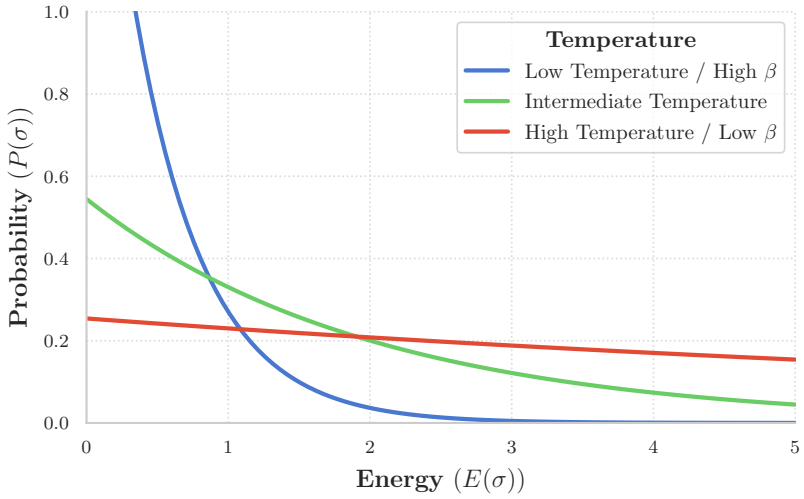



Figure 2: The Boltzmann distribution at varying temperatures. At low temperatures (high $\beta$), probability is highly concentrated on low-energy configurations. As temperature increases (low $\beta$), the distribution flattens.

### 2.5.2 Novel approach

In Baldassi et al. [15], a new method for analyzing the Binary Perceptron problem is introduced, based on a **new probability distribution** (measure) over configurations. This approach shifts the focus from simply minimizing the energy (or cost) to also incorporate the search of dense solution regions. By introducing this new distribution, the researchers were able to effectively discard the rare and isolated minima and **unveil regions of solutions** which are dense and easily accessible [27]. The new probability

distribution described in the paper is:

$$P(\sigma; \beta, y, \gamma) = Z(\beta, y, \gamma)^{-1} e^{y\Phi(\sigma, \beta, \gamma)}, \tag{7}$$

where $y$ serves a role similar to an inverse temperature ($\beta$ in Equation 6). $\Phi(\sigma, \beta, \gamma)$ is a crucial new quantity that, instead of representing the free energy of the system, indicates the "local free entropy." It is defined by:

$$\Phi(\sigma, \beta, \gamma) = \log \sum_{\{\sigma'\}} e^{-\beta E(\sigma') - \gamma d(\sigma, \sigma')}, \tag{8}$$

where $d(\cdot, \cdot)$ is any monotonically increasing function of the distance between configurations. Whereas $E(\sigma)$ in Equation 6 represented just the cost of a configuration $\sigma$, here the local entropy describes how the configuration is **placed geometrically** with respect to other configurations, explicitly prioritizing low-energy outcomes.

Specifically, $\Phi(\sigma, \beta, \gamma)$ —and consequently the probability of configuration $\sigma$— is going to be high if $\sigma$ is near (with respect to distance $d(\cdot, \cdot)$) to dense clusters of low-cost configurations, while it will decrease as $\sigma$ moves further from those regions. When $\beta \to \infty$, the sum in the Equation 8 will contain only $\sigma'$ that are global minima configurations, weighted by their distance from the configuration $\sigma$. This means that if we have a large $y$ (giving high importance to the local entropy), we will have a distribution for our configurations that peaks at positions near **high-density minima**, whose stress on distance is tuned by the parameter $\gamma$: the higher the value, the more the probability would depend on cost density of a narrower region around $\sigma$.

Like the Equation 6 described very well the cost minimization of an optimization problem, the measure in Equation 7 **shifts the focus** on dense regions of minima by incorporating geometric characteristics in the distribution of the configurations of the system. This change in perspective closes the gap between theoretical intractability and the presence of efficient algorithms, whose behavior is now captured well by the new measure. As noted in [15], these algorithms "invariably find solutions belonging to high-density regions when these regions exist, and fail otherwise".

However, computing the entropy $\Phi$ for a system is a challenging task due to the significant computational burden that brings in its definition. Some methods have successfully approximated this entropy with heuristic variations of Belief Propagation [3], but these

approaches can often be challenging to implement and lack total control. Then in [15] a new **approximation of the entropy** is defined by a system of $y+1$ interacting replicas where one of them act as a reference. Assuming $y$ to be a non-negative integer, after few mathematical steps (details in Appendix A), we can rewrite the partition function from Equation 7 as:

$$Z(\beta, y, \gamma) = \sum_{\{\sigma^\star\}} e^{\Psi(\sigma^\star;\beta,\gamma)} = \sum_{\{\sigma^\star\}} \sum_{\{\sigma^a\}} e^{-\beta \sum_{a=1}^{y} E(\sigma^a) - \gamma \sum_{a=1}^{y} d(\sigma^\star,\sigma^a)}, \tag{9}$$

where $\sigma^\star$ is the reference replica and the other $y$ are identical. Studying the equilibrium statistics of this replicated system and tracing out the replicas is equivalent to studying the original large deviations model [3].

Since the new measure naturally prioritizes dense solution regions, whose presence was verified by other studies [27], we want to design an algorithm that directly benefits from this insight. If we assume that we have an iterative algorithm that already minimizes the energy effectively, we can think of it heuristically as sampling from Boltzmann distribution at decreasing temperatures: as the algorithm runs, the configurations' probability concentrates towards the minimum because the algorithm is actively minimizing. The replicas described in Equation 9 follow a distribution that concentrates at low energies with high density through an interaction with a reference configuration. If we create $y$ copies of the same algorithm, add the interaction to their iterative move correctly, and run them simultaneously; we would create a heuristic effect of simulating the system in Equation 9 at increasing $\beta$.

This provides a simple and general framework for designing algorithms to explore dense, accessible regions of the energy landscape. This approach involves **replicating** the original model, introducing an **interaction** term with a reference configuration, and then applying the algorithm to the resulting extended system. In the original work, the measure is simplified by tracing out the reference replica, which is mathematically possible due to the structure of the distribution. While tracing out the reference replica can be convenient, in this article, the reference will be explicitly computed and integrated into the system. In the following chapters, this approach will be applied to two simple heuristic algorithms to solve the Binary Perceptron model, providing numerical evidence of the efficiency gained from replicas' interaction.

# 3 Replicated Simulated Annealing

Chapter 2 established the complexity of finding optimal solutions in discrete optimization problems like the Binary Perceptron, highlighting how the solution landscape is often dominated by numerous local minima that trap standard search algorithms. To overcome these challenges, a novel approach was introduced that focuses on accessing dense regions of solutions by modifying the probability distribution over configurations. **Practically**, the method consists of running identical algorithms simultaneously (replicas) and while each of them optimize the cost function, they **interact** with each other through a reference configuration. Building upon the theoretical foundation, Chapters 3 and 4 will introduce practical algorithmic frameworks that leverage this method, using two different algorithms as replicas. For each chapter it is first explained the vanilla algorithm implementation and then the replicated. Numerical simulations will then be conducted for both solutions and compared against their respective non-interacting version, to show an improved efficiency.

