Is Evolution an Algorithm? Effects of local entropy in unsupervised learning and protein evolution


Doctoral Dissertation
Doctoral Program in Physics (34.th cycle)

Is Evolution an Algorithm?

Effects of local entropy in unsupervised learning and protein evolution

Matteo Negri

* * * * *

Supervisor
Prof. Riccardo Zecchina,

Doctoral Examination Committee:
Prof. R.Burioni, Referee, Università degli Studi di Parma
Prof. C.Cammatora, Referee, Università di Roma Sapienza

Politecnico di Torino
May 31, 2022

I hereby declare that, the contents and organization of this dissertation constitute my own original work and does not compromise in any way the rights of third parties, including those relating to the security of personal data.

. . . . . .

Matteo Negri
Turin, May 31, 2022

Contents

Summary

In the present thesis we study artificial neural networks with the lens of statistical mechanics, field that has proven very useful in understanding the structure of the solution space of combinatorial optimization problems such as the *perceptron*, the basic unit of any artificial neural network.

The rise of artificial neural networks in the last decade has been staggering and now deep (i.e. many-layered) neural networks applications are ubiquitous in technology and science. Despite this fact, a comprehensive theory that explains this success is still missing: according to the classical statistical learning framework, deep networks should *not* have good generalization properties because they are utilized in the over-parametrized regime (i.e. much more parameters than examples), where statistical learning theory predicts overfitting for any class of models.

An interesting conjecture which has emerged in various contexts argues that the flatness of the minima can lead to good generalization in the over-parametrized regime. For this reason recently a theory has been developed that connects the generalization capabilities of artificial neural networks with the geometrical properties of the error loss functions that is minimized for learning. This theory makes use of the concept of *local entropy*, a function that counts the number of other solutions around a given solution. So far theoretical results on generalization have been limited to the basic *supervised learning* setting of the *teacher-student model*. Other empirical results are available which are not limited to a teacher-student setting, but they are still examples of supervised learning.

Supervised learning means training an artificial neural network to do some classification task. This means finding a rule to fix the parameters given a set of example patterns, in a way that the network assigns the correct label to each example. The adjective supervised refers to the fact that each example pattern must be provided with a label.

The *teacher-student model* is the prototypical classification problem. In this setting the examples are independent and identically distributed randomly-generated patterns and the labels are provided as the output of a second network (called teacher) that is randomly initialized and that has the same architecture of the first network (called student).

The main goal of this thesis is to explore the effect of local-entropy-inspired algorithms in situations that are progressively more different from the teacher-student scenario. First we study a perceptron model on a Gaussian-mixture data distribution, then we switch to an unsupervised setting (i.e. the examples are not labeled) and finally we study a system that is completely unrelated to neural networks where local entropy proves to be useful to understand the evolution of proteins.

Other setting that expand the results of the perceptron teacher-student scenario have already been studied, all of which focus on supervised learning problems. In [1, 2, 3] the authors show that developing local-entropy-based algorithms improves the generalization performance also in deep architectures. Additionally, in [4, 5, 6, 7] the authors study models where an analytical treatment is still feasible, exploring the interplay between activation functions, number of parameters of a network, classification with a margin, the existence of wide flat minima and their effect on generalization.

The thesis is organized into three parts, as it follows.

Part I is made of chapters 1 and 2 and is dedicated to introducing all the concepts necessary for understanding the results in part II.

In chapter 1 we introduce the basic ingredients of neural networks: the perceptron, classification problems, the problem of overfitting and how to build deep networks.

In chapter 2 we introduce local entropy and we review the most recent results related to it. This is useful to understand how the results in this thesis go beyond what is already known in the literature.

Part II is made of chapters 3, 4 and 5 and contains the results on the three models that are studied in this thesis.

In chapter 3 we discuss the perceptron Gaussian mixture problem, a simple model that can show how local entropy is relevant for controlling overfitting even when the training loss is a convex function. Additionally, on this model no classifier can achieve zero test error, in contrast with the classical perceptron setting. We show both analytically and numerically how to systematically improve the generalization performance in this setting by optimizing local entropy.

In chapter 4 we move on to the more complicated task of unsupervised learning of composite data. The choice of composite data allows for an assessment of the quality of the learning that goes beyond the generalization error: if we have access to the generative model of the examples we can check if the extracted features are similar to the true features. A natural candidate model for finding efficient representations are undercomplete, sparse autoencoders. We show that it is indeed possible to find representations of composite data in terms of basic features, but that this process is very sensitive to both overfitting and underfitting. Therefore we present a modified version of the autoencoder, the *replicated autoencoder*, which is designed to find good solutions in cases where overfitting is a danger. We show that the replicated autoencoder finds good representation of composite data both on synthetic data and on real-world biological data, opening up possible applications in biophysics.

In chapter 5 we generalize the algorithmic schemes introduced for sampling high

local entropy configurations in neural networks and apply them to simple models of 3D protein structures. We first consider a lattice model, showing that by sampling high-local-entropy native states we find a decrease in the chain-length separation between contact residues. Also, we show that native states with high local entropy have lower folding times when we use them to generate a Go model. Then we study all-atom representations of proteins. We find that native states of real proteins have higher local entropy than random decoys, suggesting that local entropy could be relevant in modelling protein evolution. We formulate the interpretation that the "flatness" of the energy profile in the native state can extend to the transition state, having consequences both on the thermodynamic stability of the protein, lowering the free energy of the native state, and on the folding rate, lowering the free energy of the transition state.

Part III is made of chapter 6 and is devoted to summing up results, discussing connections with the literature and drawing future developments.

Part I

Preliminaries

Chapter 1

Basics of Artificial Neural Networks

In this chapter we review the basic concepts in the study of Neural Networks. Even the most advanced deep learning applications can be understood by the means of three basic elements that are introduced in this chapter:

- the architecture;
- the loss function;
- the optimization algorithm.

The huge variety of neural networks present in the literature can be described by simply varying these three ingredients.

This introduction has different goals: first, it is necessary for a reader who knows nothing of the subject; second, it could be useful for the reader who is already familiar with the subject but is unaware of the important contributions offered by statistical physics; third, since the study of neural networks is very wide and multidisciplinary, it serves as a place to fix definitions of concepts that may appear in other works with different symbols or names.

The first section is devoted to the perceptron, the basic unit of any neural network. After giving the necessary definitions we report a sketch of the foundational results obtained with statistical physics: the calculation of the capacity of the perceptron, also known as Gardner analysis, on which results in chapter 2 build upon.

The rest of the chapter is devoted to introducing *deep* neural networks and all the other related concepts. In particular, in the last chapter we introduce regularization methods within the Bayesian framework.

1.1 Perceptron

The most basic and common task that we can use a neural network for is *classification*. Let's say that we have a collection of p data examples $\{\xi_i^\mu\}_{\mu=1}^p$ and a collection of p

corresponding labels $\{\sigma^\mu\}_{\mu=1}^p$ that describe which of the classes each example belongs to. For the sake of concreteness, let's consider a binary classification problem $\sigma^\mu = \pm 1$. We will discuss classification with more than two classes in subsection 1.4.2. The task consists in finding a configuration of the model parameters so that the model assigns the correct label to each example.

The *perceptron* is the most basic neural network and the building block of more complicated architectures. It can perform the classification task and the model is simple enough to be studied analytically in the framework of statistical mechanics. There are many other tasks that neural networks can perform, such as generating new examples, correcting errors, extracting features. Most modern technological applications of deep learning, such as speech recognition or computer vision, require huge computational resources and are beyond the scope of this thesis. Here instead we start by studying classification, as it is the most studied and simplest task that allows us to introduce gradually concepts that are of general use.

1.1.1 Model definition

A perceptron is a model that performs a weighted sum of the N input channels x_i and feeds it to a nonlinear function f, called *activation function*. A scheme is shown in figure 1.1. The output of the model y is interpreted as the predicted label:

$$y = f\left(\sum_{i=1}^{N} W_i \xi_i\right) \tag{1.1}$$

The specific choice of the activation function f depends on the situation: when the perceptron is used as a neuron inside bigger architectures (see section 1.3) common choices are the sigmoid function $f(z) = 1/(1 + e^{-z})$ or the so-called rectified linear unit (ReLU) $f(z) = \max(0, z)$. When the perceptron is used as a standalone model, the choice is to use a signum function $f(z) = \mathrm{sgn}(z)$ so that the output is $y = \pm 1$ and matches the range of the labels. In this section we discuss the standalone model with the signum function.

The weights can be either binary or real: the phase diagram of the model changes a lot depending on this choice. In this chapter we will discuss real weights, leaving the binary weights for chapter 2.

The inputs too can be either binary or real. In the following we will consider them binary for simplicity.

1.1.2 Classification and linear separability

The perceptron is capable of correctly classifying only linear-separable dataset, namely those where a hyperplane can separate the examples with $\sigma^\mu = +1$ from the examples with $\sigma^\mu = -1$.

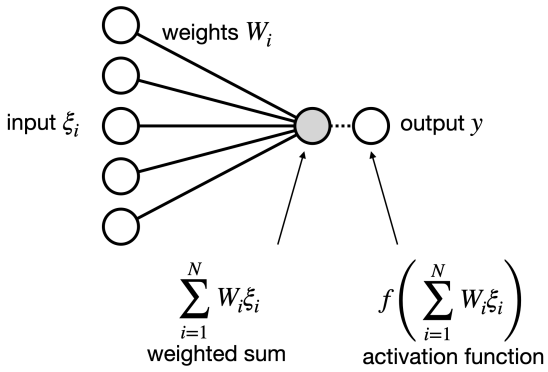


Figure 1.1: Scheme of a perceptron.

To show this property let's interpret the argument of the perceptron as a scalar product between the example and the weights:

$$y^\mu = \text{sgn}(W \cdot \xi^\mu) \tag{1.2}$$

The example ξ^μ is correctly classified if the projection of ξ^μ on the vector W has the correct sign. This means that the vector ξ^μ must lay on the correct side of the hyperplane whose versor is W (see figure 1.2). This is another way of stating the condition

$$y^\mu = \sigma^\mu \quad \forall \mu \in [1, p] \tag{1.3}$$

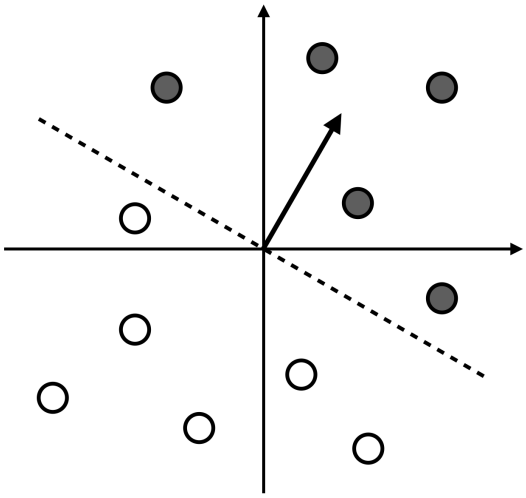


Figure 1.2: Separating hyperplane of a perceptron.

We can exploit a symmetry of the problem and multiply by σ^μ both sides of equation 1.2. We get new labels $y'^\mu := y^\mu \sigma^\mu = 1$ and new examples $x^\mu := \xi^\mu \sigma^\mu$ and the

condition 1.3 becomes

$$\text{sgn}(W \cdot x^\mu) = 1 \quad \forall \mu \in [1, p] \tag{1.4}$$

This is equivalent to saying that all the examples must lay on the same side of the hyperplane defined by W, or that they must have a positive projection on W:

$$W \cdot x^\mu > 0 \quad \forall \mu \in [1, p] \tag{1.5}$$

As for now the hyperplane must cross the origin of the axes. The model can be extended to have the hyperplane cross the axes in general positions by adding a threshold term b, called *bias*, to equation 1.1:

$$y = f\left(\sum_{i=1}^{N} W_i \xi_i - b\right) \tag{1.6}$$

Note that this is equivalent to an additional channel to the input and fixing its value to -1.

With the addition of a bias, the perceptron is capable of correctly classifying set of examples where there exists a generic separating hyperplane. Examples of impossible problems are those where the separating surface is curved and situations where the examples are not in general positions (see figure 1.3).

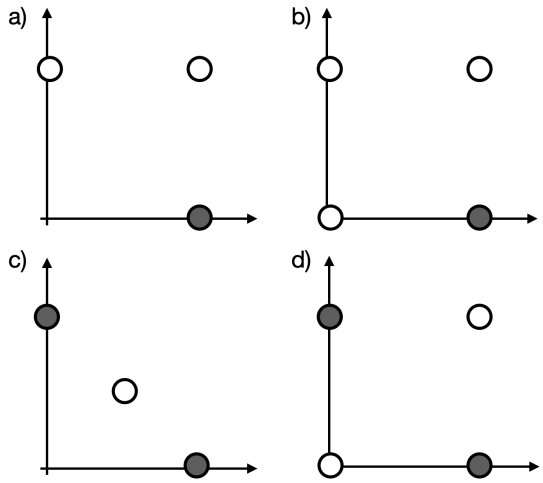


Figure 1.3: 2-dimensional examples of datasets that are or are not linearly separable. Panel a) is linearly separable even without the bias, while b) requires a nonzero bias to be. Panel c) is a case where the points are not in general positions and therefore not linearly separable. Panel d) is the XOR case, which is not linearly separable.

1.1.3 Learning algorithm

Now that we stated the problem, the question is how to find a configuration of weights W that correctly classifies the examples, given that the only information that we have are the examples themselves. This is called a *learning problem*.

One could design any rule that comes to mind and then study how effective it is (for example, ask how many steps are required to find the solution to the problem). Here we describe the most naive rule that we can imagine, just to introduce the prototype of a learning algorithm.

The idea is the following:

1. Initialize W in a random configuration.
2. Consider one example. Is the association with its label correct?
3. If it is correct, leave the weights as they are.
4. If it is not correct, add *something* to the weights
5. Go to step 2. Repeat until every example is classified correctly.

Let's add more detail to this idea. Each cycle of the algorithm consists in an update of the parameters of the form

$$W_i^{\text{new}} = W_i^{\text{old}} + \eta \Delta W_i \tag{1.7}$$

where η is a parameter called *learning rate* that controls the order of magnitude of the updates ΔW_i. We can write the updates of the i-th input channel as something proportional to the direction towards which we expect the correct W to be pointed, which means $\sim \sigma^\mu \xi_i^\mu$. Adding the modality of steps 2-3 we can write

$$\Delta W_i = \begin{cases} 2\sigma^\mu \xi_i^\mu & \text{if } \sigma^\mu \neq y^\mu \\ 0 & \text{otherwise} \end{cases} \tag{1.8}$$

Or, in a more compact form

$$\Delta W_i = (\sigma^\mu - y^\mu)\xi_i^\mu \tag{1.9}$$

An intuitive explanation of this rule is that we are making the weights "forget" the wrong direction $y^\mu \xi_i^\mu$ and "learn" the correct one $\sigma^\mu \xi_i^\mu$, and if the two are the same we are doing nothing.

We will see in section 1.3 how this rule can be included in the larger framework of gradient-based learning, which is the framework that had universal success in training deep networks.

This learning rule is known as the *perceptron rule* and it has been proved that it converges to a solution in a finite number of steps. For a more detailed discussion of this and other learning rules, and for the proof of convergence see [8].

1.2 Spherical perceptron: properties of the solution space

Here we study the two most basic problems of learning with a perceptron with continuous weights.

First, we study learning random binary patterns with random binary labels. This problem is simple enough to showcase a methodology that will be used extensively in chapter 2. It is also a good starting point in showcasing how physics had great success in describing learning problems.

Second, we study the teacher-student model, which is the most basic way of making prediction on the *generalization* properties of a network, namely how many errors a network makes on data that it has not seen before.

Before proceeding the analysis, let's review the perceptron model in a statistical physics framework.

The perceptron consist in a single set of N real weights $W_i \in \mathbb{R}$ and in a rule to compute the output y that reads

$$y(W, \xi^\mu) = \text{sgn}(W \cdot \xi^\mu - b) \tag{1.10}$$

where $\xi^\mu \in \{-1,1\}^N$ is the μ-th input pattern and $\text{sgn}(\cdot)$ is the signum function. b is the threshold, that we put to zero in this section for simplicity. Each input pattern has a corresponding label $\sigma^\mu \in \{-1,1\}$, with $\mu \in \{1, ..., p\}$. A pattern is mislabelled by the model if $\sigma^\mu \neq y(W, \xi^\mu)$.

The learning task consist in finding a configuration of the weights W such that $\mathbb{X}_\xi(W) = 1$, with

$$\mathbb{X}_\xi(W) = \prod_{\mu=1}^{p} \Theta(\sigma^\mu y(W, \xi^\mu)) \tag{1.11}$$

being the function that checks if all the patterns are correctly memorized.

An equivalent formulation of the problem is finding W such that a loss function L that counts the number of errors is minimal:

$$L(W; \{\xi^\mu\}) = \sum_\mu \Theta(-\sigma^\mu y(W, \xi^\mu)) \tag{1.12}$$

where Θ is the Heaviside step function.

If we find a solution such that $\mathbb{X}_\xi = 1$, then we have $L = 0$. This formulation is closer to statistical mechanics, since in principle it allows to study the thermodynamics of a system with energy equal to the loss function, giving information to states with any number of errors. Here we are only interested in solutions with zero errors, meaning that we will do a zero-temperature equilibrium analysis.

The quantity that we are interested into is the volume Ω of the solution space

$$\begin{aligned}\Omega(\{\xi^\mu, \sigma^\mu\}) &= \int d\mu(W) \mathbb{X}_{\xi}(W) \\ &= \int d\mu(W) \prod_{\mu=1}^{p} \Theta\left(\frac{\sigma^\mu W \xi^\mu}{\sqrt{N}}\right)\end{aligned} \tag{1.13}$$

The integration measure $d\mu(W)$ could in general be any measure, but at this stage there is no reason to choose a non-uniform measure (this will radically change in 2). In the next chapter we will consider binary weight that do not have this problem, but for continuous weights we need a finite subset of the parameter space as a domain of the measure. The most sensible choice is to constraint the weight vector on the surface of an N-sphere: since the relevant quantity to determine the separating hyperplane is the direction of W, this choice gets rid only of the inessential component of W. We write the uniform measure on the sphere as

$$d\mu(W) = \prod_{i=1}^{N} \frac{dW_i}{\sqrt{2\pi e}} \delta(\sum_{i=1}^{N} W_i^2 - N) \tag{1.14}$$

where we chose the radius of the sphere to be $\sqrt{N}$ in order to have extensive quantities as argument of exponential functions in the later steps of calculations.

We are interested into an average over the realizations of the patterns and labels, but Ω itself is not suited for this because it is exponential in the size of the system N, and therefore not self-averaging. The correct quantity to average is the entropy $\log \Omega$. The quantity $\langle \log \Omega \rangle$ is called quenched average, as opposed to the annealed average $\log \langle \Omega \rangle$. The latter is much easier to compute and can be a useful lower bound to the former. Since we have no way of manipulating the logarithm of a summation, quenched averages require the replica trick to be computed, which is a way of writing the logarithm as a limit:

$$\log \Omega = \lim_{n \to 0^+} \frac{\Omega^n - 1}{n} \tag{1.15}$$

Thanks to this we only need to compute the average $\langle \Omega^n \rangle$, which is doable.

1.2.1 Storage problem: capacity

The problem consist in learning the correct example-label association of a set of p examples ξ^μ that have no correlation with their respective labels σ^μ. The examples and labels are drawn respectively from the distributions

$$P(\xi^\mu) = \prod_{i=1}^{N} \left[\frac{1}{2}\delta(\xi_i^\mu + 1) + \delta(\xi_i^\mu - 1)\right], \tag{1.16}$$

$$P(\sigma^\mu) = \frac{1}{2}\delta(\sigma_i^\mu + 1) + \delta(\sigma_i^\mu - 1). \tag{1.17}$$

Note that we can also exploit the internal symmetry of the model: we can map $\xi \to -\xi, \sigma \to -\sigma$ without changing the error count. Therefore, we can set all the labels to $+1$ by absorbing the labels inside the examples.

Asking the perceptron to reproduce this mapping is connected to the question of how many input-output pairs can be stored in an appropriate configuration of the parameters W without making mistakes.

To answer this question means to compute the maximal number p_c of associations that are possible given the size of the model N.

The physical approach to this problem consist in computing the volume Ω of the parameter space that is compatible with the set of constraints as a function of the number of constraints, namely the number p of patterns. This calculation will be performed in the thermodynamic limit $N \to \infty$, so a better quantity to consider is the ratio $\alpha := p/N$, called *capacity* or *load*. We expect the volume to progressively shrink until it becomes zero for a certain value p_c, that will determine the critical capacity α_c.

Given the volume $\Omega(\{\xi^\mu, \sigma^\mu\})$ for a certain a realization of the examples and labels $\{\xi^\mu, \sigma^\mu\}$, we want to compute its average over the distribution of examples in order to have some information on the average problem. Note that this is a crucial difference between physics and the common approach in computer science, where analyses are worst-case rather than average case.

Given this observation, the logarithm of the volume seems a better-suited quantity to average over the "disorder" variables ξ. The average of the logarithm of the volume is called *quenched average*, as opposed to the *annealed average* that we would do if we computed the logarithm of the average of the volume. See [8] for a deeper introduction to the quenched and annealed averages.

As said above, the logarithm of the volume is the correct quantity to average. Therefore, we want to compute the quantity

$$S = \langle \log \Omega(\{\xi^\mu, \sigma^\mu\}) \rangle_{\xi^\mu, \sigma^\mu} \tag{1.18}$$

via the replica trick 1.15.

Then the average of the replicated volume reads

$$\langle \Omega(\{\xi^\mu\}) \rangle_{\xi^\mu} = \left\langle \int \prod_{a=1}^{n} d\mu(W^a) \prod_{\mu=1}^{p} \Theta\left(\frac{W^a \xi^\mu}{\sqrt{N}} \right) \right\rangle_{\xi^\mu} \tag{1.19}$$

where we exploited the symmetry of the problem and set all the labels to $+1$, and we also dropped the trivial average over the labels.

The calculation proceeds by introducing two auxiliary set of variables $\lambda^{\mu,a}$ and $\hat{\lambda}^{\mu,a}$ that allow the integral representation of the Θ step function:

$$\Theta\left(\frac{\sum_i W_i^a \xi_i^a}{\sqrt{N}}\right) = \int \frac{d\lambda^{\mu,a} d\hat{\lambda}^{\mu,a}}{2\pi} \Theta(\lambda^{\mu,a}) \exp\left(i\hat{\lambda}^{\mu,a}(\lambda^{\mu,a} - \frac{\sum_i W_i^a \xi_i^a}{\sqrt{N}})\right) \tag{1.20}$$

Now we are capable of performing the average over the patterns via standard Gaussian integration. The averaged term reads

$$\left\langle \prod_{a=1}^{n} \exp\left(i\hat{\lambda}^{\mu,a} \frac{\sum_i W_i^a \xi_i^a}{\sqrt{N}}\right)\right\rangle = \prod_{\mu} \exp\left(-\frac{1}{2}\sum_{a,b} \hat{\lambda}^{\mu,a}\hat{\lambda}^{\mu,b} \sum_i \frac{W_i^a W_i^b}{N}\right) \tag{1.21}$$

We observe that this step introduced a coupling between different replicas in the form of $\sum_i \frac{W_i^a W_i^b}{N} := q^{ab}$. The variable q^{ab} actually will be the order parameter of the model, representing the overlap between replica a and b.

In order to proceed with the calculation, we need to make an ansatz on the form of the matrix q^{ab}. The most obvious one follows from the hypothesis that the replicas are symmetric under permutations, therefore

$$q^{ab} = \begin{cases} 1 & \text{if } a = b \\ q & \text{if } a \neq b \end{cases} \tag{1.22}$$

$$\hat{q}^{ab} = \hat{q} \tag{1.23}$$

This is equivalent to assuming that the Gibbs measure cannot be decomposed into a mixture of pure-state measures, but rather it is a well-defined unique state. With these assumptions and in the limit $n \to 0$ we are able to write the volume $\langle \Omega^n \rangle_\xi$ in the form

$$\langle \Omega^n \rangle_\xi = \int dq\, d\hat{q} \exp\left(N S_{\text{RS}}(q, \hat{q}; \alpha)\right) \tag{1.24}$$

where we used the $_{\text{RS}}$ subscript for the entropy to signal that this expression depends on the *replica-symmetric* ansatz me made on the matrix q^{ab}. This integral is solved in the limit $N \to \infty$ with a saddle-point integration, meaning that the physical values of the order parameters are those that satisfy the conditions (called *saddle-point equations*):

$$\frac{\partial S_{\text{RS}}}{\partial q} = 0, \quad \frac{\partial S_{\text{RS}}}{\partial \hat{q}} = 0 \tag{1.25}$$

From these equations we can determine the overlap $q(\alpha)$ between two typical solution of the learning problem as a function of the capacity α. The complete calculation can be found in [8] or in the original paper [9]. The results are that for $\alpha = 0$ we find $q = 0$; then, as we increase α, the overlap q starts increasing too until it reaches $q = 1$.

This means that at this point two random solutions of the storage problem are exactly the same, namely there is only one possible distinct solution left. Then we can identify α_c as the value of α such that $q(\alpha) = 1$. For $\alpha > \alpha_c$ we are imposing too many constraints and the set of solutions becomes empty.

Once we conclude a replica calculation, we should also check if the ansatz that we used for the Parisi matrix is consistent. In principle one should compute the second derivatives of the saddle-point expression to see if the solution that we are considering is a minimum or a maximum, but this calculation is often involved. In practice, if the ansatz is wrong, nonphysical results are very probable to come up during the computation of the entropy or other physical observables (as we will see in chapter 2). Within this chapter we just point out when an ansatz is incorrect without discussing the matter, so that the reader can assume that the ansatz are correct unless stated otherwise. For the discussion on the stability of each ansatz check the papers cited in the corresponding sections.

We make an observation to wrap up this subsection: we learned that α is the control parameter of learning problems. In this specific problem a threshold value α_c determines whether the problem has a solution of not, but studying relevant quantities a function of α is the approach that we will keep in any learning problem. Of particular interest are phase transitions occurring with α as driving parameter. In the next subsection, for example, we will compute the generalization error as a function of α for a perceptron in the teacher-student scenario.

1.2.2 Teacher-student problem: generalization error

In this section we introduce the concept of generalization error, that is at the hearth of inference and learning.

For now, we considered a learning problem as a "memorization-without-errors" problem, where we are satisfied when the model correctly reproduces the associations in the data that we used to train it. Let's call *training set* or *trainset* this set of data. In reality, we don't actually care about the performance of the model trainset, because we are interested in the predictive capabilities of the model, namely how well the model learned something about the distribution of the data, rather than memorizing a set of input-output relations "by heart". For this reason we are interested in evaluating the loss function of the model on data that we did not use to train the model, to see how well the model *generalizes*. We call *testset* this new set of data, and we call *generalization error* the value of the loss function on the testset. In many practical cases we have a limited amount of data; we want to use as much data as possible to train the model so that we use almost all the information that we have, while also leaving a decent number of data points in the testset to reliably assess the generalization error. A rule of thumb is to use 80% of the data for the trainset and the remaining 20% for the testset.

Let's now study the simplest framework that allows for an analytical calculation of the generalization error: the teacher-student scenario.

In the teacher-student scenario the labels σ^μ are defined as the output of a second network (the teacher), which is randomly initialized and is identical to the student network. This means that we constrained the student and teacher too to be spherical with radius $\sqrt{N}$, namely the following equations hold:

$$W^2 = \sum_i W_i^2 = N \tag{1.26}$$

$$(W^T)^2 = \sum_i (W_i^T)^2 = N \tag{1.27}$$

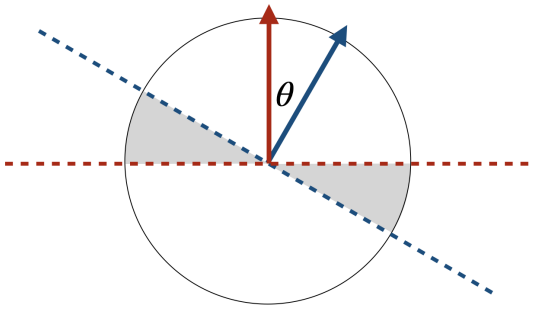


Figure 1.4: 2-dimensional representation of a teacher-student scenario: θ is the angle between the vectors corresponding to the teacher and the student. The shaded area corresponds to the region of the input space where the two models disagree. A student is in version space if the shader area is empty.

In this way we can visualize the situation as in figure 1.4: the mistakes in the student's classification are due to the mismatch between the student's separating hyperplane and the teacher's one. All the data points that line in the shaded area will be misclassified. If we assume that the data points are random, the probability ε of making a mistake is proportional to angle θ between the teacher and the student:

$$\varepsilon = \frac{\theta}{\pi} \tag{1.28}$$

We can rewrite this expression by defining the *teacher-student overlap*

$$R := \sum_i \frac{W_i^T W_i}{N} \tag{1.29}$$

which is the cosine of θ. Therefore, we see that generalization error depends only on the overlap R between teacher and student:

$$\varepsilon = \frac{1}{\pi} \arccos R \tag{1.30}$$

We call *version space* the set of students that are compatible with the teacher, given a set of examples. Our "learning procedure" here consist in uniformly sampling a solution from the version space and studying its generalization error ε.

As we did for the storage case, the strategy is to compute the volume of solutions averaged over uniform the distribution of students inside the version space. The reason is that by doing so we need to use the replica trick again, and the overlap R will appear as a natural order parameter of the model. Then we will obtain an expression for R as a function of α as a saddle-point equations, which leads to the expression for the generalization error $\varepsilon(\alpha)$ that we are looking for.

Here we only sketch the calculation. For a detailed derivation see [8].

The loss function for the teacher-student model is

$$L(W) = \sum_{\mu} \Theta(-\sigma^{\mu}\tau^{\mu}) \tag{1.31}$$

where $\sigma^{\mu} = \text{sgn}(W^T \cdot \xi^{\mu})$ is the label given by the teacher perceptron W^T and $\tau^{\mu} = \text{sgn}(W \cdot \xi^{\mu})$ is the output of the student perceptron W.

In order to compute the entropy of the zero-temperature solution space we use the replica trick. We end up with an expression similar to equation 1.19, except the integral representation of constraints now reads

$$\Theta\left(\frac{\sum_i W_i^T \xi_i^{\mu}}{\sqrt{N}} \frac{\sum_j W_j^a \xi_j^{\mu}}{\sqrt{N}}\right) = \int \frac{du^{\mu} d\hat{u}^{\mu}}{2\pi} \frac{d\lambda^{\mu a} d\hat{\lambda}^{\mu a}}{2\pi} \Theta(u^{\mu}\lambda^{\mu a}) \exp\left(i\hat{\lambda}^{\mu a}(\lambda^{\mu a} - \frac{\sum_j W_j^a \xi_j^{\mu}}{\sqrt{N}}) + i\hat{u}^{\mu a}(u^{\mu a} - \frac{\sum_i W_i^T \xi_i^{\mu}}{\sqrt{N}})\right) \tag{1.32}$$

The calculation of the replicated volume proceeds by computing the average over the disorder (the data). This operation makes us introduce the same order parameters $q^{ab} = \sum_i \frac{W^a W^b}{N}$ that we used in the storage case plus and additional set of order parameters $R^a := \sum_i \frac{W^T W^a}{N}$, which represent the overlap between the a-th replica and the teacher.

After some involved passages we make an RS ansatz

$$\begin{aligned} q^{ab} &= q \\ R^a &= R \end{aligned} \tag{1.33}$$

and we end up with a set of coupled saddle-point equations that involve q, R and α. From one of these equations we find that $q = R$, signaling an additional symmetry in

our problem. In fact, the teacher is chosen at random with uniform distribution and lies in the version space by definition. The student is itself chosen uniformly at random inside the version space, so we could have expected that the teacher has no special role in this calculation and is equivalent to just another replica, as shown by the calculation itself.

The remaining saddle-point equation reads

$$\frac{R}{\sqrt{1-R}} = \frac{\alpha}{\pi} \int Dt\, H \frac{\exp(-Rt^2/2)}{H(-\sqrt{R}t)} \tag{1.34}$$

where $Dt = \frac{dt}{\sqrt{2\pi}} \exp(-t^2/2)$ is a Gaussian measure and $H(x) = \int_x^\infty Dt$ is the complementary error function.

This equation can be solved numerically. We find that the teacher-student overlap R is zero when we do not have any data point ($\alpha = 0$) and monotonically increases when we increase the size of the trainset, until it tends to $R = 1$ for $\alpha \to \infty$. At the same time the generalization error signals a totally random prediction $\epsilon = 0.5$ for $\alpha = 0$ and goes to zero for $\alpha \to \infty$.

1.3 Gradient-based learning and deep networks

In the previous sections we introduced neural networks starting from their basic unit — the perceptron — and we introduced the concept of generalization, which is fundamental for inference and learning. In this section we study how to build more complex network and how to train these objects. The discussion on the generalization capabilities of these architectures is left for the next section.

1.3.1 Gradient descent

Instead of a single perceptron, let's consider K perceptrons with the same set on input channels. This architecture is called a *single-layer* feed-forward neural network and it is just a more general perceptron with multidimensional output (see figure 1.5 for a sketch of the architecture). The model parameters are now organized as an $N \times K$ matrix. We assume that each perceptron has the same activation function.

In this section we drop the assumption that the activation function of the perceptron is binary and we consider a generic continuous-valued activation function f. This allows us to use a different strategy for finding a configuration of W that implements a correct clarification. First, let's introduce a derivable loss function L that measures the deviation between the current output of the model and the desired output; then, we ask if we can find a solution of the problem by minimizing L. There are many possible choices for a derivable loss function, depending on the specific problem and architecture. Here we introduce the simplest one, the mean squared error.

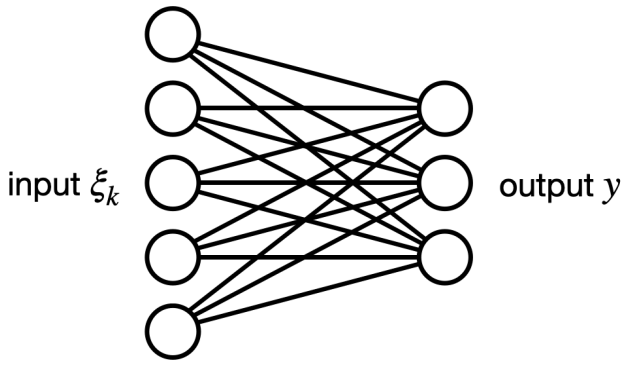


Figure 1.5: Sketch of a multidimensional perceptron.

$$L(W;\xi^\mu) := \frac{1}{2}\sum_{\mu=1}^{p}\sum_{j=1}^{K}(\sigma_j^\mu - y_j^\mu)^2 \tag{1.35}$$

There are many possible strategies for minimizing a loss function. Here we introduce the simplest one, called *gradient descent*, where the update of the parameters ΔW is given by the gradient of L with respect to the parameters themselves:

$$\Delta W_{ij} = -\frac{\partial L}{\partial W_{ij}} \tag{1.36}$$

Let's write explicitly the loss function and its derivative to compute the update term.

$$L(W;\xi^\mu) := \frac{1}{2}\sum_{\mu=1}^{p}\sum_{j=1}^{K}\left(\sigma_j^\mu - \sum_{i=1}^{N} W_{ij}\xi_i^\mu\right)^2 \tag{1.37}$$

$$\Delta W_{ij} = -\frac{\partial L}{\partial W_{ij}} = \sum_{\mu=1}^{p}(\sigma_j^\mu - y_j^\mu)\xi_i^\mu = \sum_{\mu=1}^{p}\delta_j^\mu\xi_i^\mu \tag{1.38}$$

where we defined the "errors" as $\delta_j^\mu := \sigma_j^\mu - y_j^\mu$.

Note that now the update reads as a sum of terms identical to equation 1.9, which was the basic update rule one example at a time. Now instead we are computing the gradient of the entire loss function, which consists of the sum of an error term for each example in the dataset. The gradient descent algorithm consists in updating the weights until the gradients become negligible: at that point we reached a minimum of L. We call *training epoch* each time the algorithm sees the entire dataset: in this case, the number of epoch coincides with the number of update steps. Since we are using the entire dataset to compute the gradients, this algorithm is also known as *full-batch* gradient descent, for reasons that will be clear in the next subsection.

1.3.2 Stochastic gradient descent

The sum over the examples is a universal characteristic of loss functions: a generic loss function can be written as

$$L(W;\xi^{\mu}) = \sum_{\mu=1}^{p} l(W;\xi^{\mu}) \tag{1.39}$$

Regardless of the choices of L we will need to compute a sum over the dataset to evaluate the gradient for the update rule. This operation can become very computationally expensive, especially since it must be repeated at each step of the algorithm until convergence.

For this reason the most used variation of gradient descend, called *stochastic gradient descent* (SGD), relies on just estimating the gradients on a subset of the dataset, called *mini-batch*. The update rule now reads

$$\Delta W_{ij} = -\sum_{\mu=1}^{B} \frac{\partial l(W;\xi^{\mu})}{\partial W_{ij}} \tag{1.40}$$

where $B < p$ is the dimension of the mini-batch.

SGD is important not only to reduce the computational cost, but also to avoid getting stuck in local minima of the loss function. In fact, in general the loss landscape is expected to be highly non-convex and therefore following the exact gradient does not guarantee to reach the global minimum. SGD has an intrinsic noise due to the estimation of the gradient, which gives the algorithm a chance of escaping local minima.

There are many variants of SGD in the literature (see [10] for a representative list), but they all build on this basic idea of not computing the gradient exactly but sampling it instead.

1.3.3 Deep learning and Backpropagation

Now that we described a single-layer network, we can add another layer on top of it by feeding the output of the first layer to the input of the second one (see figure 1.6). This procedure can be iterated adding any amount of layers to the network and the resulting architecture is called a *deep neural network*. This type of layer is called *fully connected layer*, since each output neuron is connected to each input one. The main other type of layer is the *convolutional layer*, which is essential for computer vision. Discussing convolutional neural networks is beyond the scope of this thesis; they are cited here just to mention that the standard architectures for image learning use a combination of convolutional and feed forward layers. Networks with too many (more than ~ 5) consecutive fully connect layers are generally more difficult to train and require special algorithms to find solutions. For a proper introduction to convolutional neural networks and for discussion about convergence of fully-connect network see [10].

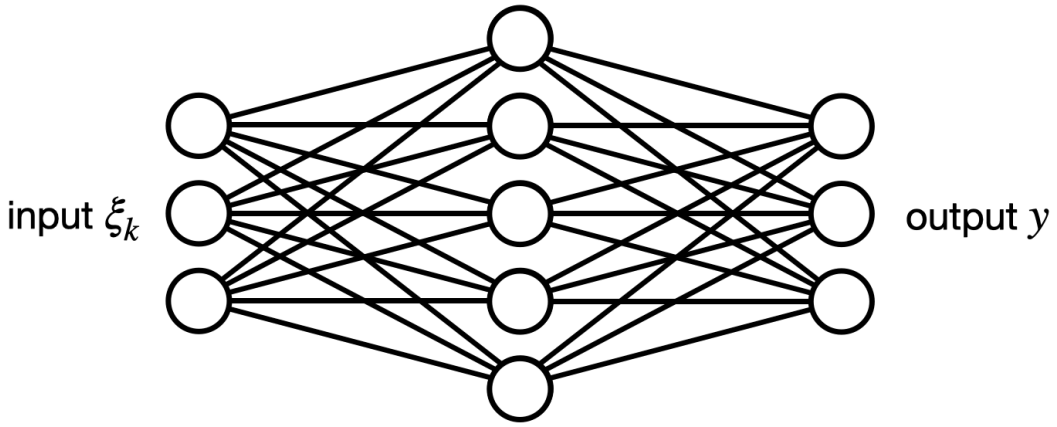


Figure 1.6: Sketch of a feed-forward network with two layers.

Stochastic gradient descent applied to deep networks is called *backpropagation*. The reason is that it can be interpreted as if we computed the errors in the output layer and we propagated them back to the input layer.

To show this, let's write explicitly the quadratic loss function for a two-layered fully-connected network and compute the gradients. In deep architectures, every layer of neurons except the first and the last one are called *hidden layers*; also note that the input layer is not counted for the total number of layers, so that the number of layers is equal to the number of parameter matrices. Let's say that $i \in [1, N]$, $j \in [1, J]$ and $k \in [1, K]$ are the index spanning respectively the input, hidden and output neurons. Let's call W_{ij} the weights of the first layer and U_{jk} the weights of the second layer. Then let's define the following quantities:

- *hidden units pre-activations*: $H_j^\mu := \sum_i W_{ij}\xi_i^\mu$
- *hidden units activations*: $h_j^\mu := f(H_j^\mu)$
- *output units pre-activations*: $Y_k^\mu := \sum_j U_{jk}h_j^\mu$
- *output units activations*: $y_k^\mu := f(Y_k^\mu)$

The loss function reads:

$$L(W;\xi^\mu) = \frac{1}{2}\sum_{\mu=1}^{p}\sum_k \left[\sigma_k^\mu - f\left(\sum_j U_{jk} f(\sum_i W_{ij}\xi_i^\mu)\right)\right]^2 \tag{1.41}$$

Now let's compute the derivative of L with respect to U_{jk} :

$$\frac{\partial L}{\partial U_{jk}} = -\sum_\mu (\sigma_k^\mu - y_k^\mu) f'(Y_k^\mu) h_j^\mu = -\sum_\mu \delta_k^\mu h_j^\mu \tag{1.42}$$

where we defined the "errors" as $\delta_k^\mu = (\sigma_k^\mu - y_k^\mu) f'(Y_k^\mu)$. Note that including $f'(Y_k^\mu)$ in the definition of δ_k^μ allows us to recognize the same form of equation 1.38.

