UNIVERSITÀ DEGLI STUDI DI PADOVA

Dipartimento di Fisica e Astronomia "Galileo Galilei"

Master Degree in Physics

Final Dissertation

Nonequilibrium training of Restricted Boltzmann Machines

Thesis supervisor

Prof. Marco Baiesi

Candidate

Andrea De Bei

Academic Year 2022/2023

Abstract

Restricted Boltzmann machines (RBMs) are a very important unsupervised learning method in the machine learning research landscape. RBMs are frequently employed is the construction of generative models, which are very important for the development of neural networks. Unlike typical generative models, RBMs allow the generation of good quality samples by doing quick sampling. In addition, the choice of this learning method is due to its proximity to statistical physics. The goal of this thesis is to train RBMs and test, by observing the generated samples, their out of equilibrium (OOE) behavior. The dataset used is MNIST (represented by number images), trained on relatively simple RBMs. First, we evaluate the reproducibility of the model for a fixed number of training samples, then study the stability and variability of the model and the equilibrium, pseudo-equilibrium and OOE regimes by training the RBM with variable samples.

Contents

Chapter 1

Introduction

Machine learning is a branch of artificial intelligence to develop algorithms that can automatically learn from data [1]. Nowadays, machine learning is increasingly being used for multiple application such as speech recognition of text, automatic driving of vehicles, classification of new astronomical structures, engineering designs, the revelation of out-of-norm charges in the financial field, etc...

Among machine learning models, we may distinguish at least two kinds: discriminative and generative. Discriminative models are designed to perceive differences between groups or categories of data, while generative models learn to represent and sample a probability distribution. A classic example of a generative model, also used in this thesis work, is image learning and reproduction from random noise.

The class of models used in this thesis work is RBM (Restricted Boltzmann Machines), energy-based stochastic latent variable machines that simplify BM (Boltzmann Machines), thus being more efficient. Increased efficiency reduces computation time, making it possible to achieve excellent results even when working with a laptop and free online servers.

Specifically, in this work, images representing numbers, ranging from 0 to 9 with a resolution of 28x28 pixels, are extracted from the MNIST dataset. These numbers are learned from the RBM and then generated from images composed of random pixels. The quality of the generated samples can be estimated by specific functions. In our case, we use the Adversarial Accuracy Indicator [2].

This thesis is mainly based on the study of equilibrium and nonequilibrium regimes using a very simple RBM, with variations on the learning methods [3]. First, the results of Ref. [3] are reproduced to prove the validity of the simplified RBM and then variations are explored.

In the second chapter of this thesis, a theoretical introduction is made to describe what a Markov random field and a Markov chain is. Gibbs sampling is described, as well as all the fundamental concepts for understanding RBM algorithms, as well as Stochastic Gradient Descent.

Chapter 3 begins by describing what an RBM is, first defining the Boltzmann Machine (BM), together with the unsupervised learning and energy-based models of which BM and RBM are part of. The training methods and algorithms used are explained. To improve the performance of a generic RBM, the centering trick is shown. After RBM training, we described how samples are generated and how their goodness is tested with appropriate functions. Mixing Time (MT) is discussed as an indicator of the RBM regimes, and finally, practical tips for working with RBMs are described, which are useful for those programming the algorithms.

In the fourth chapter, we discuss the experimental setup and the dataset used. In particular, it is explained why it is convenient to consider averages when considering the quality of an RBM. Next, variable sampling training is analyzed and regime changes are studied. The expectation was to train an RBM by variably sampling it to quickly reach the equilibrium regime without ever leaving it. However, the entry into the equilibrium regime is no more fast then the normal training. However, pseudo-equilibrium states were found. Finally, an explanation of the reaching of the pseudo-equilibrium regime is given by observing the change in the number of hidden units.

The fifth and final chapter lists the achievements and outlook of this thesis.

Chapter 2

Theoretical foreword

This chapter is entirely dedicated to mathematical and computer science concepts. These concepts are useful to define the type of model used in the following sections, i.e. Boltzmann Machines (BM) and Restricted Boltzmann Machines (RBM). The chapter begins by introducing Probabilistic Graphical Models (PGM) and defines the Markov Random Field (MRF). Next, the Markov Chain (MC) is defined to show how Gibbs sampling works, which is an algorithm that allows the training of the Machines that we use in this thesis work. Finally, Stochastic Gradient Descent (SGD) is introduced for the implementation of Gibbs sampling and Boltzmann Machines training.

2.1 Markov Random Field

The Probabilistic Graphical Model (PGM) uses graphs to simply describe a complex distribution over a high-dimensional space [4]. In the representation, the nodes are the variables in our domain and the lines connecting nodes correspond to the probabilistic interactions between them (example in Fig. 2.1). Such representations provide a simple and visual way to describe the structure of probabilistic models and are useful for designing new ones. Furthermore, simply by observing the graph, we can obtain information on the properties of a given model. Finally, complex calculations can be showed graphically, facilitating the mathematical treatment of the problem.

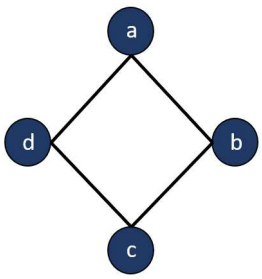

Figure 2.1: Example of probabilistic graphical model

A Markov Random Field (MRF) is an undirected graph i.e. an ordered pair $G = (V, E)$, where V is a finite set of nodes and E is a set of undirected edges (lines of interaction between nodes) [5]. An edge comprises a pair of nodes of V and the neighborhood $N_v = w \in V : \{w, v\} \in E$ of a node v is defined by the set of nodes connected to v (example in Figure 2.2).

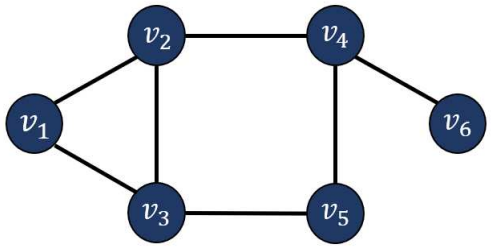

Figure 2.2: Example of an undirected graph. Node $\{v_2, v_3\}$ are a clique, but only $\{v_1, v_2, v_3\}$ is a maximal clique

A clique is a subset of V such that there exists a link between all pairs of nodes in the subset and is named maximal if no node can be added such that the resulting set is still a clique. While a sequence of connected nodes defines a path.

If we assign a random variable X_v to each node of the graph $G = (E, V)$, the set of variables $X = (X_v)_{v \in V}$ is named a Markov Random Field (MRF) if the joint probability distribution p fulfills the Local Markov Property with respect to the graph [5].

Definition I (Local Markov Property). *The joint probability distribution of a set of random variables $X = (X_v)_{v \in V}$ is said to fulfill the local Markov property with respect to a graph $G = (V, E)$, if for all $v \in V$ the random variable X_v is conditionally independent of all other variables given its neighborhood $(X_w)_{w \in N_v}$. That is, for all $v \in V$ and all $x \in \Lambda^{|V|}$ (where Λ is the state space where taking value the random variable X_v), one has that:*

$$p(x_v|(x_w)_{w \in V \setminus \{v\}}) = p(x_v|(x_w)_{w \in N_v}) \tag{2.1.1}$$

If the MRF probability distribution is strictly positive, the Local Markov Property can be extended to two other Markov properties [5].

Definition II (Global Markov Property). *The MRF is said to have the Global Markov Property with respect to a graph $G = (V, E)$, if for any three disjunct subsets $A, B, S \subset V$, such that all nodes in A and B are separated by S, the variables $(X_a)_{a \in A}$ and $(X_b)_{b \in B}$ are conditionally independent given $(X_s)_{s \in S}$, i.e., for all $x \in \Lambda^{|V|}$ one has that:*

$$p((x_a)_{a \in A}|(x_t)_{t \in S \cup B}) = p((x_a)_{a \in A}|(x_t)_{t \in S}) \tag{2.1.2}$$

Now the theorem of Hammersley-Clifford, starting from the truthfulness of Markov properties, proves the existence of a general factorization of MRF distributions [6]. The theorem is as follows:

Theorem I (Hammersley-Clifford). *A strictly positive distribution p satisfies the Markov Property with respect to an undirected graph G if and only if p factorizes over G.*

A distribution is said to factorize over an undirected graph G with maximal cliques $\mathcal{C}$ if there exists a set of non-negative functions $\{\psi \in C\}_{C \subset \mathcal{C}}$, named potential functions, satisfying:

$$\forall x_C, \hat{x}_C \in \Lambda^{|V|} : (x_c)_{c \in C}, (\hat{x}_c)_{c \in C} \implies \psi_C(\mathbf{x}_C) = \psi_C(\hat{\mathbf{x}}_C), \tag{2.1.3}$$

and

$$p(\mathbf{x}) = \frac{1}{Z} \prod_{C \in \mathcal{C}} \psi_C(\mathbf{x}_C) \tag{2.1.4}$$

where Z is named partition function [7]. In the case where p is strictly positive, the potential functions will also be strictly positive. Therefore, we can write, if $E(\mathbf{x}_C) = -\sum_{C \in \mathcal{C}} \log(\psi_C(\mathbf{x}_C))$ is the energy function [7]:

$$p(\mathbf{x}) = \frac{1}{Z} e^{\sum_{C \in \mathcal{C}} \log(\psi_C(\mathbf{x}_C))} = \frac{e^{-E(\mathbf{x}_C)}}{Z} \tag{2.1.5}$$

Therefore, thanks to Eq.(2.1.5), the joint probability distribution (strictly positive) of each MRF can be described as a product of potentials. This is named Boltzmann Distribution (or Gibbs distribution) and from it the total energy is obtained by summing the energy of each of the maximum cliques.

In an undirected graph (of which the MRF is a part), potentials have no specific probabilistic interpretation [7], whereas in directed graphs, each factor represents the conditional distribution of the corresponding variable. One consequence of no probabilistic interpretation is that product will generally not be normalised correctly. This property offers flexibility in the choice of potential functions, however, a problem arises when trying to justify the choice of a potential function for a particular application. One interpretation is to view the potential function as an expression of the type of configurations of local variables that are preferable to others [7].

2.2 Markov Chain

Markov chains provide a way to extract samples from non-trivial probability distributions, such as the Boltzmann distribution.

A Markov chain is a stochastic process describing a sequence of possible events, where the probability of each event depends only on the state reached in the previous event [4] [8]. Formally, a Markov chain is a family of random variables $X = \{X^{(k)} | k \in \mathbb{N}_0\}$ that take values in a (finite)

set Ω and for which $\forall k \geq 0$ and $\forall j, i, i_0, ..., i_{k-1} \in \Omega$ one has, if P is the probability transition:

$$\begin{aligned} p_{ij}^{(k)} &= P\Big(X^{(k+1)} = j | X^{(k)} = i, X^{(k-1)} = i_{k-1} ..., X^{(0)} = i_0\Big) \\ &= P\Big(X^{(k+1)} = j | X^{(k)} = i\Big) \end{aligned} \tag{2.2.1}$$

This property is also named the Markov property and considers the temporal neighborhood, i.e. exclusively the last step performed. In contrast, the Markov properties of Section 2.1 consider the neighborhood due to the topology of the graph.

If for all points in time $k \geq 0$, the $p_{ij}^{(k)}$ have the same value p_{ij} (i.e. the transition probabilities do not change over time) the chain is termed homogeneous and the matrix $\mathbf{P} = (p_{ij})_{i,j \in \Omega}$ is named the transition matrix of the homogeneous Markov chain [7].

The probability distribution of $X^{(0)}$ is given by the probability vector $\boldsymbol{\mu}^{(0)} = (\mu^{(0)}(i))_{i \in \Omega}$, with $\mu^{(0)}(i) = P(X^{(0)} = i)$. The distribution $\boldsymbol{\mu}^{(k)}$ of $X^{(k)}$ is given by $\boldsymbol{\mu}^{(k)T} = \boldsymbol{\mu}^{(0)T}\mathbf{P}^k$ (so after k transition). A distribution π for which $\boldsymbol{\pi}^T = \boldsymbol{\pi}^T\mathbf{P}$ is termed stationary distribution or equilibrium distribution. Once the Markov chain at time k has reached the stationary distribution $\boldsymbol{\mu}^{(k)} = \boldsymbol{\pi}$, subsequent states will have the same $\boldsymbol{\pi}$ distribution, i.e. $\boldsymbol{\mu}^{(k+n)} = \boldsymbol{\pi}$ for all $n \in \mathbb{N}$. A sufficient, but not necessary condition for a π distribution to be stationary with respect to a Markov chain described by the transition probabilities p_{ij}, $i, j \in \Omega$, is that $\forall i, j \in \Omega$

$$\pi(i)p_{ij} = \pi(j)p_{ji}. \tag{2.2.2}$$

This propriety is named detailed balance condition, and if satisfied, around any closed cycle of states, there is no net flow of probability. A Markov chain that satisfies this property is said to be reversible [7].

For a finite space of states Ω, a Markov chain has a unique stationary distribution if is irreducible. A Markov chain is irreducible or ergodic if one can go from any state of Ω to another in a finite number of transitions. Formally, a Markov chain is ergodic if $\forall i, j \in \Omega$, $\exists k > 0$ with $P(X^{(k)} = j | X^{(0)} = i) > 0$.

A chain is said to be aperiodic if each state can recur at irregular times. So, a chain is aperiodic if for all $i \in \Omega$, the Maximum Common Divisor (MCD) of all elements of the set $\{k \in \mathbb{N}_0 | P(X^{(k)} = i | X^{(0)} = i) > 0\}$ is 1.

It can be shown that an ergodic and aperiodic Markov chain on a finite state space is guaranteed to converge to its stationary distribution. If α and β are two distributions defined on a finite state space Ω, the variation distance is defined as [4]:

$$d_V(\boldsymbol{\alpha}, \boldsymbol{\beta}) = \frac{1}{2}|\boldsymbol{\alpha} - \boldsymbol{\beta}| = \frac{1}{2}\sum_{x \in \Omega} |\alpha(x) - \beta(x)| \tag{2.2.3}$$

With this definition, the following theorem can be stated [8]:

Theorem II. *Let π be the stationary distribution of an irreducible and aperiodic Markov chain on a finite state space with transition matrix $\mathbf{P}$. For an arbitrary starting distribution μ:*

$$\lim_{k\to\infty} d_V(\boldsymbol{\mu}^T\mathbf{P}^k, \boldsymbol{\pi}^T) = 0 \tag{2.2.4}$$

Therefore an irreducible and aperiodic Markov chain on a finite state space is guaranteed to converge to its stationary distribution.

Markov chain Monte Carlo (MCMC) methods use this convergence theorem to produce samples. Assuming one needs to sample from a distribution q with a finite state space Ω. It suffices to construct an irreducible, aperiodic Markov chain with a stationary distribution $\pi = q$ and, if k is sufficiently large, the state $x^{(k)}$ of $X^{(k)}$ of the constructed chain can be approximately considered as a sample of distribution q.

2.3 Gibbs sampling

Gibbs sampling [9] is an MCMC algorithm that generates transitions of a Markov chain and develops in two stages. In the first stage, a candidate state is chosen at random from a proposed distribution. In the second stage, the candidate state is accepted as a new state in the Markov chain with a certain acceptance probability that guarantees a detailed equilibrium. Gibbs sampling constructs a Markov Chain by updating each variable in a way dependent on its conditional distribution given by the state of all other variables. With this technique, it is possible to produce samples approximated by the Boltzmann Distribution of an MRF.

Going back to the representation of an undirected graph $G = (V, E)$ and taking an MRF $X = (X_1, ..., X_N)$, where the nodes are defined as $V = \{1, ..., N\}$ to simplify the notation. Now, assuming that the MRF changes its state over time, a Markov Chain is obtained from $\{X = X^{(k)} | k \in \mathbb{N}_0\}$, which takes values in $\Omega = \Lambda^N (= \Lambda^{|V|})$. A new state of the chain is produced in the following way [5]:

1. is chosen randomly X_i, $i \in V$ with probability $q(i)$ given by a probability distribution q over V

2. the new state for X_i is sampled based on its conditional probability distribution given the state $(x_v)_{v\in V\setminus i}$ of all other variables $(X_v)_{v\in V\setminus i}$, which is $\pi(x_i|(x_v)_{v\in V\setminus i}) = \pi(x_i|(x_w)_{w\in N_i})$ because of the Local Markov property of MRF, where N_i is the neighbourhood of the node i.