## 3.1 Simulated Annealing

Simulated annealing is a heuristic algorithm to solve combinatorial optimization problems inspired by the physical annealing of metals. The algorithm, designed in 1983 [28], emulates the physical process of a metal cooling that culminates in a frozen and efficient atomic structure. In our case there are no atoms to form our micro states, but weights configurations that –evolving in a physics informed manner– will gradually reach their own efficient and frozen state: the solution to the binary perceptron.

### 3.1.1 Physical annealing

To understand deeply the analogy, we first need to quickly dive into how physical annealing works at atomic level. At high temperature, the atoms in a metal have enough thermal energy to break energy barriers, which enables them to explore different states configurations. This exploration involves the system dynamically sampling configurations according to the Boltzmann distribution at each temperature level. This distribution, is again the Boltzmann distribution

$$P_T(\sigma) = \frac{1}{Z_T} e^{-\frac{1}{Tk_B}E(\sigma)} \tag{10}$$

where $Z_T$ is the normalizing factor, $k_B$ is the Boltzmann constant, $T$ is the temperature and $E(\sigma)$ is the energy of the atomic configuration. As the metal cools down, the temperature $T$ decreases, and the distribution **concentrates** around points of minimal energy, until it ultimately reaches a state where the atoms do not have enough energy to explore and the distribution degenerates to few low-energy configurations. We can now see more clearly the analogy with an optimization problem, where the atomic configurations are the variables in our problem (weights for the case of Binary perceptron), the energy is the objective function to be minimized (the cost function) while the temperature a constant to simulate.

### 3.1.2 Simulating the process

Inspired by the success of physical annealing in optimizing for energy, a group of physicists proposed **simulating** this process computationally to tackle hard optimization problems. However, simulating such a distribution for each temperature level is a non trivial problem. Calculating the partition function $Z_T$ means computing an extremely large sum over all the configurations (that rise combinatorial with the problem size). Here is where the **Metropolis-Hastings** algorithm comes in help, making approximate sampling easy with a simple propose-accept rule (more details in Appendix B). Thanks to this Markov Chain Monte Carlo method, we are able to sample approximately from the Boltzmann distribution, at decreasing temperature levels. At each temperature level, some moves are proposed randomly and they are accepted with a probability defined by the Metropolis-Hastings algorithm. This probability is such that for a reasonable amount of proposals, the accepted configurations will be distributed according to the Boltzmann distribution. This will leave enough space for the algorithm to explore and gradually narrowing to low cost solutions.

A natural question that may arise is: why not simply decrease the temperature drastically, obtaining in this way the desired distribution concentrated around the minima? This could be an efficient solution if the configurations landscape is easy enough (perfect atoms configuration do not require a lot of exploration), otherwise it is very easy to get trapped in local minima solution. In the physical setting, if there is a sudden temperature drop, the atoms will not have neither the time nor the energy to explore much different configurations, being in this way limited to small neighborhoods and favoring situations

of suboptimal minima (given the greedy and sudden search for a minimal energy configuration). Equivalently, our simulated algorithm should have the opportunity to **explore** its configuration space even taking moves that are not optimal in the short term, but that can reveal hidden global minima in the long run. The best possible scenario is an infinitely **slow cooling**, that gives the most amount of time to explore the space and gradually falling in the optimal configuration, but unfortunately this is not feasible.

## 3.2 Standard Implementation

To apply the standard implementation of Simulated Annealing to our Binary Perceptron we should define some critical variables for our problem: the cost function and the movement dynamics. While the first is needed to clearly compute the acceptance probability of the move proposal according to the Metropolis Hastings algorithm rule, the latter structurally defines the exploration dynamics of the configuration space. Both elements will be defined in the easiest forms, for brevity and clarity. The cost function $c(\mathbf{w})$ for a given vector of weights $\mathbf{w}$ will be the number of patterns that are misclassified,

$$c(\mathbf{w}) = \sum_{j=1}^{k} \Theta(-y^j \hat{y}^j) = \sum_{j=1}^{k} \Theta(-y^j H(\mathbf{x}_j^T \mathbf{w} + b)) \tag{11}$$

where $\Theta(x)$ is the same as defined in Section 2.3. Meanwhile, the move allowed at each iteration is a switch of a random weight $w_i$ from $+1$ to $-1$ or vice versa.
Given these two elements, the algorithm is straightforward to implement: we define a variable $\beta$ (representing the inverse of the temperature with the Boltzmann constant) that increases gradually during the process (cooling), and at each value of its ascent we simulate the Boltzmann distribution over problem configurations using the Metropolis-Hasting algorithm. In this way weights will explore different configurations, gradually **converging** into one. The full algorithm implementation can be seen in the Algorithm 1.

In the implementation written for this article, a notable solution is the calculation of the $\Delta c$, which is made more efficient avoiding computing the proposed cost from scratch every MCMC step. Details about the derivation of the delta cost can be found in Appendix C.1.

**Algorithm 1** Simulated Annealing

**Require:** Problem instance $\mathcal{P}$, initial inverse temperature $\beta_0$, final inverse temperature $\beta_N$, number of annealing steps $N$, number of MCMC steps $M$

Initialize configuration $w$ for $\mathcal{P}$
Store $w_{\text{best}} \leftarrow w$, $c_{\text{best}} \leftarrow \infty$
Define inverse temperature schedule: $\beta_1, \beta_2, \ldots, \beta_{N+1}$ with $\beta_{N+1} = \infty$
**for** each $\beta$ in schedule **do**
    **for** $t = 1$ to $M$ **do**
        Propose move $a$ from current configuration
        Compute cost change $\Delta c = c(w') - c(w)$ ▷ $w' = w$ after applying $a$
        **With** probability $p = \min(1, e^{-\beta \Delta c})$:
            Accept move: $w \leftarrow w'$
        **if** $c < c_{\text{best}}$ **then**
            Update best: $w_{\text{best}} \leftarrow w$, $c_{\text{best}} \leftarrow c$
            **if** $c_{best} = 0$ **then**
                **return** $w_{\text{best}}, c_{\text{best}}$
            **end if**
        **end if**
    **end for**
**end for**
**return** $w_{\text{best}}, c_{\text{best}}$

## 3.3 Hyperparameters

In this section it will be explored the impact of hyperparameters on the annealing procedure, and briefly give indications on how to adjust them to reach an optimal cost descent. Since the algorithm is a **heuristic**, there are no precise guidelines on how to define parameters, hence a trial and error approach is often used. However, we can slightly reduce the time cost of this activity if we adjust our parameters exploration based on metrics collected during the runs such as best cost and acceptance frequency. The best cost ($c_{best}$ in Algorithm 1) is the current best cost found by the algorithm at each iteration while the acceptance frequency is the ratio between the accepted moves and the total proposals (MCMC steps $M$ in Algorithm 1) in each annealing iteration.