The derivative of L with respect to W_{ij} reads

$$\begin{aligned} \frac{\partial L}{\partial W_{ij}} &= -\sum_\mu \sum_k (\sigma_k^\mu - y_k^\mu) f'(Y_k^\mu) U_{jk} f'(H_j^\mu) \xi_i^\mu \\ &= -\sum_\mu \sum_k \delta_k^\mu U_{jk} f'(H_j^\mu) \xi_i^\mu \\ &= -\sum_\mu \gamma_j^\mu \xi_i^\mu \end{aligned} \tag{1.43}$$

where we used the definition of δ_k^μ in the second line and we defined $\gamma_j^\mu := f'(H_j^\mu) \sum_k \delta_k^\mu U_{jk}$ in the third line.

Now we can observe how the update rule for $W_i j$ depends only on the i-th components of the inputs and on some "hidden layer errors" γ_j that are the result of the backward propagation of the "output layer errors" δ_k^μ.

As said above, every sum over μ can be performed on the whole dataset or on smaller mini-batches. To sum up, the resulting algorithm is sketched below.

Algorithm 1 Stochastic gradient descent with backpropagation

```
initialize the weights of each layer at random
for e = 1, ... , training epochs do
    split the dataset in mini-batches
    for b = 1, ... , number of mini-batches do
        for μ = 1, ... , size B of mini-batches do
            insert an example ξ_i^μ in the input layer
            propagate forward the signal and compute the output y_k^μ
            compute the errors δ_k^μ
            backpropagate the error and for each layer compute the updates Δw_ab^μ
        end for
        update the parameters with the rule w_ab^new = w_ab^old + Σ_{μ=1}^B Δw_ab^μ
    end for
end for
```

1.4 Overfitting and regularization

In this section we discuss the general problem of generalization in statistical inference and we introduce a Bayesian framework for neural network, which proves useful to understand the common techniques to avoid bad generalization performances.

1.4.1 General characteristics of overfitting

The problem of *overfitting* is common issue in inference: if we have a set of data points that we want to fit with a certain model, we must keep the model "simple enough" so that we do not introduce artifact features, which would cause the model to have a very bad generalization error. The trade-off is that a model that is too simple will not be able to fit the entire trainset, while a model that is too complicated will overfit the data. For the sake of concreteness let's consider the inference of a polynomial curve from a finite set of data points. In this case we can say that the measure of complexity of the model is the degree of the polynomial that we choose, that is the amount of parameters of our model. If we choose degree that is too high, the fitted curve will oscillate a lot in the gaps between the data points, while the point likely lie on a much smoother curve (see for example [11]). Note that the difficulty with neural networks is that we do not have a straightforward way to control the complexity of the fitted function, since it depends on the details of the training procedure and of the architecture.

The general way to check if the model is overfitting the data is to check the generalization error during the training. Usually we see the training error that monotonically decreases, while the generalization error follows the trend of the training error for a while then starts increasing (or stops decreasing). The difference between train error and test error is called *generalization gap*. This behavior signals that by training the model further we are generating features that are too specific to the training data.

Any modification that we introduce in the model to prevent overfitting is called *regularization*.

One obvious strategy is to stop the training procedure once we reach the generalization error minimum rather than when we reach zero training error. This is called *early stopping*. The problem with this regularization is that we are now using the testset to decide when to stop, which means using the number of training epochs as parameter to be learned. Therefore, the data points in the generalization set are not unseen by the model anymore. A more correct way of doing this is to split further the trainset creating a *validation set* used to fix the early stopping parameter, leaving the testset untouched.

In this case the early stopping parameter, namely the number of epochs, is a *hyperparameter* of the learning: it is not one of the parameters of the model but it still influences the generalization. Other common hyperparameter are the learning rate, the batch size and the various regularizer strengths, that we will discuss below.

Apart from early stopping, the most obvious regularization is called *weigh decay*: it consists in adding a penalty to the loss function proportional to the L2-norm of the weights W. This term keeps the order of magnitude of the weights finite, so that the fitted function does not oscillate too much, preventing one of the most common sources of overfitting.

1.4.2 Bayesian framework of learning

Let's quickly review the Bayes theorem, that will be useful to reinterpret the MSE loss function and the weight decay in a probabilistic framework. For a much deeper introduction to the role of Bayesian probability in machine learning see [11, 10].

Given the set H of hypotheses of our model and the dataset x, the Bayes theorem assigns a probability to the model parameters w according to the equation

$$P(w|x, H) = \frac{P(x|w, H)P(w|H)}{P(x|H)} \tag{1.44}$$

where $P(x|w, H)$ is the likelihood of the data given the model, $P(w|H)$ is the *prior* probability of the model parameters and

$$P(x|H) := \sum_{w'} P(x|w', H)P(w'|H)$$

is called *evidence*. $P(w|x, H)$ is called *posterior* probability and is the subject of study of Bayesian inference: the criterion known as *maximum a posteriori* (MAP) estimate consists in fixing the model parameters w by maximizing $P(w|x, H)$.

The advantages of MAP respect to the maximization of the likelihood (ML estimation) are many:

- we have a way of comparing different models that is built-in in the Bayesian framework;
- we can predict new samples from the generative model using a whole probability distribution over the parameters and not just a point estimate
- we can include in the inference some prior knowledge of the model parameters; this characteristic is the one that will turn out useful to design regularization strategies for neural networks.

Model comparison For instance, we observe that in H we include any assumptions that we made to use the model, included the choice of the model itself. In this paragraph we will consider also the ML estimate of the model parameters w as part of H. This allows us to compare different models and even different estimations of w within the same model. In fact, let's write the Bayes theorem with in a slightly different way:

$$P(H|x) = \frac{P(x|H)P(H)}{P(x)} \tag{1.45}$$

do that we can compare some hypotheses H_1 and H_2 using the posterior probabilities $P(H_1|x)$ and $P(H_2|x)$. Let's consider the case where we don't have any a priori reason to prefer one of the two hypotheses, so that prior probabilities are equal: $P(H_1) =$

$P(H_2) = 1/2$. In order to make the comparison we need to compute the two quantities $P(x|H_1)$ and $P(x|H_2)$, which are the evidences for the basic MAP estimation of the parameters. We can get rid of the need to compute $P(x)$ by comparing the ratio

$$\frac{P(H_1|x)}{P(H_2|x)} = \frac{P(x|H_1)P(H_1)}{P(x|H_2)P(H_2)} \tag{1.46}$$

to answer the question of which of the two hypotheses is more probable.

Predicting new occurrences Let's say that we observed p examples and we want to predict the $p+1$-th example. We can write the conditional probability as

$$p(x^{(p+1)}|x^{(p)}, \dots, x^{(1)}) = \int dw\, p(x^{(p+1)}|w)\, p(w|x^{(p)}, \dots, x^{(1)}) \tag{1.47}$$

With the traditional inference we would have used only our best guess of the parameters w to predict $x^{(p+1)}$, whereas with Bayesian inference we can exploit each set of parameters that has a non-zero probability. Additionally, since each w enters the integral with its probability, we are already incorporating the uncertainty of each estimate without the need to calculate the variance, as we would do in a traditional inference.

Of course this improvement is not free, since the computation of the integral 1.47 is often very computationally heavy. Nonetheless, this is an important result that opens up possibilities that were absent in traditional inference.

Inclusion of prior knowledge Let's now study the role of the prior in the Bayes theorem 1.44.

If the prior is flat it is easy to see that a MAP estimation coincides with a ML estimation. Things become more interesting when the prior actually shifts the likelihood towards regions of the parameters space that are preferred *a priori*, which is for example the case of the preference towards "simple enough" functions that we discussed above. The magnitude of this shift depends on how peaked the prior distribution is on certain regions, as we will see in a minute.

We now show how the MSE loss function and the weight decay regularization can be interpreted by the means of Bayesian inference. Consider a dataset $x = \{x^{(1)}, \dots, x^{(p)}\}$ of samples from an unknown distribution P_{data} and say that we estimate this distribution with a model and we get likelihood $P_{\text{model}}(x|w)$, where w are the parameters of the model. The ML estimate of w is

$$\begin{aligned} w_{\text{ML}} &= \arg\max_w P_{\text{model}}(x|w) \\ &= \arg\max_w \sum_{\mu=1}^{p} \log P_{\text{model}}(x|w) \\ &= \arg\max_w \sum_x P_{\text{data}}(x) \log P_{\text{model}}(x|w) \end{aligned} \tag{1.48}$$

where in the second line we added a log that does not change the position of the maximum; we wrote the last line to point out that maximizing the likelihood can be seen as minimizing the cross entropy (or the Kullback-Leibler divergence) between P_{data} and P_{model}. In the following we will switch the perspective from maximizing the likelihood to minimizing the negative log likelihood (NLL), to get closer to a loss-function formalism.

For the sake of concreteness let's now say that we are studying a regression problem where we observe some function $y(x; w)$ plus a Gaussian noise with zero mean and variance σ^2, which gives the following distribution of the data:

$$P(y_{\text{obs}}|x) = \mathcal{N}(y_{\text{obs}}; y(x; w), \sigma^2) \tag{1.49}$$

Let's now write our loss function, the negative log likelihood:

$$\begin{aligned} L_{\text{NLL}}(w; x) &= -\sum_{\mu=1}^{p} \log P_{\text{model}}(y^{\mu}_{\text{obs}}|x^{\mu}; w) \\ &= p \log \sigma + \frac{p}{2} \log(2\pi) + \sum_{\mu=1}^{p} \frac{1}{2\sigma} ||y^{\mu}_{\text{obs}} - y(x^{\mu}; w)||^2 \end{aligned} \tag{1.50}$$

From the last line we can finally observe that the MSE loss function is equivalent to a ML likelihood estimation where we assume that the data have been generated with the same model architecture that we are using for the inference plus a Gaussian noise.

Let's now study a MAP estimation in the same circumstance. The posterior probability reads

$$P(w|y_{\text{obs}}) \propto P_{\text{model}}(y_{\text{obs}}|w)P(w) \tag{1.51}$$

and the MAP estimator reads

$$w_{\text{MAP}} = \arg\max_{w} \sum_{\mu=1}^{p} [\log P_{\text{model}}(y^{\mu}_{\text{obs}}|w) + \log P(w)] \tag{1.52}$$

If we now choose a factorized Gaussian prior with zero mean and variance $\lambda^{-1/2}$

$$P(w_i) \propto \exp\left(-\frac{\lambda}{2} w_i^2\right) \tag{1.53}$$

and we write the corresponding loss function we obtain

$$L(w; x) = \text{const} + \sum_{\mu=1}^{p} \frac{1}{2\sigma} ||y^{\mu}_{\text{obs}} - y(x^{\mu}; w)||^2 + \frac{\lambda}{2} ||w||^2 \tag{1.54}$$

which is a MSE loss plus a weight decay regularization. We can also observe that the hyperparameter λ, which is the regularizer "strength", is nothing but the inverse variance of the Gaussian prior. A stronger regularization corresponds to a more peaked prior distribution.

1.4.3 Regularization strategies

The definition of a regularization strategy is a modification to the model that is aimed at improving the generalization error and not the training error.

We already introduced the strategies of early stopping and weight decay in subsection 1.4.1. From early stopping we learned that minimizing the loss function might not ensure the best generalization, since we saw how the minimum on the generalization error happens when the training loss is not yet optimized. This observation help to understand a general behavior of those regularizers that are terms added to the loss: they shift the position of the global minimum by a distance that depends on the regularizer strength λ. This hyperparameter is also interpretable as the Lagrangian multiplier in a constrained optimization framework.

In this class of additional-loss-terms regularizers we find all norm-based regularizers plus a number of sparsity-enforcing regularizers (we will talk about the importance of sparsity in chapter 4). Some examples are:

- L^2, which is used to keep the norm of the parameters from exploding;
- L^1, which is used to enforce sparsity in the weights;
- L^0, which is a more effective sparsity enforcer but it is not derivable, so it must be used with non gradient-based algorithms such as belief propagation;
- $(L^1)^2$, which is an example of an ad-hoc regularizer for feature extraction with restricted Boltzmann machines (see [12]);
- $L^1(h)$, which is a sparsity enforcer applied to the activation of some layer of a network rather than to the weights themselves; this is still related to feature extraction and will be the regularizer studied throughout chapter 4.

A more subtle class of regularizers are the modifications to the learning procedure that produce a regularization on the model parameters in a indirect way.

- *Noise injection* As we said in subsection 1.3.2, in general the loss landscape is complex and non-convex, so we need strategies to avoid getting stuck in local minima or to end up in very slow dynamics that take an unreasonable amount of steps to converge. The intrinsic noise of SGD helps the convergence of the dynamics, but this and other forms of noise during the training can also have a regularization effect: for example we can perturb the example patterns each time we feed them to the network so that the model is less prone to overfit the specific patterns; we can also perturb the weights at each mini batch, for example by setting to zero a small fraction of them so that no single input channel can be fundamental and therefore collective features are encouraged (this is called *dropout* technique).

- *Initialization* If loss landscape is rough and non-convex it could be hard for algorithms to move significantly away from the initial condition, therefore the initialization of neural becomes of great importance; sometimes a bad initialization prevents the model to be able to learn at all. The universal strategy for initializing neural networks is to have a very small norm from the beginning, incorporating in this way an implicit L^2 regularization in the initial condition which we already know is a good prior for our model.

Some strategies can become quite elaborate, so that they become more of a preprocessing phase:

- *Data augmentation* A strategy similar to injecting noise in the example patterns is to generate different version of the example, each with a different perturbation. A good example of this comes from computer vision: when we classify faces we expect the faces in the dataset to have slightly different poses from each other, but those are inessential to the classification. In order to prevent the model to focus on the poses, we can generate additional variations of the pictures by rotating them slightly, of reflecting them. This procedure builds an extended dataset that can be used as a normal trainset.

A comprehensive introduction many of the possible regularization strategies in deep neural networks can be found in [10].

Chapter 2

Local Entropy and neural networks

This chapter focuses on the theory of local entropy, a concept that offers a possible solution to the conundrum of why deep neural network do not overfit, in contrast to what the theoretical framework of statistical learning predicts.

The concept of local entropy has been first developed on the binary perceptron, the simplest formulation of a learning problem with constrained synapses. According to a worst-case analysis the model is intractable and no efficient (polynomial time) algorithms should exist.

An equilibrium description predicts the existence of an extensive number of metastable states that dominate the Gibbs measure and trap algorithms based on free-energy minimization. Additionally, the solutions appear to be geometrically isolated, making local-search strategies worse.

Nevertheless, the existence of a number of efficient solvers (see [13, 14]) contradicts what we just stated, suggesting that equilibrium statistical mechanics is not enough to describe learning in the binary perceptron. For this reason binary perceptron seems the perfect starting point to understand the gap between theory and numerical results in neural networks. We will discuss results for deeper networks at the end of the chapter.

We begin by presenting the surprisingly rich structure of the binary perceptron solution space. First in section 2.1 we review the standard equilibrium results for both storage and teacher-student scenarios, highlighting how the isolated nature [explain this above] of typical solution clashes with empirical results. Then in section 2.2 we carry a large-deviation analysis that introduces the idea of local entropy and clarifies how algorithms are attracted to subdominant solutions that are invisible to equilibrium statistical mechanics.

With the new understanding of the geometry of the phase space, in section 2.3 we are able to design a class of algorithms that specifically search for these subdominant solutions, improving performances. In fact, it is important to note that, from the teacher-student scenario, we learn that flat minima not only are accessible but also have good generalization properties.

In section 2.5 we briefly review the theory of local entropy for some common tools of more complicated architectures, in order to bridge the gap between the perceptron and the state-of-the-art models in deep learning.

2.1 Binary perceptron: properties of the solution space

In order to carry the calculations we need to specify the distribution of the patterns $\{\xi^\mu\}$. We know only two analytically solvable cases. The first one is to consider independent identically distributed random patterns drawn from the uniform distribution; this is called the *random storage* case. The other option is the *teacher-student* case and consist in labels generated by a teacher perceptron with weights W^T, namely $\sigma^\mu = \text{sgn}(W^T \cdot \xi^\mu)$.

2.1.1 Storage

Revisiting the Gardner analysis

This subsection contains only a sketch of the complete calculations and it follows the discussions contained in [15, 14]. For detailed calculations see those, or for example [8].

The binary storage perceptron has the same symmetry as the spherical storage perceptron: we can map $\xi \to -\xi, \sigma \to -\sigma$ without changing the error function. Since we are studying random patters, this symmetry allows us to set all the labels to $+1$ without loss of generality.

We set up the standard Gardner analysis but instead of a uniform distribution over the sphere we consider the uniform distribution on the hypercube vertices, that fixes the weights to be binary:

$$d\mu(W) = \prod_{i=1}^{N} (\delta(W_i - 1) + \delta(W_i + 1)) \tag{2.1}$$

We need to compute the following average:

$$\langle \Omega^n \rangle = \left\langle \int \prod_{a=1}^{n} d\mu(W^a) \prod_{\mu=1}^{\alpha N} \Theta\left(\frac{\sum_i W_i^a \xi_i^a}{\sqrt{N}} \right) \right\rangle_\xi \tag{2.2}$$

where $a \in \{1, .., n\}$ is the replica index. At the end of the calculation we will do the limit $n \to 0$. Note that we scaled the weights with $1/\sqrt{N}$ in order to have extensive quantities as argument of exponential functions, in the same fashion of the standard Gardner analysis.

Then we proceed in the same way as we did for the spherical perceptron: we introduce two auxiliary set of variables that allow the integral representation of the Θ step function; then we compute the average over the disorder, which introduces a coupling

between different replicas that we call overlap q^{ab}. At this point we make a replica symmetric ansatz and compute the remaining integral with a saddle-point integration. It is worth noting that these steps are exactly the same as the spherical case, except the integration measure $d\mu(W)$.

From the standard Gardner analysis (see 1.2.1) of the spherical perceptron we know that the critical capacity α_C can be identified as the α such that $q \to 1$. Solving equations 1.25 we find a solution that reaches $q = 1$ at a capacity 1.27, but this cannot be correct because the binary perceptron is composed of N bits and therefore its capacity is bounded at 1. Additionally, we find numerically that the entropy S_{RS} becomes negative after the value 0.833, signaling that the solution cannot be correct in that regime. Interestingly, we could ask if the zero-RS-entropy capacity us indeed the critical capacity itself, since we don't find any solutions after that point. This is in fact the case, but we will have a more clear picture in the next section.

These observations are pointing out that the RS ansatz is not working. They way to ensure that the RS solution is unstable is to compute the Hessian of the entropy evaluated in the solution. Numerically it is found that the RS solution is unstable for $\alpha = 1.015$.

One-step replica-symmetry breaking

We understood that we need another ansatz for the matrix q^{ab}. If we assume that because of the glassy landscape the ergodicity is completely broken, we can think of the probability measure $\mathscr{P}$ as decomposed into a convex linear combination of pure-state measures $\mathscr{P}_a$, each of those corresponding to a cluster of solutions geometrically separated from the others. We group the n replicas in n/m blocks, each containing m replicas. It is useful to define new indices that explicitly describe the organization in blocks: we split the index a into the couple (α, β), where $\alpha \in \{1, ..., n/m\}$ is the block index and $\beta \in \{1, ..., m\}$ is the inter-block index. The *one-step replica-symmetry breaking* (1RSB) ansatz consist in the following:

$$q^{\alpha\beta,\alpha'\beta'} = \begin{cases} 1 & \text{if } \alpha = \alpha' \text{ and } \beta = \beta' \\ q_1 & \text{if } \alpha = \alpha' \text{ and } \beta \neq \beta' \\ q_0 & \text{if } \alpha \neq \alpha' \text{ and } \beta \neq \beta' \end{cases} \tag{2.3}$$

$$\hat{q}^{\alpha\beta,\alpha'\beta'} = \begin{cases} \hat{q}_1 & \text{if } \alpha = \alpha' \text{ and } \beta \neq \beta' \\ \hat{q}_0 & \text{if } \alpha \neq \alpha' \text{ and } \beta \neq \beta' \end{cases} \tag{2.4}$$

With this matrix we find a new entropy S_{1RSB} and new set of saddle-point equations. Now we can try again to identify the critical capacity threshold α_C by studying the limit $q_1 \to 1$ with $q_0 < 1$. With the proper scaling the saddle-point equation for m reduces to the requirement S_{RS}, which confirms the idea that the zero-RS-entropy criterion can be used to identify α_C in the binary perceptron.

Interestingly we find only two solutions to the saddle-point equations that are stable for $\alpha < \alpha_C$: the trivial RS one with $m = 0$ and $q_1 = q_0 = q_{RS}$ and the 1RSB one with $q_1 = 1$ and $q_0 = q_{RS}$. For $\alpha \geq \alpha_C$ the entropy of the 1RSB solution is not negative but zero, thus correcting the nonphysical behavior.

The picture that emerges from this analysis is that the solutions, rather than being organized into clusters, are organized into an exponential number of point-like pure states for $\alpha < \alpha_C$.

Additionally, we see that the typical solutions can be seen both as unique pure states with solutions at overlap q_{RS} and as an ensemble of pure states that are point-like clusters, with internal overlap $q_1 = 1$ and external overlap $q_0 = q_{RS}$. These point-like clusters have zero internal entropy, but their complexity turn out to be equivalent to the RS internal entropy.

Franz-Parisi analysis

We want to confirm the picture that we built in the previous section of the geometry of the typical solutions in a binary storage perceptron. In particular, we want information about the vicinity of typical solutions. Specifically, we ask if those are really isolated, that is how far from a given solution we do find another one.

In order to do this, we compute the so-called Franz-Parisi potential in the zero-temperature limit. This analysis was done first in [15]. The idea of the Franz-Parisi potential $\mathcal{S}_{FP}$ is to consider a reference configuration $\tilde{W}$ samples from the Boltzmann distribution at temperature $1/\beta'$, then to compute the free-energy of a bigger model where a configuration W at temperature $1/\beta$ is constrained to be at distance D from the reference $\tilde{W}$:

$$\mathcal{S}_{FP} = \frac{1}{N} \left\langle \frac{1}{Z(\beta')} \sum_{\{\tilde{W}\}} e^{-\beta' E(\tilde{W})} \log \left(\sum_{\{W\}} e^{-\beta E(W)} \delta \left(d(W, \tilde{W}) - D \right) \right) \right\rangle_{\xi,\sigma} \tag{2.5}$$

where D is the Hamming distance between two configurations.

In the limit $\beta, \beta' \to \infty$ the reference configuration is sampled from a uniform distribution over the solution space. We also remind that without loss of generality we can consider all the labels $\sigma^\mu = 1 \ \forall \mu$, trivializing the average over σ. We obtain the expression:

$$\mathcal{S}_{FP} = \frac{1}{N} \left\langle \left\langle \log \sum_{\{W\}} \prod_{\mu} \Theta \left(\sum_i \frac{W_i \xi_i^\mu}{\sqrt{N}} \right) \delta \left(d(W, \tilde{W}) - D \right) \right\rangle_{\tilde{W}} \right\rangle_{\xi} \tag{2.6}$$

This expression can be expanded with the use of the replica trick. The main order parameters are

- the overlap between two typical solutions $\tilde{q}^{cd} := \sum_i \tilde{W}_i^c \tilde{W}_i^d / N$;

- the overlap between two replicas $q^{ab} := \sum_i W_i^a W_i^b / N$;
- the overlap between a replica and a solution $S^{ca} = \sum_i \tilde{W}_i^c W_i^a / N$.

The expression reduces after an involved calculation to a saddle-point integration with seven order parameters in total. The seven saddle-point equations can be integrated numerically to find the optimal value of the order parameters and consequently compute the Franz-Parisi potential.

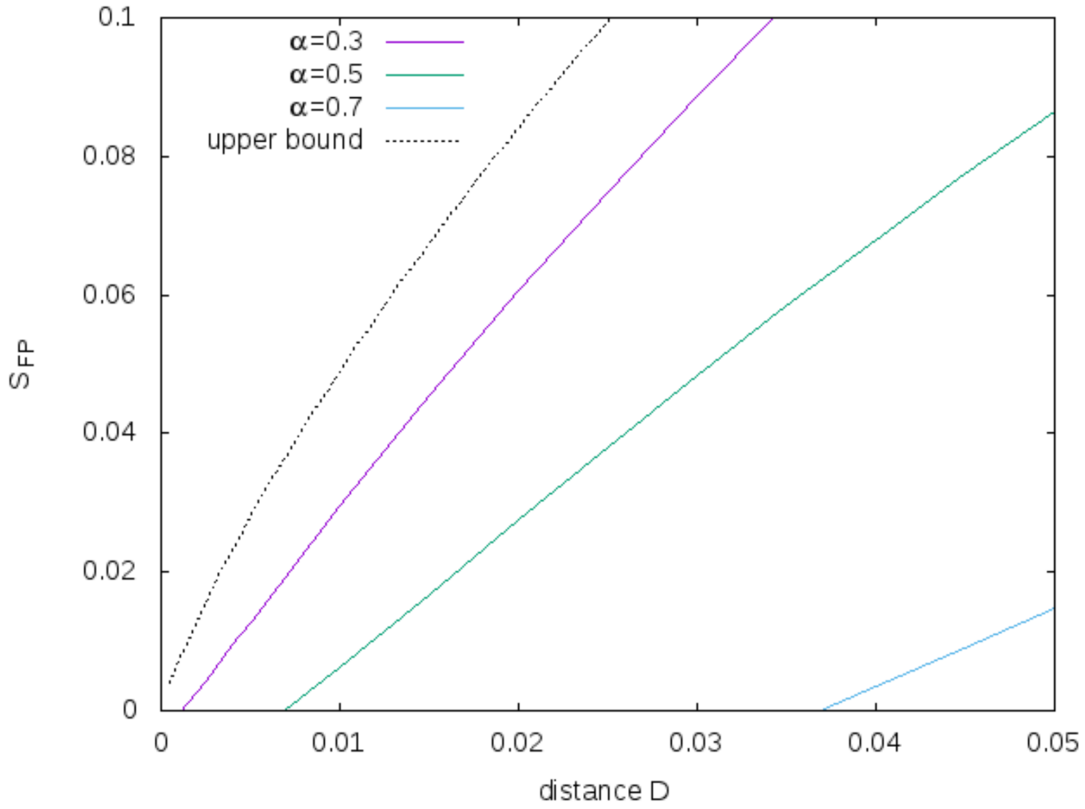


Figure 2.1: Typical solutions of a binary perceptron in the storage case are isolated. The image shows theoretical curves of Franz-Parisi entropy, which show a zero-entropy gap in the vicinity of the reference. Different colors correspond to increasing value of the capacity $\alpha = p/N$, while the dotted curve corresponds to $\alpha = 0$, which is the logarithm of the unconstrained volume of the space at a certain distance. Image from [14].

The results are shown in figure 2.1. For all values of α we find a zero-entropy gap in the vicinity of the reference, namely for small distance D. This means that there are no other solution within a certain distance, and there is no cluster of typical solutions. Additionally, we see that the gap increases as the number of pattern increases. This could be the origin of the computational hardness of the binary perceptron, since it becomes harder and harder for local-search strategies to find other solutions.

2.1.2 Teacher-student

This subsection follows the results in [16], where the equilibrium analysis of the binary perceptron was extended to the teacher-student scenario. We will see that the results do not change qualitatively from the storage scenario: typical solutions are isolated for all values of α, and the teacher itself is isolated and indistinguishable from all other typical solutions.

Equilibrium analysis

It is easy to see that the loss L is invariant under the transformation $\xi^\mu \to -\xi^\mu$. This fact allows us to fix the gauge $W_i^T = 1 \ \forall i$ without loss of generality. Therefore, the labels become $\sigma^\mu = \text{sgn}(\sum_i \sigma_i^\mu)$ and thanks to the scale invariance of the Θ function we can remove the sgn functions. The loss becomes:

$$L(W) = \sum_\mu \Theta\left(-\frac{\sum_i \xi_i^\mu}{\sqrt{N}} \frac{\sum_j W_j \xi_j^\mu}{\sqrt{N}} \right) \tag{2.7}$$

The replica calculation proceeds similarly to the storage case. We first make a RS ansatz and we find a threshold capacity $\alpha_\text{D} = 1.245$, after which the entropy becomes negative. A 1RSB approach gives the correct results of zero entropy for $\alpha > \alpha_\text{D}$ and shows that the only solution in that regime is the one with $R = 1$, namely the teacher itself.

Franz-Parisi analysis

Given that the teacher itself is the only (and therefore isolated) solution for $\alpha > \alpha_\text{D}$ it is interesting to ask whether the solution below the critical capacity are isolated too. To do this we compute again the Franz-Parisi potential, in the same fashion we did above for the storage case.

We take the expression of the Franz-Parisi entropy 2.5 and we plug in the expression for the energy of the teacher-student perceptron 2.7. The results reads

$$\mathcal{S}_\text{FP} = \frac{1}{N} \left\langle \frac{1}{Z} \sum_{\{\tilde{W}\}} \prod_\mu \Theta\left(\frac{\sum_i \xi_i^\mu}{\sqrt{N}} \frac{\sum_j \tilde{W}_j \xi_j^\mu}{\sqrt{N}} \right) \right.$$
$$\left. \log \sum_{\{W\}} \prod_\mu \Theta\left(\frac{\sum_i \xi_i^\mu}{\sqrt{N}} \frac{\sum_j W_j \xi_j^\mu}{\sqrt{N}} \right) \delta\left(d(W, \tilde{W}) - D \right) \right\rangle_\xi . \tag{2.8}$$

We call $\tilde{W}$ the pseudo teacher. Its distribution is the uniform one among all the students compatible with the original teacher (that we set to 1 with a gauge choice).

The new student perceptron is constrained to be at distance D from the pseudo-teacher and to still be compatible with the original teacher.

The distance constraint introduces a coupling between the student and the pseudo teacher, and therefore we have two additional order parameters other than the old ones $\tilde{q}^{cd}$, q^{ab} and S^{ca}:

- the overlap between the pseudo-teachers and the teacher $\tilde{R}^c := \sum_i \tilde{W}_i^c/N$;
- the overlap between a student and the teacher $R^a := \sum_i W_i^a/N$;

The saddle-point equations that we obtain for the whole set of order parameters are the same that we would have obtained if we studied the model of a student coupled to the teacher via a distance constraint, except that the overlap S would play the role of the actual teacher-student overlap, rather than being the pseudo-teacher-student overlap. By further comparing the saddle-point equations in the two cases one can note that a typical solution to the teacher student problem, which we chose as our pseudo-teacher, is indistinguishable from the teacher in everything except the generalization properties (see the Supplemental Material of [16] for the equations).

The numerical solution to the saddle-point equations is shown in figure 2.2. We see that for all $\alpha > 0$, the entropy is always negative in a neighborhood of the pseudo-teacher, meaning that all typical solutions of the teacher-student problem (including the teacher itself) are extensively isolated.

2.1.3 Contradictory numerical results

We now compare the results obtained with an heuristic algorithm and the typical solutions obtained with the equilibrium analysis of the last section. In particular, we want to check if algorithms really find isolated solutions; we intuitively expect that isolated solutions are golf-course-like holes in the loss landscape, and that for this reason they are very hard to be detected by algorithms that use local-search-based strategies.

The number of solutions as a function of the distance from a reference solution is shown in figure 2.3. We can see that there is a remarkable difference between the curve of typical solutions and the curve of algorithmic solutions, also visible if we compare the entropy as a function of the Franz-Parisi potential (inset of figure 2.3). See [16] and [14] for more details of how to compute the curves and for deeper discussions.

The observation that equilibrium statistical mechanics appears to not describe the solutions found by algorithms raises the need for a different theory that matches with the numerical experiments. This new theory is described in the next section.

2.2 Binary perceptron: subdominant states

In this section we introduce a large-deviation analysis that will reveal the existence of a class of out-of-equilibrium solutions that have radically different properties from the

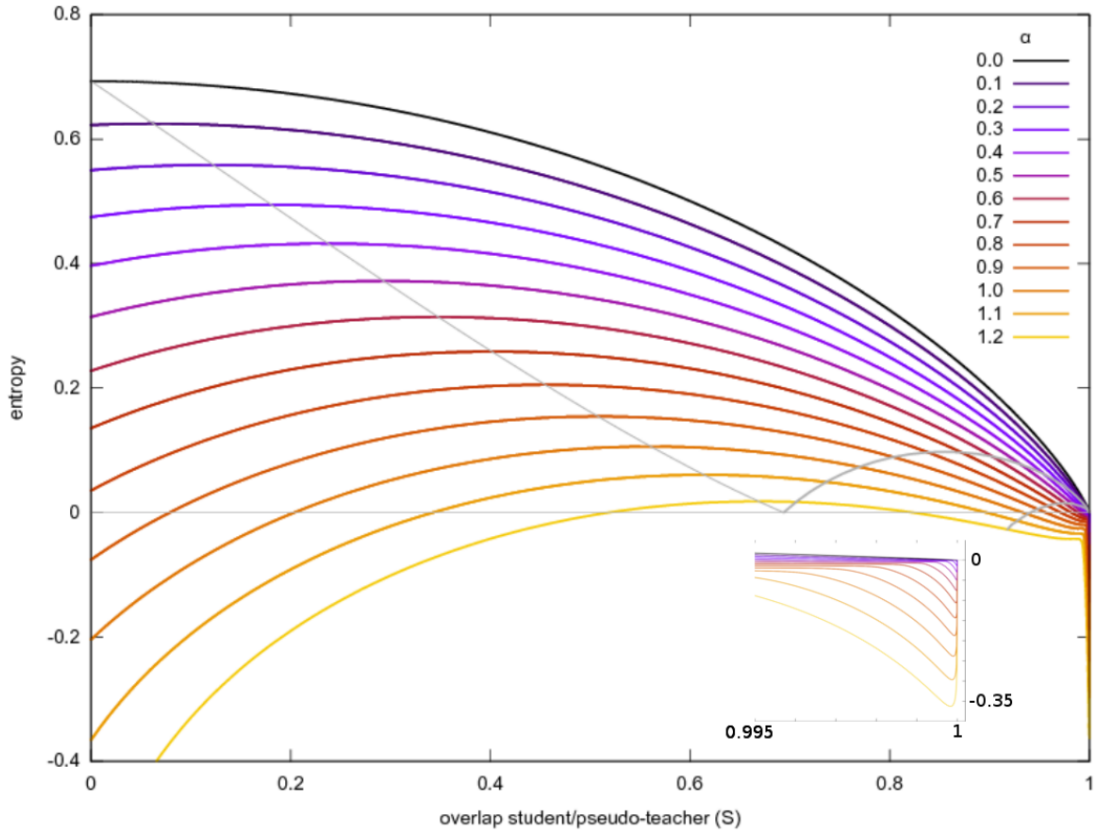


Figure 2.2: Typical solutions of a binary perceptron in the teacher-student case are isolated. The image shows theoretical curves of Franz-Parisi entropy, which show negative values in the vicinity of the reference. Different colors correspond to increasing value of the capacity $\alpha = p/N$, while the top black curve corresponds to $\alpha = 0$, which is the logarithm of the unconstrained volume of the space at a certain distance. Image from [14].

typical solutions, namely they are subdominant with respect to the Boltzmann measure and they are clustered in dense regions of the solution space. We will see that this new class turns out to have a much better agreement with numerical results, therefore we will argue these dense solutions are the ones accessible to simple learning protocols, rather than the isolated equilibrium solutions.

We start the analysis by defining a biased probability measure P biased towards solutions that have many other solutions nearby:

$$P(\tilde{W}; y, d) = \frac{\mathbb{X}_{\xi}(\tilde{W})\mathcal{N}(\tilde{W}, d)^y}{\sum_{\{\tilde{W}'\}} \mathbb{X}_{\xi}(\tilde{W}')\mathcal{N}(\tilde{W}', d)^y} \tag{2.9}$$

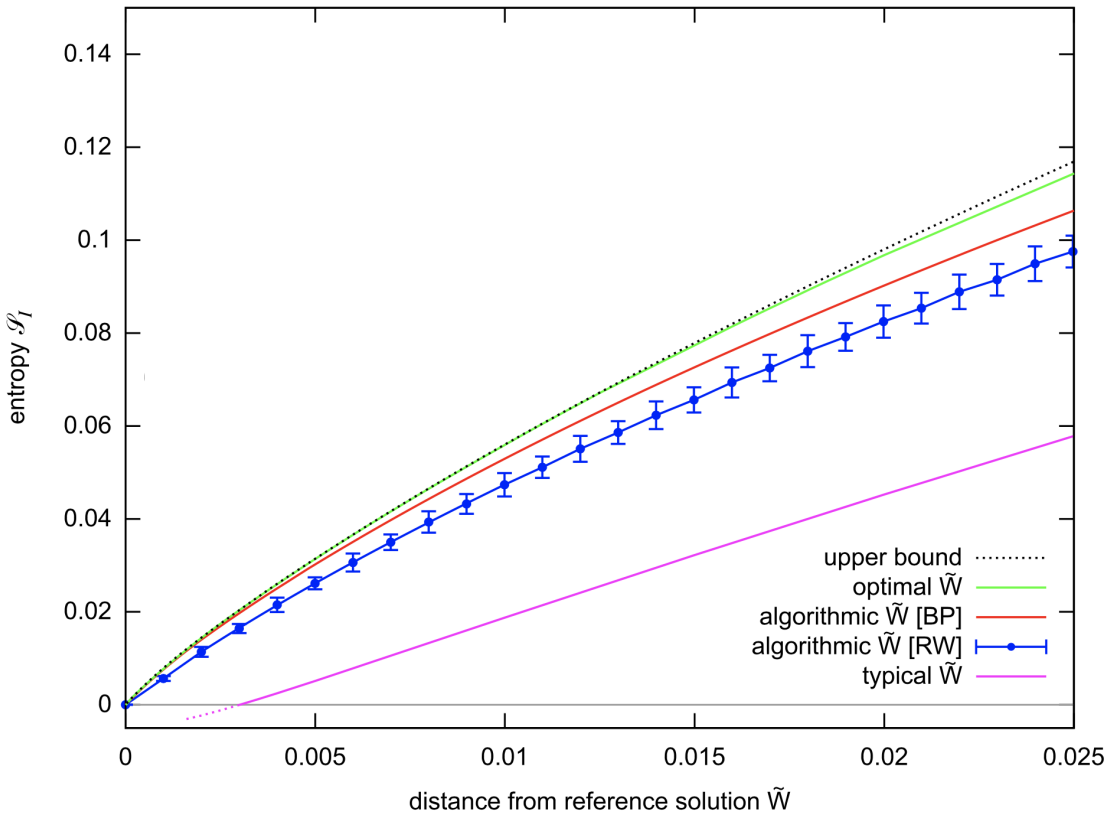


Figure 2.3: The entropy of typical solutions do not correspond to the entropy of algorithmic solutions. The image shows a comparison between curves of Franz-Parisi entropy for different references. From bottom to top: the purple curve is for typical solutions; the blue and the red curves are for algorithmic solutions (the blue curve is estimated with a Monte Carlo while the red is estimated with belief propagation, see [16]); the green curve is for the solution that optimizes local entropy. Note that the algorithm solution and the local-entropy optimal solution do not have a zero-entropy gap in the vicinity of the reference. Image adapted from [16].

where the function $\mathcal{N}(\tilde{W}, d)$ counts the number of solutions at distance d from the reference $\tilde{W}$:

$$\mathcal{N}(\tilde{W}, d) = \sum_{\{W\}} \mathbb{X}_{\xi}(W)\delta(W \cdot \tilde{W}, N(1-2d)) \tag{2.10}$$

The meaning of equation 2.9 can be understood by changing the value of the "inverse temperature" y: if we set $y = 0$ we recover the Boltzmann case, namely the flat measure in the solution space; if we increase y the measure gets more biased around dense solutions and if we send $y \to \infty$ we would expect to find the minimum of an energy-like function defined as $\mathscr{E}(\tilde{W}) = -\frac{1}{N}\log \mathcal{N}(\tilde{W}, d)$.