The transition probability $p_{\mathbf{xy}}$ for two states $\mathbf{x}$, $\mathbf{y}$ of the MRF $\mathbf{X}$ with $\mathbf{x} \neq \mathbf{y}$ is

$$p_{\mathbf{xy}} = \begin{cases} q(i)\pi(y_i|(x_v)_{v\in V\setminus i}) & \text{if } \exists i \in V \text{ so that } \forall v \in V \text{ with } v \neq i : x_v = y_v \\ 0 & \text{otherwise} \end{cases} \tag{2.3.1}$$

and the probability that the state of the MRF stays the same is:

$$p_{\mathbf{xx}} = \sum_{i \in V} q(i)\pi(x_i|(x_v)_{v \in V \setminus i}) \tag{2.3.2}$$

Recalling the statement of **Theorem II**, which proves that the Markov chain defined by these transition probabilities converges to the joint distribution π of the MRF, it must be proved that π is the stationary distribution of the Gibbs chain and that the chain is irreducible and aperiodic. To prove the stationarity of π it suffices to verify that the equilibrium condition in Eq.(2.2.2) is valid. For $\mathbf{x} = \mathbf{y}$ this follows directly. On the other hand, if $\mathbf{x}$ and $\mathbf{y}$ differ in the value of more than one random variable, it follows that, according to Eq.(2.3.1), $p_{\mathbf{yx}} = p_{\mathbf{xy}} = 0$. Finally, if $\mathbf{x}$ and $\mathbf{y}$ differ only in the state of a single variable X_i, i.e. $y_j = x_j$ for $j \neq i$ and $y_i \neq x_i$, one have:

$$\begin{aligned}
\pi(\mathbf{x})p_{\mathbf{xy}} &= \pi(\mathbf{x})q(i)\pi(y_i|(x_v)_{v \in V \setminus i}) = \\
&= \pi(x_i|(x_v)_{v \in V \setminus i})q(i)\frac{\pi(y_i|(x_v)_{v \in V \setminus i})}{\pi((x_v)_{v \in V \setminus i})} = \\
&= \pi(y_i|(x_v)_{v \in V \setminus i})q(i)\frac{\pi(x_i|(x_v)_{v \in V \setminus i})}{\pi((x_v)_{v \in V \setminus i})} = \\
&= \pi(\mathbf{y})q(i)\pi(x_i|(x_v)_{v \in V \setminus i}) \\
&= \pi(\mathbf{y})p_{\mathbf{yx}}
\end{aligned} \tag{2.3.3}$$

Thus, the detailed balance condition is fulfilled and π is the stationary distribution.

Since π is strictly positive, the conditional probability distributions of the individual variables X_i are also strictly positive. This means that every single variable can assume any state $x_i \in \Lambda$ in a single transition step and thus every state in the MRF can reach any other in Λ^N in a finite number of steps. Thus, the Markov chain is irreducible. Moreover, since $p_{\mathbf{xx}} > 0$ for all $x \in \Lambda^N$, the Markov chain is aperiodic. Therefore, from **Theorem II** the chain converges to the stationary distribution π.

2.4 Gradient Descent

In general, machine learning problems, start with a set of data X, a model $\mathcal{M}(\boldsymbol{\theta})$, which is a function of the parameters $\boldsymbol{\theta}$, and a cost function $C(X, \mathcal{M}(\boldsymbol{\theta}))$ which indicates how well the model $\mathcal{M}(\boldsymbol{\theta})$ explains the data X.

To compute the parameters that minimise the cost function, which are also those that allow the model to get closest to the data, one method used is Gradient Descent, which is based on iteratively adjusting the parameters $\boldsymbol{\theta}$ along the direction in which $\boldsymbol{\nabla} C(X, \mathcal{M}(\boldsymbol{\theta}))$ is large and negative, such that the parameters cause the cost function to fall into a local minimum. The treatment of this method is very extensive, consequently, only Stochastic Gradient Descent (SGD) is discussed. This gradient technique is used to train RBM in the thesis.

2.4.1 Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) is one of the gradient descent methods that exploits stochasticity as a strength [10]. Stochasticity is incorporated by approximating the gradient on a subset of data named mini-batch. The size of mini-batches is typically smaller than the total number of data N, with typical sizes ranging from ten to a few hundred. If there is N data in total and the mini-batch size is M, there will be $\frac{N}{M}$ mini-batches.

Mini-batches are denoted by B_k, where $k = 1, ..., \frac{N}{M}$. Thus, in SGD, at each gradient descent step, we approximate the gradient using a single mini-batch B_k. A cycle can be run on all $k = 1, ..., \frac{N}{M}$ mini-batches one at a time and use the mini-batch approximation to the gradient to update the parameters named with $\boldsymbol{\theta}$ at each k step. A complete iteration over all N data points is termed epoch. The gradient approximation to the mini-batch is:

$$\nabla_\theta E_{MB}(\boldsymbol{\theta}) = \sum_{i \in B_k} \nabla_\theta e_i(\mathbf{x}_i, \boldsymbol{\theta}) \tag{2.4.1}$$

where $e_i(\mathbf{x}_i, \boldsymbol{\theta})$ is the cost function calculated in the individual mini-batch and $\mathbf{x}_i$ are the variables of the start dataset. The SGD algorithm is:

$$\mathbf{v}_t = \eta_t \nabla_\theta E_{MB}(\boldsymbol{\theta}) \tag{2.4.2}$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \mathbf{v}_t \tag{2.4.3}$$

where η_t is a factor indicating the amplitude of the gradient named learning rate. Replacing the actual gradient with the SGD calculated with mini-batches has two important advantages. The first is that stochasticity reduces the possibility of our algorithm getting stuck in isolated local minima. Secondly, it speeds up the calculation, because it is not necessary to use all N data points for the gradient approximation. Furthermore, it is believed that the introduction of stochasticity acts as a natural regulator and prevents overfitting in isolated deep minima [11]. To make the SGD more stable, the first and second momentum of the gradient are introduced into the calculation. Specifically, the algorithm used in this thesis is the ADAM [12], which is described by the following equations:

$$\mathbf{g}_t = \nabla_\theta E_{MB}(\boldsymbol{\theta}) \tag{2.4.4}$$

$$\mathbf{m}_t = \beta_1 \mathbf{m}_{t-1} + (1 - \beta_1)\mathbf{g}_t \tag{2.4.5}$$

$$\mathbf{s}_t = \beta_2 \mathbf{s}_{t-1} + (1 - \beta_2)\mathbf{g}_t^2 \tag{2.4.6}$$

$$\hat{\mathbf{m}}_t = \frac{\mathbf{m}_t}{1 - \beta_1} \tag{2.4.7}$$

$$\hat{\mathbf{s}}_t = \frac{\mathbf{s}_t}{1 - \beta_2} \tag{2.4.8}$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta_t \frac{\hat{\mathbf{m}}_t}{\sqrt{\hat{\mathbf{s}}_t} + \epsilon} \tag{2.4.9}$$

where β_1 and β_2 set the memory lifetime of the first and second moment ($\hat{\mathbf{m}}_t$ and $\hat{\mathbf{s}}_t$) and are typically 0.9 and 0.99 respectively, while ϵ is a noise that is introduced to improve the algorithm (it is usually around 10^{-8}). Rewriting the equation as a function of variance $\boldsymbol{\sigma}_t = \hat{\mathbf{s}}_t - (\hat{\mathbf{m}}_t)^2$ the parameter update rule is given by:

$$\Delta\boldsymbol{\theta}_{t+1} = -\eta_t \frac{\hat{\mathbf{m}}_t}{\sqrt{\boldsymbol{\sigma}_t^2 + \hat{\mathbf{m}}_t^2} + \epsilon} \tag{2.4.10}$$

Analyzing what happens for small or large variances, two main advantages can be drawn from ADAM. The first is that the step size cuts the directions of large gradients (and thus prevents oscillations and divergences), secondly it measures gradients in terms of a natural length scale, the standard deviation σ_t. To obtain the best performance from the SGD algorithm, a some general rule to follow is randomize the data when forming mini-batches, because the gradient descent method can adapt to spurious correlations resulting from the order in which the data are presented if they are not randomized appropriately when forming mini-batches.

Chapter 3

Restricted Boltzmann Machine

This chapter defines and describes the Restricted Boltzmann Machines (RBM) and their properties used in this thesis. The chapter begins by defining the Model, precisely the unsupervised learning of which the RBM is a part, the energy-based models starting from the spin glass model and transposing this to a computer model and the Boltzmann Machines (BM) with the methods and the math for the training. Showing the typical problems of BM, the Restricted Boltzmann Machines (RBM) are defined as an improvement of these, specifying what Bernoulli's RBMs are (the only ones used in this thesis) and their correlations are shown for explaining the improvement respect the BM. Then the RBM training is explained and the Gibbs Sampling Block is described for implementing the training algorithm. However, this type of sampling executes infinite sampling, consequently new algorithms (CD-n, PCD-n, Rdm-n) are introduce to solve these problems. Then, the centering trick is adopted to improve the performance of a generic RBM and an algorithm to apply it is also described. Afterwards, it is described how to generate a sample starting from a random noise and we describe testing function to verify the similarity between the generated samples and the initial dataset, specifically, the Adversarial Accuracy Indicator (AAI) is explained. The mixing time of an RBM is then introduced, and used to determine the regime in which a trained RBM works. Finally, practical tips on setting up RBMs are outlined, some of which are used in the algorithms of this thesis.

3.1 Model Definition

Before defining what RBM are, it is appropriate to describe the unsupervised learning and energy-based models. Furthermore, RBMs are a special case of Boltzmann Machines (BM), which are also discussed in this section.

3.1.1 Unsupervised Learning

In unsupervised machine learning, one starts with a dataset $\mathcal{D} = \{x_d^{(k)}\}_{k=1,\dots,N_{data}}$, where $x_d^{(k)}$ is a real vector of dimension M, and it is assumed that the data have an identical and independent distribution (i.i.d.), termed $p(\mathbf{x})$. The objective of this learning is to acquire the empirical

distribution of the data and describe it with a model $p_{\boldsymbol{\theta}}(\mathbf{x})$, dependent on some parameter $\boldsymbol{\theta}$. The procedure of fitting the model to the data is named training. After training, it is possible to use the model generatively, by sampling new data that have the same characteristics as the training dataset.

3.1.2 Energy-Based Model

Energy-based models start from the probability distribution named Boltzmann Distribution defined as follows:

$$p_{\boldsymbol{\theta}}(\mathbf{x}) = \frac{1}{Z_{\boldsymbol{\theta}}} e^{-E(\mathbf{x},\boldsymbol{\theta})} \tag{3.1.1}$$

where $E(\mathbf{x}, \boldsymbol{\theta})$ is the energy function that depends on the data and parameters. The normalization factor of this probability distribution is $Z_{\boldsymbol{\theta}}$, which is termed the partition function:

$$Z_{\boldsymbol{\theta}} = \sum_{\mathbf{x}} e^{-E(\mathbf{x},\boldsymbol{\theta})} \tag{3.1.2}$$

The $\sum_{\mathbf{x}}$ operator can be replaced by the trace $\mathrm{Tr}_{\mathbf{x}}$ for discrete data or the integral $\int d\mathbf{x}$ for continuous data. Since this thesis is based exclusively on discrete data, writing $\sum_{\mathbf{x}}$ or $\mathrm{Tr}_{\mathbf{x}}$ is indifferent. So, from the equations (3.1.1) and (3.1.2) to know everything about an energy-based model, it is enough to know $E(\mathbf{x}, \boldsymbol{\theta})$. A simple choice for the energy function is given by the spin glass model. This model is used in statistical physics to describe transitions from the ferromagnetic to the paramagnetic regime or vice versa. The model is based on a lattice where at each node there is a spin particle s_i (if one dimension) with a possible value of $\{-1, 1\}$. The interaction Hamiltonian is:

$$E(\mathbf{s}) = -\sum_{(i,j)} J_{ij} s_i s_j - \sum_i b_i s_i \tag{3.1.3}$$

where $\mathbf{J}$ is the interaction matrix, $\mathbf{b}$ is the external magnetic field and the sum runs over all the connected pairs of spins (nearest neighbors) where $J_{ij} \neq 0$. Two types of configuration are favored between two neighboring spins: the first occurs when the two spins s_i and s_j are aligned if $J_{ij} > 0$, while the second occurs when the two spins are anti-aligned if $J_{ij} < 0$. Instead, the second term competes for the alignment of the spin along the direction of the field b_i at each site.

The probability distribution of the configurations of a spin system at equilibrium is given by the Boltzmann Distribution Eq.(3.1.1), with parameters $\boldsymbol{\theta} = (\mathbf{J}, \mathbf{b})$. A schematic of spin glass model in one dimension is shown in Fig. 3.1

Similarly, one can describe the data x_i (in analogy to the spin s_i of the spin glass model) using an undirected graph consisting of M connected nodes (Fig 3.2), where the nodes of the graph represent random variables associated with the M components of the data vectors. The edges of the graph represent the statistical interactions between the variables, and the strength of the

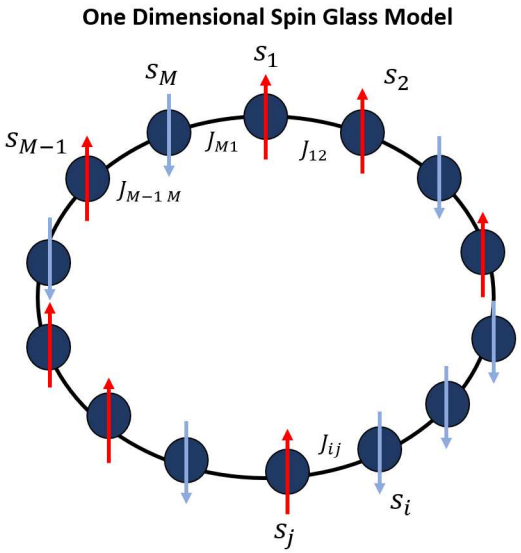

Figure 3.1: Outline of the spin glass model in one dimension, where J_{ij} describes the interaction between two neighbouring spin particles

interactions is described by the matrix of weights $\mathbf{W} = \{W_{ij}\}$, such that $W_{ii} = 0, \forall i = 1, ...M$, i.e. there is no self-interaction. Finally, the "external field" is in the equation as in Eq.3.1.3, so the energy function is:

$$E(\mathbf{x}, \boldsymbol{\theta}) = -\sum_{i,j} W_{ij} x_i x_j - \sum_i b_i x_i \tag{3.1.4}$$

which can be included in the probability distribution. Since in the case of a computer science one works with bits instead of spin, it is customary to shift and the possible values of the variables x_i from $\{-1, 1\}$ to $\{0, 1\}$. The $E(\mathbf{x}, \boldsymbol{\theta})$ has no physical meaning on the data, but is used as a tool to extract information from the data, i.e. to extract a model. A computational model that implements the probability distribution associated with an energy function of Eq.(3.1.4) is named Boltzmann machine (BM) [13].

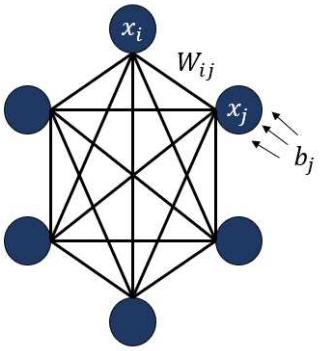

Figure 3.2: Schematic diagram of a fully connected probabilistic graph where each node represents a random variable x_j undergoing a local bias field b_j, the lines represent the interaction W_{ij} between each node.

3.1.3 Learning of a Boltzmann Machine

Given a data set $\mathcal{D} = \{x_d^{(k)}\}_{k=1,...,N_{data}}$, the goal to be achieved is to fit the parameters $\boldsymbol{\theta}$ of the generative model such that they approximate the empirical distribution of the data:

$$p_{data}(\mathbf{x}) = \frac{1}{N_{data}} \sum_{k=1}^{N_{data}} \delta(x - x_d^{(k)}) \tag{3.1.5}$$

where $\delta(x - x_d^{(k)})$ is the Delta of Dirac, which returns 0 unless $x = x_d^{(k)}$, in that case the result is 1. One way to do this is to minimize the Kullback-Leibler(KL) divergence between the empirical and the defined distribution with respect to the parameters $\boldsymbol{\theta}$, thus turning it into an optimization problem:

$$\begin{aligned} \rho_{KL}(p_{data}||p_{\boldsymbol{\theta}}) &= \sum_{\mathbf{x}} p_{data}(\mathbf{x}) \log\left(\frac{p_{data}(\mathbf{x})}{p_{\boldsymbol{\theta}}(\mathbf{x})}\right) = \\ &= \sum_{\mathbf{x}} p_{data}(\mathbf{x}) \log p_{data}(\mathbf{x}) - \sum_{\mathbf{x}} p_{data}(\mathbf{x}) \log p_{\boldsymbol{\theta}}(\mathbf{x}) \end{aligned} \tag{3.1.6}$$

The first term is (minus) the Shannon entropy of the empirical distribution (from data), and does not depend on the $\boldsymbol{\theta}$ parameters of the model. So to minimize the divergence KL it is enough to minimize the second term (or maximize it if it is considered with the positive sign) that is Log-Likelihood (LL) of the model with respect to the dataset $\mathcal{D}$:

$$\begin{aligned} \mathcal{L}(\boldsymbol{\theta}|\mathcal{D}) &= \sum_{\mathbf{x}} p_{data}(\mathbf{x}) \log p_{\boldsymbol{\theta}}(\mathbf{x}) = \\ &= \frac{1}{N_{data}} \sum_{\mathbf{x}} \sum_{k=1}^{N_{data}} \delta(x - x_d^{(k)}) \log p_{\boldsymbol{\theta}}(\mathbf{x}) = \\ &= \sum_{k=1}^{N_{data}} \log p_{\boldsymbol{\theta}}(\mathbf{x}_d^{(k)}) = \log\left(\prod_{k=1}^{N_{data}} p_{\boldsymbol{\theta}}(\mathbf{x}_d^{(k)})\right) \end{aligned} \tag{3.1.7}$$

The presence of the partition function $Z_{\boldsymbol{\theta}}$ prevents us from directly evaluating the LL (or, equivalently, the divergence KL). However, one can still minimize minus LL (so LL is maximized), using the SGD method (Section2.4.1).