Optimally, we would set the parameter $\beta_0$ such that the initial acceptance frequency is between 30-80% and then **decrease gradually** over the course of the algorithm together with the cost (*blue* in Figure 3). If the acceptance frequency is **too high** from the beginning (*red* in Figure 3), the algorithm tend to prioritize exploration of the landscape without effectively searching for minimal configurations. The lack of the optimizing behavior can be confirmed by looking at the slow descent of the best cost during the run. Otherwise, if the the acceptance ratio is **too low** (*green* in Figure 3), the algorithm could

leave no space for suboptimal search by only accepting short term best moves. If the problem is complex enough, this would likely lead to a local minima. Again, the best cost will reflect this behavior by showing a rapid decrease followed by a plateau representing the local minima.

Before doing any test we should input a considerable number of MCMC steps ($\geq 100$) to make sure the sampling produces relevant approximations. If the correct betas schedule fails to provide the desired results, annealing steps can be increased up to a feasible running time of the algorithm, to improve the performance by slowing the cooling procedure. In our specific implementation applied to the Binary Perceptron algorithm, a good set of parameters are: $\beta_0 = 0.1, \beta_1 = 5, annealing\_steps = 1000, mcmc\_steps = 200$.

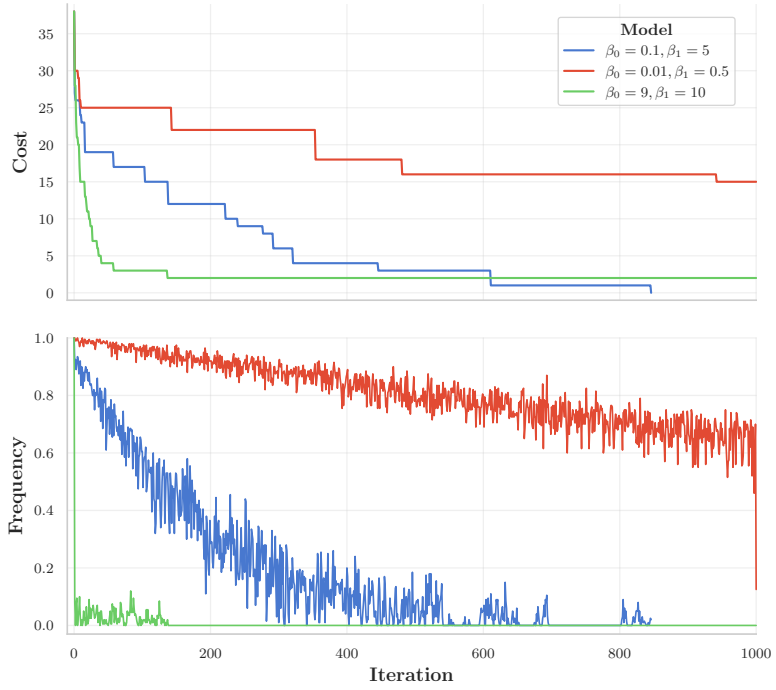



Figure 3: Comparative plot between different choices of betas. All three runs have exact same problem initialization and same parameters exept for betas. The parameters used are: size $N = 250$, $\alpha = 0.3$, $annealing\_steps = 1000$, $mcmc\_steps = 200$. (*Top*) Best cost recorded until each iteration. (*Bottom*) Acceptance frequency of the MCMC at each iteration. In red the behaviors of betas set too high, in green of betas set too low and in blue the desired behavior of correct betas.

## 3.4 Replicated case

The description of the problem using the new large deviation statistics defined in Section 2.5, provides a framework where $y$ replicas interact with a dynamic reference configuration.

We can define the probability distribution of each replica to be

$$p(W^a) \propto exp\left(-\beta E(W^a) + \gamma d(W, \overline{W})\right), \tag{12}$$

where the reference $\overline{W}_j = \frac{1}{y}\sum_{a=1}^{y} w_j^a$ for all $j \in \{1, ..., n\}$ and for the distance $d(W, W') = \frac{1}{2}\sum_{i=1}^{N}(W_i - W_i')^2$. If we compute the reference at each timestamp, the replicas will be independent and their joint probability given a fixed reference $\overline{W}$ is going to be

$$\begin{aligned} p(W^1, \ldots, W^y \mid \overline{W}) &\propto \prod_{a=1}^{y} \exp\left(-\beta E(W^a) + \gamma d(W, \overline{W})\right) \\ &= \exp\left(-\beta \sum_{a=1}^{y} E(W^a) + \gamma \sum_{a=1}^{y} d(W^a, \overline{W})\right) \end{aligned} \tag{13}$$

where the entire exponent is referred to as the Hamiltonian ($H$) of the system, as it encapsulates all energies and interactions. This new distribution is exactly the one referred in the large deviation statistics introduced in 2.5.

To modify our Simulated Annealing accordingly, we can define the move proposal $(i, a)$ in the system as the switch in the random weight $i$ for the random replica $a$ to its opposite, and then change the acceptance probability according to the new distribution 13 in order to simulate the new system correctly. The new **acceptance probability** for the proposed move $(i, a)$ is, using the Metropolis-Hastings algorithm rule, the exponential of the variation in the Hamiltonian $H$ when the system transitions with move $(i, a)$. Since the contributions of other replicas cancel out because they remain unchanged with the move, just the delta cost and delta distance regarding replica $\alpha$ will enter in the calculation. Hence,

$$\Delta H_{w_i^a \to (w_i^a)'} = -\beta \Delta c(W^a)_{w_i^a \to (w_i^a)'} + \gamma \Delta d(W^a, \overline{W})_{w_i^a \to (w_i^a)'} \tag{14}$$

After this quick realization, we can compute this quantity efficiently using the delta cost from the previous algorithm plus an additional delta distance whose derivation it is shown in Appendix C.2.

An additional feature that can be introduced is the gradual increase in the parameter $\gamma$ throughout the process. Since the exact value for $\gamma$ is peculiar to each problem specification, this "scoping" procedure (as it is called in [3]) enables the algorithm to explore different values of the parameter while gradually restricting the search radius of the repli-

cas.