The average of $-\mathscr{E}(d,y)$ over the disorder and the solution space is called *local entropy* $\mathcal{S}(d,y)$:

$$\mathcal{S}(d,y) := -\left\langle \mathscr{E}(\tilde{W}) \right\rangle_{\xi,\tilde{W}} = \frac{1}{N}\left\langle \log \mathcal{N}(\tilde{W},d) \right\rangle_{\xi,\tilde{W}} \tag{2.11}$$

The local entropy can be obtained as the first derivative of a certain average free entropy density $\mathscr{F}$

$$\mathscr{F}(d,y) = -\frac{1}{Ny}\left\langle \log\left(\sum_{\{\tilde{W}\}} \mathbb{X}_\xi(\tilde{W}) \mathcal{N}(\tilde{W},d)^y \right) \right\rangle_\xi \tag{2.12}$$

In fact

$$\begin{aligned}
\mathcal{S}(d,y) &= \partial_y(y\mathscr{F}(d,y)) \\
&= -\frac{1}{N}\left\langle \frac{1}{Z(y)} \sum_{\{\tilde{W}\}} \mathbb{X}_\xi(\tilde{W}) e^{y\log \mathcal{N}(\tilde{W},d)} \log \mathcal{N}(\tilde{W},d) \right\rangle_\xi \\
&= -\frac{1}{N}\left\langle \frac{1}{Z(y)} \sum_{\{\tilde{W}\}} \mathbb{X}_\xi(\tilde{W}) e^{y\log \sum_{\{W\}} \mathbb{X}_\xi(W)\delta(W\cdot\tilde{W},N(1-2d))} \right. \\
&\qquad \left. \log \sum_{\{W\}} \mathbb{X}_\xi(W)\delta(W\cdot\tilde{W},N(1-2d)) \right\rangle_\xi
\end{aligned} \tag{2.13}$$

Note that in the limit $y \to 0$ this expression reduces to a formula analogous to a Franz-Parisi potential. Here, instead, we leave y as a control parameter.

Another important quantity will be the *complexity* Σ, also called *external entropy*, which is defined as

$$\Sigma(d,y) = \mathscr{F}(d,y) - y\mathcal{S}(d,y) \tag{2.14}$$

The external entropy counts the number of different cluster, while the local entropy (also called *internal entropy*) counts the number of solutions inside a cluster.

Our large deviation analysis consists in calculating the average free entropy density 2.12 with the replica method. This free energy is minimal in correspondence of dense clusters of solutions which are subdominant with respect to the Boltzmann measure.

This analysis can be performed in the two scenarios of random storage and teacher-student, depending on the definition of the characteristic function $\mathbb{X}_\xi(\mathbb{W})$. We will see that the RS saddle point produces qualitatively very similar results for both the random storage and the teacher-student scenarios below the corresponding critical capacity (respectively α_c and α_{TS}).

We end this preliminary subsection by providing an alternative "soft" definition to equation 2.10 of $\mathcal{N}(\tilde{W}, d)$. This definition, or variations of it with different distances, will be useful in the next section. Instead of using the Kronecker delta-function, a more natural definition uses a Lagrangian multiplier γ to enforce a certain distance:

$$\mathcal{N}(\tilde{W}, d) = \sum_{\{W\}} \mathbb{X}_\xi(W) \exp(-\frac{\gamma}{2}(W - \tilde{W})^2) \tag{2.15}$$

This definition will come in handy when we will consider models with $\beta \neq 0$ and when we will design algorithms to actively search for dense clusters, because the Lagrange multiplier will simply be coupling between different replicas of the same model. In the thermodynamic limit, definitions 2.10 and 2.15 are equivalent.

2.2.1 Sketch of the large-deviation analysis

Here we present a sketch of the calculations of the local entropy curves that will be show in the following subsections. For the complete discussion and the detailed calculations see the original works [14, 16].

Let's call *volume* Ω the argument of the logarithm in $\mathcal{F}(d, y)$ (equation 2.12):

$$\begin{aligned} \Omega(d, y) &:= \sum_{\{\tilde{W}\}} \mathbb{X}_\xi(\tilde{W}) \mathcal{N}(\tilde{W}, d)^y \\ &= \sum_{\{\tilde{W}\}} \mathbb{X}_\xi(\tilde{W}) \left[\sum_{\{W\}} \mathbb{X}_\xi(W) \delta(W \cdot \tilde{W}, N(1-2d)) \right]^y \end{aligned} \tag{2.16}$$

To compute $\mathcal{F}(d, y)$ we need first to compute the average of the n-th power of Ω:

$$\begin{aligned} \langle \Omega(d, y)^n \rangle_\xi = \Bigg\langle \int \prod_{i=1}^N \prod_c d\mu(\tilde{W}_i^c) \int \prod_{i=1}^N \prod_{ca} d\mu(W_i^{ca}) \prod_c \mathbb{X}_\xi(\tilde{W}^c) \prod_{ca} \mathbb{X}_\xi(W^{ca}) \\ \prod_{ca} \delta(\sum_{i=1}^N W_i^{ca} \tilde{W}_i^c, 1 - 2dN) \Bigg\rangle_\xi \end{aligned} \tag{2.17}$$

where we defined two set replicas, both from the n exponent and from the y exponent. This means that we end up with two different replica indices: index $c \in 1, \dots, n$ refers to the n replicas of the reference configuration $\tilde{W}$; index $a \in 1, \dots, y$ refers to the replicas of the student configuration W, y for each reference replica, so ny in total.

In other words, we can picture the following situation: we have n reference solutions acting as pseudo-teachers for groups of y student replicas at a fixed distance d. At this stage we take only the limit $n \to 0$, while y is still a parameter of the problem.

Now we proceed in the standard way of replica calculations: we plug into equation 2.17 the integral representation of the characteristic functions $\mathbb{X}_\xi$ and of the δ function, then we perform the average over the disorder in the large N limit. At this point we have to specify an ansatz for the Parisi matrix: we choose the RS ansatz as it is the simplest one.

Interestingly we can note that, even within the RS ansatz, we already have a geometric structure in the model, namely the one described above. This structure has been produced by the reweighting term and it is formally similar to a 1RSB description, with an important difference: the distance between the student replicas is now fixed, and not a parameter of the Parisi matrix that needs to be optimized.

After choosing the ansatz we can proceed with the saddle point integration, that is solved by minimizing a free energy density with respect to the order parameters of the problem. We find 11 saddle point equations for the storage scenario and 13 for the teacher-student scenario. These equations must be integrated numerically. The form of the equations depends on the choice we make for the parameter y, which is discussed immediately below.

Once we have computed the free energy 2.12 with the replica trick, we can compute the local entropy simply as its derivative (see equation 2.11).

Problems with the large y limit, RS ansatz

As we said above one idea to compute expression 2.12 would be to take the limit $y \to \infty$, in order to find the optimal state of the local entropy 2.11. Unfortunately this limit leads to nonphysical results within the RS ansatz. In fact, if we compute the average free entropy density 2.12 and from that the complexity 2.14, we see that for all values of α and d there is a value of y beyond which $\Sigma(d, y) < 0$, which is absurd.

This behavior signals the incorrectness of the RS ansatz. This fact can be interpreted geometrically: a RS ansatz implies that there should be a unique solution with maximal local entropy (since the overlap between different replicas tends to 1); if this is not true it means that the set of solutions with maximal local entropy has a non-trivial structure, namely is not a single cluster anymore (beyond said y).

Finite y, RS ansatz

Instead of taking the 1RSB ansatz, we take a different approach. We fix the value of y to the highest possible so that the complexity is non-negative: this is called the *vanishing complexity criterion*. In practice, we must find the value $y^* = y^*(d, y)$ such that $\Sigma(d, y^*) = 0$, then we plug it in the saddle point equations so that the dependency on y is dropped.

With this method we are actually able to compute the average free energy in a meaningful way (for certain ranges of α and d that we will discuss below), and therefore we can compute the local entropy too. The main results are listed in the following

subsections. Note that the results listed in the storage subsection are also valid for the Teacher-student case. In subsection 2.2.3 we will discuss the generalization properties that are specific for this scenario.

It must be noted again that the solutions obtained with the vanishing complexity criterion are not the solutions with maximal local entropy, as those would be the ones obtained with $y \to \infty$. The values of local entropy discussed here are therefore a lower bound. However, we can note that $y^* \to \infty$ both when $d \to 0$ and $d \to 1$, suggesting that problems with the RS ansatz only appear for intermediate distances. Adjustments due to further levels of RSB would likely be small, and limited to the intermediate regions of d.

2.2.2 Storage

The results for the random storage scenario are shown in fig. 2.4.

For $\alpha < \alpha_c$ local entropy is greater than zero in a neighborhood of $d = 0$, signaling the presence of a dense cluster of solutions in correspondence of high-local-entropy regions. Additionally, the curves for different capacity collapse onto each other when $d \to 0$. Since in particular they collapse on the $\alpha = 0$ curve, we learn that the center of the cluster is extremely dense of solution, namely at the same density as if there were no constraints. The size of the cluster appears to reduce when we increase α, until it disappears when $\alpha = \alpha_c$.

For larger distances local entropy collapses onto the entropy of typical solutions (according to the Boltzmann distribution) with a second-order phase transition.

We also note that there is a capacity threshold α_U beyond which the local entropy curves stop being monotonic in d and at some point start being negative. This is when the RS ansatz starts failing again, signaling that the dense cluster is breaking into a more complex structure.

2.2.3 Teacher-student

The teacher-student scenario is useful to test the generalization properties of the solutions inside the dense cluster. It turns out that they are generally much better than the typical solutions.

In fig. 2.5 we compare the generalization error as a function of α for typical solutions, dense solutions and Bayes-optimal solutions. We see that the generalization error increases monotonically with d, until we reach the error of typical solution when $d = 0$ (as expected since the biased measure 2.9 is equivalent to the Boltzmann measure if $d = 0$).

Additionally, we see that the curve for small d is in perfect agreement with the algorithmic solutions. This result supports the idea that algorithmic results are not distributed according to a Boltzmann distribution but rather according to a local entropy distribution like 2.9.

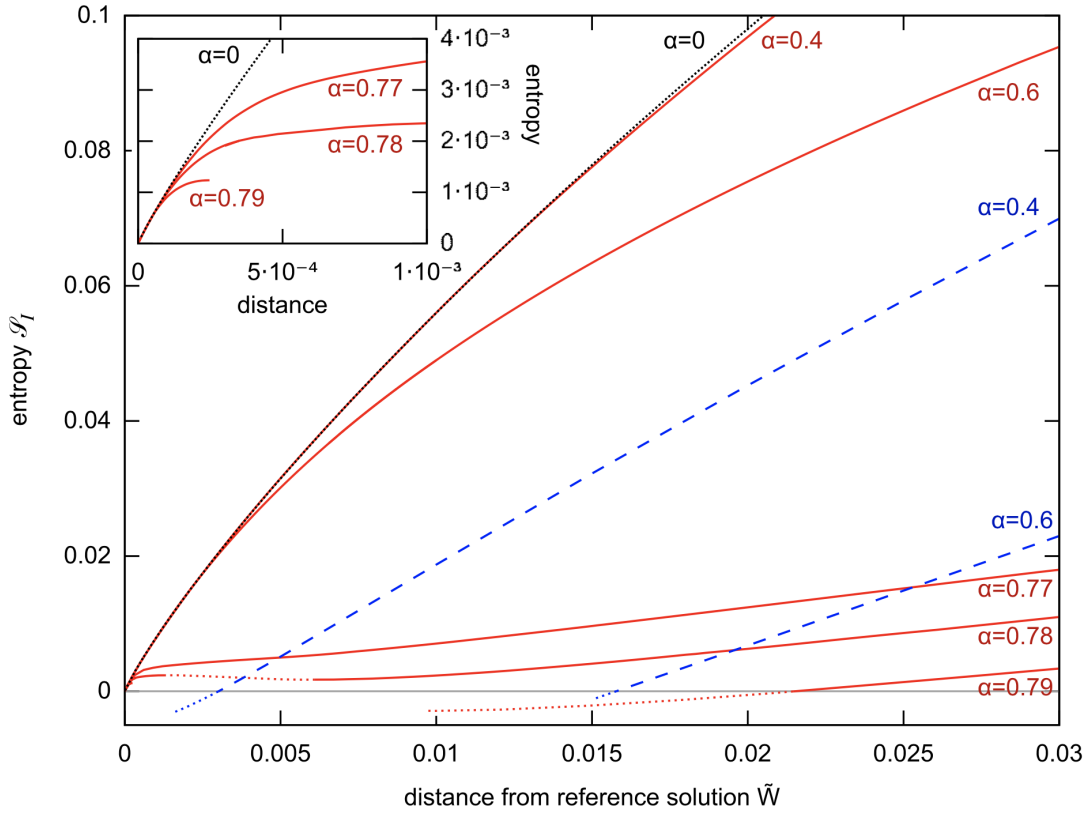


Figure 2.4: Local-entropy-optimal solutions of a binary perceptron in the teacher-student case are part of a dense cluster of solution. The image shows theoretical curves of Franz-Parisi entropy, which have a positive monotonic behavior in the vicinity of the reference. Red curves correspond to local-entropy-optimal solutions for different values of $\alpha = p/N$, while blue dotted curves correspond to typical solutions. The top black curve corresponds to $\alpha = 0$, which is the logarithm of the unconstrained volume of the space at a certain distance. Image from [16]

An intuitive way to interpret the good generalization property is a Bayesian argument: the output of a solution that lies in the middle of a cluster can be seen as a Bayesian estimator of all the solutions in the same cluster, therefore solutions towards the center of the cluster are expected to have a higher posterior weight than those at the border and even higher than those isolated.

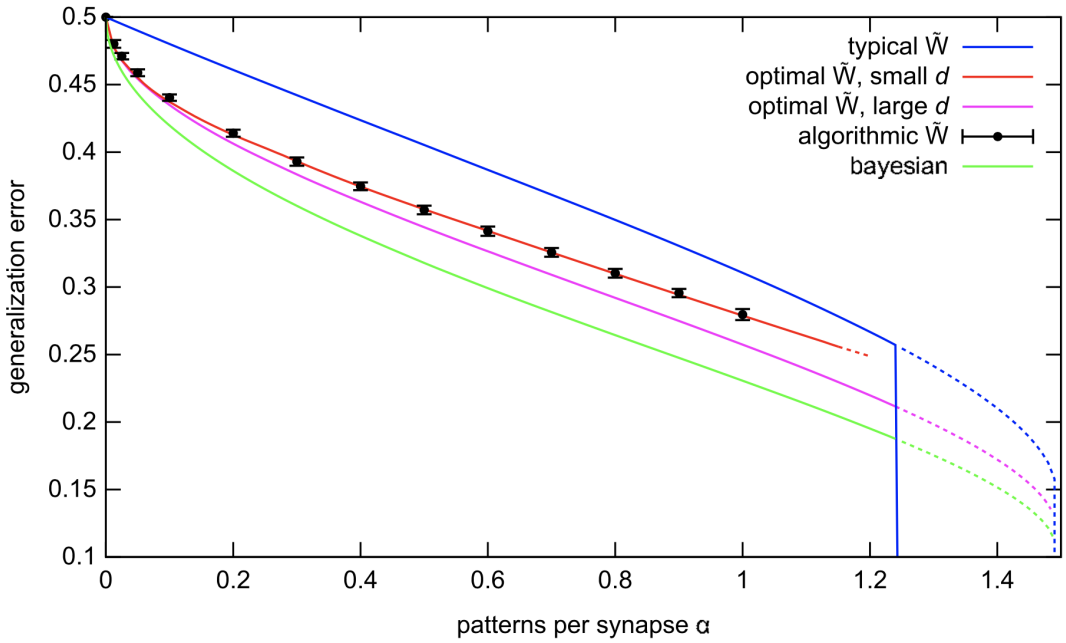


Figure 2.5: Local entropy of a solution correlates with its generalization. The image shows the generalization error as a function of α for different kind of solutions. From bottom to top: the green curve corresponds to a Bayes-optimal solution; the purple and red curves correspond to theoretical curves of local-entropy-optimal solution; the black dots correspond to algorithmic solutions; the blue curve corresponds to typical solutions. Image from [16].

2.3 Local entropy as objective function

In this section we describe how the theory introduced in the previous section can be exploited to derive a number of algorithms that actively target dense clusters of solutions.

Before introducing algorithms we want to study a slight variation of the free energy 2.12 where we remove the constraint for the reference configuration $\tilde{W}$ to be itself a solution of the problem (i.e. a ground state of the original loss function):

$$\begin{aligned} \mathscr{F}(d, y) &= -\frac{1}{Ny} \left\langle \log \sum_{\{\tilde{W}\}} \mathcal{N}(\tilde{W}, d)^y \right\rangle_\xi \\ &= -\frac{1}{Ny} \left\langle \log \sum_{\{\tilde{W}\}} e^{y \log \sum_{\{W\}} \mathbb{X}_\xi(W) \delta(W \cdot \tilde{W}, N(1-2d))} \right\rangle_\xi \end{aligned} \tag{2.18}$$

The reason for studying this model is that in this way we can use equation 2.18 as

an objective function and study the properties of $\tilde{W}$: if $\tilde{W}$ is a solution of the original problem this is due to the optimization of local entropy and not simply enforced by a constraint. Therefore, we will consider a storage scenario and compute the error rate of $\tilde{W}$.

In the following subsection we sketch this large-deviation analysis and compare the results both with the constrained case and with numerical simulation. The full discussion, as well as the algorithms used, can be found in [17, 14].

2.3.1 Sketch of the large-deviation analysis with unconstrained reference, storage

The replica calculation of equation 2.18 in a storage scenario with a RS ansatz leads to a number of nonphysical results. In fact, if we use again the vanishing complexity criterion to fix the value of y, the local entropy appears to be positive even for $\alpha > \alpha_c$, signaling the incorrectness of the RS ansatz. Additionally, the external entropy appears to be negative for all values of α and d.

Therefore, we have to make a 1RSB ansatz. The specific choice is to consider the symmetry breaking at the level of the reference configuration $\tilde{W}$ and not at the level of the "student" configurations W.

Here we have again a situation similar to equation 2.17, where we have n groups (corresponding to $\tilde{W}$) of y replicas (corresponding to W). We have two different replica indices: index $c \in 1, \dots, n$ refers to the n replicas of the reference configuration $\tilde{W}$; index $a \in 1, \dots, y$ refers to the replicas of the student configuration W, y for each reference replica, so ny in total.

In this scenario the calculation with finite y appears to be too difficult, so we resort to the limit $y \to \infty$.

The first thing to note about the results is that the external entropy is still negative for all values of α and d, even though its value tends to zero for $d \to 0$. Additionally, all the other nonphysical behaviors we had in the RS ansatz now disappear. Another couple of observations lead us to believe that this level of approximation is good enough: first, the qualitative behavior is the same as the RS calculation for the constrained model; and second, numerical experiments are in great agreement with this analysis (see figure 2.6).

In figure 2.6 the numerical results are obtained with *Entropy-driven Monte Carlo*, an algorithm first introduced in [17]. This algorithm and another one called *Entropy SGD* (first introduced in [1]) have been particularly important in the development of the theory of local entropy and therefore their general idea is described below. For more detailed descriptions see the original papers.

We will describe in mode detail other two algorithms, *Replicated Monte Carlo* and *Replicated SGD*, because they are used extensively in this thesis.

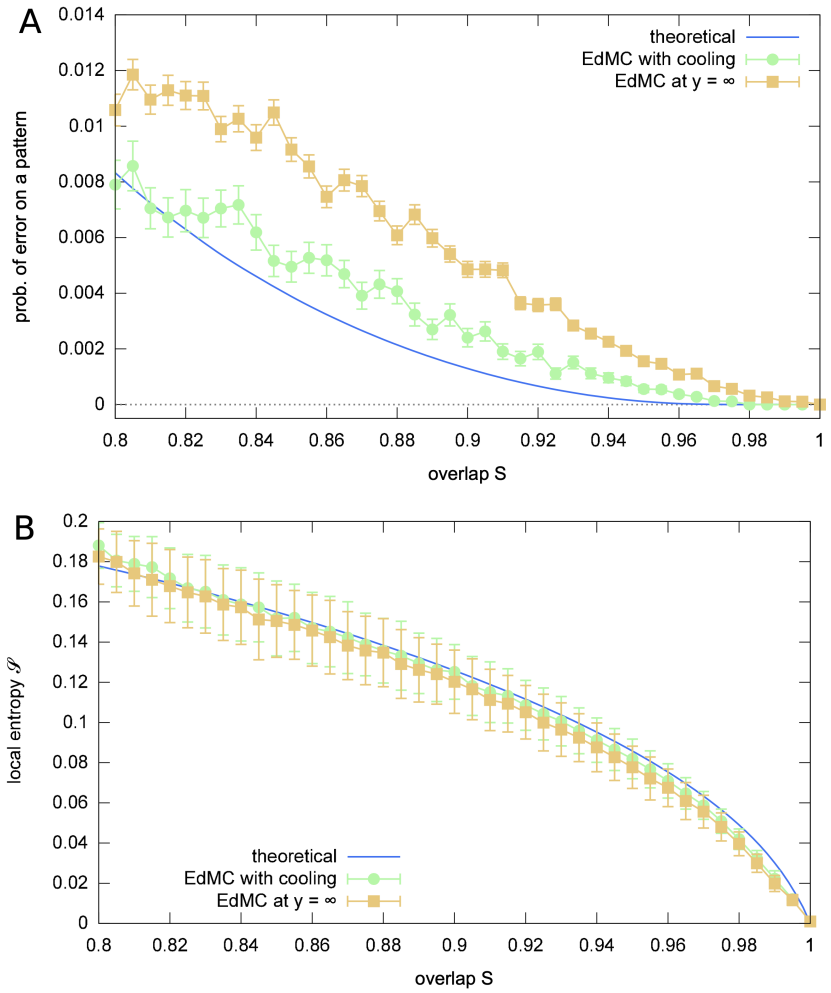


Figure 2.6: High-local-entropy solutions calculated without the constraint that the center is a zero-loss solution still have better generalization properties. Panel A shows the train error as a function of the overlap with the center. Panel B shows the corresponding local entropy. Image from [17].

2.3.2 Algorithm: Entropy-driven Monte Carlo

The idea of this algorithm is simply to perform a Markov-Chain Monte Carlo procedure with standard Metropolis-Hastings accept rule, where we use local entropy as objective function instead of the original loss function. This algorithm is called Entropy-driven Monte Carlo (EdMC).

The main technical difficulty is the estimation of local entropy of a solution at each step. The most efficient way to do so is to run a belief propagation (BP) algorithm on the model with an external field proportional to said solution.

The reason for this BP approach is that we can write a variation of the distance term

in equation 2.15 using the overlap between $\tilde{W}$ and W instead. Then if we use a Bethe approximation on the definition of local entropy we obtain modified BP equation that consist in standard equations with an additional external field.

Once we have a way of estimating local entropy, the algorithm proceeds in the standard way proposing moves to change the configurations and accepting them with a probability that depends on the difference of local entropy via the Metropolis-Hastings rule.

For a deeper description of this algorithm and of belief propagation equations, see [17].

2.3.3 Algorithm: Entropy SGD

Since we can use local entropy as an objective function, if our model has continuous parameters then we can design a gradient-based optimization of it.

We are interested the gradient of local entropy with respect to the model parameters $\tilde{W}$. Let's write local free entropy using a variation of equation 2.15 where we use a energy (loss) function $E(W;\xi)$ instead of the characteristic function $\mathcal{X}_\xi(W)$

$$\mathcal{F}(\tilde{W};\beta,\gamma) = \log \int_W e^{-\beta E(W;\xi)-\gamma(W-\tilde{W})^2} dW \tag{2.19}$$

If we compute the gradient of local entropy with respect to $\tilde{W}$ we obtain

$$\nabla_{\tilde{W}} \mathcal{F}(\tilde{W};\beta,\gamma) = \gamma(\tilde{W} - \langle W \rangle_{\beta,\gamma}) \tag{2.20}$$

where the average is computed on the distribution $P(W;\beta,\gamma,\tilde{W})$ induce by local entropy:

$$P(W;\beta,\gamma,\tilde{W}) \propto e^{-\beta E(W;\xi)-\gamma(W-\tilde{W})^2} \tag{2.21}$$

The expectation in equation 2.20 is hard to compute, therefore we resort to an estimation via Langevin dynamics, which is a MCMC algorithm for drawing samples from a Bayesian posterior distribution. The strategy consists in adding Gaussian noise into maximum-a-posteriori (MAP) updates to obtain a stochastic process that converges to distribution $P(W;\beta,\gamma,\tilde{W})$ so that we can use samples from the trajectory to compute the average numerically.

The peculiarities of this algorithm are two. First, it has been proven that we can just choose a random subset of the dataset to compute the energy $E(W;\xi)$: this is crucial to scale well on big dataset, because computing the gradient with respect to millions of inputs at each step is unfeasible. Second, it has been proven that this stochastic process does not require a Metropolis-Hastings rule in a certain limit of the MAP-update equations, which is good because if not we would have needed to compute the full energy at each step.

The minimization of the negative local entropy via Langevin dynamics that we described here is called Entropy-SGD. For a detailed description and a discussion of the approximations see [1].

2.4 Replicating models

In this section we introduce an approach to optimize local entropy alternative to the ones described above, that rely on a difficult estimation of Gibb-like integrals. The advantages of this new approach will be its simplicity and generality, that allow for a very broad application. The concepts reviewed in this section have been introduced fist in [13].

Let's write the biased measure 2.9 with all the modifications introduced in the last sections: first we drop the constraint on $\tilde{W}$, then we use the soft constraint on the distance in equation 2.15 with generic distance $d(\cdot,\cdot)$, then we consider the case $\beta > 0$ and we use an energy function $E(\tilde{W})$ instead of the characteristic function $\mathcal{X}_{\xi}$. The resulting measure reads

$$\begin{aligned} P(\tilde{W}; y, \beta, \gamma) &= \frac{1}{Z(y,\beta,\gamma)} e^{y\mathscr{E}(\tilde{W})} \\ &= \frac{1}{Z(y,\beta,\gamma)} e^{y \log \sum_W e^{-\beta E(W) - \gamma d(W,\tilde{W})^2}} \\ &= \frac{1}{Z(y,\beta,\gamma)} \Big(\sum_W e^{-\beta E(W) - \gamma d(W,\tilde{W})^2}\Big)^y, \end{aligned} \tag{2.22}$$

where $Z(y,\beta,\gamma)$ is the normalization constant.

The key observation of this section is that, if we imagine y as an integer, $Z(y,\beta,\gamma)$ appears as the partition function of an expanded model consisting in y replicas $\{W^a\}$ of the original model coupled to a "center" $\tilde{W}$:

$$\begin{aligned} Z(y,\beta,\gamma) &= \sum_{\tilde{W}} \Big(\sum_W e^{-\beta E(W) - \gamma d(W,\tilde{W})}\Big)^y \\ &= \sum_{\tilde{W}} \sum_{\{W^a\}} e^{-\beta \sum_{a=1}^y E(W^a) - \gamma \sum_{a=1}^y d(W,\tilde{W})}. \end{aligned} \tag{2.23}$$

This fact allows for a much simpler and more general approach to design algorithms that look for high local entropy solutions: we just choose an algorithm for the original model apply it to replicated model. We will see in the rest of this thesis that the exact scheme of coupling replicas is unimportant, as well as other details such as adding back an energy term on the center $\tilde{W}$.

In the following subsections we introduce the two main algorithms that are employed throughout this thesis: *Replicated SGD*, that will be used on neural networks in chapters 3 and 4, and *Replicated Monte Carlo*, that will be used on lattice proteins in chapter 5.

2.4.1 Algorithm: Replicated SGD

Given a generic neural network with weights W and loss function $L(W;\{\xi^\mu\}) = \sum_{\mu=1}^{M} l(W;\xi^\mu)$, where $\{\xi^\mu\}_{\mu=1}^{M}$ are the data points, the Replicated-SGD algorithm consists in minimizing the following replicated loss function:

$$L_{\text{rep}}(\tilde{W},\{W^a\};\{\xi^\mu\}) := \sum_{a=1}^{y}\sum_{\mu=1}^{M} l(\{W^a\};\xi^\mu) + \gamma \sum_{a=1}^{y} d(W^a,\tilde{W})^2 \tag{2.24}$$

where $\tilde{W}$ is the central replica and $d(W^a,\tilde{W})$ is a generic distance. The precise definition of d usually does not matter; the most common choice is the mean square distance. Note that $\tilde{W}$ is a parameter of the model and must be updated with its own gradient; since we do not have a loss term $L(\tilde{W};\{\xi_\mu\})$, the only force moving the center will be due to coupling with replicas only. At the end of the training we will use the center as our model and predictor. We expect the center to lay in the very middle of the cluster of solutions while the replicas stop on the border, and we know that this property is connected with better generalization properties (see [7, 6]). Different values of γ will change the final properties of $\tilde{W}$; usually a non-negligible value is required to have some effect, but not too big in order to avoid numerical problems. An annealing procedure for γ can be useful if the loss landscape appears to be particularly rough, but is not mandatory at this stage.

As we did for standard SGD, the replicated loss can be optimized in mini batches, namely $\mu \in [1, B]$ with $B < M$, so that the computation of the gradient is less heavy at the cost of introducing noise.

A different coupling scheme consists in dropping the center replica and coupling every replica with each other. The resulting loss function reads:

$$L_{\text{rep}}(\{W^a\};\{\xi^\mu\}) := \sum_{a=1}^{y}\sum_{\mu=1}^{M} l(\{W^a\};\xi^\mu) + \frac{\gamma}{2}\sum_{a,b=1}^{y} d(W^a,W^b)^2 \tag{2.25}$$

Until now we do not have a center to use as a predictor; one way to go is to do the annealing procedure of γ carefully enough to make the replicas collapse onto each other, so that they are a single solution of the original problem. Another option, if we want to avoid the γ-annealing procedure but we still want to keep this second coupling scheme, is to build a center replica on the fly as the barycenter of the replicas whenever we need to make a prediction. This happens even during the training, for monitoring purposes.

Apart from these general characteristics, each model will have specific design choices. We will describe those in the corresponding chapters.

2.4.2 Algorithm: Replicated Monte Carlo

In case we have model with discrete degrees of freedom Γ, such as the lattice polymers that we will use in section 5.2, a gradient-based optimization is not feasible. In this case we resort to optimizing a replicated energy function $E_{\text{rep}}(\Gamma_c, \{\Gamma^a\})$ via Monte Carlo with Metropolis-Hastings rule. The replicated energy function reads

$$E_{\text{rep}}(\Gamma_c, \{\Gamma^a\}) = E(\Gamma_c) + \sum_{a=1}^{y} E(\Gamma^a) + \gamma \sum_{a=1}^{y} d(\Gamma^a, \Gamma_c)^2 \tag{2.26}$$

where this time we include the energy of the center replica $E(\Gamma_c)$. The reason for including this therm is that in section 5.2 we will optimize models that have hard constraint, which must be satisfied in order to have an acceptable configuration (in particular, the hard constraints will be the condition that monomers must not occupy the same site of the lattice, this constraints modeling steric repulsion in polymers). For this reason we cannot define the center simply as the barycenter of the replicas, because there is no guarantee that it will be an acceptable configuration satisfying the hard constraints. Additionally, depending on the model and the choice of the distance, the definition of a barycenter itself may make little sense: it would require some rounding of discrete coordinates that in turn can lead to inconsistent distances between monomers. This would be acceptable if we modeled inter-monomer bonds with harmonic potentials, not in our case of a lattice model where the distance is fixed by the lattice step.

Given this ambiguity in the definition of a barycenter, one could drop the central replica and try to make replicas collapse into a single configuration by performing an annealing of γ. This strategy proved extremely difficult and inconvenient: when γ becomes high and the inter-replica distance low, most of the moves are rejected and the sampling freezes, especially if we are also have high values of β.

The solution is to consider all the replicas connected to a central one that is still endowed with its energy function, so that it is guaranteed to be an acceptable configuration. The role of center is granted through the coupling topology, so we still expect the center to have different physical properties than replicas. Another good reason to choose this coupling scheme is that it does not require an annealing of γ. This is handy when we are interested in studying the phase diagram of the replicate model, because we can simply do simulations at constant values of γ and compute thermal averages at different values of γ and β.

In section 5.2 we describe in a more detailed way the replicated Monte Carlo algorithm that we used to sample lattice polymers from a local-entropy distribution.

2.5 Elements of deep learning

An important observation is that, in deep learning, we don't have any of the certain convergence problems (some of which are of numerical nature, such as vanishing gradients) the landscape is rather smooth and algorithms often find a good solution. One way of seeing this property is that deep architectures are the results of heuristic intuitions that have accumulated in the last decade of software engineering, so that we don't know how they affect the structure of the solution space. An hypothesis is that these heuristics (such as SGD, ReLUs, regularizers, dropout, skip connections, Cross Entropy Loss) could be biasing the learning towards accessible regions, thus removing the expected computational hardness. Since local entropy has been found to be a useful concept for describing why the algorithms are not affected to the computational hardness, we are tempted to ask two questions:

- Is the design of deep networks implicitly optimizing local entropy?
- Can we improve results even further by explicitly optimizing local entropy?

In order to answer these questions we quickly review here the most recent results on the theory of local entropy, which consider typical elements of deep architectures and study their effect on the presence of wide flat minima of the loss landscape.

Cross entropy loss A first study can be done by changing the loss function: in [4] the authors study a cross-entropy loss instead of a MSE loss for a perceptron on a binary classification. The cross entropy (CE) loss for a binary classification problem is simply

$$L_{\mathrm{CE}}(w) = -\sum_{\mu=1}^{p} \log p(y^\mu(w) = \sigma^\mu) \tag{2.27}$$

where $p(y^\mu = \sigma^\mu)$ is the probability that the output y^μ the model is equal to the correct label σ^μ. This probability for a perceptron reads

$$p(y^\mu = \sigma^\mu) = \frac{e^{\gamma\sigma^\mu \frac{\sum_i w_i \xi_i^\mu}{\sqrt{N}}}}{e^{\gamma\sigma^\mu \frac{\sum_i w_i \xi_i^\mu}{\sqrt{N}}} + e^{-\gamma\sigma^\mu \frac{\sum_i w_i \xi_i^\mu}{\sqrt{N}}}} \tag{2.28}$$

Where we introduced the parameter γ that functions as an inverse temperature and can also be seen as the order of magnitude of the weights if we consider a spherical perceptron. By plugging this expression in equation 2.27 we obtain:

$$L_{\mathrm{CE}} = -\sum_{\mu=1}^{p} f_\gamma\left(\sigma^\mu \frac{\sum_i w_i \xi_i^\mu}{\sqrt{N}}\right) \tag{2.29}$$

where we defined the function

$$f_\gamma(x) := -\frac{x}{2} + \frac{2}{\gamma}\log(2\cosh(\gamma x)) \tag{2.30}$$

Now we can ask how similar are the minima of the loss function 2.29 to wide flat minima that are optimal states of local entropy, and how different they are form the sharp configurations that are typical of MSE loss. This questions can be answered by computing the Franz-Parisi potential to obtain the local entropy of solutions that optimize the CE. It turns out that there is a range of values of γ — neither too-small nor too-large values — for which the minima of the CE are indeed large and exponentially dense. Algorithmically the best procedure appears to be an annealing of γ, although this algorithm is slower and less effective than the direct optimization of local entropy.

Committee machine A further step is to consider a deeper architecture that is still analytically tractable, namely the committee machine (both real-valued and binary). The committee machine consists in K perceptrons with disjunct inputs, whose outputs are averaged to compute the output of the model. This model can be seen as a two-layer neural network where we don't learn the upper layer but instead we just set its weights to $+1$. A scheme of this architecture is shown in figure 2.7. In [4] the authors also show

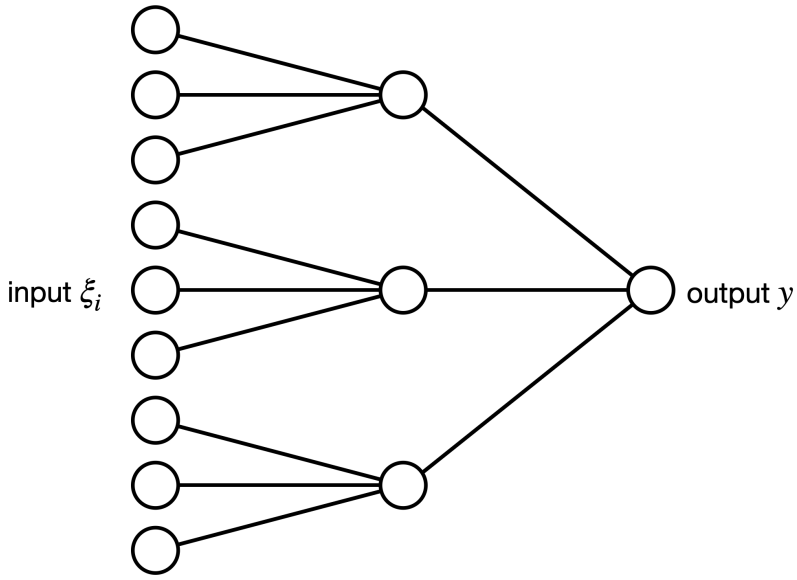


Figure 2.7: A scheme of a committee machine.

that subdominant regions with high local entropy, namely wide flat minima, still exist in a committee machine storing random patterns, suggesting that the theory of local entropy that we discussed in this section might be relevant even for deep networks.

ReLU activation function A two-layer network allows to study the effect of the choice of the activation function on the loss landscape. In [18] the authors compute the critical capacity α_c of a tree-like committee machine with ReLU activation functions and show that below α_c wide flat minima are still present. In particular, the use of ReLUs improves the robustness of the solutions respect to threshold units, measured with respect to perturbations the model weights (either binary or real-valued) or the input channels.

Learning with margin A different take on local entropy and flat regions is discussed in [7], where the authors show that, in a binary classification problem, high local entropy regions can be accessed also by imposing a margin to the classifier. This technique consists in taking the loss function 1.12 and adding an offset term k, called *margin*:

$$L(W;\{\xi^\mu\}) = \sum_\mu \Theta(-[\sigma^\mu y(W,\xi^\mu) - k]) \tag{2.31}$$

In this way we are not simply requiring the model to classify correctly the examples, by we also require that the decision boundary has a distance at least k from any data point (see figure 2.8 for a sketch of this situation). The authors show that increasing

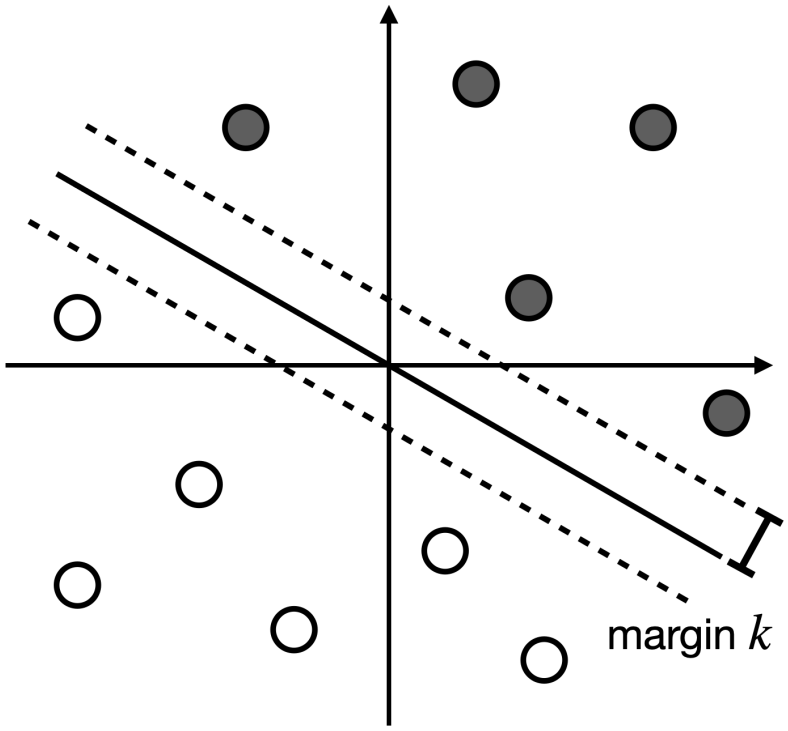


Figure 2.8: A scheme of learning a decision boundary with a margin requirement.

the margin, the distance between typical solution decreases, signaling that the solutions start clustering together. Additionally, when they study the neighborhood of a solution with margin $\tilde{k} > 0$, they find a dense cluster of solution with margin $k < \tilde{k}$. The classification problem becomes impossible when $\tilde{k}$ becomes too high; a classification with the highest possible value corresponds to optimizing the local entropy. It is interesting to note optimizing local entropy by imposing a margin makes sense only for shallow networks such as the perceptron or the committee machine, since it affects only the last layer of parameters. Imposing a margin to internal neurons of a deep network seems unfeasible, therefore the explicit optimization of local entropy appears to be a more general approach, suitable for state-of-the art architectures.