Now for each parameter θ_i we can associate a conjugate variable described by the equation:

$$C_{\theta_i}(\mathbf{x}) \equiv -\frac{\partial E(\mathbf{x}, \boldsymbol{\theta})}{\partial \theta_i} \tag{3.1.8}$$

One can now calculate the variation of a parameter by doing the derivative of the LL with respect to that parameter, and since one is working with Boltzmann Distributions the calculation is quite simple:

$$\begin{aligned}\Delta\theta_i = \frac{\partial \mathcal{L}(\boldsymbol{\theta}|\mathcal{D})}{\partial \theta_i} &= \frac{\partial}{\partial \theta_i}\left[\frac{1}{N_{data}}\sum_{k=1}^{N_{data}}\big(-E(\mathbf{x}_d^{(k)},\boldsymbol{\theta}) - \log Z_{\boldsymbol{\theta}}\big)\right] = \\ &= \frac{1}{N_{data}}\sum_{k=1}^{N_{data}} C_{\theta_i}(\mathbf{x}_d^{(k)}) - \sum_{\mathbf{x}} C_{\theta_i}(\mathbf{x})\frac{e^{-E(\mathbf{x},\boldsymbol{\theta})}}{Z_{\boldsymbol{\theta}}} = \\ &= \big\langle C_{\theta_i}\big\rangle_{data} - \big\langle C_{\theta_i}\big\rangle_{model}\end{aligned} \tag{3.1.9}$$

where $\langle\cdot\rangle_{data}$ is done on the initial data (empirical distribution), while $\langle\cdot\rangle_{model}$ is done on the probability distribution of the generative model.

Now applying this result to the previous model (3.1.4), one can obtain the variation of the parameters $\mathbf{W}$ and $\mathbf{b}$:

$$\Delta W_{ij} = \frac{\partial \mathcal{L}(\boldsymbol{\theta}|\mathcal{D})}{\partial W_{ij}} = \big\langle x_i x_j\big\rangle_{data} - \big\langle x_i x_j\big\rangle_{model} \tag{3.1.10}$$

$$\Delta b_i = \frac{\partial \mathcal{L}(\boldsymbol{\theta}|\mathcal{D})}{\partial b_i} = \big\langle x_i\big\rangle_{data} - \big\langle x_i\big\rangle_{model} \tag{3.1.11}$$

In Eq.(3.1.9), the first term of the gradient is named positive term and is generally calculable from the data. However, the second term, also named negative term, is not easily calculable, because there is partition function $Z_{\boldsymbol{\theta}}$, which is intractable in the calculation of the mean. Therefore, instead of calculating the negative term, it is approximated using dynamic Monte Carlo methods. In fact, if one can sample N identically independent distributed random variables (i.i.d.) from a probability distribution $p(\mathbf{x})$, the mean of a generic function f under p can be approximated using the empirical mean:

$$\big\langle f\big\rangle = \sum_{\mathbf{x}} f(\mathbf{x})p(\mathbf{x}) \overset{N\to\infty}{\approx} \frac{1}{N}\sum_{k=1}^{N} f(\mathbf{x}^{(k)}) = \bar{f}, \text{ where } \mathbf{x}^{(k)} \sim p(\mathbf{x}) \tag{3.1.12}$$

Thus, the problem of evaluating an intractable probability distribution became a problem of sampling i.i.d. samples from the model.

To continue solving the problem, configurations can now be sampled from $p_{\boldsymbol{\theta}}$ using the MCMC process with Gibbs sampling described in the 2.3 section. The operation of this chain consists of taking a random node x_k, and considering all other nodes $\mathbf{x}_{-k}$, as fixed. After that, the state of the node is updated using the heat bath dynamics, i.e. respecting the following conditional probability:

$$
\begin{aligned}
p_{\boldsymbol{\theta}}(x_k = 1|\mathbf{x}_{-k}) = \frac{p_{\boldsymbol{\theta}}(\mathbf{x})}{p_{\boldsymbol{\theta}}(\mathbf{x}_{-k})}\bigg|_{x_k=1} &= \frac{e^{\sum_i x_i(b_i+\sum_j W_{ij}x_j)}}{\sum_{x_k=0,1} e^{\sum_i x_i(b_i+\sum_j W_{ij}x_j)}}\bigg|_{x_k=1} = \\
&= \frac{e^{(b_k+\sum_j W_{kj}x_j)}}{\sum_{x_k=0,1} e^{\sum_i x_k(b_k+\sum_j W_{kj}x_j)}}\bigg|_{x_k=1} = \frac{1}{1+e^{-(b_k+\sum_j W_{kj}x_j)}} = \\
&= \sigma\left(b_k + \sum_j W_{kj}x_j\right), \text{ with } W_{kk} = 0
\end{aligned}
\tag{3.1.13}
$$

where $\sigma(z) = \frac{1}{1+e^{-z}}$ is the sigmoid function. Furthermore, the current reaching a node can be defined as:

$$
I_k(x_{-k}) \equiv \sum_j W_{kj}x_j, \text{ with } W_{kk} = 0 \tag{3.1.14}
$$

While $\left(b_k + \sum_j W_{kj}x_j\right)$ is the activation of the node, the sigmoid function σ is named the activation function which is the mapping from activation to the probability that the node is active.

The Boltzmann Machine has the advantage of being a simple model, whose learning dynamics can be easily interpreted if the first two moments of the distribution are made to coincide with the empirical ones. However, this type of model has some serious drawbacks. The first is that it only reproduces second-order statistics and leaves out all higher-order correlations [14]. The expressivity of the model can be improved by introducing hidden variables that "extract features", as we shall see in the Restricted Boltzmann Machines (RBM) in the following section. Secondly, BMs are computationally expensive. The sampling time scales linearly with the input size, making training difficult for high-dimensionality data. In conclusion, RBMs are the natural improvement both from a computational point of view and in the development of subsequent orders to the second.

3.2 RBM

A Restricted Boltzmann Machine (RBM) [1] is an energy-based model with l visible units $\mathbf{v} = (v_1, v_2..., v_l)$ and m hidden unit $\mathbf{h} = (h_1, h_2..., h_m)$ that interact with each other but do not interact among themselves as in Fig. 3.3. The energy of an RBM is described by the following equation:

$$
E(\mathbf{v}, \mathbf{h}) = -\sum_{i=1}^{l} a_i(v_i) - \sum_{\mu=1}^{m} b_\mu(h_\mu) - \sum_{i,\mu=1}^{l,m} W_{i\mu}v_i h_\mu \tag{3.2.1}
$$

Where $a_i(\cdot)$ and $b_\mu(\cdot)$ are functions that we are free to choose, while $W_{i\mu}$ is a parameter describing the interaction between visible and hidden variables.

A generative model with this two-level interaction structure has two main advantages. The first is to allow both pairwise and higher-order correlations between visible units to be captured.

The second is to make it easier to sample the model using Block Gibbs Sampling (section 3.3.1), which makes the model easier to train.

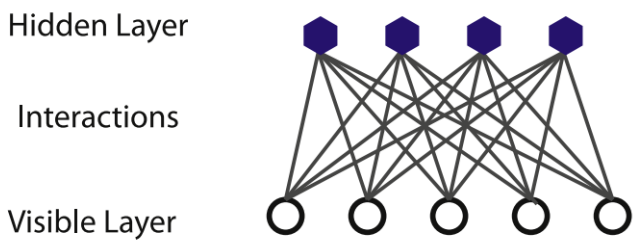

Figure 3.3: Diagram of an RBM where it is clear that interaction occurs between visible and hidden variables, but not between variables of the same type. (Image taken from Metha's review [1])

3.2.1 Bernoulli RBM

A Restricted Boltzmann Machine with visible v_i and hidden units h_μ that can only take the values $\{0, 1\}$ (discrete binary units), is named Bernoulli RBM.

In the case of a Bernoulli RBM, the functions described in the energy equation become $a_i(v_i) = a_i v_i$ and $b_\mu(h_\mu) = b_\mu h_\mu$, so the energy becomes:

$$E(\mathbf{v}, \mathbf{h}) = -\sum_{i=1}^{l} a_i v_i - \sum_{\mu=1}^{m} b_\mu h_\mu - \sum_{i,\mu=1}^{l,m} W_{i\mu} v_i h_\mu \tag{3.2.2}$$

3.2.2 RBM correlations

Starting from a probability distribution dependent on both the visible and the hidden variables $p(\mathbf{v}, \mathbf{h})$, one can calculate the resulting distribution only on the hidden variables $p(\mathbf{v})$ in the following way:

$$p(\mathbf{v}) = \mathrm{Tr}_{\mathbf{h}}\big(p(\mathbf{v}, \mathbf{h})\big) = \mathrm{Tr}_{\mathbf{h}}\left(\frac{e^{-E(\mathbf{v},\mathbf{h})}}{Z}\right) \tag{3.2.3}$$

Starting then from $p(\mathbf{v}) = \frac{e^{-E(\mathbf{v})}}{Z}$, energy can be defined as dependent only on the visible variables in the following way:

$$E(\mathbf{v}) = -\log \mathrm{Tr}_{\mathbf{h}}\left(e^{-E(\mathbf{v},\mathbf{h})}\right) = -\sum_{i=1}^{l} a_i v_i - \sum_{\mu=1}^{m} \log \mathrm{Tr}_{h_\mu}\left(e^{b_\mu h_\mu + \sum_i^l v_i W_{i\mu} h_\mu}\right) \tag{3.2.4}$$

Introducing the distribution $q_\mu(h_\mu) = \frac{e^{b_\mu h_\mu}}{Z}$ dependent on hidden units and the cumulant generating function $K_\mu(t) = \log \mathrm{Tr}_{h_\mu}(q_\mu(h_\mu) e^{t h_\mu}) = \sum_n k_\mu^{(n)} \frac{t^n}{n!}$, $E(\mathbf{v})$ can be rewritten in the following

way:

$$E(\mathbf{v}) = -\sum_{i=1}^{l} a_i v_i - \sum_{\mu=1}^{m} K_\mu\Big(\sum_{i=1}^{l} W_{i\mu} v_i\Big) \tag{3.2.5}$$

$$= -\sum_{i=1}^{l} a_i v_i - \sum_{\mu=1}^{m}\sum_{n} k_\mu^{(n)} \frac{\Big(\sum_{i=1}^{l} W_{i\mu} v_i\Big)^n}{n!} \tag{3.2.6}$$

$$= -\sum_{i=1}^{l} a_i v_i - \sum_{i=1}^{l}\Big(k_\mu^{(1)} W_{i\mu}\Big) v_i - \frac{1}{2} - \sum_{i,j=1}^{l}\Big(k_\mu^{(2)} W_{i\mu} W_{j\mu}\Big) v_i v_j + ... \tag{3.2.7}$$

where it was used $k_\mu^{(n)} = \partial_t^n K_\mu|_{t=0}$. It is seen that the energy dependent only on visible units $E(\mathbf{v})$ includes all orders of interactions between visible units, with the n-th order cumulants of $q_\mu(h_\mu)$ giving weight to the n-th order interactions between visible units. These calculations show the representative power of the Bernoulli RBM. Each hidden unit can encode interactions of arbitrarily high order. By combining many different hidden units, we can encode very complex interactions at all orders. Thus, even very simple generative models between visible and hidden variables are efficient for encoding, learning and representing complex correlations in the data.

3.3 RBM Training

RBM are a class of energy-based generative models that can be trained using the maximum likelihood estimation (MLE) procedure (see section 3.1.3). First one has to choose a cost function, which for MLE is simply the negative log-likelihood (LL), then one minimizes this cost function using one of the Stochastic Gradient Descent (SGD) methods, in our case ADAM was used. For Bernoulli RBM, which is used throughout the thesis work, the equation is:

$$\frac{\partial \mathcal{L}(\{W_{i\mu}, a_i, b_\mu\})}{\partial W_{i\mu}} = \Delta W_{i\mu} = \big\langle v_i h_\mu \big\rangle_{data} - \big\langle v_i h_\mu \big\rangle_{model} \tag{3.3.1}$$

$$\frac{\partial \mathcal{L}(\{W_{i\mu}, a_i, b_\mu\})}{\partial a_i} = \Delta a_i = \big\langle v_i \big\rangle_{data} - \big\langle v_i \big\rangle_{model} \tag{3.3.2}$$

$$\frac{\partial \mathcal{L}(\{W_{i\mu}, a_i, b_\mu\})}{\partial b_\mu} = \Delta b_\mu = \big\langle h_\mu \big\rangle_{data} - \big\langle h_\mu \big\rangle_{model} \tag{3.3.3}$$

where $\big\langle\cdot\big\rangle_{data}$ is done on the initial data, while $\big\langle\cdot\big\rangle_{model}$ is done on the model one wants to build with algorithms described in the following chapters.

3.3.1 Block Gibbs Sampling

The bipartite interaction structure, i.e. one that considers visible and hidden units, in an RBM allows the calculation of expectation values using a Markov Chain Monte Carlo (MCMC) method known as Block Gibbs sampling. The visible units $\mathbf{v}$ and the hidden units $\mathbf{h}$ do not interact with themselves, so these units are individually conditionally independent. This can be described by

the following equations:

$$p(\mathbf{v}|\mathbf{h}) = \prod_{i}^{l} p(v_i|\mathbf{h}) \tag{3.3.4}$$

$$p(\mathbf{h}|\mathbf{v}) = \prod_{\mu}^{m} p(h_\mu|\mathbf{v}) \tag{3.3.5}$$

From what was demonstrated in section 3.1.3, starting with the sigmoid function σ from Eq.(3.1.13), it can be shown that:

$$p(v_i = 1|\mathbf{h}) = \sigma\left(a_i + \sum_{\mu}^{m} W_{i\mu} h_\mu\right) \tag{3.3.6}$$

$$p(h_\mu = 1|\mathbf{v}) = \sigma\left(b_\mu + \sum_{i}^{l} W_{i\mu} v_i\right) \tag{3.3.7}$$

The sigmoid function is one of the main activation functions for artificial neurons. Thus, an RBM can be interpreted as a stochastic neural network, where nodes correspond to neurons and edges correspond to synaptic connections. Thus, a conditional probability of 1 for a single variable can be interpreted as the switching rate of a random neuron with a sigmoidal activation function.

With these expressions, expectation values can be calculated from the initial data. For the gradient, the input is a subset (mini-batch) of the observed data. For each sample in the mini-batch is simply a matter of associate the visible units v to the observed data and apply Eq.(3.3.7) using the probability for calculating the hidden variables h_μ. One then averages all samples in the mini-batch to calculate the expectation values with respect to the data.

To calculate expectation values with respect to the model, one uses Block Gibbs sampling. Due to the conditional independence between variables in the same layer, Gibbs Sampling is particularly efficient. Instead of successively sampling new values for all variables, the states of all variables in a layer can be sampled in parallel. Gibbs Sampling can be summarised as follows:

1. sampling a new state $\mathbf{h}$ for the hidden units, based on $p(\mathbf{h}|\mathbf{v})$;
2. sampling a state $\mathbf{v}$ for the visible units, based on $p(\mathbf{v}|\mathbf{h})$

The description of the Gibbs Sampling Block is as follows:

Gibbs Sampling Algorithm for RBM

Input: Initial state $(\mathbf{v}^{(0)}, \mathbf{h}^{(0)})$, Number of steps: $N \to \infty$
1) $\mathbf{h}^{(1)} \sim p(\mathbf{h}^{(0)} = 1|\mathbf{v}^{(0)})$
$\mathbf{v}^{(1)} \sim p(\mathbf{v}^{(0)} = 1|\mathbf{h}^{(1)})$

.

.

after $N \to \infty$ steps

.

.