The modified algorithm then looks as follows:

**Algorithm 2** Replicated Simulated Annealing

**Require:** Problem instance $\mathcal{R}$, initial inverse temperature $\beta_0$, final inverse temperature $\beta_N$, initial $gamma_0$, final $gamma_1$, number of annealing steps $N$, number of MCMC steps $M$

Initialize replicas configurations $[w^1, \ldots, w^y]$ for $\mathcal{P}$
Store $w_{\text{best}} \leftarrow w^1$, $c_{\text{best}} \leftarrow \infty$
Define inverse temperature schedule: $\beta_1, \beta_2, \ldots, \beta_{N+1}$ with $\beta_{N+1} = \infty$
Define inverse temperature schedule: $\gamma_1, \gamma_2, \ldots, \gamma_{N+1}$ with $\gamma_{N+1} = \infty$
**for** each $\beta, \gamma$ in schedule **do**
    **for** $t = 1$ to $M$ **do**
        Propose move $i$ for replica $a$ from current configurations
        Compute energy change $\Delta H_{w_i^a \to (w_i^a)'}$
        **With** probability $p = \min(1, e^{\Delta H_{w_i^a \to (w_i^a)'}})$:
            Accept move: $w_i^a \leftarrow (w_i^a)'$
        **if** $c^a < c_{\text{best}}$ **then**
            Update best: $w_{\text{best}} \leftarrow w^a$, $c_{\text{best}} \leftarrow c^a$
            **if** $c_{best} = 0$ **then**
                **return** $w_{\text{best}}, c_{\text{best}}$
            **end if**
        **end if**
    **end for**
**end for**
**return** $w_{\text{best}}, c_{\text{best}}$

The schedule of the $\gamma$ parameter is defined based on the input interval $[\gamma_0, \gamma_1]$ and follows the annealing update timing. The ending criteria is the solution of the problem from any replica in the system. While all the parameters follow guidelines explained in 3.3, the parameter $\gamma$ has a lower interpretability hence a trial-and-error approach is often used. Even if we can't precisely predict the validity of the parameters prior to running the algorithm, like it happens often in computer science, we can understand the algorithm behavior at the **extremes** of $\gamma$. When gamma goes to infinity the replicas are constrained to make the exact same move, while if it's zero then the replica becomes independent and becomes equivalent to running the algorithm $y$ times. The parameter $\gamma$ should then mediate between these two conditions.

## 3.5 Comparison

Numerical experiments have been conducted to evaluate the interacting approach at increasing sizes of the problem, where size it is considered the length of the weight vector $\mathbf{w}$. To correctly isolate the effect of the interaction in Algorithm 2 from the mere replication, the algorithm is compared with the same replicated instance but with a null interacting factor throughout the whole run ($\gamma_0 = 0, \gamma_1 = 0$). In this way two replicated systems are compared, where the presence of an interaction with a reference replica is the **only difference**. For each problem size, 10 samples were taken. A sample is considered based off the **same** binary perceptron problem initialized for both algorithms, with input patterns $K = \alpha N$ and $\alpha = 0.3$. Then the algorithms were run with parameters specified in Figure 4, and the number of iterations to solution is collected for both. As per iterations, it is considered the number of **annealing steps** to solution, since both algorithms are defined with same annealing schedule and both of them reach a solution before the end of annealing steps.

From Figure 4 it is shown how the interacting version **confidently outperforms** the standard instance in every size explored in the experiments. The figure clearly display the ability of the algorithm to efficiently find a solution in diverse initializations of the problem. This difference is indeed explained by its tailored designing to the solution space structure, where accessible dense region of solution are directly targeted. Notable mention is a detailed comparison by [29], that also shows the improvement of interaction in both synthetic and real datasets.

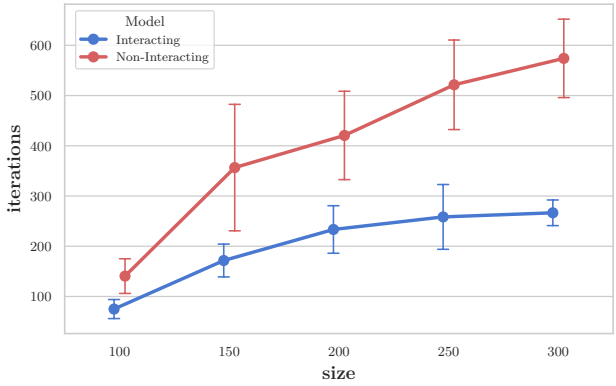



Figure 4: Comparison between interacting and non-interacting Simulated Annealing with $y = 3$ replicas. Each marker is the mean of 10 samples, while the brackets represent standard deviations. The following common parameters are considered: $\alpha = 0.3$, $annealing\_steps = 1000$, $mcmc\_steps = 200$, $\beta_0 = 0.1$, $\beta_1 = 5$. For the interacting version: $\gamma_0 = 0.6$, $\gamma_1 = 1.5$.

# 4 Replicated Gradient Descent

## 4.1 Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) is an algorithm introduced in 1951 by [30] to solve continuous optimization problems by a **stochastic variation** of the standard gradient descent procedure. While in the standard case the gradient of the loss function is taken with respect to the full training sample, with SGD the gradient is computed only with respect to a randomized subset of the input data (mini-batch). This simple modification produces extraordinary results, optimizing continuous functions with a fraction of the time. This improved efficiency made it suitable for large architectures such as deep learning applications [31]. With the SGD the descent to the minimal cost is noisier but quicker, reaching in expectations comparable or better minimization results of its standard form. The **entropic component** brought by the stochasticity of the gradient helps the algorithm escape situations of local minima and leave space for some exploration. Similarly to Simulated Annealing, the procedure of the SGD proceeds in striking a balance between exploration and exploitation, strictly focusing on the minimization task. An interesting method is described in [27] to apply the algorithm to a discrete problem, and derive a replicated version that brings computational benefits to the minimization problem of the Binary Perceptron.

As the description above can hint, the SGD is only feasible on a cost function continuous in its variables, which is contradicted by the **discrete nature** of the weights of our problem. A heuristic that circumvents the issue is defining a **two-layer weights** system connected through a discretization function: the continuous weights $\mathcal{W}$ and the discrete weights $W$ related by $W = sign(\mathcal{W})$. The procedure is simple, at each step $t$: compute the "gradient" with respect to $W$ for a given randomized batch $b(t)$, update $\mathcal{W}$ by gradient step parametrized by learning rate $\eta$, and finally update the discrete weights by the discretization procedure $W = sign(\mathcal{W})$.
The updates are then defined mathematically as

$$(\mathcal{W}_i)^{t+1} = (\mathcal{W}_i)^t - \eta \frac{1}{|b(t)|} \sum_{j \in b(t)} \frac{\partial E^j}{\partial W_i^a} (\mathcal{W})^t \tag{15}$$

$$(W_i)^{t+1} = sign\,(\mathcal{W}_i)^{t+1} \tag{16}$$

However, the gradient of a function with respect to discrete inputs, by definition, does not exist in the conventional sense. Therefore in our implementation, we adopt a heuristic definition of the gradient. We utilize the update direction from the classic **Perceptron algorithm** [13] as a proxy for this "gradient". With a slight abuse of notation, we can then define:

$$\frac{\partial E^j}{\partial W_i^a} \approx \begin{cases} -y^j\, x_i^j & \text{if } H\left(\langle \mathbf{x}^j, W^a \rangle\right) \neq y^j \\ 0 & \text{otherwise} \end{cases} \tag{17}$$

In words, if the perceptron misclassifies an input data point, the gradient points in the direction that moves the weight vector towards the correctly labeled side of the decision boundary. Specifically, if the component $w_i$ contributes to the incorrect prediction of sample $x^j$ through the term $w_i \cdot x_i^j$, then the correct update for $w_i$ depends on both the sign of $x_i^j$ and the true label $y^j$. The update should increase $w_i$ if $x_i^j$ is positively aligned with the true class (i.e., $y^j x_i^j > 0$), and decrease $w_i$ if it's negatively aligned (i.e., $y^j x_i^j < 0$). In essence, the perceptron update rule adjusts the weights in a way that reduces the misclassification error for the current data point, comparably to the effect of a gradient with respect to a continuous cost function.