Overparametrization A noteworthy property of deep network is that they are able to fit the trainset almost perfectly while mantaining good generalization performances. This property is counterintuitive given what we discussed in section 1.4: in statistical inference fully optimizing a loss function leads to overfitting, while, on the other hand, regularization strategies improve generalization but prevent the full optimization of the train loss; on the contrary, deep architectures appear to have enough parameters to do well both on the trainset and the testset. The role of the number of parameters of a neural network and its connection with local entropy has been studied in [6]. The authors analytically study an overparametrized binary perceptron (also called *random feature* model), which consists in a two-layer network where weights in the first layer are fixed and random rather than learned. This model can also be seen as a standard perceptron learning the following dataset:

$$\tilde{\xi}_i = \mathrm{sgn}\left(\frac{1}{\sqrt{D}} \sum_{k=1}^{D} F_{ki} \xi_k \right) \tag{2.32}$$

where F_{ki} is a $D \times N$ matrix whose element are sampled from a Gaussian distribution with zero mean and unit variance. The output of the perceptron reads

$$y = \mathrm{sgn}\left(\frac{1}{\sqrt{N}} \sum_{i=1}^{N} W_i \tilde{\xi}_k \right) \tag{2.33}$$

A scheme of this architecture can be found in figure 2.9. The correct labels σ^μ are assigned with a D-dimensional teacher network with random binary weights W_k^T:

$$\sigma^\mu = \mathrm{sgn}\left(\frac{1}{\sqrt{D}} \sum_{k=1}^{D} W_k^T \xi_k \right) \tag{2.34}$$

Given the number of examples p, the authors compute the phase diagram of this object as a function of the order parameters $\alpha = p/N$ and $\alpha_T = p/D$. They find that the between the SAT and UNSAT phases that we encounter by increasing α there is another phase

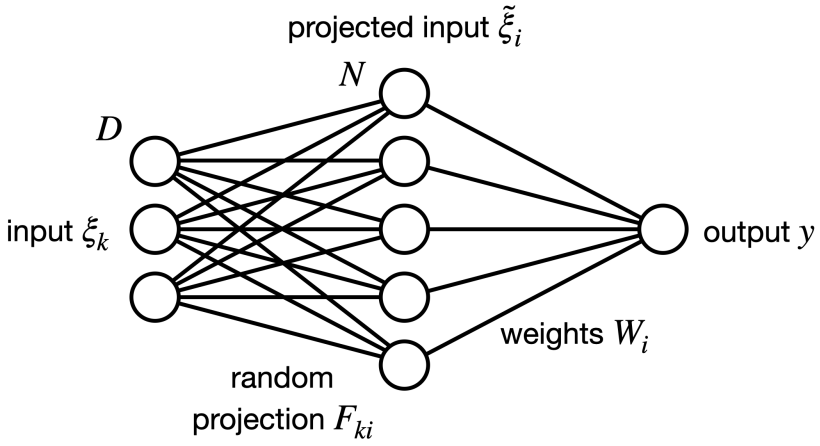


Figure 2.9: A scheme of an overparametrized perceptron.

transition, corresponding to the algorithm threshold at which algorithms stop finding solutions (called local entropy transition). Interestingly, if we start from the UNSAT phase and we increase α_T while keeping α fixed, we first encounter the LE transition line and then the SAT line. This means that by increasing the degree of overparametrization N/D we create wide flat minima in the loss function. This fact might be the explanation of the counterintuitive behavior of deep networks.

Deep architectures Finally, we see that optimizing local entropy improves performance even in state-of-the-art deep networks. In [1, 2] the authors show that implemented algorithms that explicitly optimize local entropy for deep architectures (Entropy-SGD and Replicated-SGD) improves the convergence time and the generalization error. Interestingly, a connection between different measures of flatness and generalization properties has an increasingly important focus in the recent literature (for some review on this point see [2, 19, 20, 21]), showing that the flatness of a minimum of the train loss is the best predictor for its generalization on common datasets, supporting that the theory local entropy might describe well not just shallow networks but also deep architectures.

Part II

Novel applications of Local Entropy

Chapter 3

The Gaussian Mixture Problem

The content of this chapter follow the content of my paper at ref. [5].

The first model that we study to get progressively away from the classical teacher-student scenario has two main differences: first, the problems is convex, and second, the train loss of the teacher is not zero this time, which is an interesting setting to explicitly study the effects of optimizing the train loss too much.

We discuss the connection between local entropy, flatness and generalization in a very basic model of high-dimensional statistical machine learning [22, 23, 24, 25]: Gaussian mixtures. The generative model is defined as follows. For a given problem size N, an N-dimensional vector $\mathbf{v}^\star$ is randomly generated from a standard multivariate normal $\mathcal{N}\left(\mathbf{0}, \mathrm{I}_N\right)$. Then we generate a label $\sigma = 1$ or $\sigma = -1$ with probability ρ and $1-\rho$, respectively, and we generate a pattern $\boldsymbol{\xi}$ according to the value of the label σ as $\mathcal{N}\left(\sigma \mathbf{v}^\star/\sqrt{N}, \Delta\, \mathrm{I}_N\right)$. In this way we construct a training set with $P \equiv \alpha N$ such patterns; the coordinate $i \in \{1, ..., N\}$ of pattern μ is therefore given by

$$\xi_i^\mu = \frac{v_i^\star}{\sqrt{N}}\sigma^\mu + \sqrt{\Delta}\, z_i^\mu \tag{3.1}$$

where z_i^μ are i.i.d Gaussian random variables with zero mean and unit variance. This results in two potentially overlapping clusters, with the label indicating the cluster a pattern belongs to and where Δ controls their width. We will refer to the problem with $\rho = 0.5$ as the *balanced* case; we call all other cases *unbalanced*.

As usual in statistical physics, we will consider the high-dimensional limit, where both $N \to \infty$ and $P \to \infty$ with the ratio $\alpha = \frac{P}{N}$ fixed.

In this chapter we analyze the performance of a threshold-linear classifier (a single-unit neural network). This machine is parametrized by a vector of weights $\boldsymbol{w}$ of length N and a bias b, but when studying the balanced case we always simply set $b = 0$. The machine predicts the label of a pattern $\boldsymbol{\xi}$ as:

$$\hat{\sigma}\left(\boldsymbol{\xi}; \boldsymbol{w}, b\right) = \mathrm{sign}\left(\frac{1}{\sqrt{N}}\sum_{i=1}^{N} w_i\, \xi_i + b\right) \tag{3.2}$$

We can observe that this classifier is invariant to an overall rescaling of the parameters $\boldsymbol{w}' = \kappa \boldsymbol{w}$ and $b' = \kappa b$ for any $\kappa > 0$. It is natural to identify the irrelevant degree of freedom in the parametrization with the norm of $\boldsymbol{w}$.

The most elementary metric by which to measure the performance of this classifier on the training set is the number of errors, which can be expressed as

$$\mathscr{L}_{\text{err}}(\boldsymbol{w}, b) = \sum_{\mu=1}^{P} \Theta\left[-\sigma^{\mu}\ \hat{\sigma}(\boldsymbol{\xi}^{\mu}; \boldsymbol{w}, b)\right] \tag{3.3}$$

where $\Theta(\cdot)$ is the Heaviside step function, that is $\Theta(x) = 1$ if $x \geq 0$ and 0 otherwise. This error-counting loss obviously inherits the scale invariance, but it has the drawback that it cannot be used with the gradient-based methods usually employed in training large neural networks (which is the situation about which we hope to gain the most insight from this simple model). It is therefore of interest to consider a generalized overall loss function form:

$$\mathscr{L}(\boldsymbol{w}, b) = \sum_{\mu=1}^{P} \ell\left[\sigma^{\mu}\left(\frac{1}{\sqrt{N}} \sum_{i=1}^{N} w_i\, \xi_i^{\mu} + b\right)\right] \tag{3.4}$$

where $\ell(\cdot)$ is a generic single-pattern loss function. The error-counting case corresponds to $\ell(x) = \Theta(-x)$. In what follows we analyze the mean squared error (MSE) loss $\ell(x) = \frac{1}{2}(x-1)^2$, which is a well-studied differentiable loss. As for other choices of differentiable losses (e.g. cross-entropy, hinge, etc.) the scale-invariance property is lost, and the role of norm regularization may become important.

It's important to observe that this model possesses some rather peculiar features if compared to typical classification tasks performed with neural networks, namely the training loss is convex. Indeed, Bayes-optimal performance can be achieved with a single configuration of the model parameters (instead of requiring a distribution) which can be derived analytically. Additionally, due to the overlap between the Gaussian distributions which are used to generate the data, no classifier can achieve zero test error (in the teacher-student context this would be somewhat similar to the case of having a "noisy", unreliable teacher). Therefore, care must be used when considering how the results may generalize to non-convex scenarios.

Recently, this model has been studied in [26] by using Gordon's inequality. The authors showed that the MSE loss is severely prone to overfitting, especially when $\alpha \simeq 1$[1]. They also showed however that, in spite of the fact that the output of the model is norm-independent, the generalization performance is considerably improved by adding to the loss an L_2 regularization term on the weights, $\lambda \|\boldsymbol{w}\|^2$. For large N, the parameter

[1]This value corresponds to the transition point above which the MSE loss has a unique minimum, since minimizing the MSE entails solving a system of P equations in N unknowns; in the balanced case, it is also the transition point where the data is no longer linearly separable.

$\lambda > 0$ is a Lagrange multiplier that implicitly fixes the norm of the weights. The optimal choice of λ depends in general on α and ρ. For the balanced case, $\rho = 0.5$, the optimum is obtained for all α in the limit $\lambda \to \infty$ (corresponding to vanishing values for the norm of the weights), and in that case the network reaches the Bayes-optimal generalization bound.

In light of these findings, it is interesting to further discuss the role of the norm for these class of models. As we remarked above, the output of the network (and thus the generalization error) has a scale invariance, i.e. it is independent of the norm (as long as the bias is also properly rescaled). As a consequence we need to understand which are the geometrical features of the solutions space of the classifier that are induced by the regularization of the surrogate loss function used for gradient learning.

It is worth noticing that this scenario also applies to most deep neural network models that use ReLU activations in the intermediate layers and an argmax operation to produce the output label, and are therefore invariant to uniform scaling of all their weights and biases. Since the norm cannot affect the generalization capabilities of the network, it seems unlikely that a norm-based regularization could be a valid general strategy.[2]

In this chapter, we argue for a different, more general criterion to avoid overfitting and improve generalization, proposed in several recent works [17, 4, 18], namely that of maximizing the local entropy, which is a particular measure of flatness that can be analyzed theoretically and efficiently approximated algorithmically. We refer to gradient-based algorithms that operate by maximizing (an approximation to) the local entropy as "entropic algorithms". One example is given by the replicated stochastic gradient descent (rSGD) algorithm introduced in [13], where the local entropy is targeted by using several replicas moving in the loss landscape and at the same time feeling an attraction during their dynamics. Another algorithm is entropy-SGD (eSGD) [27], where the local entropy is estimated using stochastic gradient Langevin dynamics [28]. Those algorithms have been applied to state-of-the-art deep neural networks [2], proving that they can achieve improved generalization performances.

In particular, for the balanced case, we show analytically that the minimum norm condition, which results in Bayes-optimal performance, corresponds to solutions of maximum local entropy for the classifier (which is norm invariant). We also show that these solutions can be found by entropic algorithms acting on the MSE loss function, and that these algorithms are much less sensitive to the norm.

For the unbalanced case, the authors of [26] found that, when the bias is learned, reaching the Bayes-optimal generalization error with L_2 regularization alone is impossible, and that there exists an optimal finite value of λ that minimizes the generalization

[2] There is a caveat to this statement: for particular choices of the loss, e.g. cross-entropy, it is possible to reparametrize the problem in an invariant way and interpret the norm in terms of a time-evolving parameter of the loss with a "focusing" role, see [4].

error. In this chapter we show however that there exists a choice of the bias and of λ that allows to reach the Bayes-optimal performance, and that such parameters also have a higher local entropy (measured again in a scale-invariant way). We show both analytically and numerically how to systematically improve the generalization performance in this setting. Learning the bias with entropic algorithms leads to improved performance compared to those which can be attained by the L_2 regularized loss function.

The rest of the chapter is organized as follows. In section 3.1 we briefly review the typical scenario obtained by performing a standard replica-symmetric (RS) replica calculation over the Gibbs measure. We discuss in particular how the choice of the bias is decisive for generalization performances in the unbalanced case. In section 3.2 we explore the local entropy landscape around the Bayes optimal configuration for the MSE loss function, by performing a calculation á la Franz-Parisi [29]. In section 3.3 we discuss how targeting the local entropy loss we can improve generalization. Finally, section 3.5 contains some conclusions.

3.1 Bayes-optimal configurations

The partition function of the Gaussian mixture model, with a regularization over the weights of the linear classifier can be written as

$$Z = \int \prod_i dw_i \, e^{-\beta \mathscr{L}(\boldsymbol{w},b) - \frac{\lambda}{2}\sum_i w_i^2} \tag{3.5}$$

where β is the inverse temperature. Notice that in our treatment the bias has been fixed as an external parameter. To study the case in which the bias is a learned parameter we would add another integral over b in the definition of Z. In the following we will denote the average over the training set with angle brackets: $\langle \cdot \rangle \equiv \prod_{\mu=1}^{P} \mathbb{E}_{\boldsymbol{v}^\star,\sigma^\mu} \mathbb{E}_{\boldsymbol{\xi}^\mu|\boldsymbol{v}^\star,\sigma^\mu} [\,\cdot\,]$.

The typical properties of the model are derived by computing the average log-volume $\langle \ln Z \rangle / N$, which is the (typical) free entropy of the model $-\beta f$, where f is the corresponding free energy. The free entropy can be computed in the large-N limit using the "replica trick":

$$\ln Z = \lim_{n \to 0} \partial_n Z^n \,. \tag{3.6}$$

The whole computation in the large N and β limit using an RS ansatz is reported in [5]. Here we discuss the results. In figure 3.1 we show the generalization error found by optimizing the regularized MSE loss function, in the balanced case and one unbalanced case, as a function of the bias and the squared norm (which is implicitly but monotonically controlled by λ). We also show with a black dashed line the value that the bias takes when it is learned, for any given value of the squared norm. In both the balanced and unbalanced cases there exist choices of the bias and the squared norm that achieve the Bayes optimal performance. However, an important difference can be noted: if we learn the bias in the balanced case we always find $b = 0$ for every value of the squared

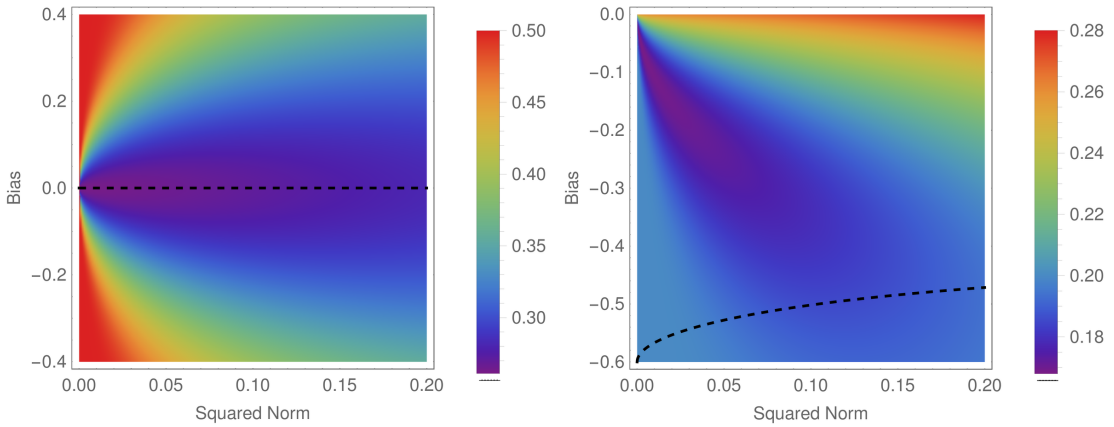


Figure 3.1: Generalization error found by optimizing the regularized MSE loss, as a function of the bias and squared norm. The black dashed line represents the value of the bias obtained by maximizing the Gardner volume. Both plots are for $\alpha = 0.7$ and $\Delta = 1$. Left: Balanced case ($\rho = 0.5$). The Bayes optimal generalization error in this case is $\epsilon_g \simeq 0.2605\ldots$ computed using its analytical expression, found in [5]. Right: Unbalanced case, with $\rho = 0.2$. The Bayes optimal generalization error in this case is $\epsilon_g \simeq 0.1679\ldots$.

norm; sending $\lambda \to \infty$ (and therefore the squared norm to zero) one recovers the Bayes optimal performance. This is not true in the unbalanced case: fixing λ and learning the bias never gives the optimal performance.

3.2 The local entropy is larger in the vicinity of the Bayes-optimal configuration

In order to quantify the local geometrical landscape around a typical configuration $\tilde{\boldsymbol{w}}$ of the Gibbs measure with loss function $\mathscr{L}_r = \sum_\mu \ell_r$, regularization parameter λ_r and inverse temperature β_r, we have studied the so-called Franz-Parisi free entropy [29, 15]. It is defined as

$$-\beta f_{\mathrm{FP}}(S) \equiv \frac{1}{N}\left\langle \frac{\int \prod_i d\tilde{w}_i\, e^{-\beta_r \mathscr{L}_r(\tilde{\boldsymbol{w}},\tilde{b}) - \frac{\lambda_r}{2}\sum_i \tilde{w}_i^2} \ln \mathscr{V}(\tilde{\boldsymbol{w}}, S)}{\int \prod_i d\tilde{w}_i\, e^{-\beta_r \mathscr{L}_r(\tilde{\boldsymbol{w}},\tilde{b}) - \frac{\lambda_r}{2}\sum_i \tilde{w}_i^2}} \right\rangle \tag{3.7}$$

where the quantity

$$\mathscr{V}(\tilde{\boldsymbol{w}}, S) \equiv \int d\mu_P(\boldsymbol{w})\, e^{-\beta \mathscr{L}(\boldsymbol{w},\tilde{b})}\, \delta\left(\sum_i w_i \tilde{w}_i - NS\right) \tag{3.8}$$

is the volume of configurations $\boldsymbol{w}$ at inverse temperature β that have overlap S with the reference configuration $\tilde{\boldsymbol{w}}$. The measure $d\mu_P(\boldsymbol{w})$ is a flat measure over a hyper-sphere

of squared radius P, i.e. the weights $\boldsymbol{w}$ have square norm $\frac{1}{N}\sum_i w_i^2 = P$. The overlap P is chosen to match the squared norm of the reference $\tilde{\boldsymbol{w}}$, that is $P = Q$. Note that Q is fixed via a soft constraint by the regularization parameter λ_r; in addition we have chosen, for simplicity, the bias of the constrained configuration $\boldsymbol{w}$ to match the one of the reference.

Notice that in equation (3.7) and (3.8) we use different losses $\mathscr{L}_r$ and $\mathscr{L}$ (and different parameters too): the landscape of which we explore the geometrical features can differ from the landscape from which we get the reference configuration.

The computation of equation (3.7) is long and involved; here we just sketch the main steps, referring to [5] for the details. The average over the disorder in equation (3.7) can be done by using two replica tricks, one for the denominator and another one for the logarithm in the numerator:

$$\frac{1}{Z} = \lim_{r\to 0} Z^{r-1} \tag{3.9a}$$

$$\ln Z = \lim_{n\to 0} \partial_n Z^n \tag{3.9b}$$

Once the average is performed, one has to introduce several order parameters in order to decouple the expressions over the size of the training set αN and of the dimension N. Using indexes a or b for replicas in $\{1,\dots,r\}$ and $c, d \in \{1,\dots,n\}$ the order parameters are $p^{cd} = \frac{1}{N}\sum_i w_i^c w_i^d$, $t^{ac} = \frac{1}{N}\sum_i \tilde{w}_i^a w_i^c$, $O^c = \frac{1}{N}\sum_i v_i^\star w_i^c$, $P^c = \frac{1}{N}\sum_i (w_i^c)^2$ and the corresponding conjugated ones. Note that P^c is just the squared norm P because of the spherical constraint inside the measure $d\mu_P(\boldsymbol{w})$. Among the conjugated order parameters, we also need to introduce an additional parameter, $\hat{S}^c$, which imposes the hard constraint on the overlap between the reference configuration $\tilde{\boldsymbol{w}}$ and $\boldsymbol{w}$.

Using an RS ansatz over the order parameters and performing the large β_r limit we obtain

$$-\beta f_{\mathrm{FP}}(S) = \mathfrak{G}_S + \alpha \mathfrak{G}_E\,, \tag{3.10}$$

where the definition of the entropic and energetic terms are reported in [5]. When $\alpha = 0$ the Franz-Parisi free entropy can be evaluated analytically

$$-\beta f_{\mathrm{FP}}(S, \alpha = 0) = \frac{1}{2}\left[1 + \ln(2\pi) + \ln\left(\frac{1}{\lambda_r} - \lambda_r S^2\right)\right]\,, \tag{3.11}$$

and it gives the logarithm of the total volume of configurations at overlap S with the reference. For a given loss $\ell(\cdot)$ the local entropy of a given configuration $\tilde{\boldsymbol{w}}$ can be computed by evaluating the local energy $\epsilon_\ell = \frac{\partial(\beta f_{\mathrm{FP}})}{\partial\beta}$ and then using

$$\mathcal{S} = \beta\left(\epsilon_\ell - f_{\mathrm{FP}}\right)\,. \tag{3.12}$$

The normalized local entropy is just the local entropy (3.12) minus the total log-volume at $\alpha = 0$ given in equation (3.11). It is the logarithm of a fraction of a volume, and thus

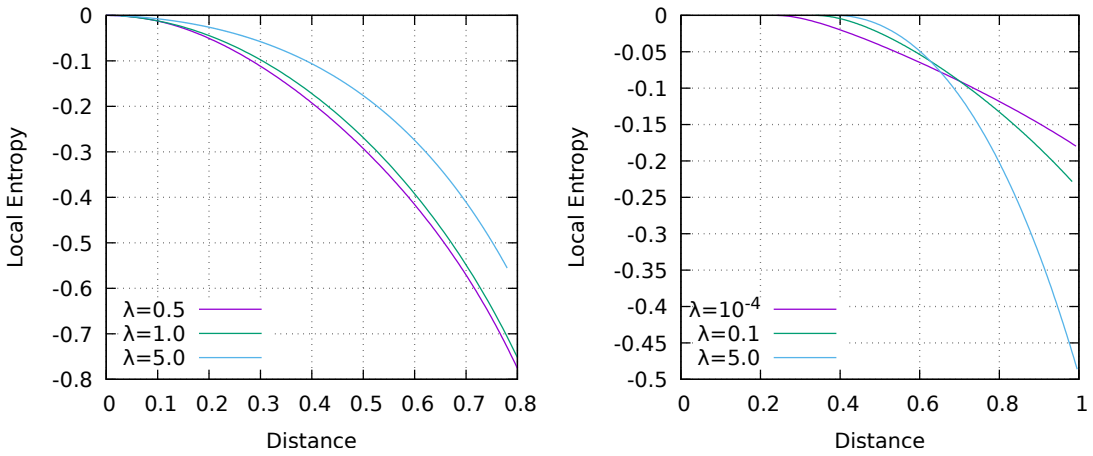


Figure 3.2: Balanced case ($\rho = 0.5$). Normalized local entropy as a function of the distance d computed from reference configurations found by optimizing the regularized MSE loss, with varying regularization strength λ. Larger values of λ correspond to minimizers with better generalization properties. In both figures $\alpha = 0.7$, $\Delta = 1$, $b = 0$. The cutoff $\overline{\epsilon}$ is chosen to be equal to the training error of the reference (left), or is given by the training error of the "oracle" $\boldsymbol{w} = v^\star$ (right panel) as in equation (3.14).

is upper bounded by zero; additionally, defining the distance as

$$d \equiv \frac{1}{2} \frac{\sum_{i=1}^{N} \left(\tilde{w}_i - w_i\right)^2}{\sum_{i=1}^{N} \tilde{w}_i^2} = 1 - \frac{S}{P} \tag{3.13}$$

the normalized local entropy is always zero for $d = 0$. For $\tilde{\boldsymbol{w}}$ located in sharp minima we expect that the normalized local entropy will have a sharp drop near $d \simeq 0$, whereas for flat minima it will be close to zero within some range of small distances.

We have explored the normalized local entropy landscape of the configurations found by optimizing the regularized MSE loss (i.e. $\ell_r(x) = \frac{1}{2}(x-1)^2$), where the *training error* is used in the local entropy definition (i.e. $\ell(x) = \Theta(-x)$). We stress that by using the error instead of the MSE we explore the properties of the model in the regime in which it operates during classification.

On the other hand, the parameter β has been chosen in such a way that the training error of $\boldsymbol{w}$ is equal to a certain cutoff $\overline{\epsilon}$.

We have analyzed two different choices for the energy $\overline{\epsilon}$:

- in the first case $\overline{\epsilon}$ is chosen to be equal to the training error of the reference. This case is depicted in the left panel of figure 3.2 for the balanced case and in the right panel of figure 3.3 for the unbalanced case. See the corresponding captions for details.

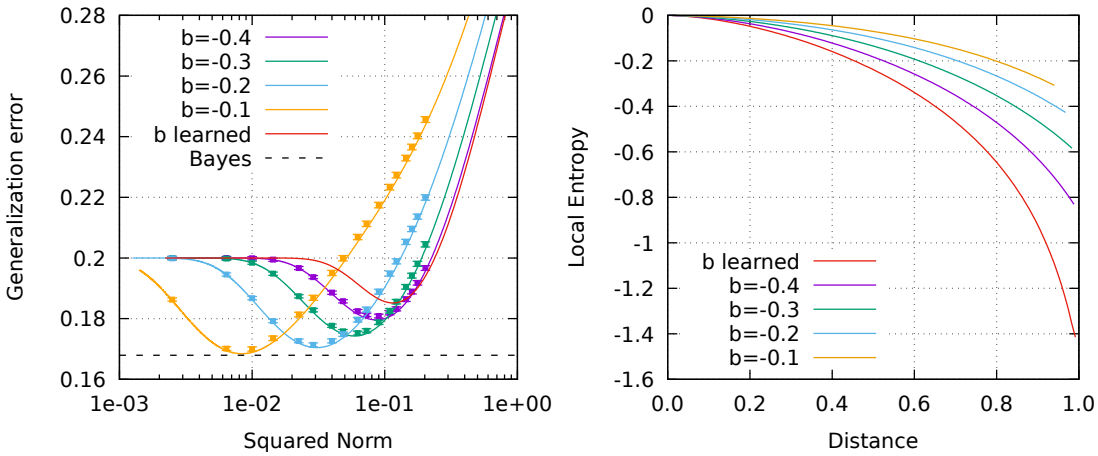


Figure 3.3: Unbalanced case ($\rho = 0.2$). In both plots $\alpha = 0.7$ and $\Delta = 1$. Left: Generalization error for a typical minimizer of the MSE loss as a function of the squared norm, for various choices of the bias b = -0.4, -0.3, -0.2, -0.1. Full lines are analytical results, points and error bars are numerical results obtained with $N = 1000$. On the red curve, instead, the bias is learned (it's the value that maximizes the free entropy) and thus it's different for every value of the squared norm. The dashed curve is the Bayesian generalization error (see [5]). Right: Normalized local entropy as a function of the squared-distance d computed from reference configurations found by optimizing the regularized MSE loss, for various choices of the bias b_r = -0.4, -0.3, -0.2, -0.1 (and $b = b_r$). The corresponding value of the squared norm has been chosen by using the one that minimizes the generalization error for that fixed value of b (see left panel). In the red curve, instead, the bias has been fixed by a saddle point equation (i.e. it is learned). The cutoff $\bar{\epsilon}$ is chosen to be equal to the training error of the reference.

- in the second case, only used for the balanced case, $\bar{\epsilon}$ is chosen as the training error of an "oracle classifier" with $\boldsymbol{w} = v^\star$, which is given by:

$$\epsilon_t^\star = \alpha \int \prod_i dv_i^\star d\xi_i P(\xi_i \mid v_i^\star) P_v(v_i^\star) \, \theta\left(-\frac{1}{\sqrt{N}} \sum_i v_i^\star \xi_i\right) = \alpha H\left(-\frac{1}{\sqrt{\Delta}}\right) \tag{3.14}$$

  This corresponds to the smallest possible test error that any linear classifier machine could achieve, which is non-zero because of the overlap between the two clusters. This case is depicted in the right panel of figure 3.2.

In both cases we clearly see that reference configurations with better generalization properties have higher local entropy curves. We remind that, in the balanced case, the configurations with better generalization properties correspond to larger values of the regularization parameter λ, whereas in the unbalanced case they correspond to

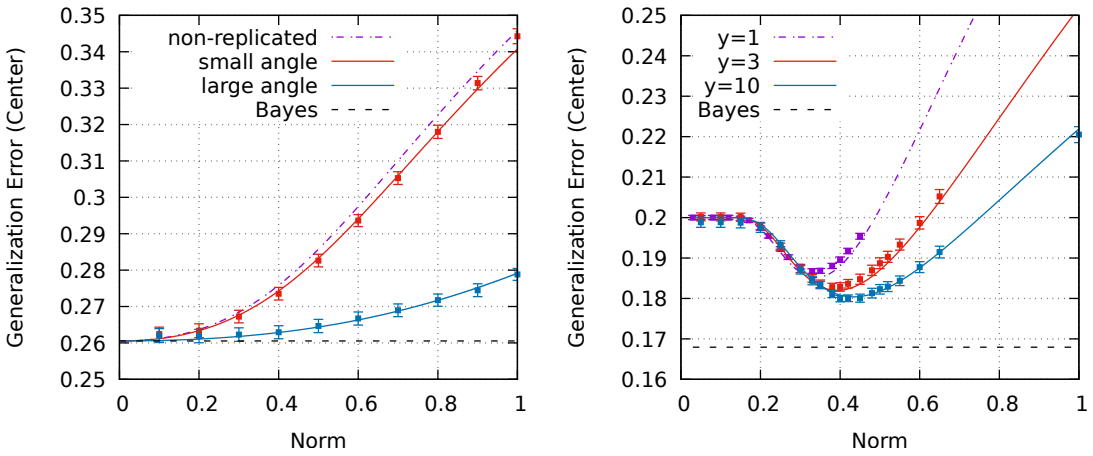


Figure 3.4: Generalization error of the center $\bar{\boldsymbol{w}}$ of a system of y replicas, each optimizing the MSE loss and with a constraint on the angle between the replicas, as a function of the norm n of the replicas. In both figures $\alpha = 0.7$; the Bayes optimal error is plotted with a dashed black line. Left: Balanced case with $y = 10$ replicas. The red curve (small-angle) corresponds to $\cos(\theta) = 0.9$; the large-angle case to $\cos(\theta) = 0.1$. Solid curves are theoretical results, points are numerical results obtained with $N = 1000$, averaged over 30 samples. In the limit $\theta = 0$ the results reproduce those of a single device; increasing θ the dependence on the norm reduces (the curve flattens onto the Bayes-optimal dashed line in the limit $\theta = \pi/2$ and $y \to \infty$). Right: Unbalanced case ($\rho = 0.2$). Solid curves correspond to analytical results, points are numerical results obtained with $N = 1000$ and respectively 100, 30, 20 samples for $y = 1,3,10$. The angle between the y replicas has been fixed to $\theta = \pi/2$.

particular fine-tuned values of the bias b and λ as already evidenced in the right panel of figure 3.1.

3.3 Algorithms that target flatter regions of the MSE landscape also generalize better

The results of the previous section confirm that the local entropy landscape constructed using the training error is a good predictor of generalization performance. However, when dealing with much more complex architectures, using the training error as the loss function in equation (3.12) is not (yet) algorithmically feasible. In particular, the entropic algorithms rSGD and eSGD must still operate on a differentiable loss. This leaves the question whether targeting high-local-entropy regions in a differentiable loss landscape can still lead to good generalization results open. We have investigated this question analytically on the Gaussian mixture model with a linear classifier and the MSE

loss, using the same technique explained in [4, 18], see [5] for details. This amounts to studying the generalization error of the barycenter of a replicated system of y classifiers, each with its own parameters $\boldsymbol{w}^a$ with $a = 1, \dots, y$, each optimizing the MSE under constraints on their norms n and on their mutual angles θ, that is: $\forall a, a' : \|\boldsymbol{w}^a\| = n,\ \boldsymbol{w}^a \cdot \boldsymbol{w}^{a'} = n^2 \cos(\theta)$. The barycenter is defined as $\bar{\boldsymbol{w}} = \frac{1}{y}\sum_a \boldsymbol{w}^a$. In this analysis we used the angle θ rather than the distance in order to compare situations with different norms.[3] Notice also that we have not imposed analytically an analogous constraint over the biases of the replicas. In other words, the bias of every replica is the same and we call it b. In order to set the bias for the barycenter we recall the fact that the error-counting loss is scale invariant, so that the value of the bias is significant only when compared to the norm of the weights. Additionally, we note that in general $\|\boldsymbol{w}^a\| \geq \|\bar{\boldsymbol{w}}\|$, so if we were to naively use b as bias of the barycenter we would change its relative magnitude respect to $\|\bar{\boldsymbol{w}}\|$. For this reason we set $\bar{b} = b\,\|\bar{\boldsymbol{w}}\| / \|\boldsymbol{w}\|$.

Our goal is to check if we can improve the generalization performances. Due to the peculiarities of this model, in the balanced case we can simply check whether the barycenter is aligned with the solution of the norm-regularized model with large λ, which we know to be the optimal classifier.

Some representative results are shown in fig. 3.4. In the left panel we analyze the balanced case. Our results indicate that, with sufficiently many replicas (even just $y = 3$) and with sufficiently large angles the generalization performance is nearly optimal and the dependence on the norm is mild, and much less pronounced than at small angles (the limit of zero angles reproduces the results of the norm-regularized analysis without replicas). The fact that for this model the best results are obtained with widely separated replicas is due to the simple nature of the problem, and we do not expect this phenomenon to just carry over as-is to the case of deep neural networks, where the landscape is non-convex and the structure of the symmetries is generally much more complex (both in terms of the scale invariance and of discrete permutation symmetries).

In the right panel of figure 3.4 we show also what happens in the unbalanced case. We have compared the performance of the typical minimizer of the norm-regularized MSE loss with the one of the replicated system. We show that by increasing the number of replicas keeping fixed the angle between them, the generalization performance is improved. The large y limit can be handled analytically, and it is indistinguishable from the results obtained with $y = 10$ replicas.

The analytical curves describe very well the numerical results that are obtained with rSGD. The algorithm consists in training y replicas of a perceptron with and additional term $\mathscr{L}_{\mathrm{d}}$ in the loss function of each model proportional to the sum of distances from the other replicas; in order to force the replicas to stay at a given distance d_0, we modify

[3] This is equivalent to using the cosine similarity, often employed in machine learning contexts. We should note however that in a multi-layer classifier the structure of the scale invariance is more complicated and the cosine similarity by itself would not be sufficient to account for it.

this term by including d_0 as offset:

$$\mathscr{L}_{\mathrm{d}}^{(a)} = \sum_{b\neq a}^{y} \left(d_{ab} - d_0\right)^2 , \tag{3.15}$$

where the index a refers to the replica of which we are computing the loss.

3.4 Numerical details

We used rSGD for all simulations reported in this work. As described in the main text, the algorithm consists in training y replicas of a perceptron each initialized differently, with an additional term in the loss function of each model proportional to the sum of distances from the other replicas. The total loss function is

$$\begin{aligned} \mathscr{L}(\{\boldsymbol{w}^a, b^a\}_{a=1}^{y}) &= \sum_{a=1}^{y} \left[\mathscr{L}_{\mathrm{MSE}}^{a} + \lambda \mathscr{L}_{\mathrm{d}}^{a}\right] \\ &= \sum_{a=1}^{y} \left[\mathscr{L}_{\mathrm{MSE}}^{a} + \lambda \sum_{a\neq b}^{y} \left(d_{ab} - d_0\right)^2\right] \end{aligned} \tag{3.16}$$

where $\mathscr{L}_{\mathrm{d}}^{a}$ is the term we introduced in order to force the replicas to stay at a given distance d_0. The bias is treated separately: in the cases where $\rho = 0.5$ it is simply set to zero; in the cases where $\rho \neq 0.5$ we add to the loss of each replica $\lambda \sum_{a\neq b}^{y} \left(b_a - b_b\right)^2$. This is done in order to match results with analytical calculations, where the replicas share the same bias.

Then we define a center model as our predictor, defined as the average of the replicas in the following way:

$$\bar{\boldsymbol{w}} = \frac{1}{y} \sum_{a=1}^{y} \boldsymbol{w}^a \tag{3.17a}$$

$$\bar{b} = \|\bar{\boldsymbol{w}}\| \, \frac{1}{y} \sum_{a=1}^{y} \frac{b^a}{\|\boldsymbol{w}^a\|} \tag{3.17b}$$

The perceptron $\left(\bar{\boldsymbol{w}}, \bar{b}\right)$ is the model we use to compute loss and error on both the testset and the trainset. Note that, as discussed in the main text, the bias of the center model is not simply the average of the biases, but rather the average of the biases weighted by the inverse norm of the weights, scaled by the norm of the predictor itself. Note that at the end of the training the replicas are expected to have the same value b of the bias, and since the norm of the replicas is fixed to some given $\|\boldsymbol{w}\|$ the bias of the center will simply be $\bar{b} = b \, \|\bar{\boldsymbol{w}}\| \, / \, \|\boldsymbol{w}\|$.

Most of the following details should not matter for the sake of generalization error because the problem is convex. They still determine the rate of convergence to the analytical solution, so we report them in detail.

The training is performed with PyTorch by using full-batch gradient descent with learning rate $1 \cdot 10^{-4}$. Initialization is standard Xavier. In the cases with $y = 1$ we train with the Adam optimizer for $2 \cdot 10^4$ epochs.

In the cases with $y > 1$ we train with the SGD optimizer with momentum 0.5 for $4 \cdot 10^4$ epochs. In those cases we increase the coupling constant λ at each epoch by a factor $\lambda_1 = 5 \cdot 10^{-3}$ starting from the value $\lambda_0 = 1 \cdot 10^{-4}$ up to a maximum $\lambda_{\max} = 1 \cdot 10^2$; namely we set $\lambda(t) = \min[\lambda_0(1 + \lambda_1)^t, \lambda_{\max}]$.

The norm is always kept fixed by renormalizing the weights to the given magnitude before each forward pass of the perceptron.

3.5 Conclusions

We have presented an analytical study concerning the connection between local entropy and optimal generalization in the case of Gaussian mixtures. Configurations of weights that reach Bayes-optimal performance were shown to be located inside regions of high local entropy, i.e. in wide flat minima of the error-counting loss function. We have also shown that targeting the wide flat minima of the differentiable loss function used for gradient learning (e.g. MSE) is a viable algorithmic strategy.

These results are relevant for the very active discussion around the success of deep learning from different points of view. First, work is in progress to extend the local-entropy-related results to deeper architectures (see [6, 1, 2]). Second, the relation between the sharpness of minima and their generalization is getting progressively more attention in the community.

A number or experimental results has been known to confirm the connection between flatness and generalization for some years [30, 31].

In [19] they make an extensive experimental study where they test many quantities to see which one correlates best with the generalization. They indeed find that a measure of the flatness around the solution works best.

In [21] they argue that neural networks have an inductive bias towards *simple* functions, where concept of simplicity is defined taking inspiration from Kolmogorov complexity. This fact proves very useful to learn real-world dataset (which seem to be simple example-label relations). Interestingly, one of the possible way to measure this simplicity is again the flatness around a weight configuration. In [20] they discuss the cases when simplicity is equivalent to flatness.

The results presented in this chapter fit in this line of works by providing a simple model where analytical calculations are available and complications of non-convexity of the landscape are absent, shining some light on how flatness influences the generalization properties.

Chapter 4

Natural Representation of Composite Data with a simple Autoencoder

The content of this chapter follow the content of my paper at ref. [32].

4.1 Introduction to unsupervised learning

We call *unsupervised learning* a learning task where the examples do not come with associated labels, so that the inference problem takes the form of extracting some features of their distribution rather that reproducing associations. Basic examples of unsupervised learning are principal component analysis (PCA) and clustering algorithms, while some neural networks that approach this task in a more advanced way are restricted Boltzmann machines (RBMs) and variational autoencoders (VAEs), that are described in detail in [10]. Here we want to study the effects of local entropy on the simplest neural network capable of solving an unsupervised-learning problem, therefore we choose undercomplete autoencoders. These are multi-layer neural networks that are trained to realize the identity function: their goal is to learn a compressed parametrization of the data distribution. To this end, the training is performed under the constraint that the internal representation in a specific low-dimensional layer called *bottleneck*.

A basic unsupervised learning problem cannot be formulated on random data, which are uncompressible and featureless by definition. We need *composite data* (also called *structured*), that is defined by having some form of correlation between its channels. This correlation also implies that data that can be thought of as a composition of basic features. For such data we expect that an efficient description can often be constructed by a – possibly weighted – enumeration of the basic features that are present in a single observation.

If the data is composite in nature, we expect a representation that captures the true underlying contribution of features to generalize well. Indeed, we can use the notion of a "correct" representation of the data as an additional — and maybe better — measure of

generalization in the context of unsupervised learning. We will study a basic example of such data in section 4.2.

One of the most promising disciplines where one can find important unsupervised learning problems is biology. As a first example, we could describe genomes of single organisms as a composition of genes and gene clusters, where the presence or absence of specific genes is determined by the evolutionary history and further reflected in the presence or absence of functions and biochemical pathways the organism has at its disposal [33, 34]. Depending on the level of description, such a composition is not necessarily a linear superposition of the basic features. It has recently been estimated, for example, that due to horizontal gene transfer the genome of Homo Sapiens outside of Africa is composed of 1.5%−2.1% of Neanderthal DNA [35], but no single genomic locus is actually a superposition. Nonetheless, such a description conveys a lot of information: a machine learning algorithm that could be trained in an unsupervised manner on a large number of genomes and automatically output such coefficients would be very valuable in the field of comparative genomics [36].