N) $\mathbf{h}^{(N)} \sim p(\mathbf{h}^{(N-1)} = 1|\mathbf{v}^{(N-1)})$

$\mathbf{v}^{(N)} \sim p(\mathbf{v}^{(N-1)} = 1|\mathbf{h}^{(N)})$

Output: Final state after N steps: $(\mathbf{v}, \mathbf{h}) \sim (p(\mathbf{v}^{(N-1)} = 1|\mathbf{h}^{(N)}), p(\mathbf{h}^{(N-1)} = 1|\mathbf{v}^{(N-1)}))$

A schematic description of Gibbs sampling is described in Fig.3.4

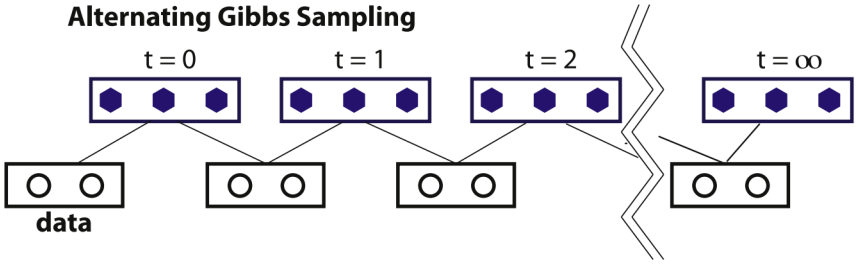

Figure 3.4: The Gibbs procedure alternately generates hidden variables and visible variables termed fantasy variables from the initial data, using the probabilities $p(\mathbf{h}|\mathbf{v})$ and $p(\mathbf{v}|\mathbf{h})$ for the generation of $\mathbf{h}$ and $\mathbf{v}$ respectively. The "time" t corresponds to the time in the MCMC and measures the number of passes between the visible and hidden layer. (Image taken from Metha's review [1])

Such sampling could continue indefinitely. The idea of Block Gibbs Sampling is to iteratively sample from the conditional distributions $\mathbf{h}_{t+1} \sim p(\mathbf{h}|\mathbf{v}_t)$ and $\mathbf{v}_{t+1} \sim p(\mathbf{v}|\mathbf{h}_{t+1})$. Since the visible and hidden units are conditionally independent, each step in this iteration can be performed by simply extracting random numbers. The samples are guaranteed to converge to the equilibrium distribution of the model in the limit where $t \to \infty$. The variables resulting from sampling are named fantasy variables. Once the RBM has been successfully trained, one can use this algorithm to generate new data from a random initial state (see section(3.5).

A shortcoming of Gibbs Sampling is that it may take many back and forth iterations to draw an independent sample. Moreover, given the finite memory and speed of the computers on which these algorithms must work, the number of samples can never be infinite.

3.3.2 Contrastive Divergence (CD)

To solve the problems caused by Block Gibbs sampling, an approximation of it named Contrastive Divergence (CD) is introduced [15]. Precisely, an algorithm that performs n Gibbs sampling blocks is referred to as CD-n. The value of n can also be chosen as 1.

When truncation is performed, no samples are extracted from the true distribution of the model. However, if the approximate gradients are reasonably correlated with the true gradient, the SGD will move in a reasonable direction. The truncation of the Gibbs algorithm prevents sampling away from the starting point because n is generally not chosen to tend to infinity. Therefore,

our generative model will be much more accurate in regions of feature space close to the training data. Hence CD-n sacrifices generalization by acquiring greater ease of training.

In the CD-n between each step of the gradient descent, there is a restart of the visible variables (and of the hidden ones) and it is a good practice to shuffle their order to facilitate a good training. Obviously, this restart does not entail a restart of the parameters a, b, W, which determine the future calculations of the gradient descent steps (calculation of the sigmoidal function used in the calculation of the hidden variables and the subsequent visible variables). The description of the CD algorithm is as follows (note that $\mathbf{h}^{(0)}$ is initialized with all values to 0):

CD Training Algorithm for RBM

Input: Initial state: $(\mathbf{v}^{(0)}, \mathbf{h}^{(0)})$ from the inital data, Initial parameter: $(\mathbf{a}_0, \mathbf{b}_0, \mathbf{W}_0)$, Number of steps: n = finite number. Epochs: N_{ep} = number of epoch

for i: **1** to $\mathbf{N_{ep}}$

1) $\mathbf{h}^{(1)} \sim p(\mathbf{h}^{(0)} = 1|\mathbf{v}^{(0)})$
$\mathbf{v}^{(1)} \sim p(\mathbf{v}^{(0)} = 1|\mathbf{h}^{(1)})$

.

.

.

n) $\mathbf{h}^{(n)} \sim p(\mathbf{h}^{(n-1)} = 1|\mathbf{v}^{(n-1)})$
$\mathbf{v}^{(n)} \sim p(\mathbf{v}^{(n-1)} = 1|\mathbf{h}^{(n)})$

SGD for updating the parameter **a**, **b**, **W**

reset of visible and hidden variables $(\mathbf{v}^{(0)}, \mathbf{h}^{(0)})$ using intial data

end

Output: RBM $(\mathbf{a}, \mathbf{b}, \mathbf{W})$ trained with CD-n algorithm.

---

where the number of epochs indicates the number of times the parameters are updated. In reality, the updating of parameters is done at each mini-batch until all data is exhausted and then there is reset. In this diagram, this step has been excluded to facilitate general understanding.

A schematic description of CD algorithm is described in Fig.3.5

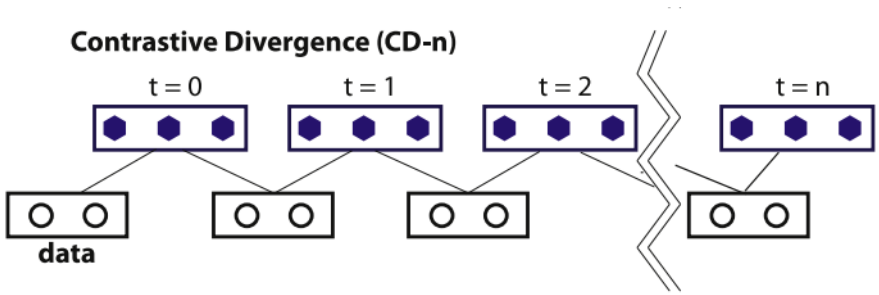

Figure 3.5: The CD procedure alternately generates hidden variables and visible variables from the initial data, using the probabilities $p(\mathbf{h}|\mathbf{v})$ and $p(\mathbf{v}|\mathbf{h})$ for the generation of $\mathbf{h}$ and $\mathbf{v}$ respectively. The "time" t corresponds to the time in the MCMC and measures the number of passes between the visible and hidden layer. (Image taken from Metha's review [1])

3.3.3 Persistent Contrastive Divergence (PCD)

A slightly different variant of the CD-n improves some of the problems present in limited sampling. This method is named Persistent Contrastive Divergence (PCD) [16]. In PCD, instead of restarting the Gibbs sampler from the data at each gradient descent step (where one step exhausts all mini-batches), Gibbs sampling starts from the fancy particles in the last gradient descent step. Since the parameters (a, b, W) change slowly with respect to Gibbs sampling, samples that have a high probability in one step of the SGD are also likely to have a high probability in the next step. Therefore, PCD does not introduce large errors in the estimation of gradients. The advantage of using fantasy unit to initialize the Gibbs sampler is that it allows the PCD to explore parts of the feature space much further away from the training dataset than could be achieved with ordinary CD-n at the same number of steps.

A schematic description of PCD algorithm is described in Fig.3.6

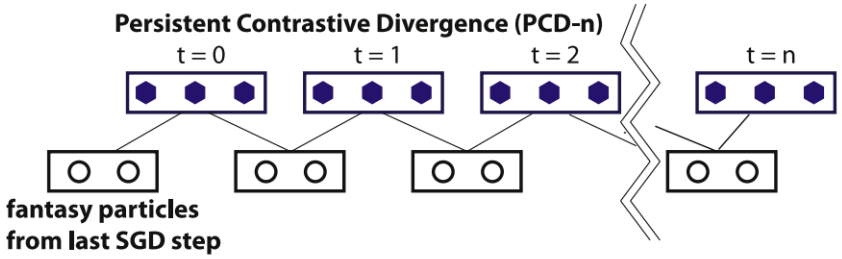

Figure 3.6: The PCD procedure alternately generates hidden variables and visible variables from the fantasy variable extracted after the SGD of the previous epoch, using the probabilities. The "time" t corresponds to the time in the MCMC and measures the number of passes between the visible and hidden layer. (Image taken from Metha's review [1])

The description of the PCD algorithm is as follows:

PCD Training Algorithm for RBM

Input: Initial state: $(\mathbf{v}^{(0)}, \mathbf{h}^{(0)})$ from the inital data, Initial parameter: $(\mathbf{a}_0, \mathbf{b}_0, \mathbf{W}_0)$, Number of steps: n = finite number. Epochs: N_{ep} = number of epoch

for i: **1** to $\mathbf{N_{ep}}$

1) $\mathbf{h}^{(1)} \sim p(\mathbf{h}^{(0)} = 1|\mathbf{v}^{(0)})$
   $\mathbf{v}^{(1)} \sim p(\mathbf{v}^{(0)} = 1|\mathbf{h}^{(1)})$

.

.

.

n) $\mathbf{h}^{(n)} \sim p(\mathbf{h}^{(n-1)} = 1|\mathbf{v}^{(n-1)})$
   $\mathbf{v}^{(n)} \sim p(\mathbf{v}^{(n-1)} = 1|\mathbf{h}^{(n)})$

SGD for updating the parameter $\mathbf{a}$, $\mathbf{b}$, $\mathbf{W}$

The new initial variable become the last fantasy variable $(\mathbf{v}^{(0)}, \mathbf{h}^{(0)}) = (\mathbf{v}^{(n)}, \mathbf{h}^{(n)})$

end

Output: RBM $(\mathbf{a}, \mathbf{b}, \mathbf{W})$ trained with PCD-n algorithm.

3.3.4 Random Scheme (Rdm)

A particular method used in the paper by Decelle, Furtlehner, Seoane [3] is to initialize the visible and hidden variables on random conditions (data with uniform distribution), perform n Gibbs sampling steps and use the final configurations to estimate the gradient with respect to the initial data. Once this is done, the initial variables are reset to random values and proceed for a given number of epochs. This algorithm is termed Rdm-n. The Rdm-n scheme is useful for two reasons: the first is that the MCMC chains are initialized from configurations not correlated with the initial dataset, and the second is that the sampling protocol used during learning is identical to that used to generate new samples from scratch with the learned RBM.

A schematic description of Rdm algorithm is described in Fig.3.7

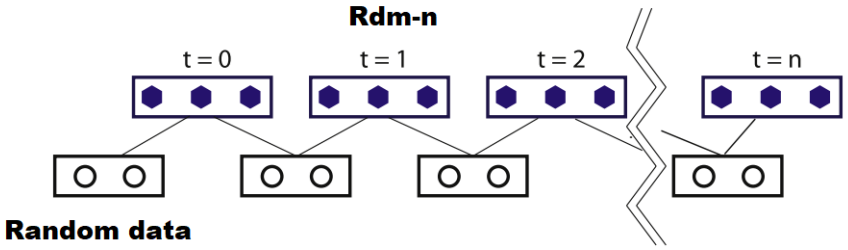

Figure 3.7: The Rdm procedure alternately generates hidden variables and visible variables from the random variable and using the probabilities distribution. The "time" t corresponds to the time in the MCMC and measures the number of passes between the visible and hidden layer.

The description of the Rdm algorithm is as follows:

Rdm Training Algorithm for RBM

Input: Initial state: $(\mathbf{v}^{(0)}, \mathbf{h}^{(0)})$ from the uniform random data, Initial parameter: $(\mathbf{a}_0, \mathbf{b}_0, \mathbf{W}_0)$, Number of steps: n = finite number. Epochs: N_{ep} = number of epoch

for i: **1** to $\mathbf{N_{ep}}$

1) $\mathbf{h}^{(1)} \sim p(\mathbf{h}^{(0)} = 1|\mathbf{v}^{(0)})$
$\mathbf{v}^{(1)} \sim p(\mathbf{v}^{(0)} = 1|\mathbf{h}^{(1)})$

.

.

.

n) $\mathbf{h}^{(n)} \sim p(\mathbf{h}^{(n-1)} = 1|\mathbf{v}^{(n-1)})$
$\mathbf{v}^{(n)} \sim p(\mathbf{v}^{(n-1)} = 1|\mathbf{h}^{(n)})$

SGD for updating the parameter $\mathbf{a}$, $\mathbf{b}$, $\mathbf{W}$

reset of visible and hidden variables $(\mathbf{v}^0, \mathbf{h}^0)$ using starting random data

end

Output: RBM $(\mathbf{a}, \mathbf{b}, \mathbf{W})$ trained with Rdm-n algorithm.

3.4 Centring Trick

RBM has over time shown two types of problems. The first is that the gradient approximation bias introduced by the use of a few Gibbs Sampling steps can lead to a divergence of log-likelihood (LL) during training. That is solved by the use of more advanced techniques (parallel tempering which is not discussed in this thesis work). The second is that the learning process is not invariant with respect to the data representation, i.e. performance is different if in a Bernoulli RBM one inverts bits of value 0 with bits of value 1 (bit flip) [5].

The solution to this problem is to subtract the mean of the data from the visible variables [17]. This solution is named the Centring Trick and makes the RBM invariant to the data representation. Furthermore, the Centring Trick can also be used on hidden variables [18].

In the Centring Trick, two offsets are defined for visible variables $\boldsymbol{\mu} = (\mu_1, \mu_2, ..., \mu_l) = \langle \mathbf{v} \rangle_d$ and hidden variables $\boldsymbol{\lambda} = (\lambda_1, \lambda_2, ..., \lambda_m) = \langle \mathbf{h} \rangle_d$. Consequently, energy can be rewritten as follows:

$$E(\mathbf{v}, \mathbf{h}) = -(\mathbf{v} - \boldsymbol{\mu})^T \mathbf{a} - \mathbf{b}^T (\mathbf{h} - \boldsymbol{\lambda}) - (\mathbf{v} - \boldsymbol{\mu}) \mathbf{W} (\mathbf{h} - \boldsymbol{\lambda}) \tag{3.4.1}$$

The conditional probabilities used in Block Gibbs Sampling are now given by:

$$p(v_i = 1|\mathbf{h}) = \sigma\left(a_i + \sum_{\mu}^{m} W_{i\mu}(h_\mu - \lambda_\mu) \right) \tag{3.4.2}$$

$$p(h_\mu = 1|\mathbf{v}) = \sigma\left(b_\mu + \sum_{i}^{l} W_{i\mu}(v_i - \mu_i) \right) \tag{3.4.3}$$

Finally, the equation for the training are:

$$\frac{\partial \mathcal{L}(\{W_{i\mu}, a_i, b_\mu\})}{\partial W_{i\mu}} = \Delta W_{i\mu} = \langle (v_i - \mu_i)(h_\mu - \lambda_\mu) \rangle_{data} - \langle (v_i - \mu_i)(h_\mu - \lambda_\mu) \rangle_{model} \tag{3.4.4}$$

$$\frac{\partial \mathcal{L}(\{W_{i\mu}, a_i, b_\mu\})}{\partial a_i} = \Delta a_i = \langle v_i - \mu_i \rangle_{data} - \langle v_i - \mu_i \rangle_{model} = \langle v_i \rangle_{data} - \langle v_i \rangle_{model} \tag{3.4.5}$$

$$\frac{\partial \mathcal{L}(\{W_{i\mu}, a_i, b_\mu\})}{\partial b_\mu} = \Delta b_\mu = \langle h_\mu - \lambda_\mu \rangle_{data} - \langle h_\mu - \lambda_\mu \rangle_{model} = \langle h_\mu \rangle_{data} - \langle h_\mu \rangle_{model} \tag{3.4.6}$$

Note that the only parameter that is affected by the centering trick, i.e. undergoes a translation of the visible and hidden variables in the averaging calculation, is the interaction between the visible and hidden levels W, to be precise its variation ΔW. It can be shown that if offsets are set to expectation values, centred RBM is invariant to any variable shift, not only to flip transformations [19].

Another method that could be used equivalent to the one just shown is that of centring the gradient instead of the parameters [19].