The algorithm proceeds by applying the updates 15 and 16 with gradients from randomized batches $b(t)$ of prespecified size $|b(t)|$. The algorithm then stops either when zero cost is reached or the maximum amount of epochs $e_{max}$ has been reached. An epoch ends when all the samples from the problem specification are used for a batch update. The learning rate $\eta$ has been considered fixed throughout all the process. The pseudocode for the algorithm implemented is described in Algorithm 3.

**Algorithm 3** Batch Gradient Descent

**Require:** Problem instance $\mathcal{P}$, learning rate $\eta$, number of epochs $e_{max}$, batch size $|b(t)|$

Initialize configuration $W$ for $\mathcal{P}$
Set $epoch \leftarrow 0$, `stop` $\leftarrow$ `False`
$c_{best} \leftarrow +\infty$, $W_{best} \leftarrow W$
**while** `not stop` **do**
    Select random batch $m(t)$ from points not seen in epoch
    Compute batch gradient w.r.t. $m(t)$
    Update weights $\mathcal{W}$
    Discretize weights $W = sign(\mathcal{W})$
    **if** cost$(W) = 0$ **then**
        `stop` $\leftarrow$ `True`
    **else if** $t \geq e_{max}$ **then**
        `stop` $\leftarrow$ `True`
    **else**
        $epoch \leftarrow epoch + 1$
        **if** cost$(W) \leq c_{best}$ **then**
            $c_{best} \leftarrow$ cost$(W)$
            $W_{best} \leftarrow W$
        **end if**
    **end if**
**end while**
**return** $W_{best}$, $c_{best}$

Learning rate $\eta$ is the crucial parameter that set the step size of the updates, hence a value which is too big could cause infinite bounces around the landscape while a value that is too low could take too much time to reach the solution. Therefore an extensive tuning of $\eta$ is needed to optimally run the algorithm.

## 4.2 Replicated

To confirm if the benefits observed with the technique applied to Simulated Annealing also extend to Gradient Descent, we can adapt the approach described in Section 3.4. Specifically, we replicate the system and add a bias towards the center of mass for each gradient update of the weights, parametralized by $\gamma$ again. This new system will stay close to the intuition of the reference paper, but with the explicit calculation of the reference configuration. The reference replica formula will be kept the same as the one described in Section 3.4. The modified update is:

$$(\mathcal{W}_i^a)^{t+1} = (\mathcal{W}_i^a)^t - \eta \frac{1}{|b(t)|} \sum_{j \in b(t)} \frac{\partial E^j}{\partial W_i^a} (\mathcal{W}^a)^t + \eta\gamma(W_i^a - \overline{W_i}) \tag{18}$$

Hence at each iteration the weights' update is limited by the distance with the reference. This mechanism ensures that while the replicas can independently explore the solution landscape, they don't go too far from the central reference.

The implementation follows the same dynamics of the standard implementation but with few modifications. In this version in each iteration the new gradient step is applied to a random replica, and the end of an epoch is the representation of all training samples to all replicas. Additionally, the $\gamma$ parameter follows a "scoping" schedule similar to the Replicated Simulated Annealing, to slowly concentrate replicas over time and explore different values for $\gamma$ during the run.
The pseudocode of the new algorithm is equivalent to Algorithm 3 with modified update at line 7.

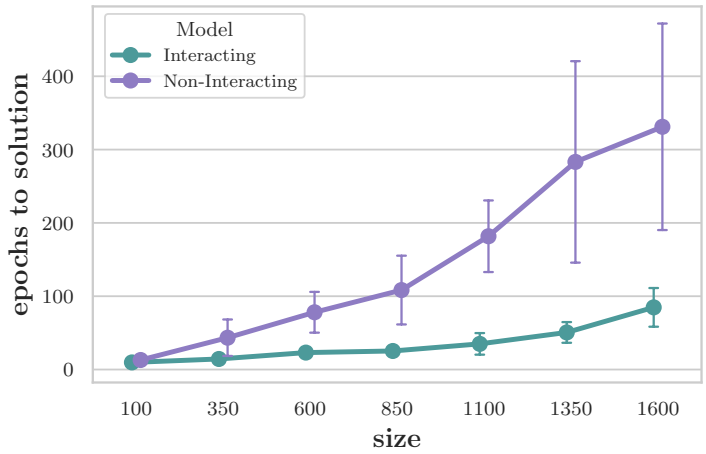
## 4.3 Comparison

To correctly evaluate the new replicated version, we compare the algorithm that incorporates the interaction term ($\gamma > 0$) against an identical version where this interacting component is removed ($\gamma = 0$). Equivalently to the experiments in Section 3.5, a sample is defined as the successful run over a single Binary Perceptron instance, with all parameters remaining identical for both algorithms. In addition to the iteration performances at increasing sizes, we considered also the case of **increasing pattern density** $\alpha$. For the first case, the epochs to solution were recorded, while the latter compares the algorithms by the **error rate** (number of misclassifications over the total patterns) of the optimal weight configuration reached after 300 epochs on a fixed size of $N = 500$.

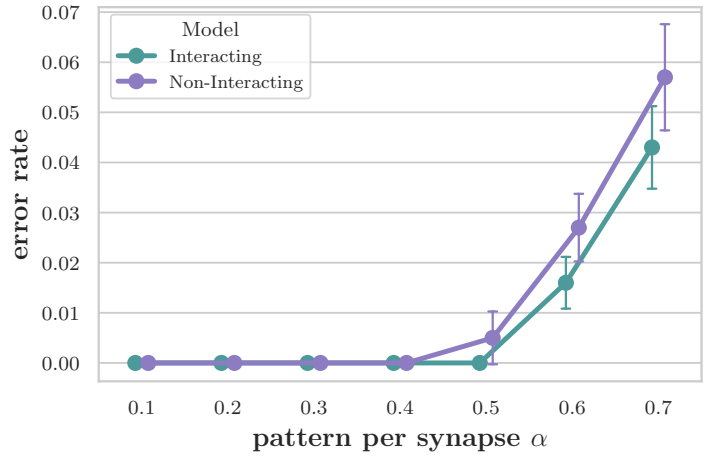
In Figure 5 we can observe the results of both experiments. The interacting version consistently demonstrates faster performance and a lower error rate across all simulated conditions. In Figure 5b it can be seen how the interacting version is able to tackle an increasing complexity of the problem with lower error rate. At $\alpha = 0.5$, where the solution landscape becomes significantly more intricate, the interacting version still consistently reaches an optimal solution. This highlights its clear superiority over the non-interacting counterpart, which becomes vulnerable to suboptimal minima under these more complex conditions.