As a further example we can take the gene expression signature of a single cell, which is determined by the activity of modules of genes that are activated depending on cell identity and the environmental conditions [37]. Since there are far fewer such gene modules than genes, the activity of these modules can be used as an efficient description of the state of the cell. The inference of such modules based on single cell genomic data and downstream tasks like clustering cells into subtypes is an ongoing field of research [38].

On a even more fine-grained level, there have been recently several successful efforts to model protein sequence data as a composition of features that arise from structural and functional constraints and are also influenced by phylogeny [12, 39]. This leads to several possible patterns of amino acids for making up functional groups, or contacts between amino acids, and the presence or absence of these patterns can be used as features and inferred from aligned sequence data of homologous proteins.

There are also many examples of composite data outside of biology. An immediate example are images that contain multiple objects. The efficient extraction of such objects, which can be seen as basic features, has important applications, for example for self-driving cars [40]. In such applications, one is of course also interested in the number and locations of the objects, but a basic description of an image, using an enumeration of objects present, can be part of a general pipeline.

As a final example, we note that in natural language processing documents are often modeled as a mixture of topics, each of which gives a contribution to different aspects of the document: for example, the authors of ref. [41] use the distribution of words. As in the case of genomes, the actual document is far from being a superposition of the topics, but such a description is nonetheless useful in fields like text classification.

Sparse autoencoders are a natural candidate model for finding efficient representations of composite data: in fact, under the assumption that only a few basic features contribute to any given observation and that the number of basic features is smaller

than the dimension of the bottleneck, such an internal representation could be expected to identify the basic features that describe the data. An example of how sparsity constraints help feature extraction is shown in figure 4.1.

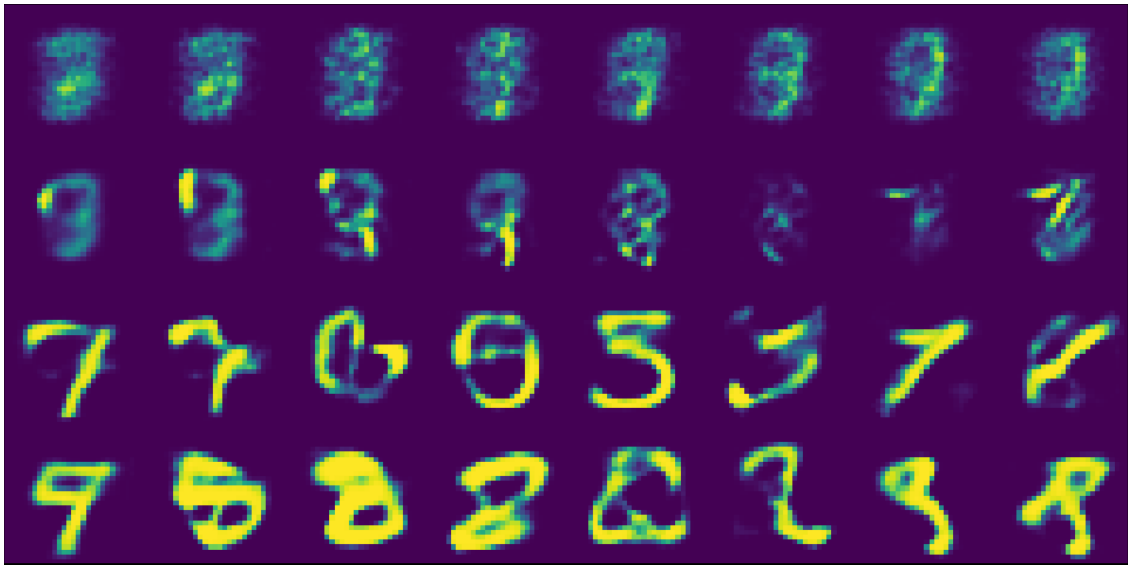


Figure 4.1: Weights corresponding to specific hidden units of a shallow autoencoder trained on images of handwritten digits from the MNIST dataset, imposing the sparsity regularization of equation 4.5. The output corresponding to activating single hidden units becomes progressively more similar to an actual example for higher values of the regularizer strength γ. Ideal features should look like the second row, where each hidden unit corresponds to a section of the input and examples are reconstructed by combining those. The first row corresponds to zero sparsity; the following rows correspond to increasing values of γ (respectively: $\gamma = 0$, $\gamma = 3.0 \cdot 10^{-2}$, $\gamma = 3.5 \cdot 10^{-1}$, $\gamma = 4.0 \cdot 10^{-1}$).

In this chapter, we present evidence that it is indeed possible to find representations of composite data in terms of basic features, but that this process is very sensitive to both overfitting and underfitting: If the imposed sparsity is not strong enough, the resulting representation does not correspond to the basic features. If it is too strong, the dictionary of basic features is not represented completely.

Therefore, we present a modified version of the autoencoder, the *replicated autoencoder*, which is designed to find good solutions in cases where overfitting is a danger. We test this hypothesis on synthetic and on real, biological data. In both cases we find more natural representations of the data using the replicated autoencoder.

4.1.1 Autoencoders

We train feed-forward autoencoders (AE) with stochastic gradient descent (SGD), minimizing the reconstruction error L_{err}.

$$L_{\text{err}}(\vec{w};\{\xi^\mu\}) = \frac{1}{M}\sum_{\mu=1}^{M}\frac{1}{N}\sum_{j=1}^{N}\left(\xi_j^\mu - y_j(\xi^\mu;\vec{w})\right)^2 \tag{4.1}$$

where y_j is the output of j-th neuron of the output layer l parametrized by the network weights $\vec{w}$:

$$y_j(\xi^\mu;\vec{w}) = f\left(\sum_{ij} w_{ij}^{(l)} h_i(\xi^\mu;\vec{w}^{(l-1)},\dots,\vec{w}^{(1)})\right) \tag{4.2}$$

where f is a given activation function, h_i is the output of the i-th neuron of the previous layer $l-1$, and $\vec{w}^{(l-1)},\dots,\vec{w}^{(1)}$ are the weights of all the layers except the last one. Note that we choose each output neuron to have the same activation function and the index j is there because the arguments of y depend on j. Furthermore, M is the dimension of the dataset and N is the dimension of the input and output layers. Since the data points in each dataset considered in this chapter have values included between 0 and 1, the choice for the activation function f of the neurons in the output layer is a logistic sigmoid function (both for shallow and deep autoencoders), defined as

$$\text{Sigm}(x) := \frac{1}{1+e^{-x}} \tag{4.3}$$

where x is the weighed sum of the inputs of a given layer. In the case of the last layer it reads $\sum_{ij} w_{ij}^{(l)} h_i$.

In the following sections we will present results obtained with two different architectures (see figure 4.2):

- a shallow AE made of an input layer, one hidden layer made by H units and one output layer, with $H < N$ (figure 4.2, left);
- a deep AE, made of an input layer, three hidden layers made respectively by K, H and K units, with $K > N > H$ (figure 4.2, right).

We use the sigmoid activation function also for the hidden layer in the case of the shallow AE. On the contrary, the activation function we use in the deep AE is the so-called rectified linear unit (ReLU), defined as

$$\text{ReLU}(x) := \max(0,x) \tag{4.4}$$

The reason for the different choice is purely based on performance: a shallow AE with ReLU activations on the hidden layer would perform badly, while the deep AE with sigmoid activations on many layers is difficult to train and achieves worse performance.

In addition to this, a regularization terms is added to the loss function to prevent overfitting but most importantly to obtain sparse codes of the inputs in the hidden layer;

for this reason we chose a L1 penalty for the activations $h_l(\xi_\mu; \vec{w})$ of the neurons of the central hidden layer:

$$L_{\text{reg}}(\vec{w}; \{\xi_\mu\}) = \frac{1}{N} \sum_{l=1}^{H} |h_l(\xi_\mu; \vec{w})| \tag{4.5}$$

To summarize, we optimize the loss function

$$L_{\text{tot}}(\vec{w}; \{\xi_\mu\}) = L_{\text{err}}(\vec{w}; \{\xi_\mu\}) + \gamma L_{\text{reg}}(\vec{w}; \{\xi_\mu\}) \tag{4.6}$$

where γ is the regularizer strength. Higher values of γ enforce lower activations of the units in the hidden layer and higher overall sparsity (for a detailed discussion on the general effects of this regularizer, see for example ref. [42]).

We observe that with higher γ there are more units that show little to no activation on any input pattern in the training set: The auto-encoder shuts down some units in a trade-off between the two regularization terms in the loss function. We call the number of active units D^* the *inferred dictionary size*, and $H - D^*$ is the number of deactivated units. We infer γ and therefore D^* from the data (see below).

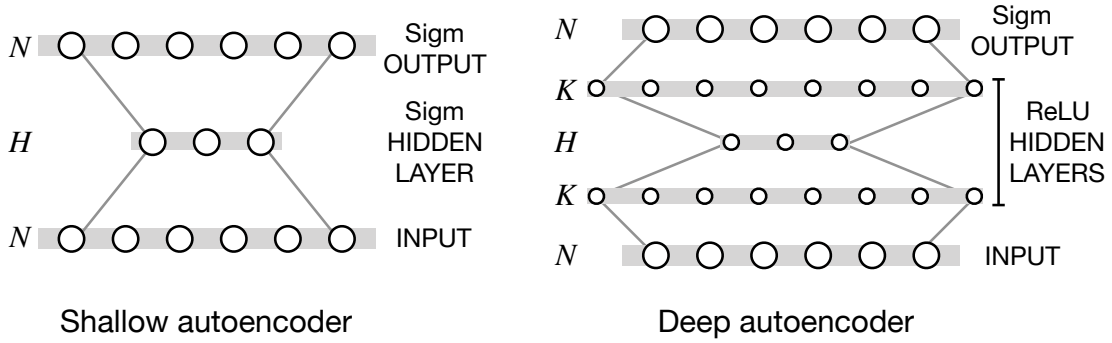


Figure 4.2: Schemes of the basic autoencoder architectures used throughout the paper.

4.1.2 Replicated systems and unsupervised learning

We are interested in the relation between the generalization error in unsupervised learning and the goodness of the internal representation of an autoencoder: we know that local entropy influences the generalization error, therefore we might expect it to influence also the internal representation of a model. For this reason we optimize local entropy in unsupervised models, and ask if increasing generalization performance constitutes evidence that the current representation in the hidden units is corresponding to the basic features in the data.

In this chapter we will sometimes refer to the optimization of local entropy as a *robust optimization*. Here we repeat for convenience (and to fix the notation) some of the definition that we introduced in chapter 2, specialized for the case of autoencoders.

The local (free) entropy of a certain configuration of the weights $\vec{w}^*$ is defined [13] as:

$$\Phi\left(\vec{w}^*;\beta,\lambda\right)=\log\sum_{\vec{w}}\exp\left(-\beta\left(L_{\text{tot}}\left(\vec{w}\right)+\lambda d(\vec{w},\vec{w}^*)^2\right)\right), \tag{4.7}$$

where the function d measures the distance between the weights: several choices are possible, but in the rest of the chapter we use exclusively the euclidean distance. The parameter λ controls indirectly the locality, i.e. the size of the portion of landscape around $\vec{w}^*$ that we are considering (a larger λ corresponds to a smaller radius). The parameter β has the role of an inverse temperature in physics, and it controls indirectly the amount of flatness required of the local landscape (a larger β corresponds to flatter landscapes).

Computing the local entropy is expensive and impractical in most cases. However, as described in detail in ref. [13], if we use the negative local entropy $-\Phi$ as an energy function (i.e. as the objective function that we wish to optimize) with an associated fictitious "inverse temperature" R that we choose to be a positive integer, the canonical partition function of the system is amenable to an equivalent description that can be implemented in a straightforward way: We add R replicas of our model, $\left(\vec{w}^{(r)}\right)_{r=1}^{R}$, and we add an interaction between each replica r and the central (original) configuration $\vec{w}^*$ that forces them to be at a certain distance. We thus end up with the new replicated objective function

$$L_{\text{R}}=\sum_{r=1}^{R}L_{\text{tot}}^{(r)}+\lambda\sum_{r=1}^{R}d(\vec{w}^{(r)},\vec{w}^*)^2, \tag{4.8}$$

where $L_{\text{tot}}^{(r)}$ is the total loss of the replica r. It is important at this stage to observe that the canonical physical description presupposes a noisy optimization process where the amount of noise is regulated by some inverse temperature β, while in this chapter (following ref. [13]) we will be relying on the noise provided by SGD instead, thereby using the mini-batch size and the learning rate as "equivalent" control parameters. Relatedly, we should also note that, although the interaction term is purely attractive, the replicas won't collapse unless the coupling coefficient λ is very large, due to the presence of noise in the optimization. Thus, in our protocol, the coefficient λ is initialized to some small value and gradually increased at each training epoch.

4.1.3 Learning algorithm

The robust optimization protocol that we use throughout this chapter is a version of replicated SGD (introduced in subsection 2.4.1) and can be then summarized as follows. We train R autoencoders with different initialization coupled with a central autoencoder $\vec{w}^*$, which we call the center. Every replica is trained on batches from the training set with normal SGD, but we add a gradually increasing coupling term between every replica and the central autoencoder, see equation (4.8). At the end of the training procedure, we have R trained replicas and one center. All of these $R+1$ models are

autoencoders that can be used for prediction or representation. We typically discard all replicas and only use the center. We call an autoencoder that is trained using this procedure a *replicated* autoencoder (R-AE).

The algorithm we use to train a R-AE consists in iterating two alternating steps: a step of SGD on each replica computed on its own reconstruction loss, followed by a step in which each replica is pushed towards the center and the center towards the replicas. In practice this procedure is similar to elastic-averaging SGD [43], which in turn is related to the optimization of local entropy [13]. The pseudocode for the algorithm is sketched in alg. 2.

Algorithm 2 Training procedure for replicated autoencoder

Input: current weights $\vec{w}^{(r)}, \vec{w}^*$
Hyper-parameters: batch size, learning rate η, coupling λ

for $i = 1, \dots,$ steps **do**
  for $r = 1, \dots,$ replicas **do**
    $x \leftarrow \text{minibatch}[i, r]$
    $\vec{w}^{(r)} \leftarrow \vec{w}^{(r)} - \eta \vec{\nabla}_w L(\vec{w}^{(r)}; x)$
  end for
  for $r = 1, \dots,$ replicas **do**
    $\vec{w}^{(r)} \leftarrow \vec{w}^{(r)} - \lambda(\vec{w}^{(r)} - \vec{w}^*)$
    $\vec{w}^* \leftarrow \vec{w}^* + \lambda(\vec{w}^{(r)} - \vec{w}^*)$
  end for
end for

We impose an exponential scheduling on the coupling λ between the replicas and the center, namely we take $\lambda(t) = \lambda_0(1 + \lambda_1)^t$, where t is the time step of the training in units of epochs.

The training of a *single* autoencoder (S-AE) is performed with the same procedure with just one replica and setting $\lambda = 0$.

In order to set the values for the many hyperparameters of these algorithms, we selected one prototype case among the synthetic data and one among the protein data, and we proceed by trial and error in order to find a regime in which the training converges and has good performance; once we found these values, we assume that the general performances should not be sensitive to the fine-tuning of the hyperparameters: for this reason we use the same set of hyperparameters for every synthetic dataset and the other set of hyperparameters for all the protein families. We observe them to work well in the majority of cases.

For synthetic data we set $\eta = 2.5 \cdot 10^{-4}$, $\lambda_0 = 4 \cdot 10^{-2}$, $\lambda_1 = 3 \cdot 10^{-2}$ and train the shallow autoencoder for 350 epochs and the deep autoencoder for 700 epochs. For all protein data we set $\eta = 5, \cdot 10^{-4}$, $\lambda_0 = 8 \cdot 10^{-3}$, $\lambda_1 = 3 \cdot 10^{-2}$ and train for 300 epochs. The training epochs are sufficient to reach convergence of the training loss. The batch size is fixed to 50 for all trainings.

All the results for replicated models in this chapter are obtained using $R = 5$ replicas.

We do not use any momentum, which would produce a deterioration of performances across every region of parameters and non-convergence. The reason for this behavior could be that the loss landscape for this optimization problem appears to be highly non-convex, especially when the bottleneck size is close to the intrisic dimension of the data (this happens at a specific value γ^*, discussed below); momentum-related techniques, on the other hand, are designed to work well when the loss landscape is sufficiently smooth [42].

In the rest of this chapter we ask if this robust optimization is helpful for finding a natural representation of composite data. We test this idea first on synthetic data where we control the generative process and then extend the approach to protein sequence data. In the latter case the exact generative process is unknown, but a coarse approximation to the basic features can be found in the taxonomic labels.

4.2 Preliminary studies

4.2.1 Synthetic data

Following ref. [44], we generate synthetic datasets $X = \{\vec{x}^\mu\}_{\mu=1}^{M}$ of examples $\vec{x}^\mu$ obtained as superposition of basic features, modeled as follows. We consider a dictionary of basic features $\{\vec{v}_d\}_{d=1}^{D}$, where D is the size of the dictionary. In this setup, we choose $\vec{v}_d$ as a random binary (0 or 1) sparse vector of length N. We use binary weights α_d^μ to control the contribution of the basic feature $\vec{v}_d$ on the observation $\vec{x}^\mu$ and set only a small number of the weights to 1 for each observation. The final observation is defined to be

$$\vec{x}^\mu = \min\left(1, \sum_{d=1}^{D} \alpha_d^\mu \vec{v}_d\right) \qquad \mu \in \{1 \dots M\} \tag{4.9}$$

Note that this is not a simple linear superposition due to the element-wise min function. The purpose of this way of generating data is to let all basic features have a potential impact on every observation while keeping the task of inferring their contributions and the basic features themselves non-trivial. A possible representation of the data is one where each feature $\vec{v}_d$ in the dictionary corresponds to a single hidden unit in the central layer of the autoencoder. We call this the *natural* representation of the synthetic dataset. This representation needs D hidden units. For this reason we expect the autoencoder to be able to find the natural representation when $H \geq D$, given that an appropriate value for γ has been used.

We generate synthetic data points according to equation (4.9) with the following characteristics: each example has 784 components, we generate training sets with $M_{\text{train}} = 60000$ examples and a test set with $M_{\text{test}} = 10000$ examples. We used four different datasets with dictionary sizes $D = 80$, 160, 240 and 320.

The architecture (figure 4.2) is fixed: we used intermediate hidden layers with $K = 1000$ and a bottleneck with $H = 400$ for all our experiments with synthetic data.

We chose the feature vectors $\vec{v}_d$ to be random with binary (0 or 1) independent entries, with a fixed average fraction p_v of non-zero components. The coefficients α_d^μ are also binary, sparse and random: they were generated with a probability p_d of being non-zero. However, in order to make the retrieval problem sufficiently difficult, for each pattern $\vec{x}^\mu$ we ensured that it contained *at least* three features (i.e. we discarded and resampled those that didn't meet the criterion $\sum_d \alpha_d^\mu \geq 3$). The generation of a dataset is therefore parametrized by N, M_{train}, M_{test}, D, p_v and p_d. In this chapter, we fixed the sparsity of the features at $p_v = 0.1$ and the sparsity of the coefficients at $p_d = 0.01$. We always chose $M \gg D$.

Since we work with binary patterns, the activation function of the output layer is chosen to be $\text{Sigm}(x) := 1/(1+e^{-x})$, which sets the range of each output unit between 0 and 1. The loss function of choice for these datasets is mean square error (MSE), which is simply the squared difference between a unit in the input layer and the corresponding unit in the output layer, summed over all the units.

4.2.2 A more distributed representation improves performance of shallow autoencoders

We trained the models with the single (S-AE) and robust (R-AE) algorithms for many values of the regularizer strength γ, stopping the training after a fixed number of epochs sufficient to reach convergence in train loss, test loss and regularization loss. It is worth to note that the robust algorithm could speed up the training with a more precise scheduling of the coupling hyperparameter λ, but we chose to keep the same number of epochs of the single algorithm in order to keep the comparison between the two as clean as possible. Therefore, we chose a schedule for λ such that the replicas collapse together withing the chosen number of epochs. A similar reasoning holds for the learning rate: better result could be obtained for some values of γ using a fine-tuned scheduling of the learning rate, especially in the deep architecture (see section 3.1). Nonetheless, for each architecture we found a learning rate appropriate for all values of γ and kept it fixed during the training, in order to highlight the effects of the optimization of the local entropy.

Since in principle same values of γ can have different effects on the two different training algorithms, we compare the results on equal terms of the L1 norm. High values of γ correspond to forcing low values of the regularization loss.

It is convenient to define some words to describe the states of neurons of the hidden layer. Let's say that a neuron *dead* if its output is zero for every example in the dataset; this can be a way in which the network minimizes the L1 norm: it effectively reduces the size of the hidden layer by killing some of its units. Let's say that a neuron is *active* if it is not dead. An active unit still can have zero output on some examples and non-zero output on some other examples. We say that the neuron is *firing* for an input if its

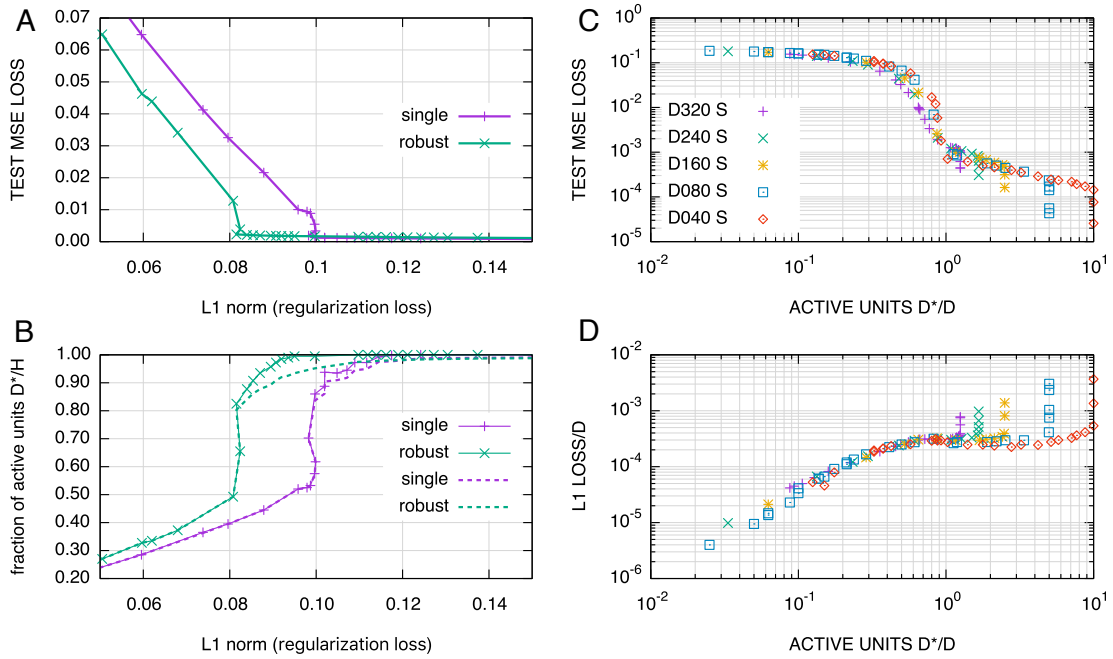


Figure 4.3: The sparsity-inducing regularizer turns off $H - D^*$ hidden units; the reconstruction performance starts deteriorating for a number D^* of active units less than the dimension of the dictionary of the dataset D. **A)** The performance robust AE starts deteriorating at a lower value of the regularization loss L1. **B)** The shallow AE shows a regime in which some hidden units are activated in a compositional way: the **solid line** describes the number of active units which are not turned off, while the **dashed line** describes the average number of firing units per pattern; a higher difference between these two quantities means a representation of input more distributed across the active hidden units. The robust AE shows this regime for more units and for smaller values of L1 norm; this is the reason why it achieves better performance for smaller values of the regularization loss. **C-D)** Reconstruction loss of single autoencoders as a function of the number of active units, for many values of the dictionary dimension D: the error starts growing quickly when $D^*/D < 1$. The over-parametrized region $D^*/D > 1$ corresponds to a plateau in the regularization loss L1. These trainings were performed using a shallow autoencoder $784 > 400 > 784$ trained for 350 epochs on datasets with different D but the same number of patterns $M = 6 \cdot 10^4$. For the sake of simplicity, we limited each pattern ξ^μ to be composed of exactly three features v^d randomly chosen with uniform probability $p_u = 0.01$. The trainings in A-B were performed for fixed $D = 320$.

output is non-zero.

As seen in figure 4.3A, in both cases the test loss curve as a function the L1 norm has two well recognizable regimes: at high values of L1 norm (low γ), the test loss

increases slowly up to a critical value; then it changes drastically the slope and the reconstruction performance rapidly deteriorates (the outputs are very bad for value of MSE greater than $\sim 3 \cdot 10^{-2}$).

In figure 4.3B (solid line) we can see that the critical value of L1 norm corresponds to a point in which increasing γ does not decrease the regularization loss but decreases only the number of active hidden units: in this regime the autoencoder only uses a number of hidden units $D^* < H$, while the other $H - D^*$ are dead (both trainset and testset).

The striking difference between the robust algorithm and the single algorithm is that the former is able to achieve a good performance at higher sparsity; this is connected to the fact that, at L1 norm between ~ 0.08 and ~ 0.10, the robust autoencoder has a number of active units much larger than the single autoencoder. To explain how this is obtained, we consider the average number of active units *per pattern*, reported in figure 4.3B (dashed line): in the robust autoencoder this number is lower than D^*, while in the single autoencoder it is much closer to D^*. Given that the two hidden representation have the same L1 norm, we deduce that, while the single autoencoder always fires all the active units, the robust autoencoder fires only a fraction of the active units, namely it uses a distributed representation. The values of L1 norm in which this behavior happens is the same where the robust AE performs much better than the single AE.

It is important to note that, in this simple architecture, a proper sparse representation of data (which is close to how these data were generated) is achievable only by the robust AE, since the L1 regularizer on the single AE mainly kills hidden units. This difference offers us an interpretation of the results on deeper architectures, where a robust training helps to actually retrieve the original features of the sparse data: this could be due to robust training being more biased towards deeper architectures.

4.2.3 Shallow autoencoders can infer the intrinsic dimension of the dataset

The region of increased performance in the robust AE ends when the number of active units D^* becomes lower than the dimension D of the dictionary of the dataset (in figure 4.3B this happens when $D^*/H = 0.80$, since we used $D = 320$). We repeated the same experiment for different values of D. The result is that we can exploit the number of active units in the hidden layer to infer the intrinsic dimension of the dataset: if we plot both the MSE loss or the L1 loss as a function of D^*/D, for every value of D we observe a point at $D^*/D = 1$ at which the curve changes slope (figure 4.3C and 4.3D). Since the performance starts to deteriorate at that point, it is reasonable to think that the AE is going from an over-parametrized regime to an under-parametrized regime, and that the turning point corresponds to the intrinsic dimension of the dataset.

4.2.4 Deep autoencoders can retrieve more original features of the dataset

We performed the same kind of experiments for the deep AE: training the architecture at various sparsity using datasets with various dimensions D of the dictionary. This time we compared the results for same values of the parameter γ and not the L1 norm; the reason is that where the function L1(γ) is stable, it has the same value for every value of D, and for $\gamma > 10^{-2}$ it is unstable. This means it that is impossible to use the L1 norm as a useful quantity to plot on the x axis (figure 4.4C).

Let us first consider the optimization of single AE using different values of D (see figure 4.4A): the common behavior is that the MSE loss slowly increases with γ up to a critical value, from which the performance starts deteriorating very quickly; in contrast to the shallow AE, now the values of the MSE loss are decreasing for decreasing D. Another difference with the shallow AE is that, after a peak of instability in the training at a certain value of γ, the MSE loss shows a plateau where the performance is bad, but not fully deteriorated.

To understand what is happening in terms of the inference of the original features, we consider the number of active hidden units (figure 4.4B) and the number of highly activated hidden units per pattern (figure 4.4D), measured with the participation ratio (PR):

$$\mathrm{PR}_a(h) = \frac{\left(\sum_{l=1}^{H}(h_l)^a\right)^2}{\sum_{l=1}^{H}(h_l)^{2a}} \tag{4.10}$$

with $a = 2$. We find that, for sufficiently small D (in particular $D = 80$), the deep AE is able to find a range of values of γ where $D^*/D = 1$; for the same values of γ the number of active hidden units when we present a feature v^d as an input is precisely 1. This suggests that each active hidden unit is assigned to a feature v^d of the dictionary, and that the AE is able to represent the patterns ξ^μ with the exact structure in which they are created. It is interesting to note that the function $D^*(\gamma)$ is qualitatively different when the AE finds the ground truth representation and when it does not: for $D = 80$ the function is concave and has a plateau, while for $D > 80$ the function is convex. This could suggest the presence of some sort of crossover between two regimes of AEs. Irrespectively of whether the AE finds the ground truth or not, the L1 loss has a local minimum for values of γ corresponding to $D^* \simeq D$.

The robust AE improves the performance in different ways. As shown in figure 4.4F, the autoencoder activates the correct number of hidden units on a wide range of values of γ, while the single AE does not even have a plateau in D^*. Additionally, as shown in figure 4.5B, the robust AE it is able to fully retrieve the dictionary of features for bigger dictionaries than single AE. This could be connected to the fact that robust AE spontaneously finds more distributed representations, so that could be more adapt to retrieve features from correlated data. Another improvement is observable in the MSE loss (figure 4.4E): increasing γ both the test and the train loss increase smoothly

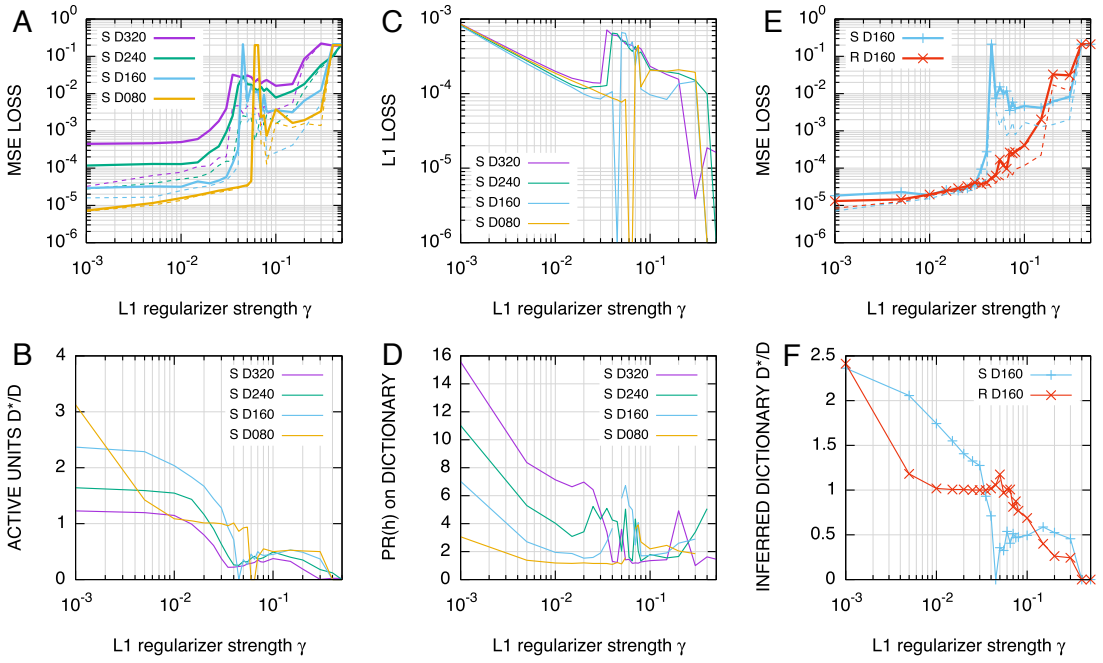


Figure 4.4: The deep AE is able to retrieve all the features of a sufficiently small dictionary, at a specific value of the regularizer strength γ^*. For $\gamma > \gamma^*$ the performance quickly deteriorates. The robust AE is able to fully retrieve bigger dictionaries. **A-D)** The performance of a single deep AE trained on a dataset Ξ_D depends on D: for smaller dictionaries the test loss (**solid line**) is smaller and the rough region appears for a higher value of γ; the train loss (**dashed line**) shows a weaker dependence on D. The retrieval of features improves for smaller dictionaries up to $D = 80$, when the AE is able to fully retrieve all the dictionary: the curve of active units (B) develops a plateau at the value $D^*/D = 1$ and each active unit corresponds to one feature of the dictionary (D). **E)** For a robust deep AE, both the training loss (**red dashed line**) and the test loss (**red solid line**) does not present a rough region for high values of γ: the test error starts increasing smoothly at a certain value of γ. **F)** That value of γ corresponds to the point at which there are less active units D^* than the dimension of the dictionary D. The curve of active units exhibits a plateau at $D^* = D$, meaning that the robust deep AE is able to fully retrieve a dictionary of $D = 160$ features, while the single deep AE is not.

until the reconstruction fully deteriorates, skipping the instabilities and the plateau.

Again, we find that the main difference between the R-AE and the S-AE is that the R-AE is able to achieve a better reconstruction performance at high sparsity, in the region where $\gamma \gtrsim 0.03$ (see figure 4.5A). This is connected to the observation that the R-AE has a number of active units D^* that is significantly larger than the S-AE while keeping a similar L1 norm for most inputs. This might sound paradoxical, but we recall here that

D^* is the number of units in the bottleneck that show a significant activation for at least one input from the training set. This is not directly suppressed by the L1 regularization on the bottleneck, which penalizes cases in which many units are activated for a single input. There are thus different ways to realize the same overall L1 norm. The S-AE kills more units, while using a larger fraction of the remaining active units on the inputs on average. The R-AE, on the other hand, kills fewer units completely, using a smaller fraction of the active units for every input. Another way of stating this fact is that the R-AE uses representations that are more distributed over all available units and keeps D^* closer to D (see 4.6 for an example of this behavior). Another interesting difference is the dynamics of the two algorithms, which is shown in figure 4.7.

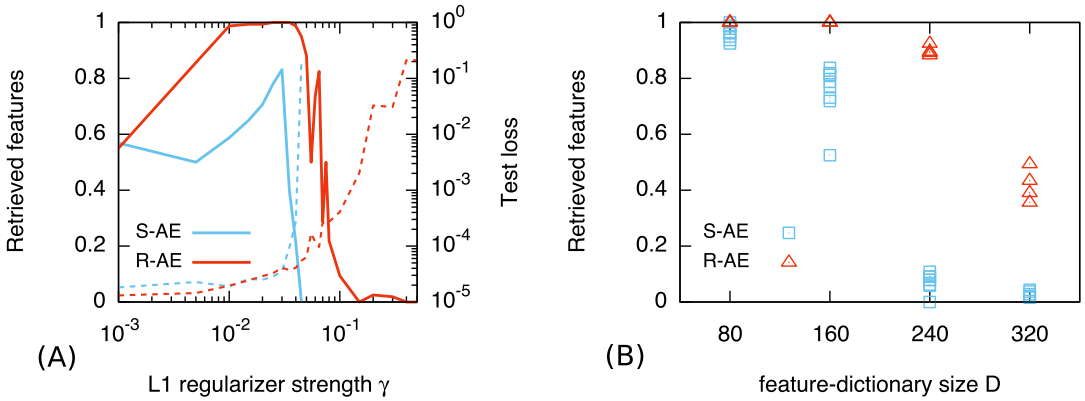


Figure 4.5: **The AE is able to retrieve all the features of a sufficiently small dictionary. (A)** The test loss (**dashed line**, right y-axis) increases slowly with γ, up to a certain knee point γ^*, which corresponds to the point where the fraction of retrieved features has a maximum (**solid line**, left y-axis); for $\gamma > \gamma^*$ the performance quickly deteriorates. The curves are obtained with one execution of the training for each value of γ. The dimension of the dictionary is fixed to $D = 160$. **(B)** The performance of the AE trained on a dataset X_D (y-axis) depends on the dimension of the dictionary D (x-axis): the retrieval of features is better for smaller dictionaries, and the robust AE is able to fully retrieve bigger dictionaries (for $D = 80$ both single and robust AE retrieve 100% of the features, while for $D = 160$ only the robust AE is able to do so). For each D the plot shows 9 results with S-AE and 4 with R-AE, each one corresponding to different realizations of the same training procedure. The regularizer strength is set in the proximity of the knee point, namely $\gamma = 3 \cdot 10^{-2}$.

If we plot the loss as a function of γ, we observe that it grows slowly up to a certain knee point γ^* (figure 4.5A, dashed line). This point coincides with the maximum number of retrieved features. This can be interpreted as a phase transition between overfitting and underfitting, and for $\gamma > \gamma^*$ the performance deteriorates quickly.

In general, the retrieval of features is better for smaller dictionaries for both models, but for larger dictionary sizes the R-AE retrieves a higher number of features, see

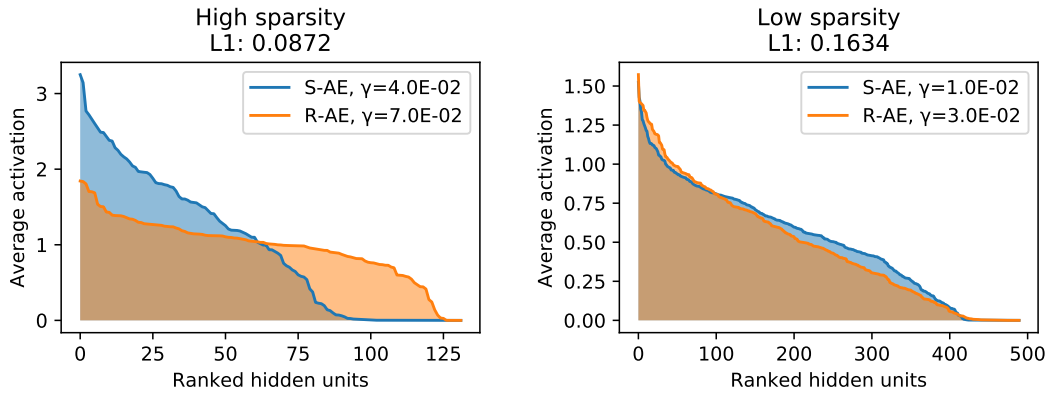


Figure 4.6: There are different ways to realize the same overall L1 norm of the units in the bottleneck layer. The figure shows rank plots for different AE. On the x-axis there are the hidden units ranked by their average activation: the units on the right are the most active on average and the ones on the far right are the ones that are always deactivated (their signal is next to zero across all the dataset). On the y-axis there is the average activation of the units. In the high sparsity case we can see that S-AE deactivates more units completely, while R-AE, on the other hand, deactivates fewer units completely. This effect disappears at lower sparsity, far from the knee point of the loss curve. The dataset used for these result is PF01978.19.

figure 4.5B: for $D = 80$ both the S-AE and the R-AE retrieve 100% of the features, while for $D = 160$ only the R-AE is able to do so. For $D \geq 240$ the R-AE finds $\sim 40\%$ more features than the S-AE.

4.3 Application to biological data: protein families

In this section we test the capability of the R-AE to infer basic features on real data. We use sequence data of homologous proteins because they allow a reasonable interpretation of composition: due to co-evolution of residues that are part of structural contacts or functional groups, certain patterns of amino acids arise. These patterns can be exploited for the prediction of contacts with the structure of a single protein [45, 46], infer protein interaction networks [47, 48] and paralogs [49, 50], model evolutionary landscapes [51] and predict pathogenicity of mutations in humans [52, 53]. Since these patterns are inheritable, we expect their presence to be partly determined by the phylogenetic history of the organism and therefore to be correlated with its taxonomy. We therefore argue that a 'natural' representation of an amino acid sequence should be correlated with taxonomy of the organism.