To show the complete algorithm exploited in the Centring Trick, the following diagram can be

observed (algorithm also used in the programmes of the thesis work):

Centering Trick Training Algorithm for RBM

Input: Input Data: $(\mathbf{v}^{(0)}, \mathbf{h}^{(0)})$, Input parameter: $\mathbf{a}_0, \mathbf{b}_0, \mathbf{W}_0, \boldsymbol{\mu} = \langle \mathbf{v} \rangle_d, \boldsymbol{\lambda} = \langle \mathbf{h} \rangle_d$, Number of data: N, Number of steps: n = finite number, Sample for each mini-batch = M Number of mini-batch = $\frac{N}{M}$, Epochs: N_{ep} = number of epoch, Moving average factor: ξ_μ, ξ_λ

1. **for i**: **1** to $\mathbf{N_{ep}}$
2. **for k**: **1** to $\frac{\mathbf{N}}{\mathbf{M}}$
3. 1) $\mathbf{h}_k^{(1)} \sim p(\mathbf{h}_k^{(0)} = 1|\mathbf{v}^{(0)})$
4. $\mathbf{v}_k^{(1)} \sim p(\mathbf{v}_k^{(0)} = 1|\mathbf{h}^{(1)})$
5. .
   .
   .
6. n) $\mathbf{h}_k^{(n)} \sim p(\mathbf{h}_k^{(n-1)} = 1|\mathbf{v}^{(n-1)})$
7. $\mathbf{v}_k^{(n)} \sim p(\mathbf{v}_k^{(n-1)} = 1|\mathbf{h}^{(n)})$
8. Calculation of averages in the minibatch:
9. $\boldsymbol{\mu}_k = \langle \mathbf{v}_k \rangle_d$
10. $\boldsymbol{\lambda}_k = \langle \mathbf{h}_k \rangle_d$

    Updating of parameters **a** and **b** to the new offset (minibatch):
11. $\mathbf{a} = \mathbf{a} + \xi_\lambda \mathbf{W}(\boldsymbol{\lambda}_k - \boldsymbol{\lambda})$
12. $\mathbf{b} = \mathbf{b} + \xi_\mu \mathbf{W}^T(\boldsymbol{\mu}_k - \boldsymbol{\mu})$

    Updating of parameters $\boldsymbol{\mu}$ and $\boldsymbol{\lambda}$ to the new offset (minibatch):
13. $\boldsymbol{\mu} = (1 - \xi_\mu)\boldsymbol{\mu} + \xi_\mu \boldsymbol{\mu}_k$
14. $\boldsymbol{\lambda} = (1 - \xi_\lambda)\boldsymbol{\lambda} + \xi_\lambda \boldsymbol{\lambda}_k$
15. SGD for updating the parameter **a**, **b**, **W** (ADAM algorithm described in section 2.4.1)
16. reset of visible and hidden variables $(\mathbf{v}^0, \mathbf{h}^0)$ according to the chosen algorithm i.e. CD-n, PCD-n, Rdm-n
17. **end**
18. **end**

Output: RBM $(\mathbf{a}, \mathbf{b}, \mathbf{W})$ trained with one generic algorithm using Centering Trick.

Note that offsets are updated using a moving average with sliding factors $\xi_\mu, \xi_\lambda \in (0, 1)$ (usually $\xi \sim 0.01$). This is done to obtain a more uniform approximation of the parameter updates in case the approximation of the mean values may be distorted.

3.5 RBM testing

Testing a Restricted Boltzmann Machine helps understand the quality of the training.

Since the Restricted Boltzmann Machine is a generative model, to evaluate the goodness of the training, the sample must be generated and compared with the initial learning data, which are

also used to calculate the maximum likelihood gradient.

To generate a sample, as mentioned in the section 3.3.1, we just start from random noise and apply the Gibbs Sampling Block. However, computationally it is impossible to reach $N_{step} \to \infty$, so the total number of samplings, defined as $t_{g,max}$ will be chosen according to the interest of the particular type of test and the type of training done.

The sample generative algorithm for a RBM is as follows:

Generative Algorithm for RBM

Input: Initial state: $(\mathbf{v}^{(0)}, \mathbf{h}^{(0)})$ from the random data, Parameter of RBM trained: $(\mathbf{a}, \mathbf{b}, \mathbf{W})$, Number of steps: $t_{g,max}$

1) $\mathbf{h}^{(1)} \sim p(\mathbf{h}^{(0)} = 1|\mathbf{v}^{(0)})$
$\mathbf{v}^{(1)} \sim p(\mathbf{v}^{(0)} = 1|\mathbf{h}^{(1)})$

.

.

.

$t_{g,max}$) $\mathbf{h}^{(t_{g,max})} \sim p(\mathbf{h}^{(t_{g,max}-1)} = 1|\mathbf{v}^{(t_{g,max}-1)})$
$\mathbf{v}^{(t_{g,max})} \sim p(\mathbf{v}^{(t_{g,max}-1)} = 1|\mathbf{h}^{(t_{g,max})})$

Output: Sample generated $(\mathbf{v}^{(t_{g,max})}, \mathbf{h}^{(t_{g,max})})$ starting from trained RBM.

To compare the generated sample with the initial dataset, specific functions (Section 3.5.1) are used which allow to evaluating of how far the dataset is from the generated model. These functions are described in the next section, where it is also explained why the choice made for this thesis work falls exclusively on one of these functions. A detail to note is that the calculation of the distance function between the initial dataset and the generated model can also be done between one sampling step and another, in such a way as to be able to build a curve on how the goodness of the generated sample changes as the number of samplings MCMC, termed t_G.

3.5.1 Testing Function

There are many functions or observables that make it possible to understand the closeness between the generated images and those of the initial dataset, such as Log-Likelihood (LL), Energy, Error of the Second or Third Moment, Spectral Power Density, Entropy, etc... [3].

Adversarial Accuracy Indicator

The function chosen is named Adversarial Accuracy Indicator [2] and is used to measure the similarity of the data obtained from a generative model to the training dataset that was used to train the model. The metric is based on the idea of nearest-neighbor data. Two sets are defined: the first set that only contains samples generated $T \equiv \{s^{(m)}_{RBM}\}^{N_s}_{m=1}$ and the second set that only contains samples from the initial dataset $S \equiv \{s^{(m)}_{D}\}^{N_s}_{m=1}$ where N_s is the number of samples generated (and randomly selected from the initial dataset). In the analyses of this thesis work, $N_s = 100$ was chosen for the speed of computation.

The next step is to calculate the Minimum Euclidean Distance for each sample m in T, from a sample in S, i.e.:

$$d_{TS}(m) = min_n \left|\left| s_{RBM}^{(m)} - s_D^{(n)} \right|\right| \tag{3.5.1}$$

Therefore, the "nearest-neighbor" distance between the generated sample and the original dataset is taken. The same calculation of the distance can be performed from the dataset to the generated sample, finding $d_{ST}(m)$, but also between a sample m and another sample from the same dataset, i.e. between generated samples or between samples from the initial dataset, thus obtaining $d_{TT}(m)$ and $d_{SS}(m)$ respectively (in the latter two cases $m \neq n$). The calculations are summarized in the following equations:

$$d_{ST}(m) = min_n \left|\left| s_D^{(m)} - s_{RBM}^{(n)} \right|\right| \tag{3.5.2}$$

$$d_{TT}(m) = min_n \left|\left| s_{RBM}^{(m)} - s_{RBM}^{(n)} \right|\right| \tag{3.5.3}$$

$$d_{SS}(m) = min_n \left|\left| s_D^{(m)} - s_D^{(n)} \right|\right| \tag{3.5.4}$$

Once these four distances have been obtained, the frequency with which one finds the nearest-neighbors of a data point in the same set that already contained that point can be estimated, i.e:

$$\mathcal{A}_T = \frac{1}{N_s} \sum_m^{N_s} \mathbf{1}(d_{TT}(m) < d_{TS}(m)) \tag{3.5.5}$$

$$\mathcal{A}_S = \frac{1}{N_s} \sum_m^{N_s} \mathbf{1}(d_{SS}(m) < d_{ST}(m)) \tag{3.5.6}$$

In case the generated samples are statistically indistinguishable from those in the initial dataset, these two measures $\mathcal{A}_T$ and $\mathcal{A}_S$ should converge to the value of a random guess, i.e. 0.5, for $N_s \to \infty$. Furthermore, $\mathcal{A}_S$ close to 1 means that the generated samples are very different from the true ones while, if close to 0, it means that the generated samples are so close to the true ones that overfitting of the initial dataset occurs. Conversely, if $\mathcal{A}_T$ is close to 1, the generated samples are all very close to each other, which may be related to a lack of diversity in the reference set or to the fact that the generated samples are very far from the true ones, instead if $\mathcal{A}_T$ is close to 0 is overfitting again.

In this thesis work, the Mean Quadratic Deviation (MSE) estimate of the two indicators is used for simplicity:

$$\mathcal{E}^{(AAI)} = (\mathcal{A}_S - 0.5)^2 + (\mathcal{A}_T - 0.5)^2 \tag{3.5.7}$$

For this quantity $\mathcal{E}^{(AAI)}$, the closer it is to 0, the more it will indicate closeness between the initial dataset and the generated samples. In general throughout the thesis work, acceptable generated samples are considered if the values of $\mathcal{E}^{(AAI)} < 7 \cdot 10^{-2}$

3.6 RBM Mixing Time

The mixing time (MT) is the number of Monte Carlo iterations required to sample independent configurations of a model. It can be shown that MT plays a crucial role in the dynamics and stability of a trained model. Furthermore, from empirical findings, MT increases with learning, so the number of epochs chosen for a given learning could change the regime of the RBM. The RBM regimes are generally two: equilibrium and out of equilibrium (OOE) and this is determined by the mixing time and the number of steps, n, used in Gibbs sampling [3].

The MT can be defined as the relaxation time of the MCMC dynamics of our RBM. It depends on N_{ep} (number of epochs) or t_{up} (number of updates of parameters), and it evaluates the correlation between the first and subsequent samples generated up to about 10^5 MCMC samples. What is expected is that the correlation decreases as the number of samples increases [3].

The steps to calculate the MT are summarized in the following procedure [3]:

1. training of the RBM using Rdm-n with the same parameters as the machine (N_{ep}, η, N_h) on which the mixing time is to be calculated, thus obtaining the a, b, W values of the trained RBM. From now on, all calculations of the following steps will be done on the following trained machine.

2. start of sampling and calculation of magnetisation for approximately each sampling up to $t_G \sim 10^5$ in the following way: $m_i(t) = \sigma\big(\sum_\mu^{N_h} W_{i\mu}(h_\mu(t) - \lambda_\mu) + a_i\big)$ where $m_i(t)$ is averaged over N_{samp} samples;

3. calculation of $C_m(t) = \frac{1}{N_v}\sum_i^{N_v}(m_i(t) - \bar{m}_i)(m_i(0) - \bar{m}_i)$, where $\bar{m}_i = m_i(10^5)$;

4. calculation of $\rho_m(t) = \frac{C_m(t)}{C_m(0)}$;

5. fit of $\rho_m(t)$ with $Ae^{-\frac{t}{t_\alpha}}$, where t_α is mixing time searched and A is another parameter of the fit (example in Fig.3.8).

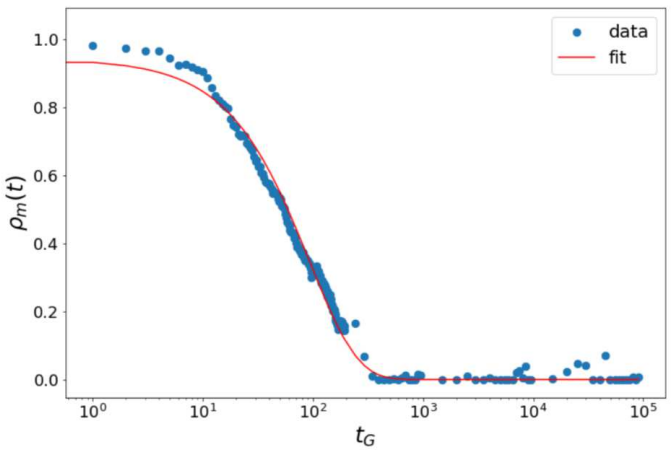

Figure 3.8: fit of $\rho_m(t)$ vs t_G with the exponential curve $Ae^{-\frac{t_G}{t_\alpha}}$. This calculation is derived from an RBM with $N_h = 90$, a training of 30 epochs (that have 200 minibatch for each, so are 6000 parameter updates) using the mixture Rdm-100 algorithm, calculated on 100 generated samples of MNIST dataset (section 4.2). The result gives a mixing time $t_\alpha \sim 95$.

Note that the mixing time should be calculated for each trained machine to ensure the type of regime it is in. Since this is not the main purpose of the thesis, and since the validity of Mixing Time has already been proved [3], the type of regime (OOE or equilibrium) will be deduced from the observation of $\mathcal{E}^{(AAI)}$. Although Mixing Time should be proved for every RBM.

This follow subsection describes how to recognise the type of regime the RBM is in, given the mixing time.

3.6.1 Out of equilibrium regime

The classical training procedures of an RBM can lead to two distinct regimes: an equilibrium and an out of equilibrium (OOE) regime. The equilibrium regime is observed if the number of MCMC steps, to train the machine and estimate the gradient (n), is greater than the algorithm's Mixing Time (MT), $n > t_\alpha$. Instead, the OOE regime occurs for $n < t_\alpha$. This means that learning an RBM that operates in the equilibrium regime is expensive and means that most of the work in the literature undoubtedly operates in the OOE regime [3]. In this thesis we have also found an intermediate regime between OOE and equilibrium, termed pseudo-equilibrium. We assume that in this regime $n \sim t_\alpha$.

In the OOE region, the equilibrium distribution of the machine at the end of learning (after t_G very large) is significantly different from that of the initial dataset. However, it can be shown that the OOE regime can generate good samples with short training times: to be precise for $t_G \simeq n$, where t_G is the number of MCMC steps to generate a sample and n is the number of MCMC steps exploited in the Gibbs Sampling Block in training.

3.7 Practical advice on RBM

For RBM training, there is a summary of useful tips compiled by Geoffrey Hinton [20]. Obviously, these tips are not exhaustive and must be contextualized to the particular training and testing procedures and the type of dataset.

The most useful tips are as follows:

1. **Size of mini-batches**: for datasets containing a small number of equiprobable classes, the ideal size of mini-batches is often equal to the number of classes. Furthermore, each mini-batch should contain an example of each class to reduce sampling error when estimating the gradient for the entire training set. For other datasets, it is necessary to first randomize the order of the training examples and then use mini-batches of approximately 10 units.

2. **Initialisation of parameters**: ensure that the weights have random initial values to break the symmetry. Hinton suggests taking the weights W_{ij} from a Gaussian with zero mean and standard deviation $\sigma = 0.01$. Instead, Glorot and Bengio [21] propose the standard deviation $\sigma = \frac{2}{\sqrt{N_v+N_h}}$, where N_v and N_h are the numbers of visible and hidden units respectively, to scale with the size of the layers. The bias of the hidden units b is initialized to zero.

3. **Number of hidden units**: starting from the number of bits required to describe each data vector (N_v). Then one should multiply this estimate by the number of training cases and use N_h lower by about an order of magnitude than the value obtained from multiplication. In cases of redundancy in the dataset, this number can be lowered.

4. **Monitoring Overfitting**: the main idea would be to calculate and compare the average free energy of the model while it is training and that of the initial dataset, and if this difference starts to increase, overfitting occurs.

5. **Learning rate**: in general, it is useful to reduce the learning rate in the later stages of training, because as the number of epochs increases, the gradient becomes small, and consequently fewer steps are needed to bring the function to be minimized closer to the minimum.

Chapter 4

Numerical Experiment

This thesis aims to reproduce and study the behavior of a simplified RBM in the out of equilibrium (OOE) and equilibrium regimes, comparing the results with literature [3]. Once the appropriate comparisons have been made, the behavior of an RBM will be studied by varying the number of sampling within the Gibbs Sampling Block in the same training procedure.

The algorithm mainly used in the thesis is the Rdm-n, which allows an easy study of the OOE regime and demonstrates the closeness to the literature [3]. PCD-n algorithm is also briefly studied just for comparison because works on an equilibrium regime.

This chapter begins by talking about the setup of the numerical experiment. We provide full details, so that the numerical experiment can be reproduced. Next, the dataset used for the whole experiment (MNIST) is introduced. In addition, the dataset is simplified so that the training can be computed more quickly. Once this is done, the number of machines needed that can guarantee statistically relevant data is verified. At this point, we introduce an innovative training method, working with mixtures of Rdm-n, and studying how the RBM goes from the OOE regime to the pseudo-equilibrium regime and vice versa. A brief report is made on the number of hidden units to try to justify the results obtained in the study of regimes. Finally, some Python codes used in this thesis are shown.

4.1 Setup

The entire thesis is based on the execution of computer programs of the algorithms described in the previous sections using Python as a programming language. Consequently, the only device used is a computer. The choice of the characteristics of the computer or server relied upon is crucial, as considerable computing capacity is required to train many RBMs. To work faster, the "tensorflow" library is typically used, which makes use of the GPU (Graphics Processing Unit), which allows parallel calculations. However, the choice was made to build simplified RBMs and work exclusively with the CPU, using a Laptop with Google Colab, which allows to execution of the required code. This choice was made to demonstrate the stability and reproducibility of the RBM results by exploiting a code created practically from scratch. In terms of technical

characteristics, the basic Google Colab servers typically provide $1.4 \div 1.6\ GB$ of RAM with rare peaks at $1.9\ GB$, while the disk space is $23.62\ GB$. An important work-accelerating property of Google Colab is that it can run a maximum of 5 programs in different windows from the same account. As a result, it is possible to train several machines in parallel and consequently the work time for training multiple RBM is reduced.