These results verify the strength of the approach proposed by [3], even in a simplified version such as the one implemented in this article. The algorithm's targeted search for

dense minima successfully provides robust solutions with **fewer representations** of the training set. The limited experiments also provide good evidence of the applicability of the method to distinct algorithmic solutions, and validity under algorithmic variations.




(a) By size




(b) By $\alpha$

Figure 5: Comparison metrics between Replicated Gradient Descent with and without interaction, with $y = 3$ number of replicas. In both plots the markers represent mean and bars standard deviation. For both cases the parameters were fixed at: $\gamma_0 = 0.01$, $\gamma_1 = 1$, $\eta = 0.01$, $max\_epochs = 1000$, batch size $|b(t)| = 10$. (a): it was used a pattern to synapse ratio of $\alpha = 0.4$. (b): the size of the problem was fixed at $N = 500$

# 5 Conclusion

## 5.1 Additional applications

The change in perspective of the replicated approach uncovered in Baldassi et al. [3] brought new insights into the difficult task of solving a network with binary synaptic weights, and provide numerical evidence of the benefits of shifting to an entropy based exploration of the configuration space. Those results are proven in the original paper to hold for a fully connected committee machine, signaling a possible **extension** of the benefits to more complex architectures, such as continuous neural networks. This could explain also the success of federated learning approaches like Elastic Averaging SGD ([32]).

In addition to the two algorithms seen in this dissertation the researchers design also a modified version of **Belief Propagation** called Focusing BP, that provides a more clear explanation of its effectiveness with respect to other variants such as the Reinforced BP. Its explanation is rooted in the search for dense and robust region of solutions, equivalently to the principles guiding the two heuristics reviewed in this article, Replicated Simulated Annealing and Replicated Gradient Descent.

In practical applications, the algorithms provided by the work of Baldassi et al.[3] can be applied to **other discrete problems** with similar solution guarantees and characteristics of the binary networks, and reach efficient and generalizable solutions. This is the case studied in [33], where the authors provide both numerical and analytical evidence that Replicated Simulated Annealing (RSA) achieves Bayes-optimal performance in hard inference problems, specifically the planted coloring problem on sparse random graphs.

## 5.2 Methodology's legacy

The theoretical and numerical results of the replicated approach from Baldassi et al. [3] inspired new algorithms such as ENTROPY-SGD [34], which extend the idea of favoring wide, dense minima in the energy landscape to modern **deep neural networks**, leveraging Langevin dynamics to efficiently approximate local entropy without the additional computational effort of replicated training.

Another application that extends the entropy-based methodology is the Sharpness-Aware Minimization process, laid out by Google Research in 2021 [35, 36]. The protocol,

inspired by the works of Baldassi et al. on the **geometry** of networks' solution landscape, intelligently minimizes the loss while prioritizing regions of high density of solutions, reaching state of the art performances on a variety of benchmarks.

Additional progress was also inspired on the specifics of solution landscape of the binary perceptron problem, in [37] it is shown that when the perceptron is over-parameterized (low pattern to synapse ratio $\alpha$) it exists a subdominant yet connected cluster of solutions with nearly maximal diameter, where "an efficient multiscale majority algorithm is able to find solutions within this cluster with high probability"[37].

## 5.3 Final remarks

The methodology reviewed in this dissertation exemplifies how algorithms can be derived from a **deep understanding** of a problem. In this case, the meticulous study of the solution landscape brought valuable insights into designing simple algorithmic procedures while bridging the gap between the theoretical intractability and practical solutions. The new large deviation statistics defined in Section 2.5 led to findings about the solution landscape that drove the design of new algorithms, while explaining how effective algorithms could find solutions in the harsh conditions of the Binary Perceptron landscape. Moreover, it is an example of how a single fundamental principle can **inspire** new algorithms and variations that effectively improve performances. In Figures 4,5 can be observed how our simple variations of heuristics can have significant impact also at small scale of the problem.

The work then reminds ourselves about the importance of extensive **explorations** of a problem, strongly justifying the works that drives in that direction. While powerful heuristics -mostly not fully understood- advance the performances in a variety of pattern recognition tasks (such as image recognition), a full understanding of those complex mechanisms still lacks. The paper from Baldassi et al. is evidence of the impact of understanding and how much value is lost by this lack of comprehension.

The work by Baldassi et al. is part of a research segment that continues to inspire **theoretically grounded** algorithmic approaches, which gradually aspire to bring order to the modern heuristics of deep learning discipline. The reviewed methodology contributes to these aspirations by creating frameworks and fostering connections between disciplines, all aimed at providing practical solutions to real-world problems.

# References


[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2023.

[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015.

[3] C. Baldassi, A. Ingrosso, C. Lucibello, L. Saglietti, and R. Zecchina, "Local entropy as a measure for sampling solutions in constraint satisfaction problems," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2016, p. 023301, Feb. 2016.

[4] L. Tonelli, "Navigating complex energy landscapes: Replicated approaches to the binary perceptron," 2024.

[5] M. Mézard and G. Parisi, "Replicas and optimization," *Journal de Physique Lettres*, vol. 46, no. 17, pp. 771–778, 1985.

[6] M. Mézard, G. Parisi, and M. A. Virasoro, *Spin Glass Theory and Beyond*, vol. 9 of *Lecture Notes in Physics*. World Scientific, 1987.

[7] A. D. Gioacchino, *Euclidean Correlations in Combinatorial Optimization Problems: A Statistical Physics Approach*. Phd thesis, University of Milan, Milan, Italy, 2020. PhD School in Physics, Astrophysics, and Applied Physics, Cycle XXXII. Supervisor: Prof. Sergio Caracciolo.

[8] K. A. Zweig, G. Palla, and T. Vicsek, "What makes a phase transition? analysis of the random satisfiability problem," *Physica A: Statistical Mechanics and its Applications*, vol. 389, no. 8, pp. 1501–1511, 2010.

[9] M. Vujošević-Janičić, J. Tomašević, and P. Janičić, "Random k-gd-sat model and phase transition," *Journal of Universal Computer Science*, vol. 13, no. 4, pp. 572–591, 2007. Submitted: 16/2/07, accepted: 25/4/07, appeared: 28/4/07.

[10] R. Monasson and R. Zecchina, "Statistical mechanics of the random¡mml:math xmlns:mml="http://www.w3.org/1998/math/mathml" display="inline"¿¡mml:mi¿k¡/mml:mi¿¡/mml:math¿-satisfiability model," *Physical Review E*, vol. 56, p. 1357–1370, Aug. 1997.