We thus proceed as follows. We consider a wide variety of protein families and we use aligned sequences in a one-hot encoding as the input of the autoencoders. Each

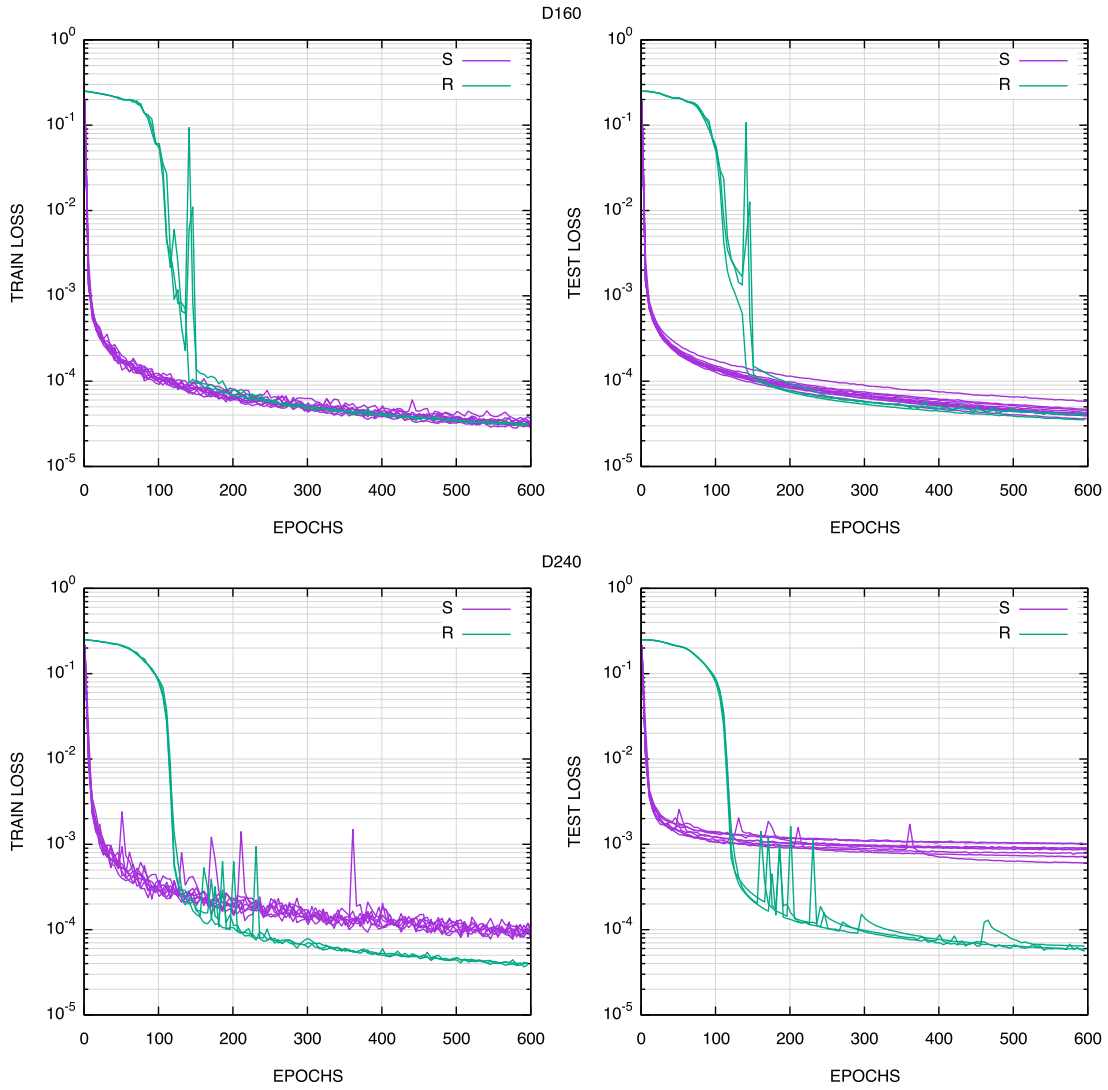


Figure 4.7: Example trajectories of the loss during the training; ten trajectories are shown for S-AE and three for R-AE. The panels on the left show the train loss, the right ones show the test loss. Here we show a case where S-AE and R-AE have the same performance (D=160, top line) and one case where R-AE has a much better performance (D=240, top line). Note that the improvement is greater for the test loss, showing that R-AE generalizes better. These trajectories refer to figure 2B in the main text. The regularizer strength is set in the proximity of the knee point, namely $\gamma = 3 \cdot 10^{-2}$.

family is partitioned in train set, test set and validation set in the proportion 80%-10%-10%. We then test two different measures of correlation between the representations of the S-AE and the R-AE of the sequences with their taxonomic labels. Note that, analogously to the case of synthetic data, the training of the autoencoders is agnostic

about these labels.

4.3.1 Protein data

We considered 18 protein families from the PFAM database (tab. 4.1) selected according a number of criteria: we want many types of proteins represented, as well as families covering many different partitions of the tree of life; additionally, we chose families with a sufficient number of sequences and species, varying the ratio between these two numbers. We use aligned data (multiple sequence alignment, or MSA).

DATASET	n. seq	n. species	n. amm	DATASET	n. seq	n. species	n. amm
PF01978.19	4531	1806	68	PF04545.16	35976	8384	50
PF09278.11	6117	2867	65	PF00805.22	38453	3485	40
PF00444.18	6551	5971	38	PF07676.12	48848	6060	38
PF03459.17	8823	4066	64	PF00356.21	49284	5450	46
PF00831.23	9782	9209	57	PF03989.13	60674	8153	48
PF00253.21	10577	8650	54	PF01381.22	72011	9760	55
PF03793.19	20495	4026	63	PF00196.19	85219	6666	57
PF10531.9	22080	7683	58	PF00353.19	101177	2304	36
PF02954.19	35339	5079	42	PF04542.14	110168	8385	71

Source: https://pfam.xfam.org/

Table 4.1: List of dataset used for training the AE, listed by their number of sequences. The protein families of ribosomal domains are highlighted; notice that, for these families, the ratio of the number of different species over the number of sequences is higher than for the other families.

Given a sequence $S = \{a_i\}_{i=0}^{A}$ of length A to the AE, we represent each amino-acid a_i with a 21-components one-hot encoding: each input sequence is thus a binary vector of length $N = A \times 21$, and the entire dataset with a matrix (M, N). The architecture is rescaled according to the sequence length A: we set $K = 1.1 \times N$, and the number of units in the bottleneck to $H = 0.4 \times N$.

Since each amino acid is a categorical variable represented by a one-hot encoding, a common way to compute the reconstruction error $L^{(i)}$ for a single amino acid a_i is the cross entropy between the input and the output. To do this, we consider the 21 units $\{z_j^{(i)}\}_{j=1,\dots,21}$ in the output layer that describe the site i, then we apply a softmax operation so that each unit can be interpreted as a probability

$$\text{Softmax}(z_j^{(i)}) := \frac{e^{z_j^{(i)}}}{\sum_{j=1}^{21} e^{z_j^{(i)}}} \tag{4.11}$$

and finally we compute the cross entropy

$$L^{(i)} = -z_{j^*}^{(i)} + \log \sum_{j=1}^{21} e^{z_j^{(i)}} \tag{4.12}$$

where j^* is the index corresponding to the true value of the amino acid i. The complete loss function is the summation of the cross entropy losses for each amino acid of the sequence:

$$L = \sum_{i=1}^{A} L^{(i)}. \tag{4.13}$$

Here we choose a linear activation function for the units in the output layer.

4.3.2 The internal representation correlates with the natural one

The behavior of the autoencoders trained on protein sequence data is qualitatively similar to what we saw for synthetic data: there is always a knee point in the curve of the loss (both train and test) as a function of γ, see 4.8, 4.9, 4.10. We expect that the range of values around the knee point corresponds to a representation that is close to the underlying biology.

We determine the knee point γ^* for a given protein family by fitting the error curve (not directly the loss) on the test set by two connected line segments and then use the point where they intersect as γ^*. All the subsequent analysis is done on the validation set, using the autoencoder with the identified γ^*.

Knee Point Identification

An important part of our approach is identifying the knee point γ^* in the loss curve. To this end, we consider the reconstruction error curve on the test set in dependence of γ. The curve has two parts, separated by the knee point: A slow increase in reconstruction performance (decrease in error) and a drastic decrease in reconstruction performance (drastic increase in error) when γ becomes too high. We fit the region around the knee point with the function:

$$f(\gamma) = \begin{cases} a_1 \left(\gamma - \gamma^*\right) + b & \text{if } \gamma < \gamma^* \\ a_2 \left(\gamma - \gamma^*\right) + b & \text{if } \gamma \geq \gamma^* \end{cases} \tag{4.14}$$

which is simply the equation of two straight lines passing from the same point at γ^*. From the fit over the four parameters a_1, a_2, b, γ^* we obtain the estimation of the position of the knee point.

We use the error curve (the number of wrong amino acids in the reconstruction) rather than the loss directly, since the error curve is better approximated by two line segments and therefore easier fitted by our approach, leading to better approximations of the knee point.

The knee point is different for each protein and we expected it to be also different for S-AE and R-AE. Empirically, however, we obtained the best results across all the protein families by using the γ^* of the S-AE also for the R-AE.

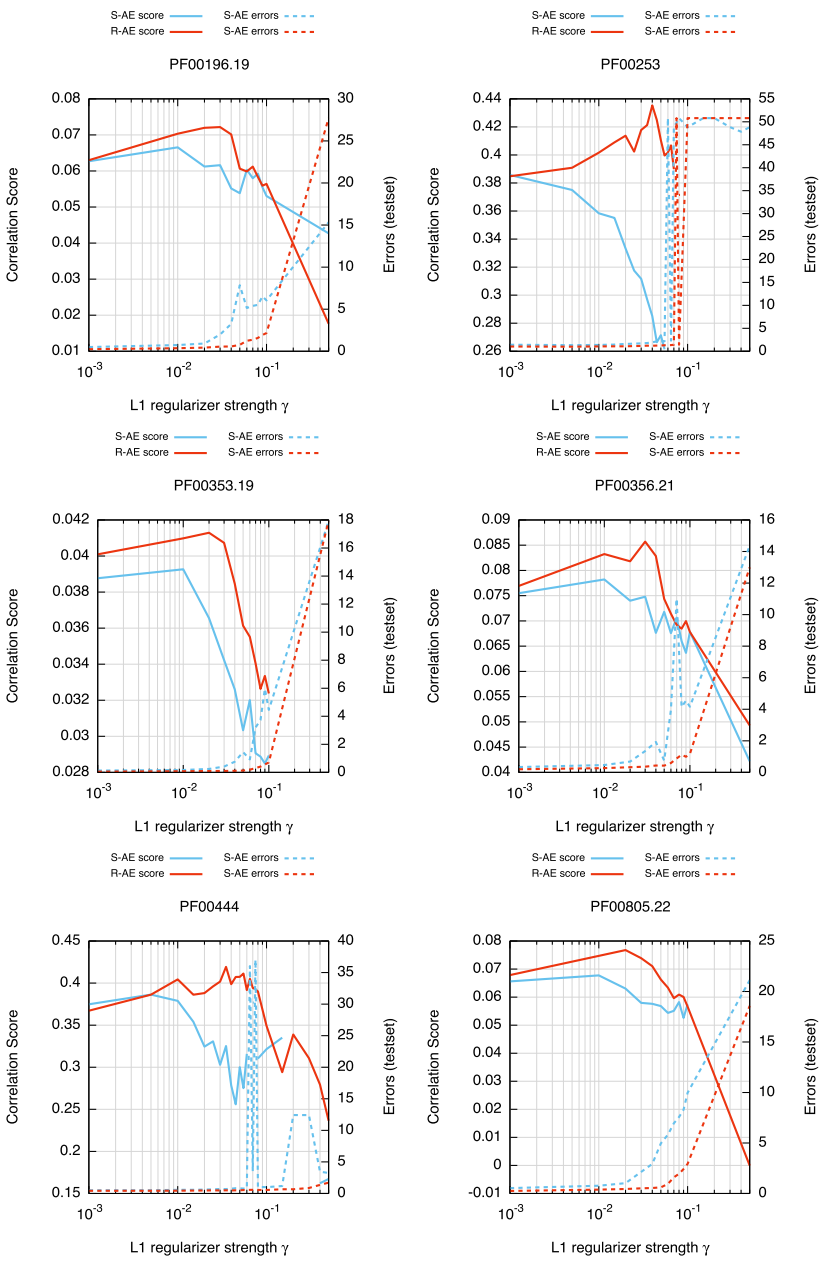


Figure 4.8: Part 1: The behavior of the autoencoders trained on protein sequence data is qualitatively similar to what we saw for synthetic data: there is always a knee point in the curve of the loss as a function of γ, corresponding to the maximum correlation with the taxonomic labels.

Captured taxonomic information

We measure the taxonomic information captured by the hidden layer in two ways: First, in analogy with synthetic data, under the very hopeful hypothesis that each taxonomic

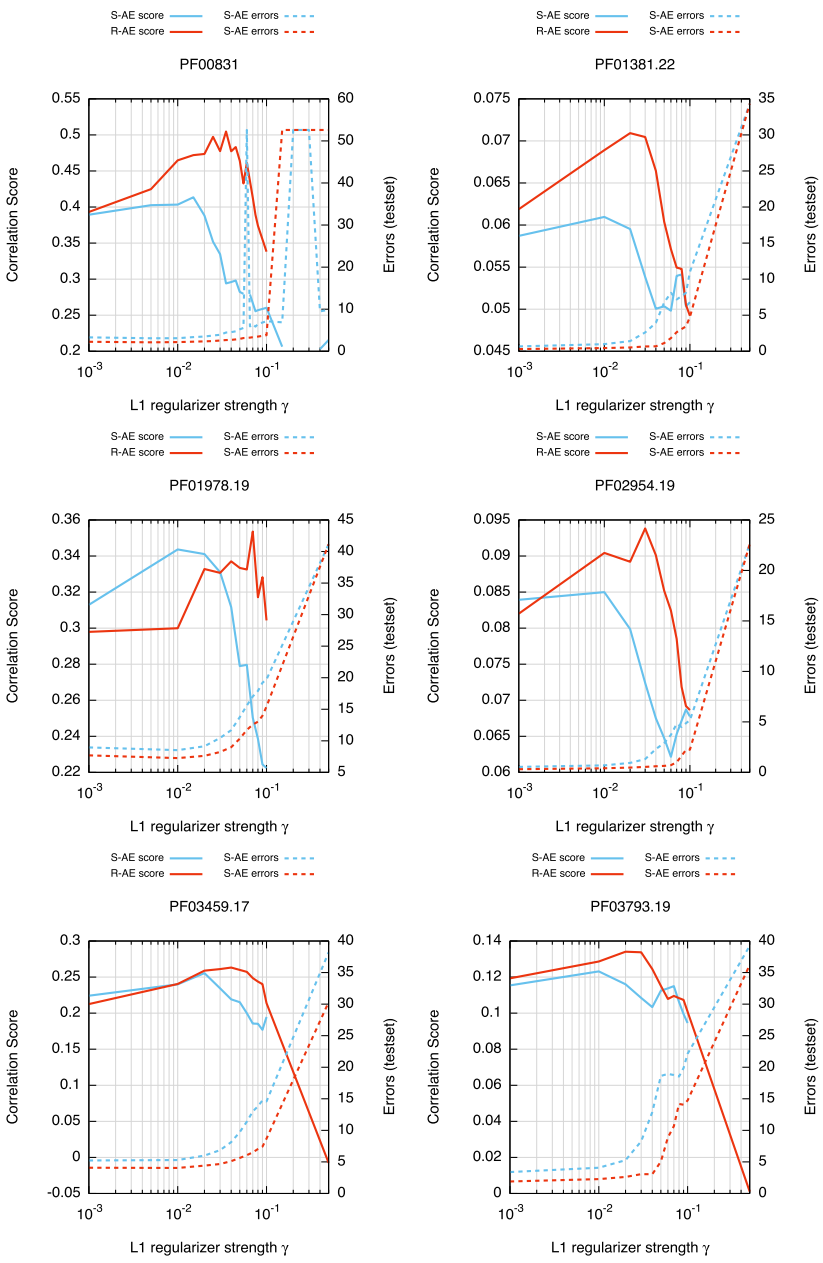


Figure 4.9: Part 2

label corresponds to a single hidden unit. We test this idea in the next paragraph. Secondly, we ask how well a clustering of the sequences based on the hidden representations correlates with the taxonomic labels in comparison to a clustering based directly on the amino acid sequences.

Since the taxonomic classification is modeled by a tree, we consider the labels as organized by their depth d, that is their distance from the root of the tree. For example,

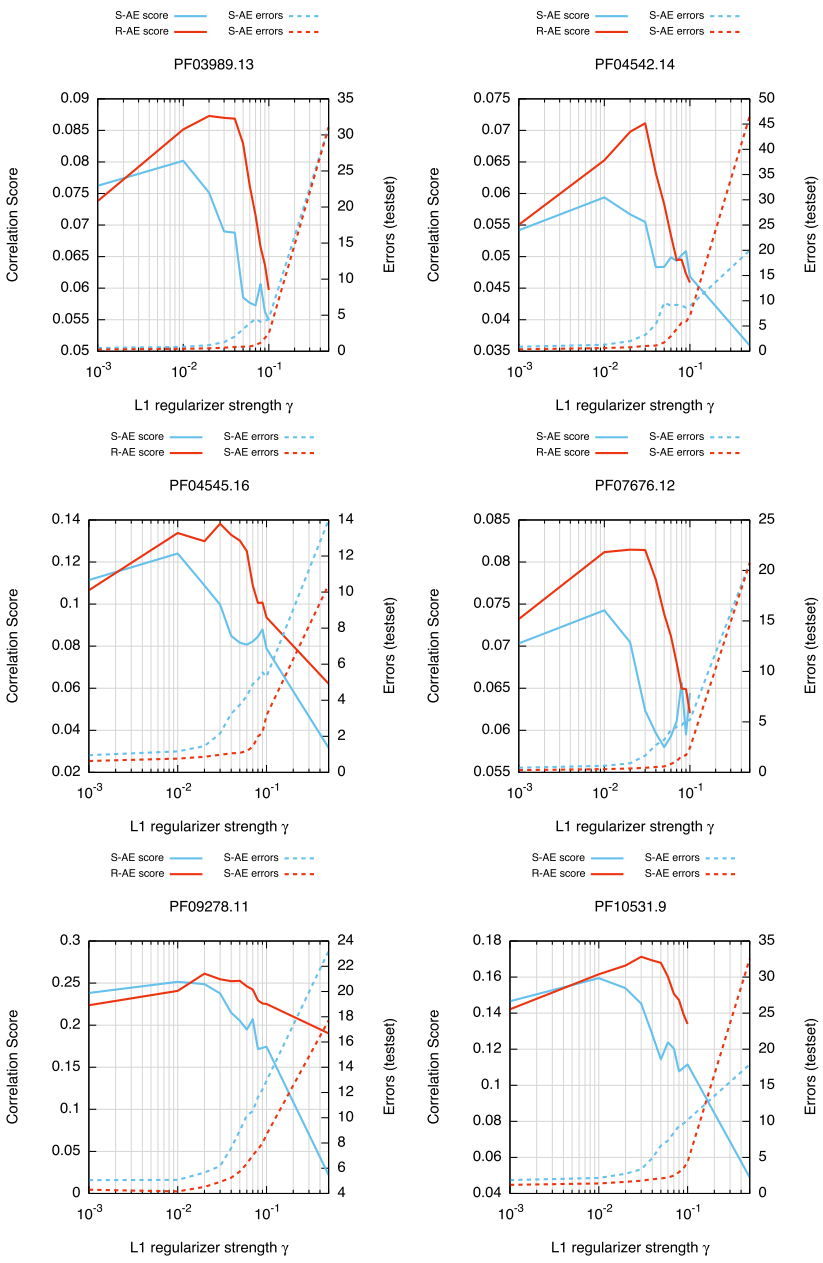


Figure 4.10: Part 3

the root has $d = 0$, the label 'Bacteria' has $d = 1$, 'Proteobacteria' has $d = 2$. Every label is associated with one or more sequences in the training set and every sequence corresponds to several labels (see figure 4.11 for a sketch and figure 4.12 for how different subsets are represented in the AE hidden layer). The labels near the root are the most populated, while the labels deeper in the tree are sparsely populated. We expect the labels in the first few levels to be more correlated with the hidden unit since

deeper labels correspond to only a few sequences in the training set (see figure 4.13). For these reasons we restricted the following analysis to labels up to depth $d = 5$, with the additional condition that they must contain at least 20 sequences from the training set.

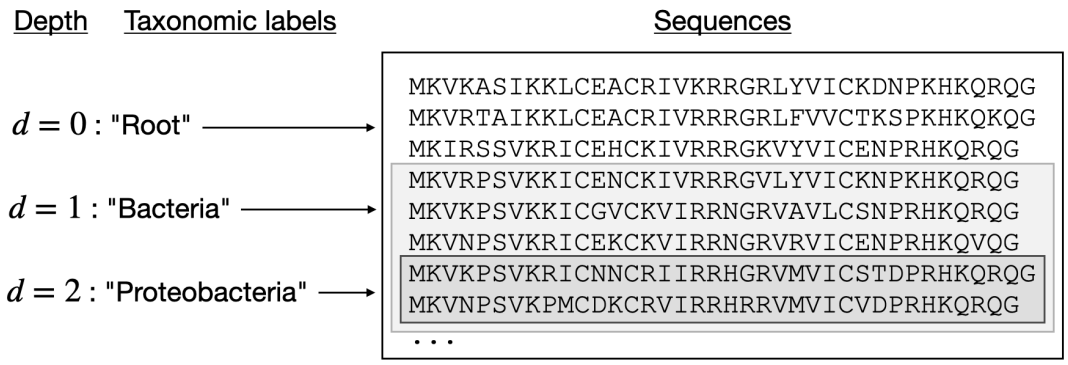


Figure 4.11: **Scheme of the data organization.** Since the taxonomic labels live on a tree, each amino-acid sequence correspond to multiple labels on different depths of the tree.

Neuron-Taxon correlation

Given a taxonomic label indexed by l, we consider the binary variable $y_l(s^\mu)$ that, for each sequence s^μ in the dataset, is equal to 1 if the sequence belongs to that taxon and is equal to 0 otherwise; then, after the AE has been trained, we compute (on the training set) the correlation matrix $C_{l,k}$ between the variables $\{y_l\}$ and the activations $\{h_k\}$ of the hidden units. For every label l we select the most correlated unit $k^*(l)$. Then we define a score Q as the average correlation (on the test set) of the most correlated units for every label:

$$Q := \frac{1}{L}\sum_{l=1}^{L} C_{l,k^*(l)} \tag{4.15}$$

where L is the total number of labels considered in a dataset.

The results are shown in figure 4.14: R-AE consistently finds a higher score than S-AE. It is useful to note the general trend of this score: the more sequences in the dataset, the worse the score. Additionally, the protein families of ribosomal domains have a much higher score, which is probably due to the fact that ribosomes are well sampled (see the 4.4 section for more on this).

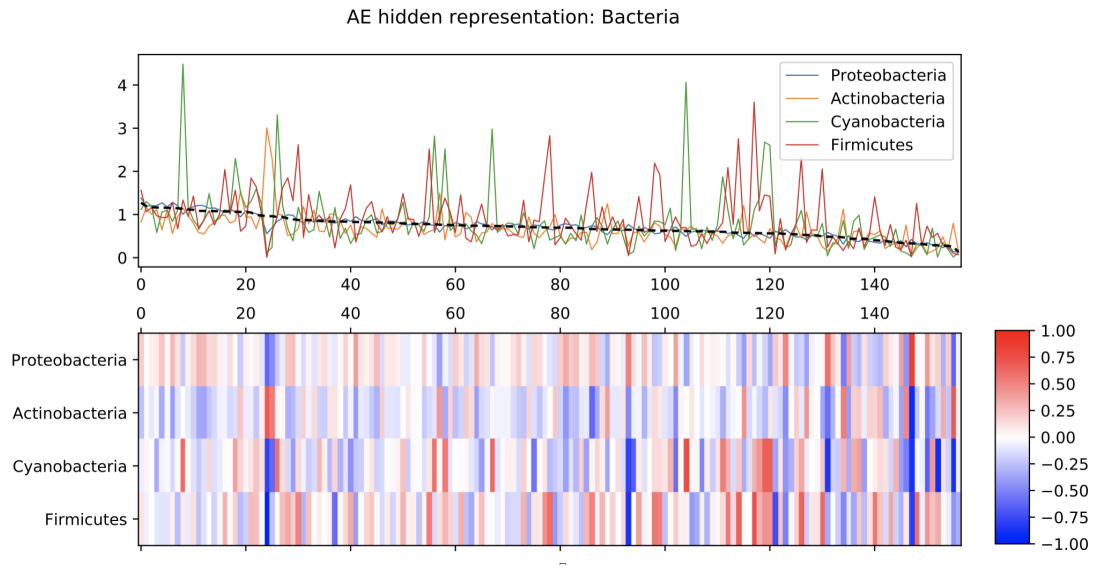


Figure 4.12: **Different taxonomic subsets are represented in different ways in the AE hidden layer**. Upper panel: the average activation for a given subset (y axis) is shown for the D^* active hidden units of the hidden layer (x axis). The dashed line correspond to the average activation for the father label "Bacteria". Lower panel: correlation coefficient between the average activation of hidden units and the taxonomic labels defining the subsets.

Clustering data in the latent space

For a given label l at depth d, we consider the sub-labels l' at depth $d+1$ branching from l; we select the subset of the training set corresponding to the label l, then we compute the centroids of the clusters corresponding to the sub-labels l' by averaging the sequences with that sub-label. Given a new sequence from the test set belonging to l, we assign the sub-label l' according to the closest centroid. In order to perform this clustering procedure on disjoint subsets in such a way that the accuracies are independent of each other, we fix the depth d and consider only labels l found at that depth. We choose $d = 2$, because it provides the most variety of sub-labels with a high number of examples in the protein families we considered.

First we run this procedure using the original sequences, the same ones on which we trained the AEs. Then we repeat the clustering using, for each sequence, its representation in terms of the hidden units of the AEs. We ask whether this representation improves the accuracy of the clustering, depending on whether we use the representation from R-AE or S-AE.

The results are shown in figure 4.15: the representation learned by R-AE does improve the accuracy for the majority of labels both respect to S-AE (bottom-left panel) the to input space (bottom-right panel).

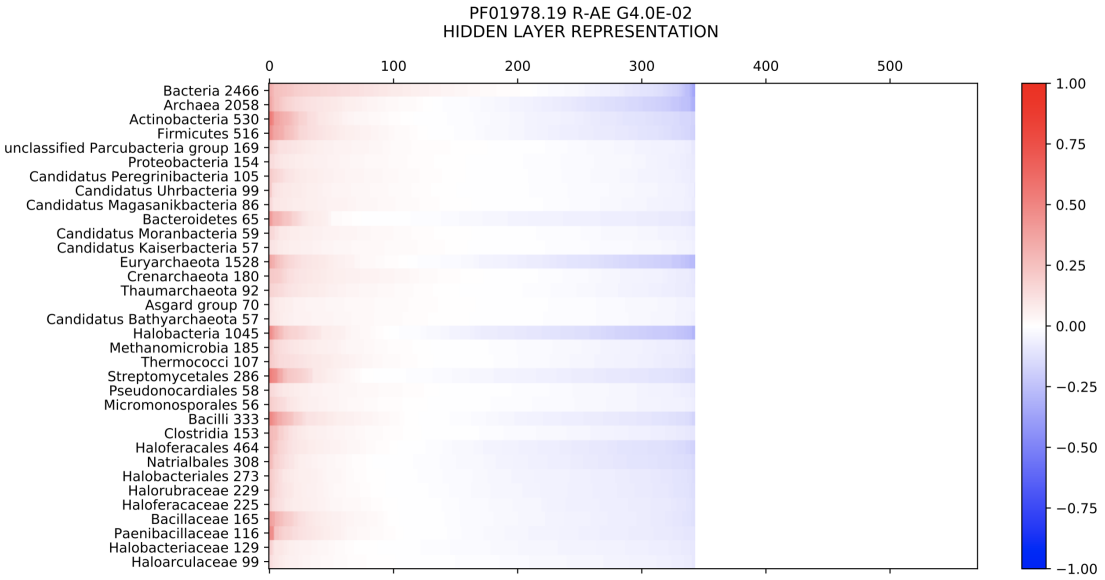


Figure 4.13: **Less populated subsets are more difficult to identify**. Each line of the matrix shows the correlation coefficients between a taxonomic label and all the active units of the AE hidden layer, sorted from the most correlated to the most uncorrelated. The empty space corresponds to the $H - D^*$ dead hidden units. Each taxonomic label is shown with its populations. The labels with small populations have lower correlations with hidden units.

4.4 Conclusions

In this chapter we have presented a method to extract representations of composite data that connects to the structure of the underlying generative process. To this end, we combined two techniques that allowed us to recover such representations from the bottleneck of autoencoders trained on the composite data: The first is a regularization that forces the autoencoder to use a sparse representation. The second is the replicating of the autoencoder, which changes the properties of the solutions found. We showcased the method on two different datasets. In the first dataset, where we controlled the generative process, we showed that the replication allows to extract the underlying basic features also in cases where the sparsity constraints are too strong for a single autoencoder. After a closer analysis, we found that replication enables the system to effectively disentangle basic features and specialize parts of the internal representations.

In a second step, we applied the same method to protein sequence data. Since patterns of amino acids are inheritable, we used the correlation between the extracted representations and the phylogenetic labels as a metric for assessing the quality of the representation. We found that the replication of the autoencoder resulted in representations that are closer to biological reality and that the qualitative characteristics of the loss function and internal representations are similar to autoencoders trained on

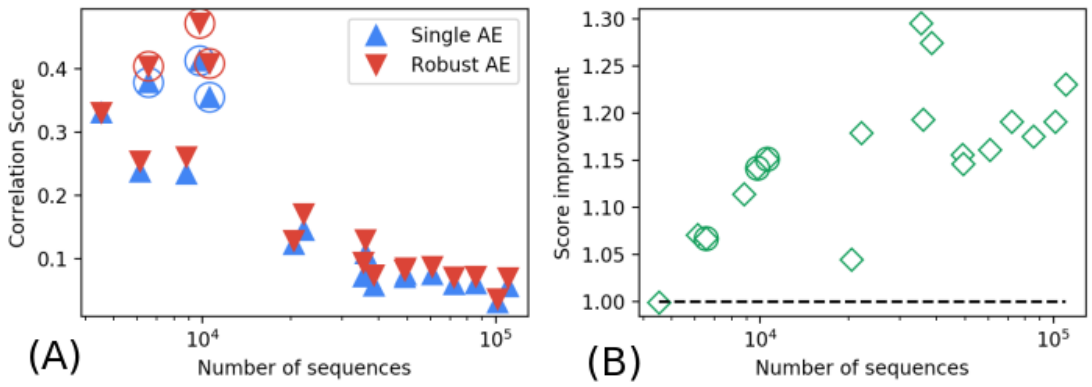


Figure 4.14: **The robust AE consistently captures more biological information in most of the protein families considered. (A)** The panel shows an aggregate score of correlation between the hidden units of the network and the taxonomic labels present in each protein family (equation (4.15)); the families are show on the x-axis according to their number of sequences. The circled points correspond to ribosomal domains, which appear to be the datasets with the highest performance of our method. **(B)** The panel shows, for each family, the score improvement gained by training the robust AE respect to training the single AE.

synthetic data.

One intriguing observation is that the point where the internal representation becomes correlated with the basic features is identifiable: The knee point in the loss curve in dependence of the regularization parameter corresponds to the peak performance in feature retrieval. For synthetic data we were able to verify this directly. Near this knee point, each hidden unit represented one basic feature. This also allowed us to infer the number of basic features present in the data (i.e. its inherent dimensionality). For protein sequence data, we observed that the internal representation becomes correlated with taxonomic labels at the knee point. After this knee point, the loss deteriorates quickly, indicating that the autoencoder starts dropping important information from the internal representation.

Interestingly, we found that feature retrieval on synthetic data became more difficult for increasing dictionary sizes. This could be addressed either by using a larger and therefore more expressive architecture or by using a larger training set. We generally expect the size of the training set necessary for the extraction of the basic features to scale with the size of the dictionary [54, 55].

Regarding the difficulty of the feature extraction task, we found a similar behavior on the protein families: Families with more sequences also contain a higher number of labels and are expected to have a wider variety of features. It is further noteworthy that the correlation between taxonomic labels and internal representations was more pronounced for ribosomal domains than for other families with a similar number of

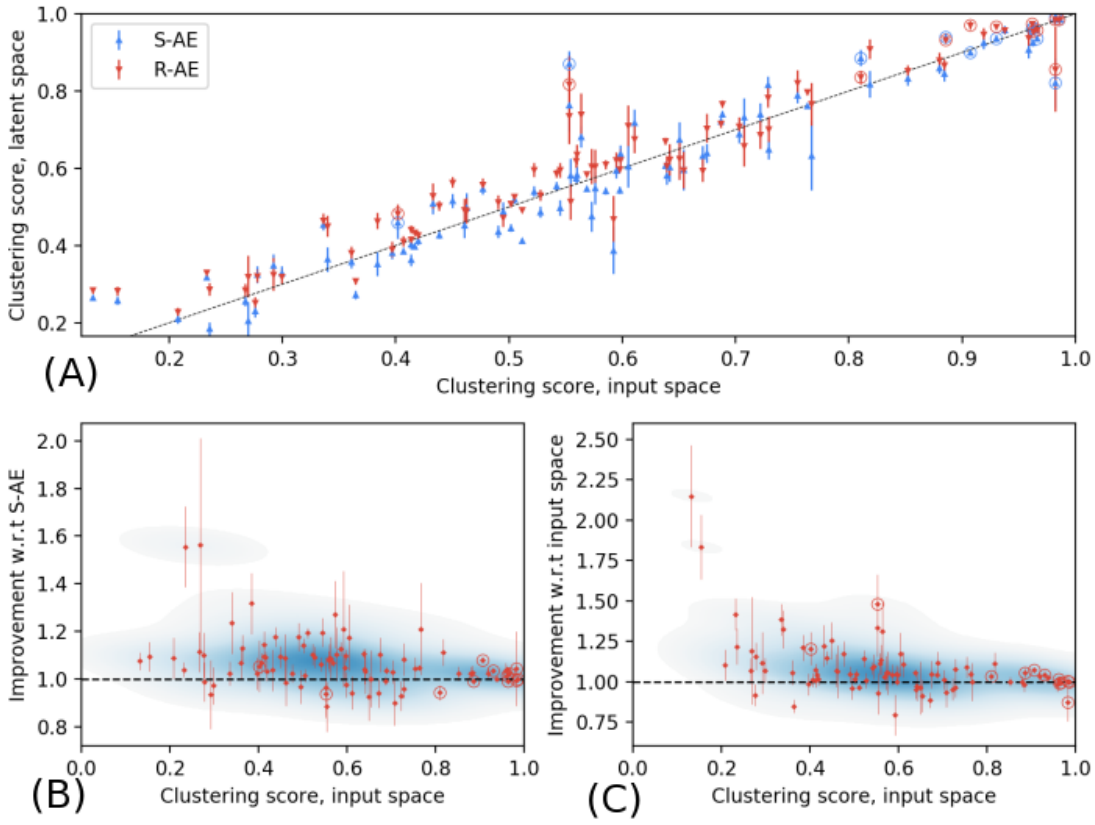


Figure 4.15: **The representation of the AE improves a clustering algorithm. (A)** The panel shows the accuracy of a clustering algorithms run into the latent space of the neural network versus the accuracy of a clustering algorithms run on the original data point. Each point corresponds to a label in a protein family: given the subset of a protein family corresponding to that label, we consider the task of clustering that subset according to the subcategories of that label. **(B,C)** The panels show two 2D density plots of the score improvement as a function of the score on the original data points, respect to S-AE (B) and to the input space (C).

sequences (see figure 4.14, circled markers). This is probably due to the fact that ribosomes are well studied systems in many species and that in the databases we used there are more species per sequence for these domains (see tab. 4.1). This indicates that a well-balanced dataset is an additional factor in the inference of basic features from composite data.

In conclusion, we have shown that replicated autoencoders are better suited for extracting natural representations of composite data. In particular, we positively tested this capability on protein sequences and their taxonomic labels. This result is a proof of concept that this method works even outside synthetic problems, and that the details of the procedure could be further optimized to obtain even better results.

On amino-acid data, the difference between single and robust autoencoders is less evident than on synthetic data. Nevertheless, the fact that we see there is some improvement was not necessarily expected, and its presence seem to confirm the strong hypotheses that we made: namely that taxonomic labels can be associated to single hidden units, and that amino-acid sequences can be seen as superposition of features specific to taxonomic labels.

Some parts of this method could be improved in future works. For example, the relation between the sparsity of the autoencoder representation and the level of the taxonomic tree could be explored. In fact, one expects that features of an AE with a smaller bottleneck would correspond to labels closer to the root of the taxonomic tree and, on the contrary, a bigger bottleneck would describe labels closer to the leaves. If we were able to characterize this dependence we could also choose an appropriate subset of taxonomic labels to test. Additionally, the quality of the representation could be tested by comparing it to other sequence representations that are becoming the standard in bioinformatics (see for example [56]). Finally, note that we used a very simple neural network (especially compared those used in [56]). In particular, moving from shallow to deep autoencoders allowed us to identify which feature is connected to which hidden unit and made a robust model essential on synthetic data. It is reasonable to think that using more advanced architectures the difference between a single model and a robust one would improve even on amino-acid sequences data.

The approach presented in this chapter is very general, since we used no prior knowledge about the biology involved. This fact encourages us to believe that the method could be useful for other data in biology where the representations and basic features extracted might lead to new biological insights. Another possible direction of future research we point to the increasing number of measurements coming from single-cell transcriptomics [57]. In these data, the basic features are conceptually clearer than in protein sequence data and we suspect that representations of cell states in terms of gene expression modules would be found. Such representations could in turn be used to cluster cell types or analyze pathologies like Alzheimer's Disease [58].

Chapter 5

Evolution optimizes Local Entropy

The content of this chapter follow the content of my paper at ref. [59].

Proteins are the machinery of life. In order to perform their functions, the majority of proteins fold into a compact native state that is intimately linked with their polypeptide conformation, as it has been known for thirty years [60]. The goal of this study was to rationalize the observation that the number of protein sequences is largely more abundant than the number of protein conformations, causing substantial degeneracy in the map between sequence and structure. The problem has regained importance in more recent years, when it has become feasible to design proteins *de novo* with custom functions, and thus it has become critical to understand which conformations can be best designed [61].

Most theoretical results point to the conclusion that we can observe a small subset of all possible protein conformations because they exhibit physical properties that make them biologically more fit rather than just because they are poorly sampled by evolution. Some conformations therefore appear to be more "designable" than others [62]. Some works justify the better designability of existing conformations by their greater kinetic accessibility during the folding process, often associated with conformational symmetries such as secondary structures [63, 64, 65]. An optimal balance between local and nonlocal contacts would make the folding rate of some native conformations particularly fast. This hypothesis is supported by the correlation observed in proteins between the average linear separation between residues in contact in the native state and the rate of folding [66]. Other works relate the design of a native conformation to its thermodynamic stability, thus focusing on its equilibrium rather than kinetic characteristics. The basic idea is that stable proteins exhibit a large gap between the energy of the native state and those of competing conformations [67] and this gap can accommodate a large number of sequences that fold to the same native state [68, 69]. Different conformational properties may contribute to this enhanced thermodynamic stability. More compact conformations are more stable because they can exhibit more attractive interactions and because they protect more efficiently hydrophobic residues from the solvent. Daisy-like conformations that maximize the trace of the eighth power of the

contact matrix have also been shown to be particularly stable [70]; in fact, this quantity has been shown to correlate with the evolutionary age of the proteins [71]. Similarly, the presence of loops of specific sizes has been found to improve thermodynamic stability and justified by [72] energy arguments.

Still, the fact that specific protein conformations can be particularly stable at equilibrium is usually associated with the property of amino acid sequences to exhibit markedly low potential energy in those conformations, thus more efficiently minimizing system frustration. On the other hand, although the native state of proteins is usually considered macroscopically unique, its entropy is not negligible compared to the competing denatured state [73]. This entropy arises from the constellation of conformations, structurally similar to the ground state, that lie beyond the transition state. They certainly include several vibrational states, conformational sub-states [74], and perhaps other conformations whose contribution to the partition function cannot be separated from that of the ground state.

The hypothesis we wish to further investigate in the present work is whether the entropy of the native state of natural proteins, and not just the potential energy, is particularly optimized compared to that of random conformations. This hypothesis was suggested several years ago [75] by a qualitative computational analysis in which some of the most mobile dihedrals of small proteins were changed and the number of conformations within 4Å from native conformations and devoid of steric clashes was estimated. It was found that natural proteins exhibit more neighboring conformations than random decoys. More recently, a knowledge–based local–entropy parametrization was used to predict contact changes between amino acids during protein conformational changes [76].

From a statistical–physics point of view, this quantity is captured by the so-called *local entropy*, i.e., the log of the number of low energy configurations within a given distance from a reference configuration (for continuous systems the definition can be generalized straightforwardly). Recently, it has been seen that the notion of local entropy plays a central role in systems that exhibit a potentially very complex energy landscape and at the same time possess highly accessible states that correspond to high local entropy minima [16, 13]. The latter turn out to be accessible by a multitude of dynamical processes which are not designed to have the Gibbs distribution as stationary probability measure, due e.g. to non-thermal external perturbations. Systems of this type are non-convex models of artificial neural networks (including deep neural networks), in which entropic phenomena play essential roles [4] for the (unexpected) success of the current learning processes, largely based on non-equilibrium variants of gradient descent.

Here we want to generalize the algorithmic schemes introduced for sampling high local entropy ground states in neural systems [17] and apply them to simple models of 3D protein structures. Considering both lattice model and all-atom representations of proteins, we show that by sampling native states with high local entropy (which are in principle rare compared to states that dominate the Gibbs measure) we find a decrease in

the linear separation between contact residues. In addition, the "flatness" of the energy profile in the native state can extend to the transition state, having consequences both on the thermodynamic stability of the protein, lowering the free energy of the native state, and on the folding rate, lowering the free energy of the transition state. The generality of the sampling method would allow it to be used in conjunction with any structure prediction method, such as e.g. Alpha–fold [77] and Rosetta Fold [78], and could aid in the search for sequences folding onto the most designable structures.