4.2 Dataset

For this work, the MNIST dataset [22] is used, which consists of 60000 greyscale numeric images of the handwritten digits 0 to 9 in a 28x28 pixel grid, which can be converted into a string of 784 numeric values in the range of 0 to 255 indicating the intensity of the greys. Working exclusively with Bernoulli RBM, these numerical values must first be scaled to a range between 0 and 1. This is done by simply dividing the values in the string by 255. Then they must be converted to bits 0 (black) or 1 (white). For this step, an interval must be chosen such that a given grey value falls either into white (number) or black (background) and to make the numbers visually classifiable, all values above 0.3 are taken as white (equal 1), while the rest are considered black (Fig. 4.1).

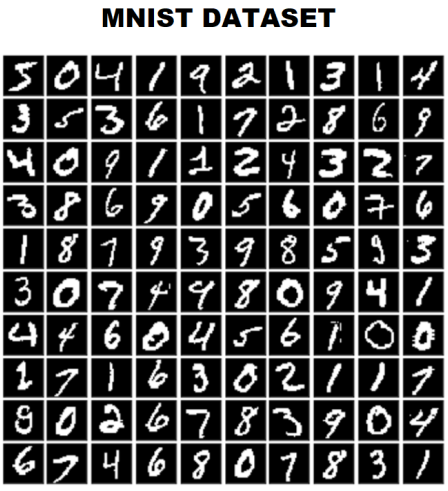

Figure 4.1: 100 sample are taken from the MNIST dataset, already scaled and converted to bits 0 (black or background) and 1 (white or number). With this configuration, the numbers are visually distinguishable.

As mentioned in section 4.1 the available calculation capacities are limited, therefore, we chose firstly, to take $N = 10000$ samples instead of 60000, secondly, to reduce the number of bits from a 28x28 pixel grid to a 14x14 pixel grid. This procedure is performed by averaging the values of 4 neighboring pixels over the entire image, resulting in 196 visible units for the sample. This procedure still keeps the images of the numbers distinguishable (Fig. 4.2).

The purpose of a generic RBM in this thesis is to learn the number images of the MNIST dataset reduced to 14x14 pixels during the training procedure, to generate new number images from random noise and to test the distance between the generated images and the starting images of the dataset using testing function described in section 3.5.1.

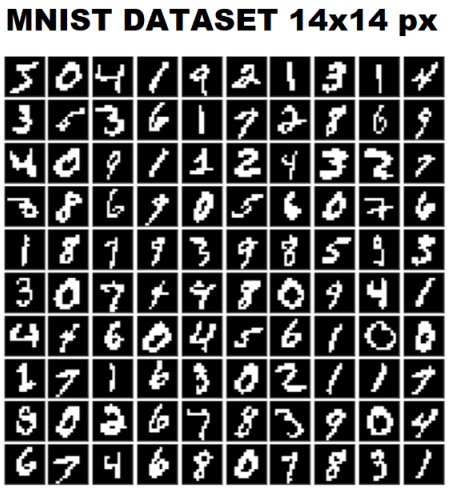

Figure 4.2: 100 sample are taken from the MNIST dataset with 14x14 pixel.

4.3 Training Setup

In section 3.7, several practical tips for improving efficiency in RBM training were discussed. These hints were initially put into practice. However, some modifications were necessary to adapt to the specific problem. This section will give a brief account of the choices made in this thesis.

4.3.1 Number of mini-batches

The recommended number of samples within a mini-batch for our particular case would be one equal to the number of digits to be learned. In our case, with the MNIST dataset, there are 10 digits (i.e., 0 to 9), consequently, the simple choice would be to put 10 samples per mini-batch. However, this choice leads to a degradation of the RBM performance and this is because in these 10 samples, one should use all different digits. Therefore, each sample should be a different digit.

To avoid increasing the number of operations that the program should perform, it was chosen to take $M = 50$ samples within each mini-batch, to speed up the execution and be sure to have the same diversity of digits on average. Since the number of data considered is $N = 10000$, one have that the total number of mini-batches is $N_{MB} = \frac{N}{M} = 200$.

4.3.2 Initial Parameters

In the thesis case, the initialization of Glorot and Bengio [21] was followed, with initial parameters a and W being initialized randomly between 0 and 1, but multiplied by $\sigma = \frac{2}{\sqrt{N_v+N_h}}$. In addition, the factor 2 over sigma was varied (between 0.1 and 8) to check the difference in the quality of the RBM. However, it seems that with these parameter initialization, in the case dataset and RBM, it is not so relevant.

To have initial starting conditions of the parameters as similar as possible between one RBM and another, the following was used $\sigma = \frac{0.1}{\sqrt{N_v+N_h}}$.

Note that as suggested in the 3.7 section the b parameter of hidden units is initialized to 0.

4.3.3 Number of Hidden Unit

For training a model, the choice of the number of hidden units is crucial. Generally, if supported by the right number of epochs, N_h sufficiently large, allows complex patterns to be derived from within the visible units or data.

In this work, the number of hidden units $N_h = 90$ for Rdm-n was chosen as a trade-off between computation speed and complexity collection. While for PCD-10 test has been chosen $N_h = 50$ for the same reason.

A brief separate section describes the choice of the number of hidden units and the consequences this has on the quality of the training and the generated samples.

4.3.4 Learing Rate

The learning rate η allows the intensity of parameter updating to be adjusted and is generally recommended to be decreasing as the number of epochs increases, that is, when it is closer to the local minimum.

In this thesis, the ADAM algorithm is used in the SGD, which does not require η modification because the way it is structured works efficiently anyway. In any case, its value was set at $\eta = 0.008$.

Also if is recommended in the ADAM algorithm to use $\epsilon = 10^{-8}$, better results were seen using $\epsilon = 10^{-10}$ for the Rdm-n algorithm, while for PCD-n that value was kept.

4.3.5 Number of epochs

The number of epochs is important to understand how many times the parameters are updated.

Again, the number of epochs was chosen to ensure the goodness of the generated samples without drastically increasing the computation time. Accordingly, the number of epochs was set to $N_{ep} = 30$ for Rdm-n and since there is $N_{MB} = 200$ for each epoch in which parameters are updated, 6000 parameter updates occur in the 30 epochs. While PCD-10 test has been chosen $N_{ep} = 100$ (20000 parameters updates) because fewer N_{ep} gave poorly reproducible results.

For an in-depth study of regime changes, machines were trained at $N_{ep} = 150$ epochs (30000 parameters updates).

4.4 Average goodness of the training

The first step to be checked is the reproducibility of the individual RBM trained, i.e. whether the individual RBM trained can individually acquire a high statistical weight.

The procedure that tries to prove reproducibility consists of training a number of RBMs one after another while keeping fixed all values of learning rate (η), number of hidden units (N_h)

and number of epochs (N_{ep}). Next, we generate samples from the machine trained with Gibbs sampling, compute $\mathcal{E}^{(AAI)}$ for checking the goodness-of-fit of the RBMs and looking for whether one or more of the many trained machines deviates heavily from the behavior of all the others.

4.4.1 Rdm-10 test

To perform this test, the Rdm-n algorithm was chosen. In this case, it was picked with $n = 10$ (so the algorithm is Rdm-10), which works in the out of equilibrium (OOE) regime. For all simulations using Rdm-n the number of visible and hidden units are kept respectively: $N_v = 196(= 14^2)$, $N_h = 90$. The learning rate is $\eta = 0.008$ and the SGD algorithm used is ADAM. In the literature [3], the behavior obtained from this algorithm is that of a minimum in the function $\mathcal{E}^{(AAI)}$ for $n \simeq t_G$, which is also obtained in our case. However, as can be seen, already after 10 trained RBMs, there is one curve that deviates from the behavior of all others (Fig.4.3 on the left).

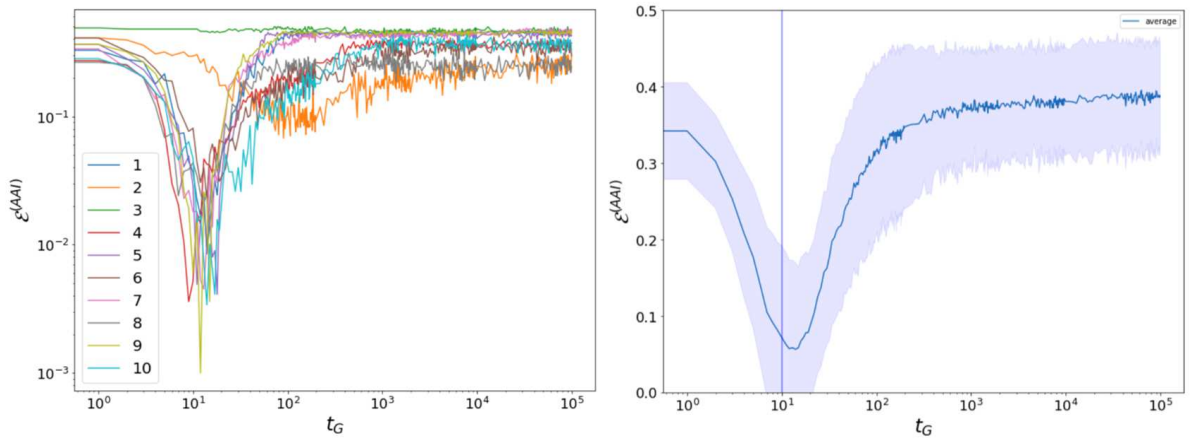

Figure 4.3: The image on the **left** shows $\mathcal{E}^{(AAI)}$ vs t_G both in logarithmic scale of 10 RBMs trained with $n = 10$ of which two (curves number 2 and 3) deviates heavily from the behaviour of all the others. As can be seen, almost all of the RBMs exhibit a minimum for $t_G \simeq n$. The image on the **right** shows $\mathcal{E}^{(AAI)}$ vs t_G, in linear scale of the average of 20 RBMs trained with $n = 10$. The average curve has a minimum at $t_G \simeq n$, which is in line with the results obtained in the literature. The vertical line is the curve $t_G = n$, while the blue band indicates the standard variation of the individual RBM from the mean (standard deviation)

Consequently, it was chosen, to have statistical relevance, to work with at least 10 RBM, and to consider the average of the $\mathcal{E}^{(AAI)}$ curves to discuss the data obtained.

An example of 100 samples generated by an Rdm-10 machine starting from random noise are described in Fig.4.4 and shows entry into the minimum at $t_G \simeq n$ and then enters in the OOE regime represented by all digits of the value "1" and remaining there for the rest of the sampling until $t_G = 10^5$.

One thing that helps to discuss the data is to consider the standard deviation ($\sigma_{\mathcal{E}}$). By calculating the standard deviation on a certain number of $\mathcal{E}^{(AAI)}$ curves for each t_G, it is possible to show graphically the variability in quality, of the generated images, around the mean value of the curve

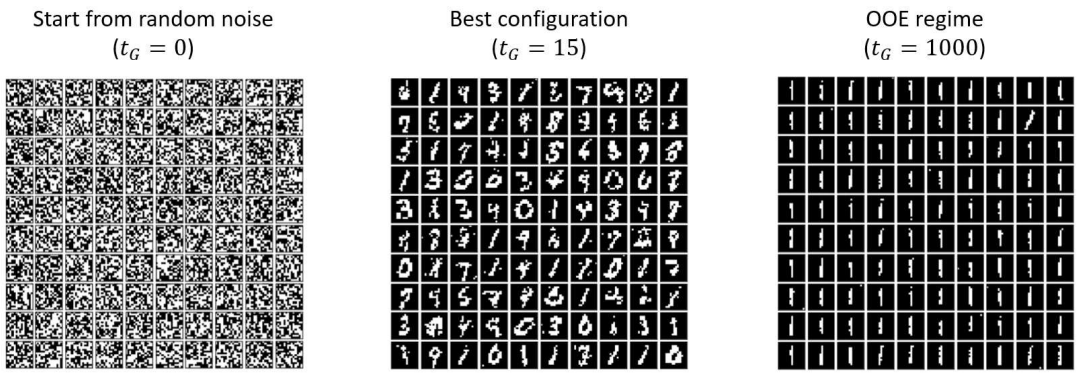

Figure 4.4: example of 100 samples generated from random noise ($t_G = 0$). The RBM is trained for 30 epochs with Rdm-10. As can be seen, the best configuration is reached for $n = 15$, very close to $n = 10$ of the Rdm algorithm. It is not shown in the figure, but already after $t_G = 100$ samplings the figures are very similar to those for $t_G = 1000$ demonstrating stabilisation in the Out of Equilibrium regime with all digits equal "1".

$\bar{\mathcal{E}}^{(AAI)}$. In fact, the band is constructed for each t_G by taking the two extremes: $\bar{\mathcal{E}}^{(AAI)} - \sigma_{\mathcal{E}}$ and $\bar{\mathcal{E}}^{(AAI)} + \sigma_{\mathcal{E}}$. It can be seen (Fig. 4.3 on the right) that for example in the average of 20 RBM trained with Rdm-10, the band delimited by the standard deviation has extremes that trace the shape of the curve.

Probably more epochs could reduce the anomalies on the single RBM. In fact, in the following sections, graphs on Rdm-100 working at $N_{ep} = 150$ epochs show a decrease in standard deviation as the number of epochs increases (subsection4.5.2).

We will see that in almost all curves studied, around the minimum, the standard deviation exceeds the value of the mean, thus trespassing the band into a non-physical zone (where $\mathcal{E}^{(AAI)} < 0$). Consequently, the lower band is truncated at the minimum physical possibility, i.e. $\mathcal{E}^{(AAI)} = 0$. Even if the minimum were exactly at 0, this would indicate that the samples generated and MNIST dataset are indistinguishable from the generated samples, further confirming the effectiveness of the RBM training. Since the bands have an amplitude with respect to the mean value that is considered acceptable(Fig. 4.3 right), the average of $\mathcal{E}^{(AAI)}$ of at least 10 RBM will be used in the following sections for the analysis of Rdm-n data.

4.4.2 PCD-10 test

In the case of the PCD-10, training can be done using a lower number of hidden units, i.e. $N_h = 50$, but $N_{ep} = 100$ number of epochs because for lower epochs the reproducibility is lost. Furthermore, in the PCD case, the updating of parameters is slower because each epoch restarts from the visible variables of the previous epoch, instead of resetting to the initial data. For this reason, PCD-10 works in the equilibrium regime. The equilibrium regime is shown when the function $\mathcal{E}^{(AAI)}$ after falling to a minimum value, stabilizes at that value and does not rise again as the number of samplings increases, which happens in PCD-10 (Fig. 4.5).

Again a few RBMs behave differently from all the others. In fact, there are curves that do not

descend and behave at the equilibrium regime. Consequently, the average of at least 10 RBMs is also considered to be of significant statistical weight here. A graph of the behavior just described can be seen in the Fig.4.5 on the left.

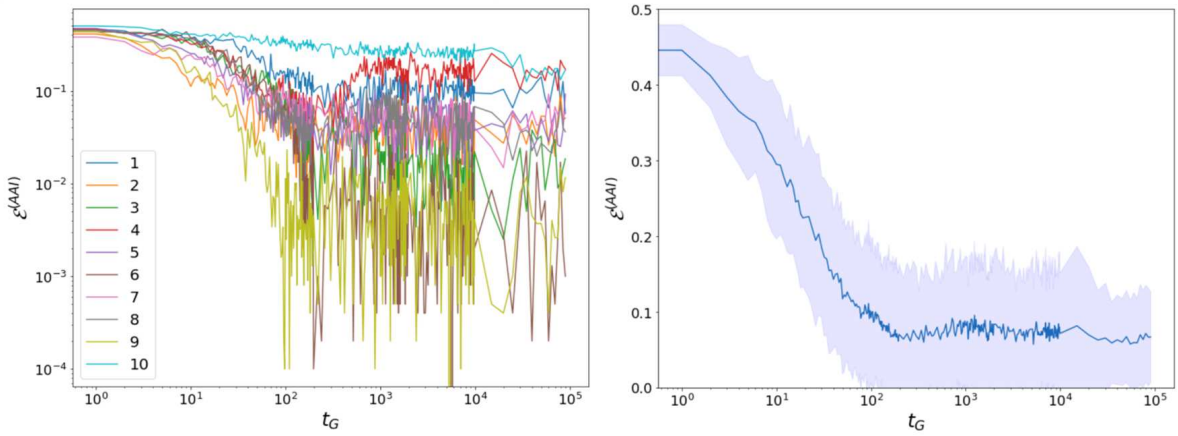

Figure 4.5: The image on the **left** $\mathcal{E}^{(AAI)}$ vs t_G in logarithmic scale for 10 RBMs trained with PCD-10. Curve number 10 (see the legend) deviates heavily from the behaviour of all the others. The average of these ten curves and the standard deviation indicating their variability are shown on the **right** with $\mathcal{E}^{(AAI)}$ in linear scale. The regime is clearly one of equilibrium. Note that in these graphs to save computation time, for $t_G > 10^4$, points were taken every 5000 samplings, i.e. $\Delta t_G = 5000$.