[11] I. P. Gent and T. Walsh, "The tsp phase transition," *Artificial Intelligence*, vol. 88, no. 1, pp. 349–358, 1996.

[12] S. Mertens, "Phase transition in the number partitioning problem," *Physical Review Letters*, vol. 81, p. 4281–4284, Nov. 1998.

[13] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain," *Psychological Review*, vol. 65, no. 6, pp. 386–408, 1958.

[14] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The Bulletin of Mathematical Biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[15] C. Baldassi, C. Borgs, J. Chayes, A. Ingrosso, C. Lucibello, L. Saglietti, and R. Zecchina, "Unreasonable effectiveness of learning neural networks: From accessible states and robust ensembles to basic algorithmic schemes," *Proceedings of the National Academy of Sciences*, vol. 113, pp. E7655–E7662, November 2016.

[16] S. Li, T. Schramm, and K. Zhou, "Discrepancy algorithms for the binary perceptron," 2025.

[17] D. L. K. Yamins, H. Hong, C. F. Cadieu, E. A. Solomon, D. Seibert, and J. J. DiCarlo, "Performance-optimized hierarchical models predict neural responses in higher visual cortex," *Proceedings of the National Academy of Sciences*, vol. 111, no. 23, pp. 8619–8624, 2014.

[18] E. Gardner, "Maximum storage capacity in neural networks," *Europhysics Letters*, vol. 4, no. 4, pp. 481–485, 1987. EDINBURGH-87-395.

[19] W. Krauth and M. Mézard, "Storage capacity of memory networks with binary couplings," *Journal de Physique*, vol. 50, no. 20, pp. 3057–3066, 1989.

[20] G. Parisi, "Infinite number of order parameters for spin-glasses," *Phys. Rev. Lett.*, vol. 43, pp. 1754–1756, Dec 1979.

[21] E. Amaldi and V. Kann, "The complexity and approximability of finding maximum feasible subsystems of linear relations," *Theoretical Computer Science*, vol. 147, no. 1, pp. 181–210, 1995.

[22] H. Huang, K. Y. M. Wong, and Y. Kabashima, "Entropy landscape of solutions in the binary perceptron problem," *Journal of Physics A: Mathematical and Theoretical*, vol. 46, p. 375002, Aug. 2013.

[23] H. Huang and Y. Kabashima, "Origin of the computational hardness for learning with binary synapses," *Phys. Rev. E*, vol. 90, p. 052813, Nov 2014.

[24] A. Braunstein and R. Zecchina, "Learning by message passing in networks of discrete synapses," *Phys. Rev. Lett.*, vol. 96, p. 030201, Jan 2006.

[25] C. Baldassi, A. Braunstein, N. Brunel, and R. Zecchina, "Efficient supervised learning in networks with binary synapses," *Proceedings of the National Academy of Sciences*, vol. 104, no. 26, pp. 11079–11084, 2007.

[26] C. Baldassi, "Generalization learning in a perceptron with binary synapses," *Journal of Statistical Physics*, vol. 136, p. 902–916, Sept. 2009.

[27] C. Baldassi, A. Ingrosso, C. Lucibello, L. Saglietti, and R. Zecchina, "Subdominant dense clusters allow for simple learning and high computational performance in neural networks with discrete synapses," *Physical Review Letters*, vol. 115, Sept. 2015.

[28] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

[29] V. Gripon, M. Löwe, and F. Vermet, "Some remarks on replicated simulated annealing," *Journal of Statistical Physics*, vol. 182, Mar. 2021.

[30] H. E. Robbins, "A stochastic approximation method," *Annals of Mathematical Statistics*, vol. 22, pp. 400–407, 1951.

[31] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533–536, 1986.

[32] S. Zhang, A. Choromanska, and Y. LeCun, "Deep learning with elastic averaging sgd," 2015.

[33] M. C. Angelini and F. Ricci-Tersenghi, "Limits and performances of algorithms based on simulated annealing in solving sparse hard inference problems," *Physical Review X*, vol. 13, Apr. 2023.

[34] P. Chaudhari, A. Choromanska, S. Soatto, Y. LeCun, C. Baldassi, C. Borgs, J. Chayes, L. Sagun, and R. Zecchina, "Entropy-sgd: Biasing gradient descent into wide valleys," 2017.

[35] P. Foret, A. Kleiner, H. Mobahi, and B. Neyshabur, "Sharpness-aware minimization for efficiently improving generalization," 2021.

[36] J. Zhuang, B. Gong, L. Yuan, Y. Cui, H. Adam, N. Dvornek, S. Tatikonda, J. Duncan, and T. Liu, "Surrogate gap minimization improves sharpness-aware training," 2022.

[37] E. Abbe, S. Li, and A. Sly, "Binary perceptron: efficient algorithms can find solutions in a rare well-connected cluster," 2021.

# Appendices

## A Replicated approach extra derivations

We start from the unnormalized probability distribution $P(\sigma; \beta, y, \gamma) \propto e^{y\Phi(\sigma,\beta,\gamma)}$ as given by Equation 7 (excluding the normalization constant for now). The associated partition function $Z(\beta, y, \gamma)$ is defined as the sum over all possible configurations $\sigma$:

$$Z(\beta, y, \gamma) = \sum_{\{\sigma\}} e^{y\Phi(\sigma,\beta,\gamma)}$$

Substitute the definition of $\Phi(\sigma, \beta, \gamma)$ from Equation 8:

$$\Phi(\sigma, \beta, \gamma) = \log \sum_{\{\sigma'\}} e^{-\beta E(\sigma') - \gamma d(\sigma,\sigma')}$$

Substituting this into the expression for $Z(\beta, y, \gamma)$:

$$Z(\beta, y, \gamma) = \sum_{\{\sigma\}} e^{y \log\left(\sum_{\{\sigma'\}} e^{-\beta E(\sigma') - \gamma d(\sigma,\sigma')}\right)}$$

Using the property $e^{k \log A} = (e^{\log A})^k = A^k$:

$$= \sum_{\{\sigma\}} \left( \sum_{\{\sigma'\}} e^{-\beta E(\sigma') - \gamma d(\sigma,\sigma')} \right)^y$$

Since $y$ is a positive integer, we can define it to represent the number of identical replicas. Therefore, the term in parentheses, raised to the power $y$, can be expanded as a product of $y$ independent sums. Since the replicas are identical, we can write the variable $\sigma'$ for

each of these $y$ sums as $\sigma^a$ for $a = 1, \ldots, y$:

$$\begin{aligned}
&\left( \sum_{\{\sigma^1\}} e^{-\beta E(\sigma^1) - \gamma d(\sigma, \sigma^1)} \right) \times \cdots \times \left( \sum_{\{\sigma^y\}} e^{-\beta E(\sigma^y) - \gamma d(\sigma, \sigma^y)} \right) \\
&= \sum_{\{\sigma^1, \ldots, \sigma^y\}} \prod_{a=1}^{y} e^{-\beta E(\sigma^a) - \gamma d(\sigma, \sigma^a)} \\
&= \sum_{\{\sigma^1, \ldots, \sigma^y\}} e^{-\beta \sum_{a=1}^{y} E(\sigma^a) - \gamma \sum_{a=1}^{y} d(\sigma, \sigma^a)}
\end{aligned}$$