5.1 Local entropy and protein structures

To define a probability measure that ignores narrow ground states and enhances the statistical weight of large dense regions of ground states, we can consider the local free-entropy

$$S_{\text{loc}}(\Gamma, \gamma, \beta) = \log \int d\Gamma' \exp\left[-\beta U(\Gamma') - \gamma d(\Gamma, \Gamma')\right], \tag{5.1}$$

where $d(\Gamma, \Gamma')$ is any metrics suitable for the protein-structure model under consideration and γ is its conjugate Lagrange multiplier. We describe the explicit choice of d in section 5.2 (see equation 5.7). Here and in the following, we shall set Boltzmann's constant to 1. In the limit of $\beta \to \infty$, this expression reduces (up to an additive constant) to a "local entropy": it counts the number of minima Γ' of the energy, weighing them (via the parameter γ) by the distance to a reference configuration Γ. For continuous variables, the local entropy becomes the log of a weighted volume around a reference configuration. We can then define the probability distribution

$$P(\Gamma; y, \gamma, \beta) = \frac{1}{Z(y, \gamma, \beta)} e^{y S_{\text{loc}}(\Gamma, \gamma, \beta)} \tag{5.2}$$

where y determines the degree of concentration of the probability distribution on high local entropy regions. When y is large, only the configurations Γ that are surrounded by an large number of local minima will have non-negligible weight. By increasing the value of γ, it is possible to focus on tighter neighborhoods around Γ, and at large values of γ the target Γ will also share with high probability the properties of the surrounding minima. From an algorithmic perspective we can use the high local entropy probability distribution as a starting point for designing a Markov Chain, in the same way that simulated annealing uses the Gibbs measure. One possibility [13, 17] is to observe that if we take y to be a non-negative integer we can rewrite the partition function as a product of identical systems connected by a distance constraint

$$\begin{aligned} Z(y, \gamma, \beta) &= \int d\Gamma e^{y S_{\text{loc}}(\Gamma, \gamma, \beta)} \\ &= \int d\Gamma_c \prod_{a'=1}^{y} d\Gamma_{a'} \; e^{-\beta \sum_{a=1}^{y} U(\Gamma_a) + \gamma \sum_{a=1}^{y} d(\Gamma_a, \Gamma_c) - \beta U(\Gamma_c)} \end{aligned} \tag{5.3}$$

where Γ_c is the "central" reference configuration, and $\{\Gamma_a\}$ are the configurations of the replicated systems. This partition function describes a system of $y+1$ interacting replicas of the initial system, one of which acts as the reference system, while the other y systems are subject to a distance constraint with respect to the reference system. This gives us a very simple and general scheme to direct algorithms to sample wide minima of the energy landscape: replicate the model, add an interaction term with a reference configuration, and run the algorithm on the resulting extended system. In practice, we only need to consider the following effective system

$$U_{\text{eff}}(\Gamma_c, \{\Gamma_a\}) = \sum_{a=1}^{y} \left[U(\Gamma_a) - \frac{\gamma}{\beta} d(\Gamma_a, \Gamma_c) \right] + U(\Gamma_c) \tag{5.4}$$

and run our preferred Monte Carlo (MC) Markov Chain update. Replicas are initialized at random, while the center configuration is initialized at the average of the replica configurations. It is worth noting that y controls the value of the local entropy and that relatively small values of y are sufficient to obtain the results we are interested in. By taking the inverse temperature of the individual systems β to be large, we focus the sampling on flat ground states.

5.2 Effects of the local entropy on the equilibrium conformations of polymers

We first tested the effect of controlling the local entropy of a polymer on a simple cubic–lattice model, which offers the advantage of a fast sampling of the conformational space of the system and of defining unambiguously the zero of the entropy. We employed a standard model on a cubic lattice [67] in which the beads, sitting on the vertices of the lattice, cannot overlap. They interact with a contact potential depending on the conformation $\Gamma = \{\vec{x}_i\}_{i=1}^{N}$

$$U(\Gamma) = \frac{1}{2} \sum_{ij=1}^{N} J_{ij} \Delta_{ij}(\Gamma), \tag{5.5}$$

where the contact function $\Delta_{ij}(\Gamma)$ is 1 if the ith and jth beads are neighbors in space and $|i-j| > 2$ and zero otherwise.

The chain is simulated with a standard Monte Carlo algorithm that includes the corner flip, the crankshaft and rotations of the ends as elementary moves. At each step, a monomer is chosen at random for it with flat *a priori* probability and a random move is chosen randomly with uniform probability among those that are possible, accepting it with the standard Metropolis probability. The initial conformation is generated from a random self–avoiding walk in the lattice. The simulations for computing equilibrium quantities consist of 10^7 steps, recording the conformation every 10^4 steps. See Fig. 5.1 for example trajectories that reach equilibrium. Simulations we use to compute the folding time consist of 10^6 steps, recording the conformation every 10^2 steps.

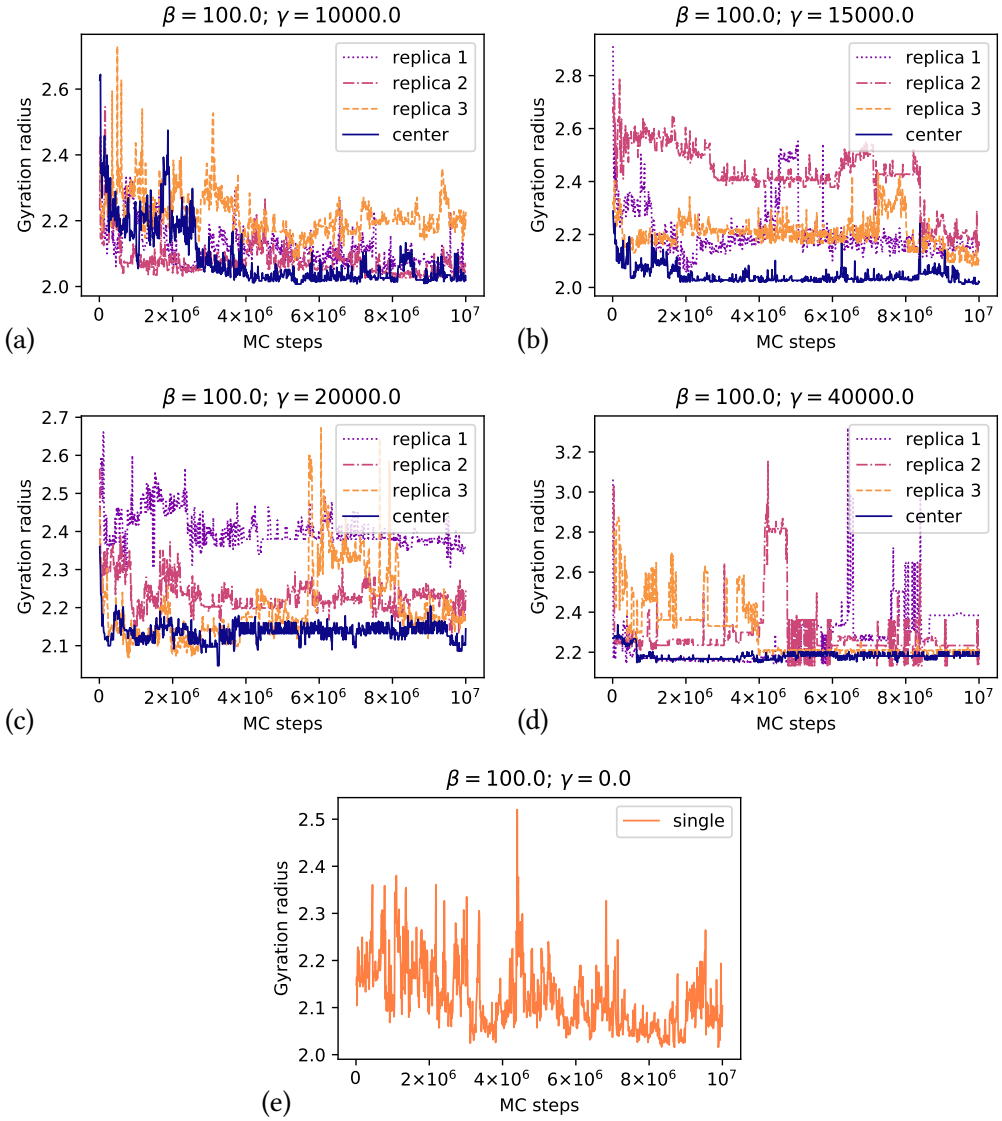


Figure 5.1: **Examples of trajectories of the gyration radius.** (a-d) The solid blue curve corresponds to the gyration radius of the center, while the other curves correspond to the radius of the three replicas. (e) The curve corresponds to the gyration radius of a single configuration obtained with plain Monte Carlo. The radius of gyration is recorder every 10^4 MC steps. To compute the thermal averages of the phase diagram in Fig. 1a of the main text we used the last 300 samples of the trajectory of the center (namely from MC steps $> 7 \cdot 10^6$ to the end of the trajectory). We observe that the radius of the centers has lower variability than the one of plain Monte Carlo and it decreases faster.

In order to control the local entropy of the system, we performed a Monte Carlo simulations of y replicas of the system starting from independent conformations and interacting with the potential defined in equation (5.4). The Metropolis acceptance rate now reads $p_{\text{accept}} = \min\left(1, e^{-\Delta}\right)$, where $\Delta = \beta\Delta E + \gamma\Delta d$ and

$$\Delta d = \begin{cases} d(\Gamma_a^{\text{new}}, \Gamma_c) - d(\Gamma_a^{\text{old}}, \Gamma_c), & \text{if we are moving a replica } a \\ \frac{1}{y}\sum_y [d(\Gamma_a, \Gamma_c^{\text{new}}) - d(\Gamma_a, \Gamma_c^{\text{old}})], & \text{if we are moving the center } c. \end{cases} \tag{5.6}$$

5.2.1 Increased local entropy depletes long–range contacts in homopolymers

The simplest polymer model that can be studied is the lattice homopolymer, obtained setting $J_{ij} = -1$ in equation (5.5) for each pair i, j. We compared the results of a standard Monte Carlo sampling of the Boltzmann distribution with a replicated Monte Carlo, which controls the local entropy through the parameter γ, as described in Sect. 5.1. To define the local free entropy we adopt for this system a distance function defined by

$$d(\Gamma_1, \Gamma_2) = 1 - \frac{1}{N_c}\sum_{i<j}^{N} \Delta_{ij}(\Gamma_1)\Delta_{ij}(\Gamma_2), \tag{5.7}$$

that is the fraction of different contacts between the two conformations Γ_1 and Γ_2; here N_c is the maximum number of contacts that the chain can build. As displayed in Fig. 5.2, we have checked that the results are robust with respect to the distance function (e.g. by comparison with root-mean-square distance, RMDS) and to a different coupling scheme of the replicas. What might be an optimal definition of distance is an interesting problem that goes beyond the scope of our study and that might deserve further study.

In a homopolymer the only relevant equilibrium effect is the coil–globule transition. A comparison between the transition described by a Boltzmann sampling and that obtained controlling the local entropy (cf. Sect. 5.1) is shown in Fig. 5.4a for a polymer with $N = 70$ (see Fig. 5.3 for two example configurations). In the case of Boltzmann sampling ($\gamma = 0$) the coil–globule transition is marked by a jump of the radius of gyration (R_g) at an inverse temperature $\beta \approx 30$. In the lattice model we measure lengths in units of the lattice step, which is set to 1 so that R_g is dimensionless. Increasing the bias associated with the local entropy (i.e., by increasing γ) leads to a stabilization of the globule, the transition temperature increasing with γ. At the same time, we observe a compaction of the coil state, associated with the fact that at short distances more compact conformations have certainly a larger number of neighboring other conformations (cf. equation 5.7) and thus a larger local entropy. The globular phase, in which the polymer is maximally compact, is weakly affected.

In Fig. 5.4b we compared the distributions of contact range (i.e., of the values of $|i - j|$ when i and j are in contact) for fixed inverse temperature $\beta = 100$, at which the system is in the globule phase at all values of γ we simulated. A homopolymeric

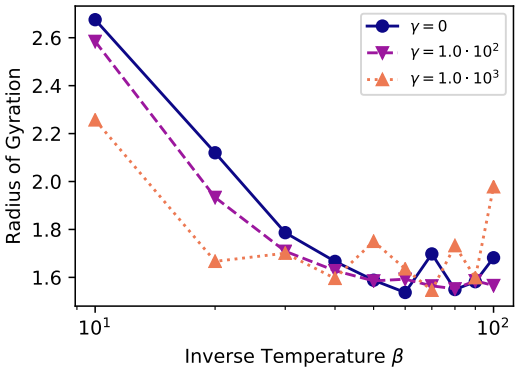


Figure 5.2: **The phase diagram is consistent with respect to choices of distance and replica coupling scheme**. This portion of the phase diagram has been computed using Root Mean Square Distance (RMSD) as distance and by coupling replicas with each other, without a center. RMSD between two polymers is calculated as $d_{\text{RMDS}}(\Gamma, \Gamma') = \frac{1}{N}(\sum_{i=1}^{N} ||\vec{r}_i^{\,\Gamma} - \vec{r}_i^{\,\Gamma'}||^2)^{1/2}$, where N is the number of residues and $\vec{r}_i^{\,\Gamma}$ is the position of the i-th residue of the structure $^{\Gamma}$. The effects of local entropy on the transition with these choices are qualitatively the same as in Fig. 1 of the main text.

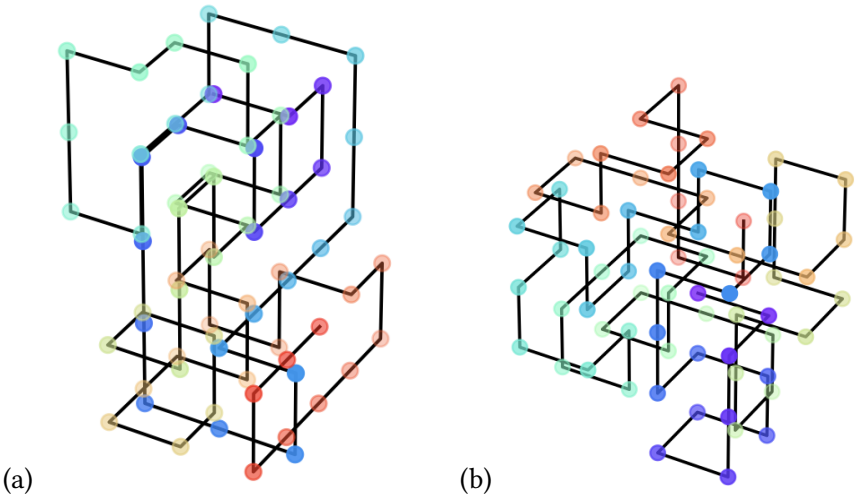


Figure 5.3: **Examples of compact configurations in the lattice model.** Panel (a) is obtained with plain Monte Carlo, panel (b) is obtained by sampling local entropy. The color of the beads represents the index of each bead, from 0 (red) to 70 (purple).

globule is expected to display an initial decrease of the contact probability up to the distance corresponding to the diameter of the globule, followed by a flat distribution [79]. At $\gamma = 0$ the distribution displays overall the expected character but is quite irregular

because of the constraints imposed by the cubic lattice on the R_g of highly–compact conformations (i.e., there is a non–monotonic relation between R_g and energy). Increasing γ, we observed a regular increase of short–range contacts at the expense of long–range contacts.

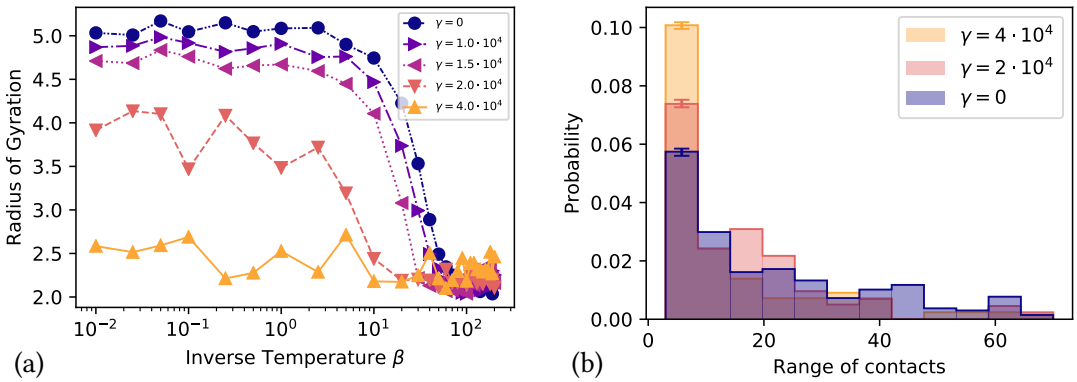


Figure 5.4: **Local entropy changes the properties of a lattice homopolymer.** (a) Radius of gyration as a function of the inverse temperature for increasing values of γ. R_g is dimensionless, since measure lengths in units of the lattice step which is set to 1. The points represent thermal averages. We used $y = 3 + 1$ replicas, the last one being the center. (b) Distribution of the range of contacts $|i - j|$ at $\beta = 100$ for increasing values of γ. Increasing γ suppresses long-range contacts and favors short-range ones. The error bars on the fist bin are generated by bootstrapping.

5.2.2 Increased local entropy simultaneously stabilizes and decreases folding time of model proteins

To obtain a clear picture of the effects of the local entropy on the folding properties of model proteins, we calculated the equilibrium stability and the folding rate of a Go model [80] on lattice. This is defined choosing a target conformation Γ_0 and setting

$$J_{ij} = -J \cdot \Delta_{ij}(\Gamma_0) \tag{5.8}$$

in equation (5.5), where J defines the energy scale and was set to 1. With this choice of J_{ij}, the target conformation is by definition the ground state of the system, and thus the equilibrium state at low temperature. The reason for using a Go model is to decouple the effect of protein sequence from that of protein structure, focusing our attention only on the latter. Here the protein sequence is described effectively, assuming that evolution has minimized energetic frustration to the maximum degree [81].

We chose as target conformation Γ_0 either conformation sampled by the homopolymeric model according to Boltzmann distribution ($\beta = 100$ and $\gamma = 0$) or biasing their

local entropy ($\beta = 100$ and $\gamma > 0$), in all cases taking care of selecting only globular conformations (our choice for the cutoff is $R_g < 2.5$). We thus obtained different potentials that depend on γ through the choice of Γ_0.

We then simulated at low temperature ($\beta = 120$) the dynamics of the system starting from a random, high–temperature, coil state with the standard Metropolis scheme, that at fixed temperature approximates the Smoluchowski equation and thus reports realistic trajectories of the system [82]. From each trajectory we obtained the fraction of native contacts $f_N(t)$ (that for the Go model is $= U(\Gamma(t))/U(\Gamma_0)$, where $\Gamma(t)$ is the conformation of the chain at time t), we calculated the average $\overline{f_N(t)}$ over 40 simulations and fitted them by a two–state kinetics

$$f_N(t) = f_{\text{eq}} \cdot (1 - e^{-t/\tau}) + f_0 \tag{5.9}$$

where τ is the mean folding time, f_0 is the residual fraction of native contacts in the initial conformation, f_{eq} is the equilibrium similarity to the target conformation, that in the two–state approximation is equal to the equilibrium probability of the native state (cf. Fig. 5.5). We show in Table 5.1 the average and standard deviation of the mean square error for each fitted curve.

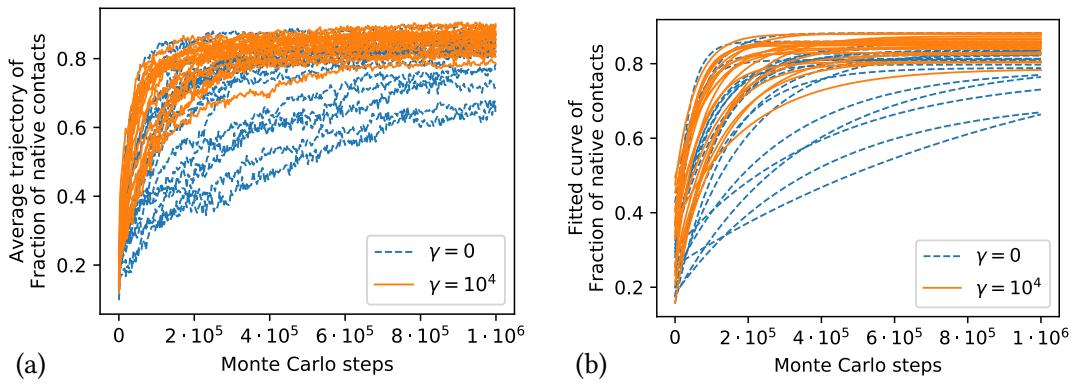


Figure 5.5: **Examples of averaged folding trajectories of lattice Go models and corresponding fitted curves.** The left panel shows trajectories of the fraction of native contacts as a function of Monte Carlo steps. The blue dashed ones correspond to references with $\gamma = 0$, the orange solid ones correspond to references with $\gamma = 1e4$. The right panel shows the corresponding curves obtained by fitting the trajectories with an exponential curve.

In Fig. 5.6a we plotted the values of τ (in MC steps) and f_{eq} as a function of the parameter γ that controls the local entropy of the target conformation Γ_0. The values are medians over 60 realization of Γ_0. The corresponding standard deviations are shown in Table 5.4. It is shown that the stability of the native state increases with γ, while the folding time displays a non–monotonic behavior with a minimum at $\gamma \approx 1.2 \cdot 10^4$.

γ	std(f_{eq})	std(τ)	median(MSE)	std(MSE)
$5.0 \cdot 10^2$	$8.2239 \cdot 10^{-}2$	$1.0669 \cdot 10^3$	$2.7821 \cdot 10^{-}4$	$1.5583 \cdot 10^{-}4$
$2.0 \cdot 10^3$	$1.3406 \cdot 10^{-}1$	$8.9720 \cdot 10^2$	$2.2772 \cdot 10^{-}4$	$1.7510 \cdot 10^{-}4$
$8.0 \cdot 10^3$	$1.1693 \cdot 10^{-}1$	$4.3120 \cdot 10^3$	$1.8888 \cdot 10^{-}4$	$1.0092 \cdot 10^{-}4$
$2.5 \cdot 10^3$	$7.5114 \cdot 10^{-}2$	$2.2077 \cdot 10^3$	$2.3513 \cdot 10^{-}4$	$1.1705 \cdot 10^{-}4$
$5.0 \cdot 10^3$	$1.0108 \cdot 10^{-}1$	$1.2295 \cdot 10^3$	$2.8181 \cdot 10^{-}4$	$2.1203 \cdot 10^{-}4$
$6.0 \cdot 10^3$	$1.0941 \cdot 10^{-}1$	$2.6071 \cdot 10^3$	$2.0890 \cdot 10^{-}4$	$1.2907 \cdot 10^{-}4$
$7.0 \cdot 10^3$	$1.5255 \cdot 10^{-}1$	$1.8627 \cdot 10^4$	$3.0163 \cdot 10^{-}4$	$2.1992 \cdot 10^{-}4$
$9.0 \cdot 10^3$	$1.2876 \cdot 10^{-}1$	$2.1858 \cdot 10^3$	$2.0844 \cdot 10^{-}4$	$2.2282 \cdot 10^{-}4$
$1.0 \cdot 10^4$	$9.0906 \cdot 10^{-}2$	$3.8539 \cdot 10^2$	$2.4447 \cdot 10^{-}4$	$1.1748 \cdot 10^{-}4$
$1.1 \cdot 10^4$	$8.6400 \cdot 10^{-}2$	$4.3558 \cdot 10^2$	$2.0593 \cdot 10^{-}4$	$1.5520 \cdot 10^{-}4$
$1.2 \cdot 10^4$	$1.0607 \cdot 10^{-}1$	$3.9646 \cdot 10^2$	$2.6716 \cdot 10^{-}4$	$1.1724 \cdot 10^{-}4$
$1.3 \cdot 10^4$	$1.3475 \cdot 10^{-}1$	$6.4258 \cdot 10^2$	$2.3607 \cdot 10^{-}4$	$1.8014 \cdot 10^{-}4$
$1.4 \cdot 10^4$	$7.7247 \cdot 10^{-}2$	$3.8472 \cdot 10^2$	$3.0565 \cdot 10^{-}4$	$1.3042 \cdot 10^{-}4$
$1.5 \cdot 10^4$	$9.7204 \cdot 10^{-}2$	$2.9444 \cdot 10^2$	$2.4793 \cdot 10^{-}4$	$2.0832 \cdot 10^{-}4$
$2.0 \cdot 10^4$	$6.7892 \cdot 10^{-}2$	$7.8561 \cdot 10^2$	$2.8288 \cdot 10^{-}4$	$1.4823 \cdot 10^{-}4$
$3.0 \cdot 10^4$	$8.9116 \cdot 10^{-}2$	$6.2010 \cdot 10^2$	$2.8535 \cdot 10^{-}4$	$1.7384 \cdot 10^{-}4$

Table 5.1: **Folding times and stabilities are less variable for higher values of γ.** The first two columns of the table show standard deviation of stabilities and folding times shown in Fig. 2a of the main text. The last two columns show median and standard deviation of mean square error of fits averaged over the samples, attesting the goodness of the fits.

To rationalize these results, we estimated what is the radius of the neighborhood of the target conformation that is affected by the increase in local entropy at varying γ. In Fig. 5.6b we plotted the average inter–conformation distance obtained from the replica simulation from which we obtained the conformations Γ_0 (cf. equation 5.4) as a function of γ. For large γ ($> 1.5 \cdot 10^4$) the average distance displays a plateau at $d \approx 0.6$; decreasing γ the average distance increases, overcoming 0.9 for plain Monte Carlo simulations. In the plot is also marked the transition state at $d = 0.7$, that separates the denatured state from the native basin.

Both the stabilization and the kinetic effects of the local entropy can be clarified noting what part of the free–energy profile $F(d) = E(d) - TS(d)$ is affected by the local entropy. At large γ the local entropy affects only the close neighborhood of the Γ_0 ($d \lesssim 0.4$), decreasing its free energy and thus stabilizing it with respect to the denatured state, which is unaffected. The transition state is not affected as well, so the folding rate, that according to Kramers theory depends on the free energy of the transition state calculated with respect to the denatured state, is similar to that at $\gamma = 0$. When γ is decreased, the neighborhood of Γ_0 affected by the increase in local entropy reaches

the transition state ($d \simeq 0.7$) and lowers it, decreasing the folding time. The non–monotonicity of the folding time arises from the fact that making γ even smaller, also the denatured state is affected and then folding barrier grows again.

The degree of cooperativity of the folding transition decreases slightly with γ (see Table 5.2). This is not unexpected because $\gamma > 0$ leads to the stabilization of conformations with varying distance from the native one, thus decreasing the two-state character of the folding transition.

γ	cooperativity κ
0.0	1.21 ± 0.01
$0.5 \cdot 10^4$	1.35 ± 0.04
$1.0 \cdot 10^4$	1.28 ± 0.03
$2.0 \cdot 10^4$	1.42 ± 0.07

Table 5.2: **Local entropy leads to less cooperative transitions.** The table shows the estimated average cooperativity of the coil globule transition for go models on references at various values of γ. The cooperativity is defined with the Privalov coefficient κ, that is 1 in case of perfect first-order transition and greater than one the less cooperative the transition is. The definition is $\kappa := \int dT C_v(T)/2T_f\sqrt{C_v(T)}$, where T_f is the folding temperature and C_v is the specific heath of the transition.

5.3 The local entropy of natural proteins is larger than that of random decoys

We then tested the hypothesis that the native state of natural proteins displays a larger local entropy than random polypeptidic conformations with the same density. We studied seven natural proteins and a stable protein designed *de novo* (HHH) [61].

To estimate the local entropy associated with the native state we need to describe the energy of the protein in the neighborhood of the crystallographic structure. For this purpose we made use of an all–atom Go model [80], that is expected to be particularly realistic in the native basin. At the same time, it allows one to decouple the effect of the sequence from that of the native conformation in the calculation of the local entropy.

In order to evaluate the local entropy, we performed molecular dynamics (MD) simulations with Gromacs 2020.4 [83] using the all–atom Go model obtained by Smog2 [84]. The native conformations of the proteins were: protein G (pdb code 1pgb, 56 residues), ACBP (pdb code 2abd, 87 residues), CI2 (pdb code 2ci2, 83 residues), src-SH3 (pdb code 1srl, 64 residues), villin headpiece (pdb code 5vnt, 63 residues), barnase (pdb code 1bnr, 110 residues) and HHH (artificial protein, pdb code 5uoi, 43 residues). MD simulations were performed in the range of temperature from 1 to 200 (in energy units) for $2 \cdot 10^5$ steps of time step $5 \cdot 10^{-4}$ ps with stochastic dynamics [83]. The Go potential

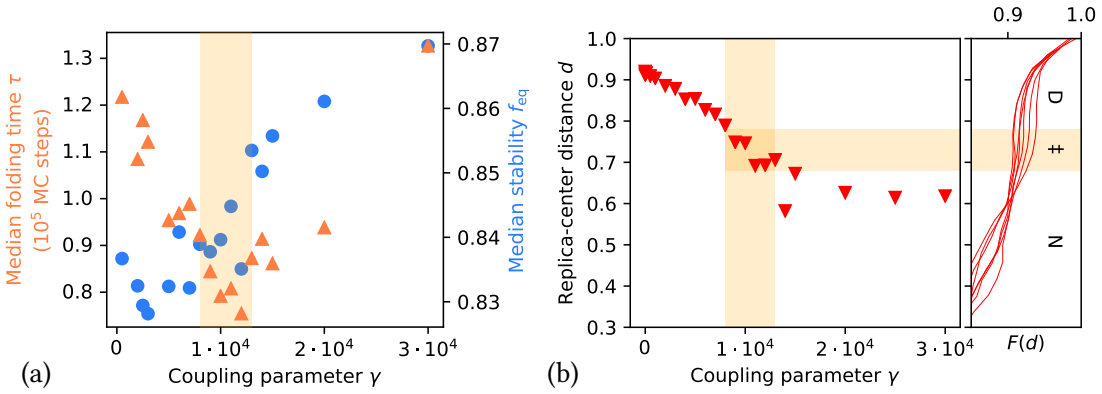


Figure 5.6: **Structures with high local entropy are more stable and fold faster.** (a) The average folding time (in MC steps, orange triangles, left y-axis) for target conformations Γ_0 obtained at different values of γ and the average stability of the structure (blue circles, right axis). The orange bar indicates the transition state ‡ calculated in the right panel. (b) The average distance between the central conformation and the replicas at different values of γ obtained from the simulation used to generate the Γ_0. Aligned with the right axis we show the free energy of a Go model as function of the distance from the native state for various structures at low γ. Both the distance and the free energy are dimensionless given the definition of the model. We can see that the region around $\gamma = 1 \cdot 10^4$, marked with an orange bar, corresponds to the typical distance between native and transition state.

of each decoy, whose parameters in Smog2 depend on the number of residues and of native contacts in the native conformation, is rescaled to that of the corresponding native protein in order to facilitate the comparison among them. The microcanonic entropy $S(E)$ is extracted from all the simulations performed at different temperatures for the same protein with the maximum–likelihood code developed in ref. [85].

As a control model we generated putative native conformations from a homopolymeric model, derived from the original models as follows. Starting from the crystallographic structure of each protein, we generated an all-atom model similar to that described above (i.e., with the same atomic structure), but where each pair of atoms interacts instead in the same way with the Lennard–Jones potential

$$V_{\mathrm{LJ}}(r_{ij}) = C_{ij}^{(12)}/r_{ij}^{12} - C_{ij}^{(6)}/r_{ij}^{6} \tag{5.10}$$

where for all i, j we set $C_{ij}^{(6)} = 1.4{\cdot}10^{-2}\,\mathrm{kJ\,mol^{-1}\,nm^6}$ and $C_{ij}^{(12)} = 1.0{\cdot}10^{-4}\,\mathrm{kJ\,mol^{-1}\,nm^6}$. No potential is applied to the dihedrals at this stage. The parameters $C^{(6)}$ and $C^{(12)}$ are chosen with a grid search so that, after an annealing MD of the chain, the number of contacts in the putative conformations is not smaller than in the original model (see Fig. 5.7), to rule out trivial effects in the calculation of the local entropy. The resulting

conformation is used as putative Γ_0 (examples of these configurations can be found in Fig. 5.8).

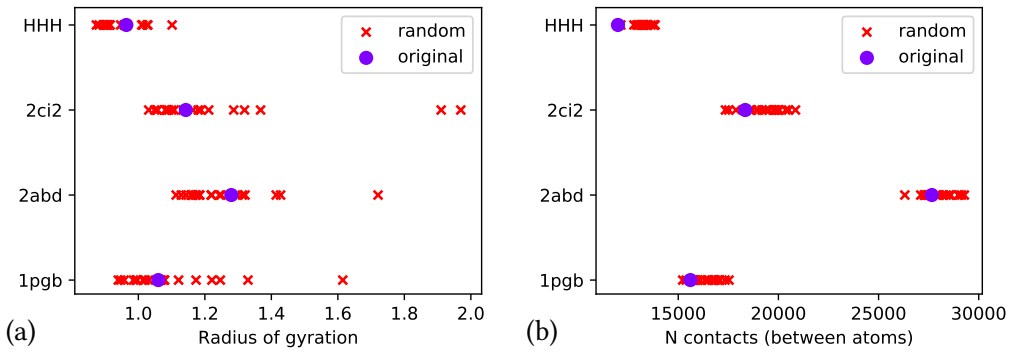


Figure 5.7: **The decoys have similar compactness than the native structures.** We show radius of gyration and number of atomic contacts for the proteins under study and for the associated random decoys. In order to obtain this distribution of properties for decoy configurations we set $C_{ij}^{(6)} = 1.4 \cdot 10^{-2}\,\mathrm{kJ\,mol^{-1}\,nm^6}$ and $C_{ij}^{(12)} = 1.0 \cdot 10^{-4}\,\mathrm{kJ\,mol^{-1}\,nm^6}$ in the Lennard-Jones potentials between pairs of atoms.

The entropy $S(E)$ for four proteins is displayed in Fig. 5.9(c–f) (see also Fig. 5.10) and compared with the random decoys of each of them, matching the different curves at infinite temperature (cf. the insets). In most cases, proteins display in the native energy region (below the transition state) an entropy that is larger than that of the random decoys. This effect is summarized Fig. 5.9(a), that displays the density of local entropy

$$s_{\mathrm{loc}} = \frac{1}{N} \log \sum_{E=E_{\mathrm{N}}}^{E_{\ddagger}} e^{-\beta E + S(E)}, \tag{5.11}$$

where E_{N} is the minimum energy of the system and $E_{\ddagger}$ is the energy of the transition state, approximated as the average energy at the transition temperature (cf. caption of Table 5.3). The local entropy is calculated at low temperature ($\beta = 10^{-1}$) at which all proteins and decoys are stable (cf. Fig. 5.11). Equation (5.11) is the microcanonical counterpart of equation (5.1). The density of local entropy of native proteins is always larger than the average $\overline{s}$ of the decoys, in five cases out of eight for more than one standard deviation σ_s. From $\overline{s}$ and σ_s we estimated the p–values associated with the null hypothesis that the entropy density of the native protein is smaller than that of the decoys within a Gaussian approximation, that is $p = [1 - \mathrm{erf}((s - \overline{s})/\sqrt{2}\sigma_s)]/2$, where erf is the error function. The p–values are rather low, except for CI2 (2ci2) and for the CHE–Y (1cye).

The protein HHH, designed *de novo* and not shaped by natural evolution, seems to display the same features of natural proteins. However, one has to consider that the

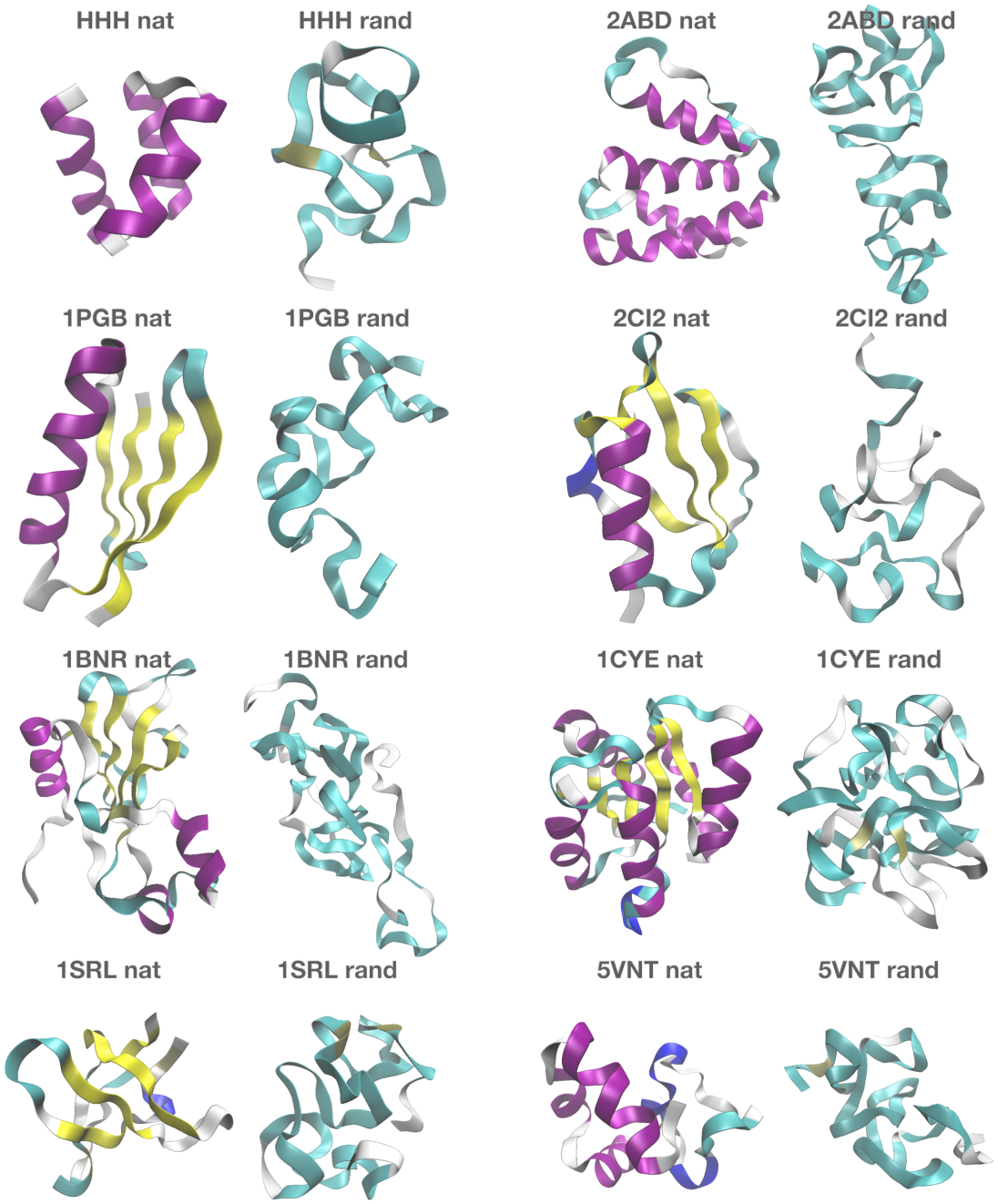


Figure 5.8: **Examples of compact configurations in the all-atom model.** Different colors correspond to secondary structures, showing that random decoys show little to no secondary content.

scaffold they used for the design is a helix bundle typical of natural proteins.