In Fig.4.5 on the right the variability (standard deviation) of the RBM is similar to the Rdm-10 case, but the quality between the 100 samples generated at nearby t_G can vary a lot for the single RBM, and this is well described by the fluctuations in the graph. This means that in this case, the PCD-10 algorithm, despite the different number of epochs (N_{ep}) and hidden units (N_h), is probably less stable than the Rdm-10 algorithm.

In Fig.4.6 one can see some samples generated by the RBM trained with the PCD-10 algorithm at various t_G. Sample generation starts from random noise and the equilibrium regime leaves the samples unchanged after $\Delta t_G = 9 \cdot 10^4$ samplings, starting from $t_G = 10^4$.

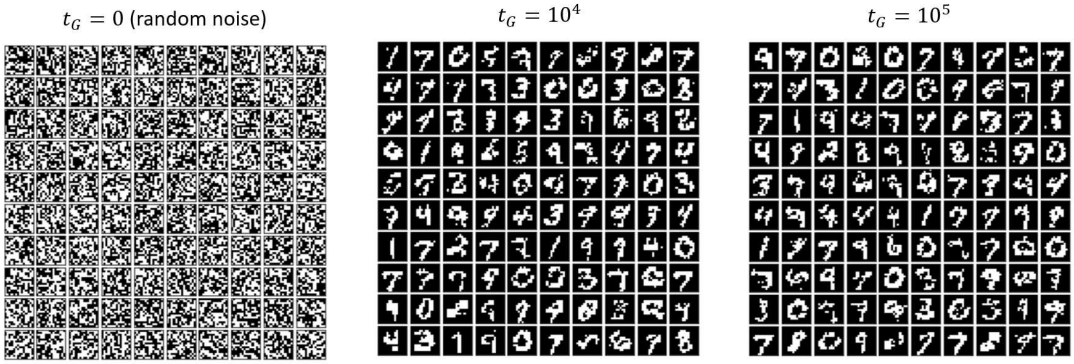

Figure 4.6: example of 100 samples generated from random noise ($t_G = 0$). The RBM is trained for 100 epochs with PCD-10. As can be seen in Fig.4.5, the equilibrium configuration is reached already for $t_g = 100$, but here it is shown that between $t_G = 10^4$ and $t_G = 10^5$ there is no substantial difference between the generated samples, because they are at equilibrium regime.

In the following sections, the PCD-n algorithm is set aside to discuss exclusively the Rdm-n case on which studies can be done on mixtures of algorithms and regime switching. In addition, the PCD-n case is much more computationally expensive if a large study is attempted, because $N_{ep} > 100$ epochs in the single machine for obtaining reasonable results.

4.5 Gibbs Sampling n-variable in training

Once it has been decided to take the $\mathcal{E}^{(AAI)}$ results as an average, the next step is to propose variable training, i.e. within the Gibbs Sampling Block in the training procedure, the number of iterations is changed during the same training. In practice Rdm-m/Rdm-n mixtures will be taken. The first part of the training starts by training the machine for $N_{ep,m}$ (number of epochs referred to the first Rdm-m algorithm) and the second part continues by training the machine for $N_{ep,n}$ (number of epochs referred to the second Rdm-n algorithm). To guarantee that the variation depends only on m and n of the chosen algorithm, all parameters such as hidden units, learning rate, type of SDG algorithm are kept constant. In addition, the total number of epochs $N_{ep} = N_{ep,m} + N_{ep,n}$ is also held constant, while only the values of $N_{ep,m}$ and $N_{ep,n}$ vary. Performing this kind of training on various Rdm-m/Rdm-n mixtures, several interesting results can be found.

The mixes of Rdm-n mainly considered are:

- Rdm-10 mixed with Rdm-5
- Rdm-10 mixed with Rdm-20
- Rdm-10 mixed with Rdm-50
- Rdm-10 mixed with Rdm-100

4.5.1 Minimum shift

In this thesis the mini-batch size is $M = 50$ and there are $N = 10000$ samples from the MNIST dataset, this means that for each epoch there are $t_{up} = \frac{N}{M} = 200$ parameters updates.

An interesting result is found by looking at the mixtures between Rdm-10 and Rdm-5, 20, 50, 100 separately for a number of epochs $N_{ep} = 30$, precisely by training the machine for $N_{ep,10} = 29$ with Rdm-10 and for a number of epochs $N_{ep,n} = 1$ for Rdm-n, where n can be equal to 5, 20, 50, 100.

Analyzing these RBMs by averaging for 20 of them, it turns out that it only takes a single epoch, which is equivalent to 200 parameter updates (time for the SDG procedure to go through the entire MNIST dataset considered), to shift the minimum in the n of the last Rdm-n that it trains. This result is well described in Fig.4.7, where it is evident that the minimum is in the position of the second Rdm-n trained.

Note that this effect only occurs for an already trained machine for 29 epochs with Rdm-m (in this case Rdm-10). In the case of starting with random parameters, the number of epochs to

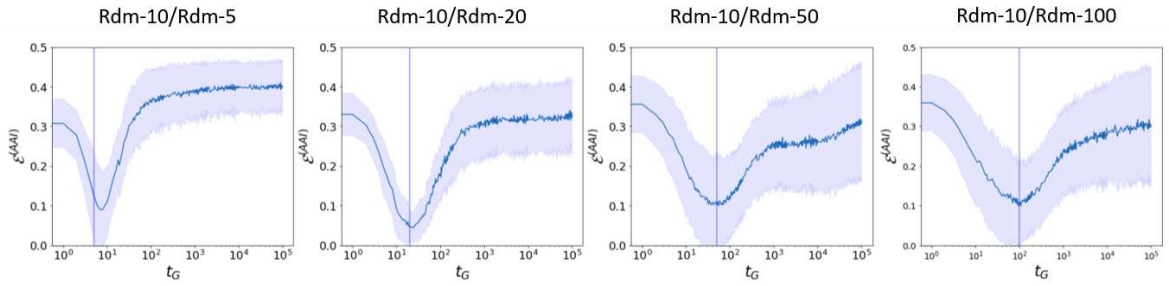

Figure 4.7: average $\mathcal{E}^{(AAI)}$ vs t_G (sampling time) of samples generated from random noise of 20 different RBMs. The mixes Rdm-10/Rdm-5, Rdm-10/Rdm-20, Rdm-10/Rdm-50, Rdm-10/Rdm-100 were used for each of the graphs, where in the first 29 epochs Rdm-10 was used and in the last epoch the other $(29 + 1)$. The vertical line indicates where one would typically expect the minimum using the second Rdm-n. These graphs demonstrate the shift of the minimum in a single epoch of a trained RBM.

generate a minimum of epochs in the average graph of $(t_G, \mathcal{E}^{(AAI)})$ is about 15.

In this type of training, it can be inferred from Fig.4.8 that by putting all the curves in the same graph, there is no real trend followed as far as the height of the minima is concerned, while the final plateau seems to fall as the n in the second Rbm-n increases. Moreover, since the minimum to be reached is at a lower t_G, the time of descent in the minimum is faster for lower n in the second Rdm-n.

The height of the minimum, in this case, is higher than in the training of a single Rdm-10 at 30 epochs because the second Rdm-n needs more epochs ($N_{ep,n}$) to learn the images of the MNIST dataset numbers, despite generating the best samples for $t_G \simeq n$.

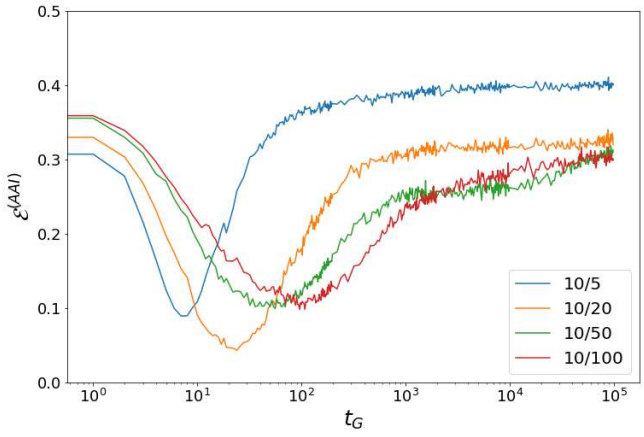

Figure 4.8: average $\mathcal{E}^{(AAI)}$ vs t_G (sampling time) of samples generated from random noise of 20 different RBMs. The mixes Rdm-10/Rdm-5, Rdm-10/Rdm-20, Rdm-10/Rdm-50, Rdm-10/Rdm-100 were used in the same graphs, where in the first 29 epochs Rdm-10 was used and in the last epoch the other $(29 + 1)$. The legend describes the type of Rdm-n mix divided into 29 epochs in the first and 1 epoch in the second. For example 10/5 indicates the training of the RBM machine with Rdm-10 for the first 29 epochs and Rdm-5 for the last epoch.

4.5.2 OOE vs Pseudo-Equilibrium Regime

In this subsection, the types and regime changes of the various mixtures are studied. Considering a generic Rdm-10/Rdm-n mixture, a study is made of the function $\mathcal{E}^{(AAI)}$ based on the distribution of epochs in which the two algorithms are run. First, intervals are chosen keeping $N_{ep} = 30$, then, to further study, these epochs are increased to $N_{ep} = 150$, and alternating and non-alternating algorithm is done to study the regime in which an average RBM enters.

This subsection shows the transition of the out of equilibrium state, to the pseudo-equilibrium state achieved by increasing the value of n in the Rdm-10/Rdm-n mixture, which corresponds on average to an increase in the value of one n of a pure Rdm-n.

Rdm-10/Rdm-5

The first step is to study the Rdm-10/Rdm-5 mixture at $N_{ep} = 30$ total epochs, but with $N_{ep,10}$ and $N_{ep,5}$ taken at different intervals described in Table 4.1.

Number of test	$N_{ep,10}$	$N_{ep,x}$
1	29	1
2	28	2
3	25	5
4	15	15
5	5	25
6	2	28
7	0	30

Table 4.1: test divisions by number of epochs of the Rdm-10/Rdm-x mixture. Note that test number 7 in the table is actually not a mixture, but a test in which the second Rdm-n is used for all the epochs in the training procedure.

It can be seen from Fig. 4.9 that the regime in which these RBMs operate is certainly out of equilibrium (OOE). In fact, an RBM works in the OOE regime when a minimum is present in the graph of $\mathcal{E}^{(AAI)}$ vs t_G for $t_G \simeq n$, where n is the index of the Rdm-n algorithm used for training. In addition, for sampling $t_G > n$ there is an ascent that stabilizes in a plateau.

The height and position of the minimum trained in this way is not so different when $N_{ep,5}$ varies. However, the height of the plateau for $N_{ep,5} \geq 25$ increases as the number of samples t_G increases, but still stabilizes at the value of the others. This different trend could be due to the low n number of the second Rdm-n algorithm, in fact, with the other mixtures this behavior does not occur. However, the value at $t_G = 10^5$ is very similar for all epoch intervals. Knowing that the axis of t_G is in logarithmic scale, it can be seen that for $t_G \to 0$, i.e. when starting from random noise without having yet sampled anything, a certain value in $\mathcal{E}^{(AAI)}$ is obtained. In the case of the Rdm-10/Rdm-5 mixture, the value of $\mathcal{E}^{(AAI)}$ of the random noise has better quality than the last generated samples and this even more strongly indicates the exit from an equilibrium regime.

The indicator $\mathcal{E}^{(AAI)}$ takes values in the range $[0, 0.5]$, but for random noise images it does not

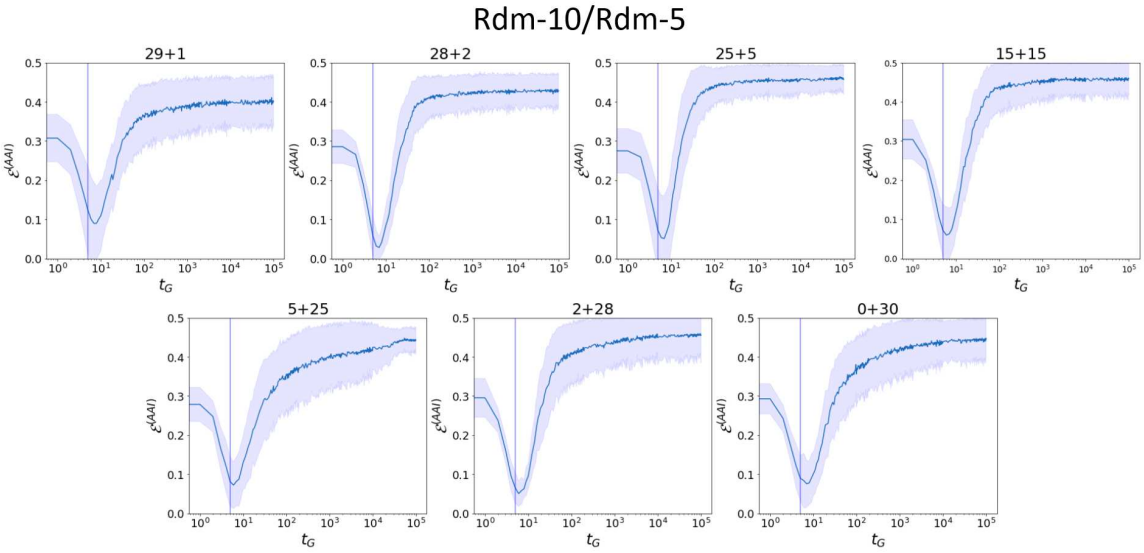

Figure 4.9: average $\mathcal{E}^{(AAI)}$ vs t_G (sampling time) of samples generated from random noise of 20 different RBMs. The mix is Rdm-10/Rdm-5, where the numbers above each graph indicate the combination of epochs also shown in table 4.1 The vertical line indicates where one would typically expect the minimum using the second Rdm-5 of the mix of two. The blue band indicates the variability of RBM and was calculated as the standard deviation on the individual RBM.

reach the maximum value, because in the calculation of $\mathcal{A}_T = \frac{1}{N_s}\sum_m^{N_s} \mathbf{1}(d_{TT}(m) < d_{TS}(m))$, for some m, the computation of the Euclidean distance between a random noise image and all other noise $d_{TT}(m)$, is less than that computed between the whole dataset and a random noise image $d_{TS}(m)$. In the Rdm-10/Rdm-5 case, the value of $\mathcal{E}^{(AAI)}$ in the palteau exceeds the value of the random noise, and is probably due to a decorrelation of the sampled images greater than random noise. The $\mathcal{E}^{(AAI)}$ indicator may exceed the level of random noise even in the overfitting cases.

The standard deviation indicating the variability, for the last two cases ($N_{ep,5} \geq 28$) there is less uncertainty about the height of the minimum than the height of the plateau.

So by observing such configurations in which the height of the minimum is acceptable already for $N_{ep,5} \geq 2$, it can be said that the OOE regime, in which there is a minimum for $t_G \simeq n$ of the second Rdm-n, is reached.

Rdm-10/Rdm-20

The next step is to study the Rdm-10/Rdm-20 mixture at $N_{ep} = 30$ total epochs, with $N_{ep,10}$ and $N_{ep,20}$ taken at different intervals (Table. 4.1).

The graphs for epoch intervals are described in Fig.4.10. Again, the regime is OOE with this time a plateau for all $N_{ep,20}$. However, the plateau height in the $\mathcal{E}^{(AAI)}$ function is lower than that in the random noise, indicating a higher quality of the out of equilibrium samples. This behavior can give an important clue to the start of a regime change. Moreover, such a clue occurs just as the value of n of the second mixture is increased, and, remembering that one switch to the equilibrium regime, generally, for $n > t_\alpha$, where t_α is mixing time, the lowering

of the plateau at the increase of n could confirm that the value of n begins to be similar of the mixing time.