Now, by relabeling the summation variable $\sigma$ as $\sigma^\star$ (reference), we obtain the full partition function for the replicated system:

$$Z(\beta, y, \gamma) = \sum_{\{\sigma^\star\}} \sum_{\{\sigma^a\}_{a=1}^{y}} e^{-\beta \sum_{a=1}^{y} E(\sigma^a) - \gamma \sum_{a=1}^{y} d(\sigma^\star, \sigma^a)}$$

This expression explicitly describes a system of $y + 1$ interacting replicas: one reference replica $\sigma^\star$ and $y$ identical replicas $\sigma^a$, with interactions given by their energies and distances to the reference.

# B Metropolis-Hastings Acceptance Rule

The Metropolis-Hastings (M-H) algorithm is an MCMC method for sampling from a target probability distribution $P(x)$, especially when direct sampling is difficult. Given a current state $x$ and a proposed state $x'$ (drawn from a proposal distribution $Q(x'|x)$), the acceptance probability $A(x \to x')$ is:

$$A(x \to x') = \min \left( 1, \frac{P(x')Q(x|x')}{P(x)Q(x'|x)} \right)$$

This rule ensures the Markov chain converges to $P(x)$. If accepted, the chain moves to $x'$; otherwise, it remains in $x$. In the context of Simulated Annealing, the M-H rule is adapted for minimizing a cost function $c(x)$. The implied target distribution is proportional to $e^{-c(x)/T}$, where $T$ is the temperature. Assuming a symmetric proposal ($Q(x|x') = Q(x'|x)$), the acceptance probability simplifies to:

$$A(x \to x') = \min \left( 1, e^{-\frac{c(x') - c(x)}{T}} \right)$$

Or, using $\beta = 1/T$ and $\Delta c(x \to x') = c(x') - c(x)$:

$$A(x \to x') = \min\left(1, e^{-\beta \Delta c(x \to x')}\right)$$

This allows for probabilistic acceptance of higher-cost moves, enabling the algorithm to escape local optima.

# C Efficient implementations

## C.1 Efficient delta cost

The predictions in the Binary Perceptron are obtained by $\mathbf{Xw}$. This matrix-vector product can be viewed as a linear combination of the columns of $\mathbf{X}$, where each column $\mathbf{X}_j$ is scaled by its corresponding weight $w_j$:

$$\mathbf{S}_{\text{current}} = \mathbf{Xw} = \sum_{j=1}^{n} w_j \mathbf{X}_j$$

When a single weight, say $w_k$, is flipped to $-w_k$, only its contribution to this linear combination changes. The new prediction vector $\mathbf{S}_{\text{new}}$ becomes:

$$\mathbf{S}_{\text{new}} = \sum_{j=1, j\neq k}^{n} w_j \mathbf{X}_j + (-w_k)\mathbf{X}_k$$

The change in the prediction vector, $\Delta_{\text{pred}} = \mathbf{S}_{\text{new}} - \mathbf{S}_{\text{current}}$, is therefore:

$$\begin{aligned} \Delta_{\text{pred}} &= (-w_k)\mathbf{X}_k - w_k \mathbf{X}_k \\ &= -2w_k \mathbf{X}_k \end{aligned}$$

This simple relation shows that the change in the entire vector of predictions is directly proportional to the flipped weight $w_k$ and its corresponding input column $\mathbf{X}_k$. This allows for an extremely efficient update of predictions after a single weight flip, significantly reducing computational cost compared to recalculating the full matrix-vector product from scratch.

## C.2 Efficient delta distance

In the Replicated Simulated Annealing the total cost function for each replica includes not only its cost (energy) but also a term that penalizes based on the distance from a reference configuration. Then to calculate the acceptance probability of the M-H we need to compute the quantity from 14 at each proposed move. The delta cost is defined as above, while for the delta distance $\Delta d(W^a, \overline{W})_{w_i^a \to (w_i^a)'}$ (or $\Delta_{\text{distance}}$ as we will call it here) can be computed efficiently in the following way. Note that the replica index $a$ is omitted for a matter of clarity. The distance between a replica's weight vector $\mathbf{w}$ and the reference weight vector $\overline{\mathbf{w}}$ is defined as:

$$d(\mathbf{w}, \overline{\mathbf{w}}) = \frac{1}{2} \sum_{i=1}^{n} (w_i - \overline{w}_i)^2$$

The contribution of this distance to the total cost is $\gamma \cdot d(\mathbf{w}, \overline{\mathbf{w}})$, where $\gamma$ is a parameter that controls the strength of this interaction. When a single weight, say $w_k$, in a replica's vector $\mathbf{w}$ is flipped from $w_k$ to $-w_k$ (while all other $w_j$ for $j \neq k$ remain unchanged), only the $k$-th term in the sum for $d(\mathbf{w}, \overline{\mathbf{w}})$ is affected. Let's calculate the change in the $\gamma$-weighted distance, $\Delta_{\text{distance}}$, when $w_k$ changes to $-w_k$. The original $k$-th term in the sum for the distance contribution was $\frac{\gamma}{2}(w_k - \overline{w}_k)^2$. The new $k$-th term, after $w_k$ becomes $-w_k$, will be $\frac{\gamma}{2}((-w_k) - \overline{w}_k)^2 = \frac{\gamma}{2}(w_k + \overline{w}_k)^2$. Therefore, the change in the distance contribution $\Delta_{\text{distance}}$ is:

$$\begin{aligned}
\Delta_{\text{distance}} &= \frac{\gamma}{2}(w_k + \overline{w}_k)^2 - \frac{\gamma}{2}(w_k - \overline{w}_k)^2 \\
&= \frac{\gamma}{2}\left[(w_k^2 + 2w_k\overline{w}_k + \overline{w}_k^2) - (w_k^2 - 2w_k\overline{w}_k + \overline{w}_k^2)\right] \\
&= \frac{\gamma}{2}\left[w_k^2 + 2w_k\overline{w}_k + \overline{w}_k^2 - w_k^2 + 2w_k\overline{w}_k - \overline{w}_k^2\right] \\
&= 2\gamma w_k \overline{w}_k
\end{aligned}$$

This $\Delta_{\text{distance}}$ is the exact change in the distance-related cost component for the proposed move. By calculating only this local change, the algorithm avoids re-summing the entire distance for all $n$ components, which significantly make the process more efficient for the Metropolis-Hastings steps in the replicated system.