Finally (cf. Fig. 5.11), we observed that the peaks in the specific heat of the four

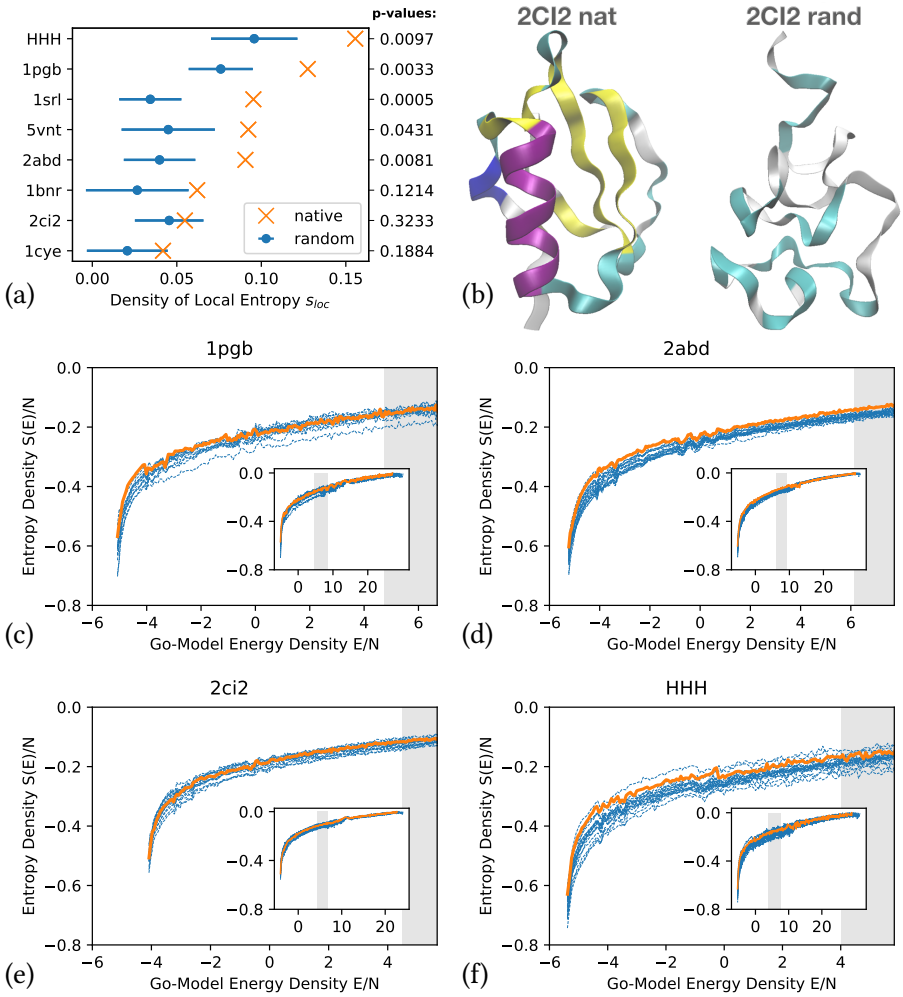


Figure 5.9: **Natural structures have higher local entropy than random ones at low energy.** (a) The density of local entropy of the native state (orange crosses), the average density of local entropies of the decoys (blue circles) and the associated standard deviation (blue bars) for each protein. On the right axis are reported the associated p–values. (b) the native conformation of CI2 (2ci2) and an example of decoy. (c–e) The entropy density calculated as a function of energy for four proteins (solid orange curves) and for the associated decoys (dashed blue curves). The gray vertical bars indicate the region where the transition states are (the band is centered on the average over the protein and the decoys, its width is twice a standard deviation). In the inset is displayed the whole entropy density, while in the main figures only the region of the native state. Note the energy density has been rescaled for every structure as described in section 5.3, so the energy units are arbitrary.

proteins tend to be wider and less pronounced, in agreement with what we found in

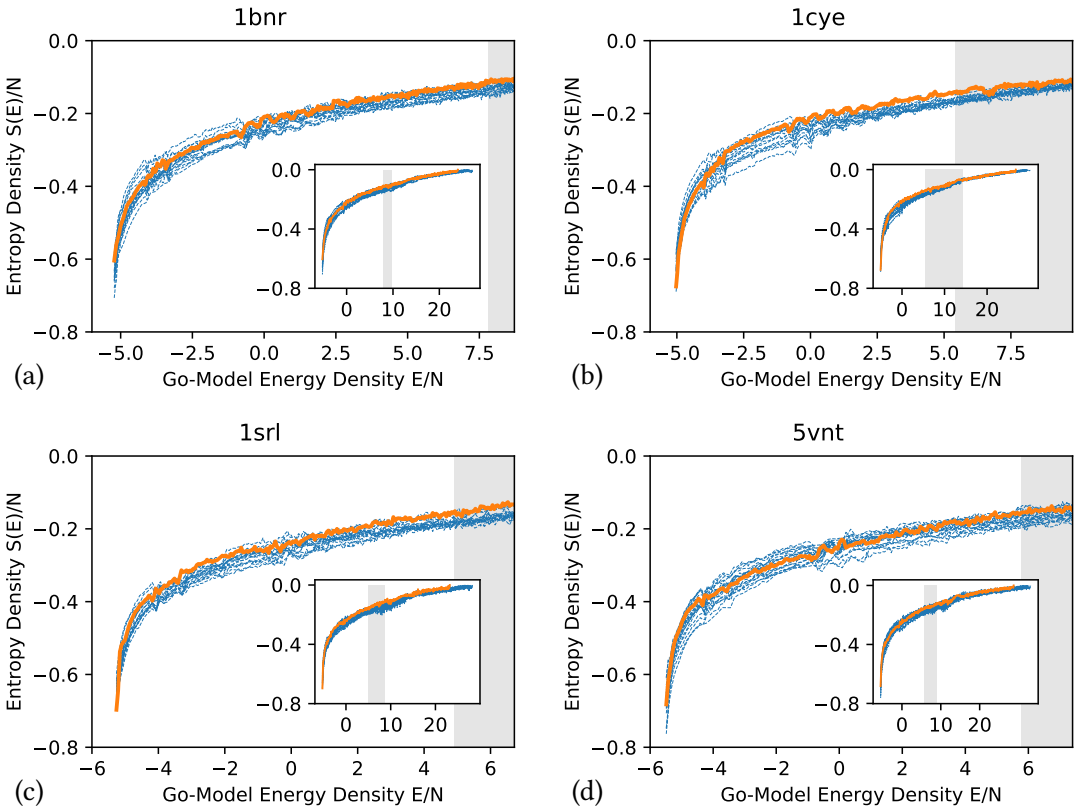


Figure 5.10: Entropy density calculated as a function of energy for four proteins (solid orange curves) and for the associated decoys (dashed blue curves), for the remaining four proteins.

subsection 5.2.2: the wider the specific heat, the less cooperative the transition [86] (a measure of cooperativity of the transition can be found in Table 5.2). At the same time, the majority of native proteins display a larger folding temperature (cf. the main peak in the specific heat in Fig. 5.11), in agreement with the fact that a larger local entropy stabilizes the native state.

5.4 The local entropy depends on the topology of contacts

Finally, one can investigate whether there is an elementary way to reinterpret why natural proteins display a larger local entropy than random conformations. This aspect can be studied easily in the context of a Go model in which the frustration associated with the sequence is minimized [81].

In fact, in this case one can insert in the definition of local entropy of equation (5.1)

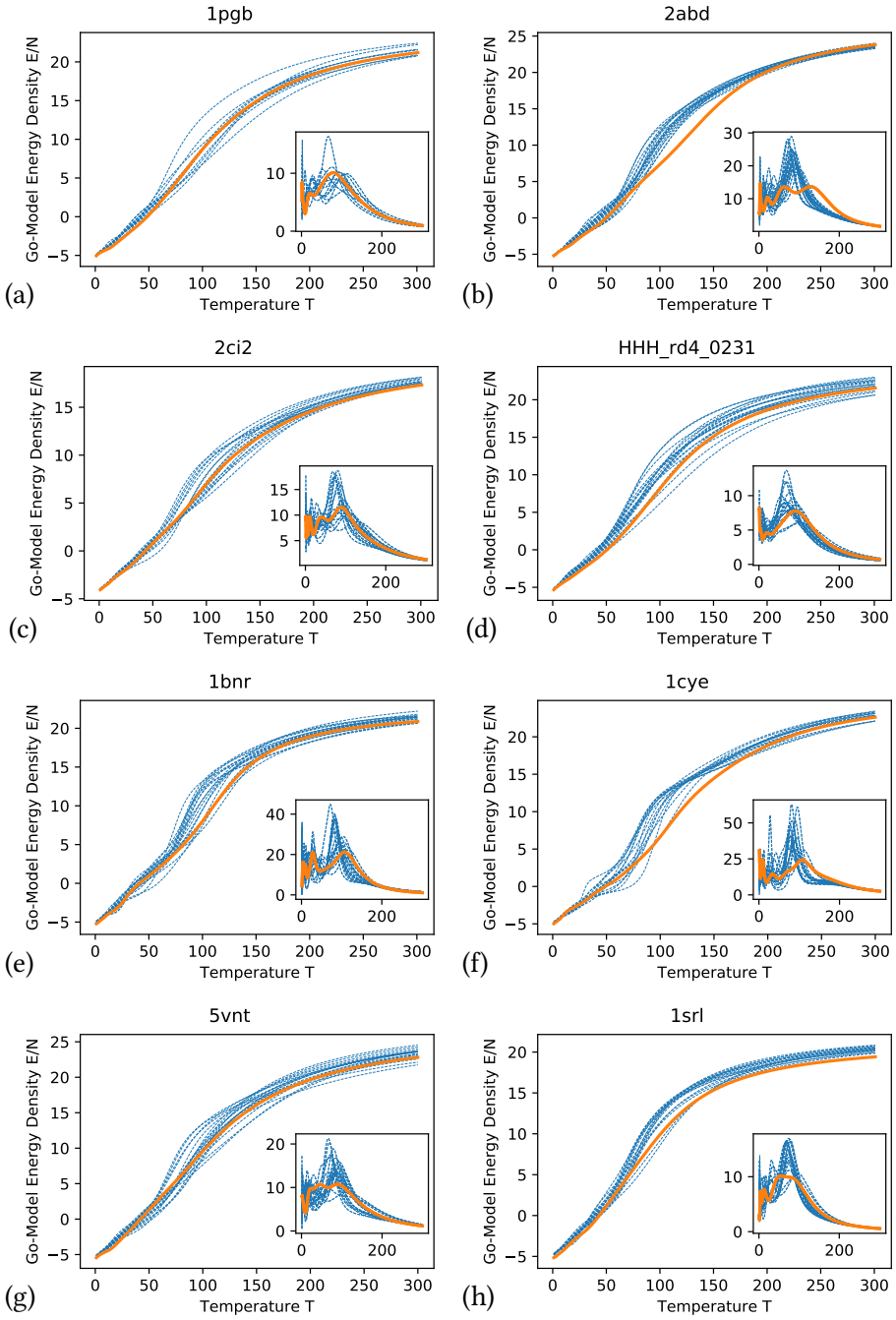


Figure 5.11: **Thermodynamics of all-atom models.** Each plot show the internal energy and the specific heat (inset) as a function of the temperature for the four proteins we studied (solid orange curves) and for their random decoys (dashed blue curves).

the expressions of equations (5.5), (5.7) and (5.8), obtaining

$$S_{loc}(\Gamma_0) = \tag{5.12}$$
$$= \log \int d\Gamma \exp\left[\left(\beta J + \frac{111\gamma}{N_c}\right) \sum_{i<j}^{N} \Delta_{ij}(\Gamma_0)\Delta_{ij}(\Gamma)\right],$$

Protein	Mean(E_{tr})	Var(E_{tr})	Mean(T_{tr})	Var(T_{tr})
1pgb	6.69	1.95	89.16	18.76
2abd	7.73	1.60	88.76	14.75
2ci2	5.67	1.18	88.43	14.80
HHH	5.84	1.83	77.45	14.84
1bnr	8.72	0.92	92.74	12.06
1cye	9.81	4.40	107.24	32.00
5vnt	7.40	1.65	83.81	12.13
1srl	6.72	1.82	72.88	9.95

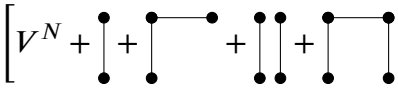
Table 5.3: **Average transition temperatures and corresponding transition energies.** We identified transition temperatures for each native and random structure as the position of the first peak in the specific heat (coming from high temperatures). The corresponding transition energy is determined with the energy curves shown in Fig. 5.11. We average those energies for each protein to determine the region where the transition states are in Fig. 3c-f in the main text. This "transition region" is plotted as a gray band.

where $d\Gamma \equiv dx_1 dx_2 \ldots$ and an immaterial constant has been disregarded. This is essentially the free energy of the Go model at a rescaled temperature. The exponential in the integrand can be expanded in series,

$$e^{\cdots} = 1 + \left(\beta J + \frac{\gamma}{N_c}\right) \sum_{i<j}^{N} \Delta_{ij}(\Gamma_0)\Delta_{ij}(\Gamma) + \tag{5.13}$$
$$+ \frac{1}{2}\left(\beta J + \frac{\gamma}{N_c}\right)^2 \sum_{\substack{i<j \\ k<l}}^{N} \Delta_{ij}(\Gamma_0)\Delta_{kl}(\Gamma_0)\Delta_{ij}(\Gamma)\Delta_{kl}(\Gamma) + \ldots$$

Moving the $\Delta_{ij}(\Gamma_0)$ out of the integral and defining the interaction volume as $v \equiv \int d\Gamma \Delta_{ij}(\Gamma)$ and the total volume available to each degree of freedom as V, one obtains

$$S_{loc}(\Gamma_0) = \log\left[V^N + \right. \tag{5.14}$$



$$\left. + \ldots\right],$$

where the graphs indicate the terms of the expansions. For example,

$$= \left(\beta J + \frac{\gamma}{N_c}\right) V^{N-1} v \sum_{i<j}^{N} \Delta_{ij}(\Gamma_0) \tag{5.15}$$

is the contribution of a single contact, that depends on the number $\sum \Delta_{ij}(\Gamma_0)$ of contacts of the native conformation. Thus, the larger is the number of contacts on the native

conformation, the larger is the local entropy. Similarly,

$$= \frac{1}{2}\left(\beta J + \frac{\gamma}{N_c}\right)^2 V^{N-2} v^2 \sum_{i<j<k}^{N} \Delta_{ij}(\Gamma_0)\Delta_{jk}(\Gamma_0), \tag{5.16}$$

where $\sum \Delta_{ij}(\Gamma_0)\Delta_{jk}(\Gamma_0)$ is the number of triples of nodes (e.g., amino acids) interacting pairwise. For a graph with loops, for example,

$$= \frac{1}{3!}\left(\beta J + \frac{\gamma}{N_c}\right)^3 V^{N-2} A_{\triangleright} v^2 \sum_{i<j<k}^{N} \Delta_{ij}(\Gamma_0)\Delta_{jk}(\Gamma_0)\Delta_{ki}(\Gamma_0), \tag{5.17}$$

where $A_{\triangleright} = 3/4$ is a parameter that arise in the integration of the contact functions Δ from the constraints given by looped graphs. In fact,

$$\int dx_i dx_j dx_k \Delta(|x_i - x_j|)\Delta(|x_j - x_k|)\Delta(|x_k - x_i|) = V \cdot \frac{3}{4} v^2. \tag{5.18}$$

For a generic graph, $A \leq 1$ and is equal to the unity if the graph does not contain loops because the variables can be integrated sequentially. For a fully–connected graph of n nodes, that in the language of network theory is called a clique, $A = n/2^{n-1}$.

By the linked cluster theorem, the sum of graphs in equation (5.13) is the exponential of the sum of connected graphs, so the local entropy results simply the sum of connected graphs. Note that the connected graphs are different from zero only if the associated $\sum \Delta_{ij}(\Gamma_0)\Delta_{jk}(\Gamma_0)\ldots$ is different from zero, that is the corresponding structure is in the native protein. The goal is to spot what are the most important graphs. The general form of a graph is

$$\frac{A}{l!}\left(\beta J + \frac{\gamma}{N_c}\right)^l V^{N-n+1} v^{n-1} \cdot (\#\text{ of instances in } \Gamma_0), \tag{5.19}$$

where n is the number of interacting nodes, l is the number of links (i.e., interactions between nodes) and A depends on how links loop together. Each term is also proportional to the number of instances that the specific graph appears in the native conformation of the protein.

Proteins will display a large local entropy if they are rich of graphs with large values of

$$w \equiv \frac{A}{l!} B^l V^{N-n+1} v^{n-1}, \tag{5.20}$$

where we defined $B \equiv \beta J + \gamma/N_c$. Thus, local entropy depends on the topology of native contacts. For unlooped graphs ($A = 1$), for fixed n, w has a maximum at $l^* = B$. Typically, for proteins βJ is of the order of 1; γ/N_c for the model of Fig. 5.6b is of the order of $10^4/10^2 = 10^2$, and consequently $l^* \sim 10^2$.

However, for graphs composed of l^* links, fully–connected graphs display a larger weight w. In fact, if we compare the value of w_{clique} for a fully connected graph ($l \sim n^2$)

with that w_{unloop} of an unlooped graph ($l \sim n$) at $l = B$, one obtains $w_{\text{clique}}/w_{\text{unloop}} = (V^N/v)^{B/2} B^{-1/2}$ that is large for $B \gg 1$.

It was shown [70] that $\mathrm{Tr}\,[\Delta_{ij}(\Gamma_0)]^n$ for $n \gg 1$ is a good determinant of protein designability. Since this quantity counts the number of closed paths of length n in the interaction network of the protein, it will be correlated with the number of clusters with large w; consequently, the tendency of evolution to maximize $\mathrm{Tr}\,[\Delta_{ij}(\Gamma_0)]^8$ [71] is tantamount to the optimization of the local entropy. Notice that for the lattice–model proteins discussed above, the average values of $\mathrm{Tr}\,[\Delta_{ij}(\Gamma_0)]^n$ with $n = 4{,}8$ are increasing with γ, that is with the bias towards having high entropy at short distances (cf. Table 5.4).

γ	Tr M^4	Tr M^8	cont. range
0.0	784.4 ± 0.8	41150 ± 90	19.28 ± 0.05
$0.5 \cdot 10^4$	790.3 ± 0.9	41524 ± 99	19.80 ± 0.05
$1.0 \cdot 10^4$	821.3 ± 0.9	44814 ± 112	17.96 ± 0.05
$1.5 \cdot 10^4$	794.7 ± 0.9	42511 ± 98	16.81 ± 0.04

Table 5.4: **Properties of the contact map M of lattice structures.** The table shows the fourth and eighth power of the contact map and the mean contact range as a function of γ. The quantities are averaged over 60 structures.

5.5 Conclusions

The concept of local entropy was used to explain the properties of complex systems like artificial neural networks [16, 13], in which the "energy" landscape is highly roughed and the system learning dynamics leads to atypical configurations with respect to the Gibbs measure. In particular, states characterized by a large local entropy can be easily accessed in spite of the abundance of competing states that tend to block the learning procedure.

In the present work we claimed that the concept of local entropy plays an important role also in systems that can move in a smoother energy landscape and thus are not in a glassy regime. In particular, we focused our attention on the three–dimensional conformation of proteins, that evolved along the eons through a complex dynamics described by Darwinian evolution. The high diversity of native conformations of known proteins and their redundancy (i.e., the existence of analogous proteins) suggests that evolution explored a large part of conformational space and thus that protein conformations can be regarded as stationary realizations of the evolutive dynamics.

In the present work, we made use of a Go model to estimate the local entropy of proteins, in order to separate the evolutionary problem of sequence design from that of conformational selection, removing in this way frustration from the system [87].

Although it is known [88] that frustration plays an important role in determining the features of the denatured state up to the formation of the transition state, one expects it to be less relevant to determine the local properties of the native state and thus the estimation of its local entropy, in agreement with the principle of minimal frustration of the native state [81]. We thus think that the Go model is particularly suitable for the specific problem we face.

A relevant question is then whether the ensemble of native conformations can be described by Boltzmann statistics or the evolutionary accessibility of native conformations is important to define the fitness of proteins. The latter case would imply that out-of-equilibrium effects affect the set of existing protein conformations. In the cases we studied, the local entropy of proteins is indeed larger than that of random decoys displaying the same length and density. As already mentioned in the introduction, these ideas are not new. However, our aim was to provide a simple proof of concept with a novel technique that can be generalized to more realistic settings for protein design. While previous estimation of the local entropy are either qualitative [75] or based on a knowledge–based, empirical function [76], the methods we suggest inspired by the study of complex systems [16] make use of a direct calculation of the local entropy. Being based on simple statistical–mechanical concepts, we expect this method to be widely applicable to different class of models.

In fact, we showed that the large local entropy of model proteins has consequences both on their thermodynamic and equilibrium properties. It straightforwardly stabilizes the native state, decreasing its free energy, but extending to the transition state can also improve the folding rate. From this point of view, the concept of evolutionary optimization of the local entropy is related to the dynamical variational principle stated in ref. [65]. In fact, also the mechanistic reason for the increase in the folding rate is the same: local entropy biases the stabilizing contacts to be closer along the sequences, making their formation kinetically faster [66].

Code availability The scripts and the main data used and produced in this work can bee freely downloaded at https://gitlab.com/bocconi-artlab/local-entropy-proteins.

Part III

Conclusions

Chapter 6

Discussion and future perspectives

In this thesis we studied the effects of optimizing local entropy in a wide variety of systems, each of those farther form the context where the concept was originally developed, namely simple neural networks trained on supervised problems.

We found that looking for wide flat minima in neural networks is relevant even when the problem is convex while the naive optimization of the loss is not sufficient to generalize well (chapter 3).

We also showed that wide solutions of sparse autoencoders implement representation of data with different properties than typical solutions, and we saw that these representations are closer to the actual structure of data when such structure is known (chapter 4). We explored some real-world applications of this improved feature extraction on amino-acid sequences of homologous proteins families.

Finally, we abandoned neural networks and applied the concept of local entropy to the problem of protein folding. In particular, we disentangled the study of the sequence from that of the three-dimensional structure and showed how structures that maximize local entropy seem to be better suited from the point of view of evolutionary fitness (chapter 5).

This last model is particularly interesting for several reasons. First, it is the first finite-dimensional model where local entropy is studied, and there were no a priori reasons for it to be effective. Second, if a similar study is conducted on more realistic models on proteins (for example using the recent deep learning tools [78, 77] to bridge the gap between the structures and the sequences), it could be a case of a real-world (meaning not from computer science) system where physics of subdominant states plays an important role.

Moreover, the connection between learning algorithms and evolution might be deeper than just both solving optimization problems: this connection resides in neutral theory.

The neutral theory of protein evolution (see [89]) starts from the observation that most of the mutations that can happen to a protein sequence are *neutral*, namely they do not change the fitness of the sequence. A consequence of this fact is that most of the volume of mutations is due to random drift rather than selective pressure. This fact is

related to the designability (sometimes called *evolvability*) of a protein, since a sequence that accepts more neutral mutations is less prone to destructive mutations, therefore is more stable [90]. On the other hand, a protein *structure* that hardly changes – i.e. a more *robust* structure – is less probable to reach beneficial mutations and therefore seems a pitfall for the evolution process.

The interplay between robustness and designability of protein structures has been clarified in [91], by separating the two concepts for the genotype (the sequences) and the phenotype (the structures): the author showed that robust structures possess a large neutral network, which is the set of all sequences forming the same structure and connected by neutral mutations. They also showed that a large neutral network improves designability, because a sequence folding to such structure can change a lot before producing a new phenotype and therefore as access to a bigger set of different phenotypes, with this set proportional to the size of the neutral network.

It is clear that natural sequences are very different from random ones [92]. Can local entropy describe the structures that are good for evolution as if the evolution was an algorithm that can only visit particularly accessible structures, in analogy with neural networks? What dynamics do we expect to be involved in the process of evolution?

From the point of view of sequences, we don't expect to be a process of retaining neutral or beneficial mutation to be a stochastic process that reached equilibrium, just because it has been estimated that evolution only explored a fraction of the possible sequences (see [93]). It is plausible that evolution can only visit those sequences that are well-connected via neutral paths to the root of the evolutionary tree. In this picture, a particularly fit structure that is isolated in the fitness landscape effectively does not exist for evolution.

From the point of view of structures, we expect that a protein should be able to fold in a variety of situations: some in vivo examples are the folding that happens as soon as the amino-acid chain is assembled reading the RNA, or folding reactions that are catalyzed by enzymes in the crowded cell environment; moreover, a protein folds also in vitro inside pure solutions The requirement to be a good folder in many different situations could create an evolutionary pressure for the structures to be almost "universal folders", namely to be the ending point of the different dynamics that can come up inside a cell, in analogy with high-local-entropy states being attractors for many of the heuristic algorithms that are used to train neural networks.

In this thesis we showed that good folders show higher local entropy than random structures, but whether this effect is due to evolution optimizing it or due to evolution happening in the high-local-entropy subset of the sequence space is yet to be understood.

To wrap up this discussion, another way of studying the role of local entropy in protein folding would be to ask if local entropy could be a null model for natural selection by capturing the geometrical requirements (see [65, 94]) of the fitness score, which cause the existence of neutral networks.

A theory that relies on similar hypotheses have been formulated using the concept

of Kolmogorov complexity, both for genotype-phenotype maps ([93, 95, 21]) and neural networks ([21, 20]). A study of the connection between local entropy and algorithmic complexity could be an interesting development of this thesis.

Bibliography

[1] Pratik Chaudhari et al. "Entropy-sgd: Biasing gradient descent into wide valleys." In: *Journal of Statistical Mechanics: Theory and Experiment* 2019.12 (2019), p. 124018.

[2] Fabrizio Pittorino et al. "Entropic gradient descent algorithms and wide flat minima." In: *arXiv preprint arXiv:2006.07897* 2021.12 (2020), p. 124015.

[3] Fabrizio Pittorino et al. "Deep Networks on Toroids: Removing Symmetries Reveals the Structure of Flat Regions in the Landscape Geometry." In: *arXiv preprint arXiv:2202.03038* (2022).

[4] Carlo Baldassi, Fabrizio Pittorino, and Riccardo Zecchina. "Shaping the learning landscape in neural networks around wide flat minima." In: *Proceedings of the National Academy of Sciences* 117.1 (2020), pp. 161–170.

[5] Carlo Baldassi et al. "Wide flat minima and optimal generalization in classifying high-dimensional Gaussian mixtures." In: *Journal of Statistical Mechanics: Theory and Experiment* 2020.12 (2020), p. 124012.

[6] Carlo Baldassi et al. "Learning through atypical"phase transitions"in overparameterized neural networks." In: *arXiv preprint arXiv:2110.00683* (2021).

[7] Carlo Baldassi et al. "Unveiling the structure of wide flat minima in neural networks." In: *Physical Review Letters* 127.27 (2021), p. 278301.

[8] Andreas Engel and Christian Van den Broeck. *Statistical mechanics of learning*. Cambridge University Press, 2001.

[9] E Gardner and B Derrida. "Optimal storage properties of neural network models." In: *Journal of Physics A: Mathematical and General* 21.1 (Jan. 1988), pp. 271–284.

[10] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning." In: *nature* 521.7553 (2015), p. 436.

[11] David JC MacKay, David JC Mac Kay, et al. *Information theory, inference and learning algorithms*. Cambridge university press, 2003.

[12] Jérôme Tubiana and Rémi Monasson. "Emergence of compositional representations in restricted Boltzmann machines." In: *Physical review letters* 118.13 (2017), p. 138301.

[13] Carlo Baldassi et al. "Unreasonable effectiveness of learning neural networks: From accessible states and robust ensembles to basic algorithmic schemes." In: *Proceedings of the National Academy of Sciences* 113.48 (2016), E7655–E7662.

[14] Luca Saglietti. "Out of Equilibrium Statistical Physics of Learning." PhD thesis. Politecnico di Torino, 2018.

[15] Haiping Huang and Yoshiyuki Kabashima. "Origin of the computational hardness for learning with binary synapses." In: *Physical Review E* 90.5 (2014), p. 052813.

[16] Carlo Baldassi et al. "Subdominant Dense Clusters Allow for Simple Learning and High Computational Performance in Neural Networks with Discrete Synapses." In: *Phys. Rev. Lett.* 115 (12 Sept. 2015), p. 128101.

[17] Carlo Baldassi et al. "Local entropy as a measure for sampling solutions in constraint satisfaction problems." In: *Journal of Statistical Mechanics: Theory and Experiment* 2016.2 (Feb. 2016), P023301.

[18] Carlo Baldassi, Enrico M. Malatesta, and Riccardo Zecchina. "Properties of the Geometry of Solutions and Capacity of Multilayer Neural Networks with Rectified Linear Unit Activations." In: *Phys. Rev. Lett.* 123 (17 Oct. 2019), p. 170602.

[19] Yiding Jiang et al. *Fantastic Generalization Measures and Where to Find Them.* 2019.

[20] Shuofeng Zhang et al. "Why flatness does and does not correlate with generalization for deep neural networks." In: *arXiv preprint arXiv:2103.06219* (2021).

[21] Kamaludin Dingle, Chico Q Camargo, and Ard A Louis. "Input–output maps are strongly biased towards simple outputs." In: *Nature communications* 9.1 (2018), pp. 1–7.

[22] Xiaoyi Mai and Zhenyu Liao. "High Dimensional Classification via Empirical Risk Minimization: Improvements and Optimality." In: *arXiv preprint arXiv:1905.13742* (2019).

[23] Marc Lelarge and Leo Miolane. "Asymptotic Bayes risk for Gaussian mixture in a semi-supervised setting." In: *arXiv preprint arXiv:1907.03792* (2019).

[24] Zeyu Deng, Abla Kammoun, and Christos Thrampoulidis. "A Model of Double Descent for High-dimensional Binary Linear Classification." In: *arXiv preprint arXiv:1911.05822* (2019).

[25] Thibault Lesieur et al. "Phase transitions and optimal algorithms in high-dimensional Gaussian mixture clustering." In: *2016 54th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. IEEE. 2016, pp. 601–608.

[26] Francesca Mignacco et al. "The role of regularization in classification of high-dimensional noisy Gaussian mixture." In: *arXiv preprint arXiv:2002.11544* (2020).

[27] Pratik Chaudhari et al. "Entropy-SGD: Biasing Gradient Descent Into Wide Valleys." In: *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[28] Max Welling and Yee W Teh. "Bayesian learning via stochastic gradient Langevin dynamics." In: *Proceedings of the 28th international conference on machine learning (ICML-11)*. 2011, pp. 681–688.

[29] Silvio Franz and Giorgio Parisi. "Recipes for metastable states in spin glasses." In: *Journal de Physique I* 5.11 (1995), pp. 1401–1415.

[30] Gintare Karolina Dziugaite and Daniel M. Roy. *Entropy-SGD optimizes the prior of a PAC-Bayes bound: Data-dependent PAC-Bayes priors via differential privacy*. 2018.

[31] Nitish Shirish Keskar et al. "On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima." In: *CoRR* abs/1609.04836 (2016).

[32] Matteo Negri et al. "Natural representation of composite data with replicated autoencoders." In: *arXiv preprint arXiv:1909.13327* (2019).

[33] Andrea Mazzolini et al. "Statistics of shared components in complex component systems." In: *Physical Review X* 8.2 (2018), p. 021023.

[34] Ricard Albalat and Cristian Cañestro. "Evolution by gene loss." In: *Nature Reviews Genetics* 17.7 (2016), p. 379.

[35] Kay Prüfer et al. "The complete genome sequence of a Neanderthal from the Altai Mountains." In: *Nature* 505.7481 (2014), p. 43.

[36] Ross C Hardison. "Comparative genomics." In: *PLoS biology* 1.2 (2003), e58.

[37] Eran Segal et al. "Module networks: identifying regulatory modules and their condition-specific regulators from gene expression data." In: *Nature genetics* 34.2 (2003), p. 166.

[38] Cole Trapnell. "Defining cell types and states with single-cell genomics." In: *Genome research* 25.10 (2015), pp. 1491–1498.

[39] Rémi Monasson Jérôme Tubiana Simona Cocco. "Learning protein constitutive motifs from sequence data." In: *eLife* 8 (2019), e39397.

[40] Joseph Redmon et al. "You only look once: Unified, real-time object detection." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 779–788.

[41] David M Blei, Andrew Y Ng, and Michael I Jordan. "Latent dirichlet allocation." In: *Journal of machine Learning research* 3.Jan (2003), pp. 993–1022.

[42] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[43] Sixin Zhang, Anna E Choromanska, and Yann LeCun. "Deep learning with elastic averaging SGD." In: *Advances in Neural Information Processing Systems*. 2015, pp. 685–693.

[44] Marc Mézard. "Mean-field message-passing equations in the Hopfield model and its generalizations." In: *Physical Review E* 95.2 (2017), p. 022117.

[45] Faruck Morcos et al. "Direct-coupling analysis of residue coevolution captures native contacts across many protein families." In: *Proceedings of the National Academy of Sciences* 108.49 (2011), E1293–E1301.

[46] Simona Cocco et al. "Inverse statistical physics of protein sequences: a key issues review." In: *Reports on Progress in Physics* 81.3 (2018), p. 032601.

[47] Qian Cong et al. "Protein interaction networks revealed by proteome coevolution." In: *Science* 365.6449 (2019), pp. 185–189.

[48] Christoph Feinauer et al. "Inter-protein sequence co-evolution predicts known physical interactions in bacterial ribosomes and the Trp operon." In: *PloS one* 11.2 (2016), e0149166.

[49] Thomas Gueudré et al. "Simultaneous identification of specifically interacting paralogs and interprotein contacts by direct coupling analysis." In: *Proceedings of the National Academy of Sciences* 113.43 (2016), pp. 12186–12191.

[50] Anne-Florence Bitbol et al. "Inferring interaction partners from protein sequences." In: *Proceedings of the National Academy of Sciences* 113.43 (2016), pp. 12180–12185.

[51] Matteo Figliuzzi et al. "Coevolutionary landscape inference and the context-dependence of mutations in beta-lactamase TEM-1." In: *Molecular biology and evolution* 33.1 (2015), pp. 268–280.

[52] Thomas A Hopf et al. "Mutation effects predicted from sequence co-variation." In: *Nature biotechnology* 35.2 (2017), p. 128.

[53] Christoph Feinauer and Martin Weigt. "Context-aware prediction of pathogenicity of missense mutations involved in human disease." In: *arXiv preprint arXiv:1701.07246* (2017).

[54] Mark A Davenport and Justin Romberg. "An overview of low-rank matrix recovery from incomplete observations." In: *IEEE Journal of Selected Topics in Signal Processing* 10.4 (2016), pp. 608–622.

[55] Sanjeev Arora et al. "Simple, Efficient, and Neural Algorithms for Sparse Coding." In: *CoRR* abs/1503.00778 (2015).

[56] Daria Grechishnikova. "Transformer neural network for protein-specific de novo drug generation as a machine translation problem." In: *Scientific reports* 11.1 (2021), pp. 1–13.

[57] Amit Zeisel et al. "Cell types in the mouse cortex and hippocampus revealed by single-cell RNA-seq." In: *Science* 347.6226 (2015), pp. 1138–1142.

[58] Hansruedi Mathys et al. “Single-cell transcriptomic analysis of Alzheimer’s disease.” In: *Nature* (2019), p. 1.

[59] Matteo Negri, Guido Tiana, and Riccardo Zecchina. “Native state of natural proteins optimizes local entropy.” In: *Physical Review E* 104.6 (2021), p. 064117.

[60] Alexei V. Finkelstein, Alexander M. Gutun, and Azat Ya Badretdinov. “Why are the same protein folds used to perform different functions?” In: *FEBS Lett.* 325.1-2 (1993), pp. 23–28.

[61] Aaron Chevalier et al. “Massively parallel de novo protein design for targeted therapeutics.” In: *Nature* 550.7674 (Oct. 2017), pp. 74–79.

[62] Hao Li et al. “Emergence of preferred structures in a simple model of protein folding.” In: *Science* 273.5275 (1996), pp. 666–669.

[63] Sridhar Govindarajan and R. A. Goldstein. “Why are some proteins structures so common?” In: *Proc. Natl. Acad. Sci.* 93.8 (Apr. 1996), pp. 3341–3345.

[64] Peter G. Wolynes. “Symmetry and the energy landscapes of biomolecules.” In: *Proc. Natl. Acad. Sci. U. S. A.* 93.25 (1996), pp. 14249–14255.

[65] Amos Maritan, Cristian Micheletti, and Jayanth R. Banavar. “Role of secondary motifs in fast folding polymers: A dynamical variational principle.” In: *Phys. Rev. Lett.* 84.13 (2000), pp. 3009–3012.

[66] K W Plaxco, K T Simons, and D Baker. “Contact order, transition state placement and the refolding rates of single domain proteins.” In: *J. Mol. Biol.* 277.4 (Apr. 1998), pp. 985–994.

[67] E Shakhnovich. “Proteins with selected sequences fold into unique native conformation.” In: *Phys. Rev. Lett.* 72.24 (June 1994), pp. 3907–3910.

[68] R A Broglia et al. “Stability of Designed Proteins against Mutations.” In: *Phys. Rev. Lett.* 82.23 (June 1999), pp. 4727–4730.

[69] G Tiana, R A Broglia, and D Provasi. “Designability of lattice model heteropolymers.” In: *Phys. Rev. E* 64.1 (June 2001), p. 011904.

[70] Jeremy L England and Eugene I Shakhnovich. “Structural determinant of protein designability.” In: *Phys. Rev. Lett.* 90.21 (May 2003), p. 218101.

[71] G Tiana et al. “Imprint of evolution on protein structures.” In: *Proc. Natl. Acad. Sci. U. S. A.* 101.9 (Mar. 2004), pp. 2846–2851.

[72] I N Berezovsky, A Yu Grosberg, and E N Trifonov. “Closed loops of nearly standard size: common basic element of protein structure.” In: *FEBS Lett.* 466 (Jan. 2000), pp. 283–286.

[73] M Karplus, T Ichiye, and B M Pettitt. “Configurational entropy of native proteins.” In: *Biophys. J.* 52.6 (Dec. 1987), pp. 1083–1085.

[74] H Frauenfelder, F Parak, and R D Young. "Conformational Substates in Proteins." In: *Annu. Rev. Biophys. Biophys. Chem.* 17.1 (June 1988), pp. 451–479.

[75] D Shortle, K T Simons, and D Baker. "Clustering of low-energy conformations near the native structures of small proteins." In: *Proc. Natl. Acad. Sci. U. S. A.* 95.19 (Sept. 1998), pp. 11158–11162.

[76] Kannan Sankar, Kejue Jia, and Robert L. Jernigan. "Knowledge-based entropies improve the identification of native protein structures." In: *Proc. Natl. Acad. Sci. U. S. A.* 114.11 (2017), pp. 2928–2933.

[77] John Jumper et al. "Highly accurate protein structure prediction with AlphaFold." In: *Nature* (2021), pp. 1–11.

[78] Minkyung Baek et al. "Accurate prediction of protein structures and interactions using a three-track neural network." In: *Science* (2021).

[79] P G de Gennes. *Scaling Concepts in Polymer Physics*. Cornell University Press, Jan. 1979.

[80] N Go. "Theoretical Studies of Protein Folding." In: *Annu. Rev. Biophys. Bioeng.* 12.1 (June 1983), pp. 183–210.

[81] J D Bryngelson and P G Wolynes. "Spin glasses and the statistical mechanics of protein folding." In: *Proc. Natl. Acad. Sci. U. S. A.* 84.21 (Nov. 1987), pp. 7524–7528.

[82] G Tiana, L Sutto, and R.A. Broglia. "Use of the Metropolis algorithm to simulate the dynamics of protein chains." In: *Phys. A Stat. Mech. its Appl.* 380.1-2 (July 2007), pp. 241–249.

[83] Mark James Abraham et al. "GROMACS: High performance molecular simulations through multi-level parallelism from laptops to supercomputers." In: *SoftwareX* 1 (2015), pp. 19–25.

[84] Jeffrey K Noel et al. "SMOG 2: a versatile software package for generating structure-based models." In: *PLoS computational biology* 12.3 (2016), e1004794.

[85] Guido Tiana and Ludovico Sutto. "Equilibrium properties of realistic random heteropolymers and their relevance for globular and naturally unfolded proteins." In: *Physical Review E* 84.6 (2011), p. 061910.

[86] P L Privalov and N N Khechinashvili. "A Thermodynamic Approach to the Problem of Stabilization of Globular Protein Structure: A Calorimetric Study." In: *J. Mol. Biol.* 86 (Jan. 1974), pp. 665–684.

[87] Payel Das, Silvina Matysiak, and Cecilia Clementi. "Balancing energy and entropy: A minimalist model for the characterization of protein folding landscapes." In: *Proceedings of the National Academy of Sciences* 102.29 (2005), pp. 10141–10146.

[88] Emanuelel Paci et al. "Determination of a transition state at atomic resolution from protein engineering data." In: *J. Mol. Biol.* 324.1 (Nov. 2002), pp. 151–163.

[89] Motoo Kimura. *The neutral theory of molecular evolution*. Cambridge University Press, 1983.

[90] Jesse D Bloom et al. "Protein stability promotes evolvability." In: *Proceedings of the National Academy of Sciences* 103.15 (2006), pp. 5869–5874.

[91] Andreas Wagner. "Robustness and evolvability: a paradox resolved." In: *Proceedings of the Royal Society B: Biological Sciences* 275.1630 (2008), pp. 91–100.

[92] Davide De Lucrezia et al. "Do natural proteins differ from random sequences polypeptides? Natural vs. random proteins classification using an evolutionary neural network." In: *PLoS One* 7.5 (2012), e36634.

[93] Kamaludin Dingle, Steffen Schaper, and Ard A Louis. "The structure of the genotype–phenotype map strongly constrains the evolution of non-coding RNA." In: *Interface focus* 5.6 (2015), p. 20150053.

[94] Trinh Xuan Hoang et al. "Geometry and symmetry presculpt the free-energy landscape of proteins." In: *Proceedings of the National Academy of Sciences* 101.21 (2004), pp. 7960–7964.

[95] Chico Q Camargo and Ard A Louis. "Boolean threshold networks as models of genotype-phenotype maps." In: *Complex Networks XI*. Springer, 2020, pp. 143–155.

This Ph.D. thesis has been typeset by means of the TEX-system facilities. The typesetting engine was LuaLATEX. The document class was `toptesi`, by Claudio Beccari, with option `tipotesi=scudo`. This class is available in every up-to-date and complete TEX-system installation.