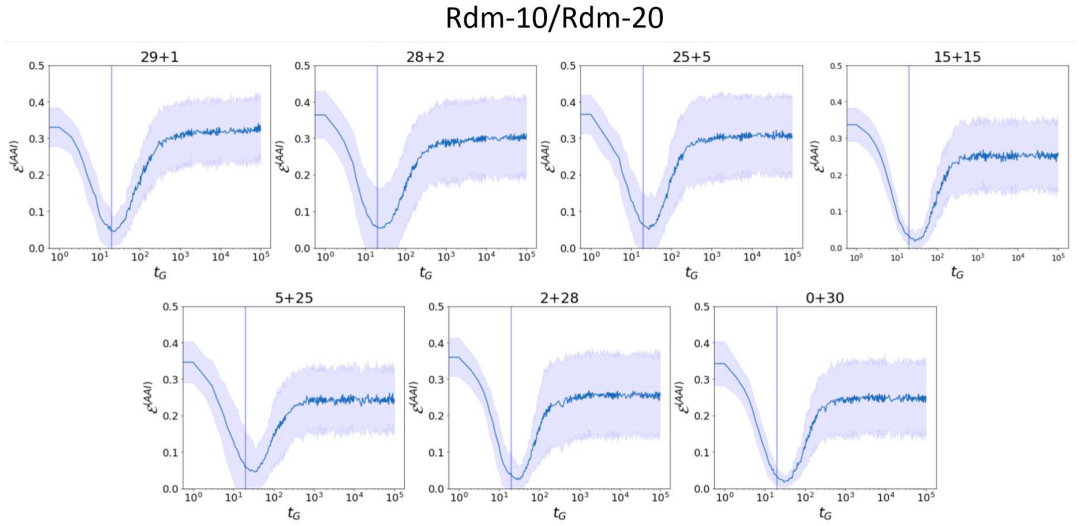

Figure 4.10: average $\mathcal{E}^{(AAI)}$ vs t_G (sampling time) of samples generated from random noise of 20 different RBMs. The mix is Rdm-10/Rdm-20, where the numbers above each graph indicate the combination of epochs also shown in table 4.1 The vertical line indicates where one would typically expect the minimum using the second Rdm-20 of the mix of two. The blue band indicates the variability of RBM and was calculated as the standard deviation on the individual RBM.

In this case, the standard deviation in plateau calculated over 20 RBMs is larger than in the Rdm-10/Rdm-5 case and this indicates greater variability when entering the OOE regime. This variability shows that the quality of the generated samples is less stable after passing the minimum at $t_G \simeq 20$.

For $N_{ep,20} \geq 15$ the height of the minimum is lower then the case for $N_{ep,20} < 15$ denoting a better average agreement between the generated samples and those of the MNIST dataset. Furthermore, for $N_{ep,20} \geq 28$, the standard deviation in the minimum is very low, leaving no doubts about the overall quality of the samples generated by this mixture.

Rdm-10/Rdm-50

The Rdm-10/Rdm-50 mixture at $N_{ep} = 30$ total epochs is then studied, with $N_{ep,10}$ and $N_{ep,50}$ taken at different intervals (Table. 4.1).

The first thing that can be seen by looking at the graphs in Fig.4.11 is that in this case as $N_{ep,50}$ increases, the plateau begins to lower, stabilizing at a certain value for $N_{ep,50} \geq 15$. This behavior indicates that the RBM is somewhere between an OOE and an equilibrium regime, because the plateau stabilizes at a rather low level of generated sample quality, although not as low as the minimum at $t_G = 50$. For convenience, such a regime is named pseudo-equilibrium. Next, it is shown how in the pseudo-equilibrium regime, the numbers generated after the minimum have slightly different behavior from those in the OOE regime, but also different from the equilibrium regime seen in the PCD-10 case.

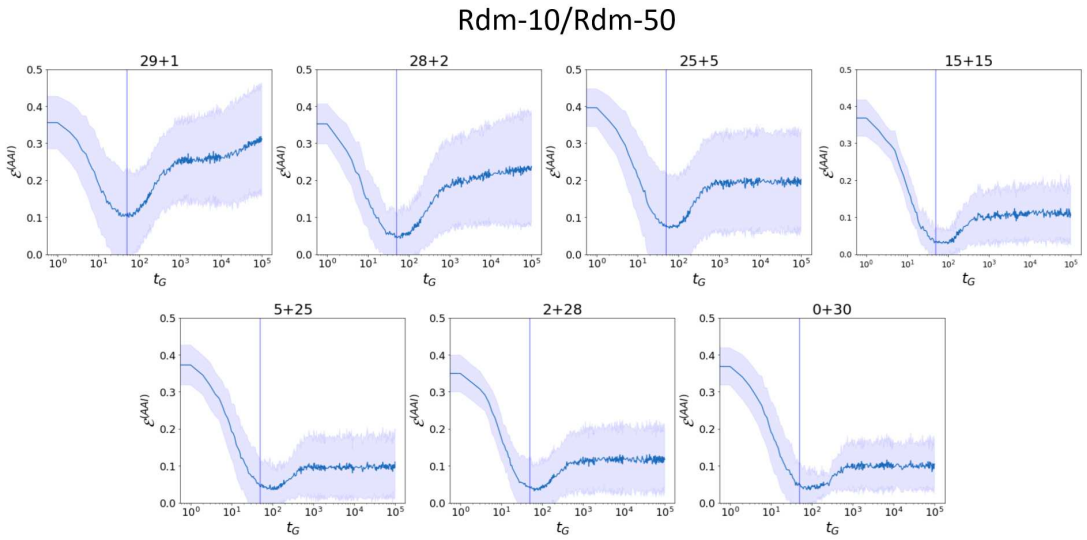

Figure 4.11: average $\mathcal{E}^{(AAI)}$ vs t_G (sampling time) of samples generated from random noise of 20 different RBMs. The mix is Rdm-10/Rdm-50, where the numbers above each graph indicate the combination of epochs also shown in table 4.1 The vertical line indicates where one would typically expect the minimum using the second Rdm-50 of the mix of two. The blue band indicates the variability of RBM and was calculated as the standard deviation on the individual RBM.

As $N_{ep,50}$ increases, the plateau decreases until it reaches a stable value for $N_{ep,50} \geq 15$. The variability (standard deviation) in the minimum is higher than the case of Rdm-10/Rdm-20, suggesting that there is worse stability of training in the minimum. While, the variability of the plateau, for $N_{ep,50} \geq 15$, is lower than in all the graphs of the Rdm-10/Rdm-20 case and this indicates a better training stability in the minimum.

Once again by increasing the average n of the mixture, it is easy to see that the plateau reaches even lower values than the Rdm-10/Rdm-20 case denoting the fact that value of n begins to be similar of the mixing time (t_α) and moves further and further away from the out of equilibrium regime. Therefore, we still increase the value of n to continue the study of the Rdm-10/Rdm-100 mixture.

Rdm-10/Rdm-100

Studying the Rdm-10/Rdm-100 mixture with $N_{ep} = 30$ total epochs, but with $N_{ep,10}$ and $N_{ep,100}$ taken at different intervals, one can see as increasing of $N_{ep,100}$ the change from the OOE regime to the pseudo-equilibrium regime as for Rdm-10/Rdm-50. This change is indicated by the disappearance of the minimum and the appearance of a plateau that rises even less than the Rdm-10/Rdm-50 case. The intervals taken of $N_{ep,10}$ and $N_{ep,100}$ are summarised in Table 4.1 This described trend can be seen in Fig. 4.12 where in this case 100 RBMs were trained for each curve.

This effect is because as the value of n increases in the algorithm change from 10 to 100, it goes from $\bar{n} < t_\alpha$ (OOE regime) to $\bar{n} \simeq t_\alpha$ (pseudo-equilibrium regime), where $\bar{n}$ is calculated as $\bar{n} = \frac{mN_{ep,m}+nN_{ep,n}}{N_{ep}}$ in a Rdm-m/Rdm-n mixture. Since in the mixtures $\bar{n}$ is a weighted average

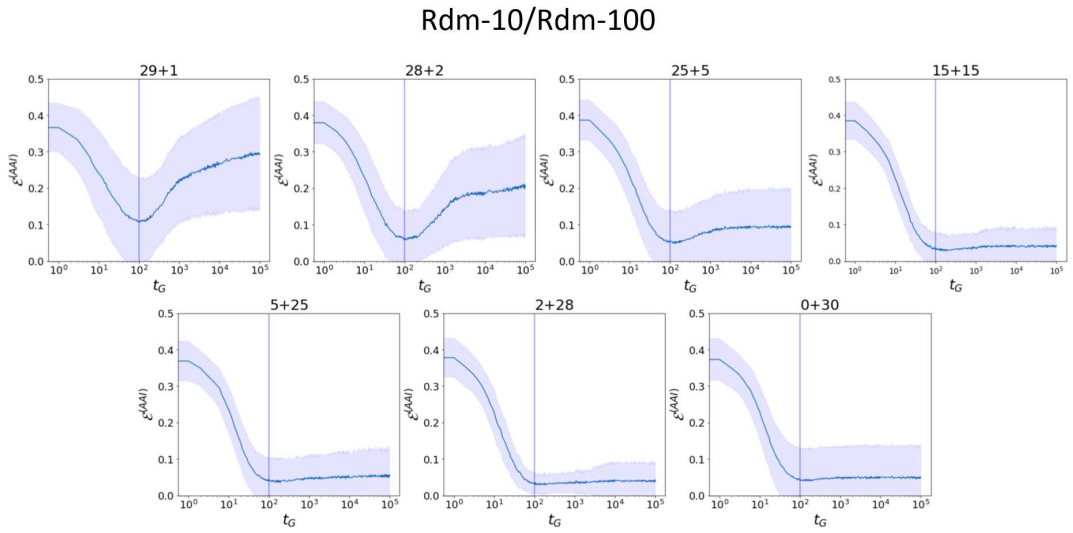

Figure 4.12: average $\mathcal{E}^{(AAI)}$ vs t_G (sampling time) of samples generated from random noise of 100 different RBMs. The mix is Rdm-10/Rdm-100, where the numbers above each graph indicate the combination of epochs also shown in table 4.1 The vertical line indicates where one would typically expect the minimum using the second Rdm-100 of the mix of two. The blue band indicates the variability of RBM and was calculated as the standard deviation on the individual RBM.

between the two Rdm-m and Rdm-n values, it takes about 15 epochs for changing regime to happen. Actually, it is not clear whether the value of $\bar{n}$ is the only one that affects the regime type of an RBM. In fact, in the section 4.6, we can see that the choice of the number of hidden units also affects the type of regime.

On the other hand, for $N_{ep,100} \geq 15$, i.e. for the regime at pseudo-equilibrium, the heights of the minima are roughly interchangeable and it can be seen that the standard deviations (variability of single RBM) are generally smaller then the case for $N_{ep,100} < 15$.

Once the overall graph has been constructed (Fig. 4.13), on average $N_{ep,100} \geq 15$, the height of the minimum remains quite high. Moreover, the height of the plateau decreases as $N_{ep,100}$ increases (mainly for $N_{ep,100} < 15$), indicating pseudo-equilibrium regime. Looking at the height of the plateaus (Fig. 4.13), it can be seen more clearly on a logarithmic scale that the regime is not yet equilibrium as there is still a short rise denoting a pseudo-equilibrium regime.

To better describe the pseudo-equilibrium regime with generated samples, one can look at the images in Fig. 4.14 showing the samples generated at various sampling steps using the Rdm-100 algorithm for 30 epochs. Up to $t_G \sim n$ the sampling gives some variety on the generated digits, while for $t_G = 1000$, the frequency of appearance of the digits with the number "1" is higher than all the others, but it does not appear in all generated images as in the case of the Rdm-10 in section 4.4.1. Since the frequency of the digits "1" is maintained even for $t_G = 100000$, this regime is nemed pseudo-equilibrium. Summarizing, a pseudo-equilibrium regime does not guarantee an equal diversity of digits, but it does guarantee a certain distribution of digit frequency over time, provided that in 100 sample generated, all digits are present. The absence of more than one digit is not accepted.

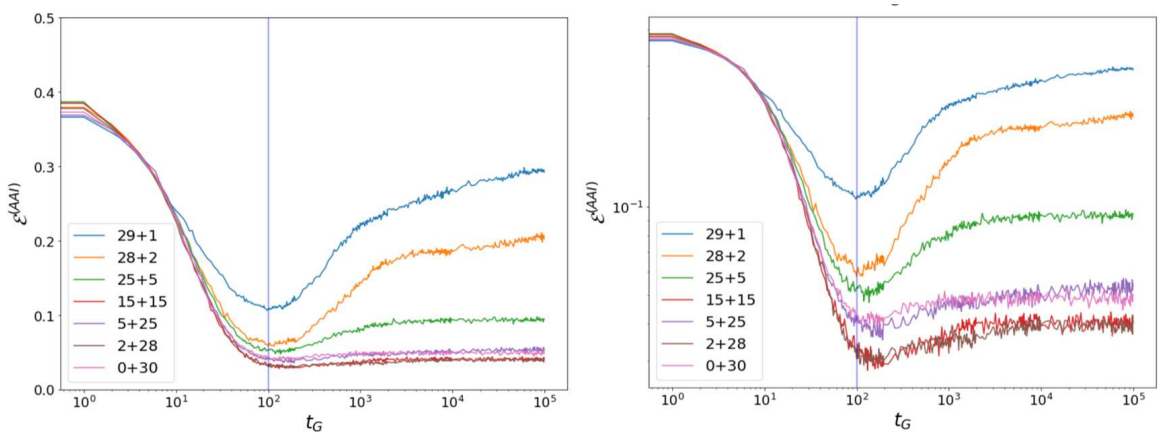

Figure 4.13: This image shows the average $\mathcal{E}^{(AAI)}$ vs t_G (sampling time) of samples generated from random noise of 100 different RBMs for each curve. The graph is shown on the left with a linear scale in $\mathcal{E}^{(AAI)}$, and on the right with a logarithmic scale. The mix is Rdm-10/Rdm-100, where the numbers in the legend indicate the combination of epochs also shown in table 4.1. The vertical line indicates where one would typically expect the minimum using the second Rdm-100 of the mix of two.

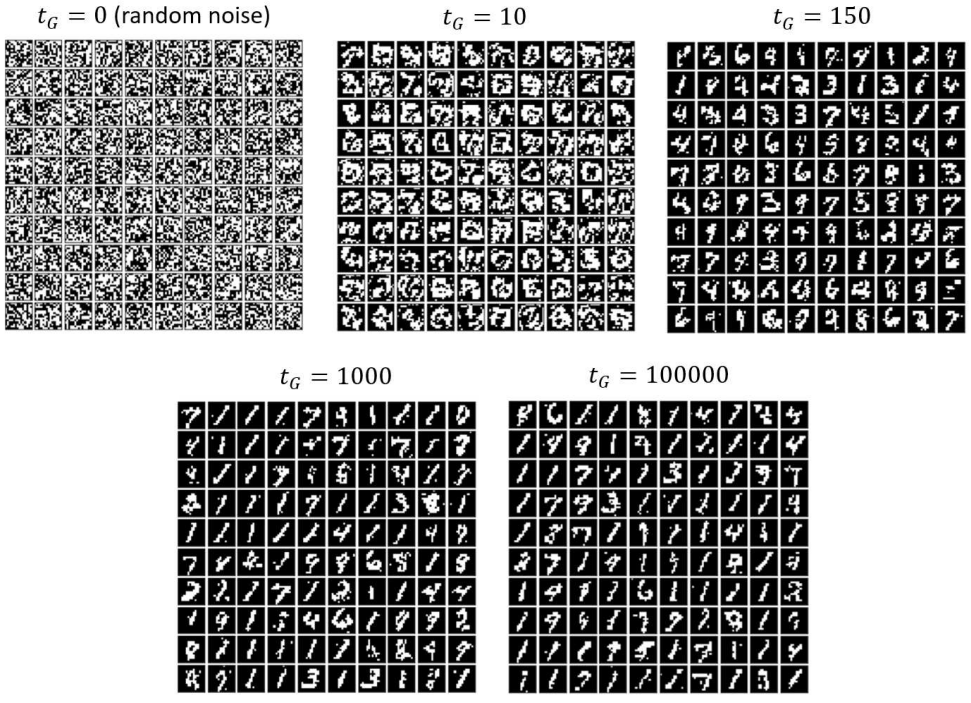

Figure 4.14: This image is an example of 100 samples generated strating from random noise ($t_G = 0$). The RBM is trained for 30 epochs with Rdm-100. As can be seen, the best configuration is reached for $n = 150$, very close to $n = 100$ of the Rdm-100 algorithm. It can be seen that already by $t_G = 1000$ the similar numbers increase, consequently the RBM is entering to the pseudo-equilibrium plateau which is already consolidated at $t_G = 100000$. Note that all digit types are present.

An important new feature introduced by the mixing method is about the quality of the generated samples. After a certain number of $N_{ep,100}$, the generated samples have the same quality as those trained using only Rdm-100 algorithm for $N_{ep} = 30$ epochs. Consequently, such mixtures could be used to accelerate the training time since training an RBM with Rdm-10 is much faster than training it with Rdm-100. Nevertheless, the regime in which RBM is found could change by not working with a pure n, but with a mixture of the two algorithms, so $\bar{n}$.

Observation on plateau

An interesting plot is in Fig.4.15, which represents $\mathcal{E}^{(AAI)}$ vs t_G when considering the average of 20 RBM with all Rdm-n algorithm used (without algorithm change during training), where a single value of n is kept constant for the 30 epochs. Apart from the position and depth of the minimum, which corresponds to $t_G \simeq n$, it can be seen very well that as the value of n increases, the transition from the OOE regime to the pseudo-equilibrium regime takes place as previously described.

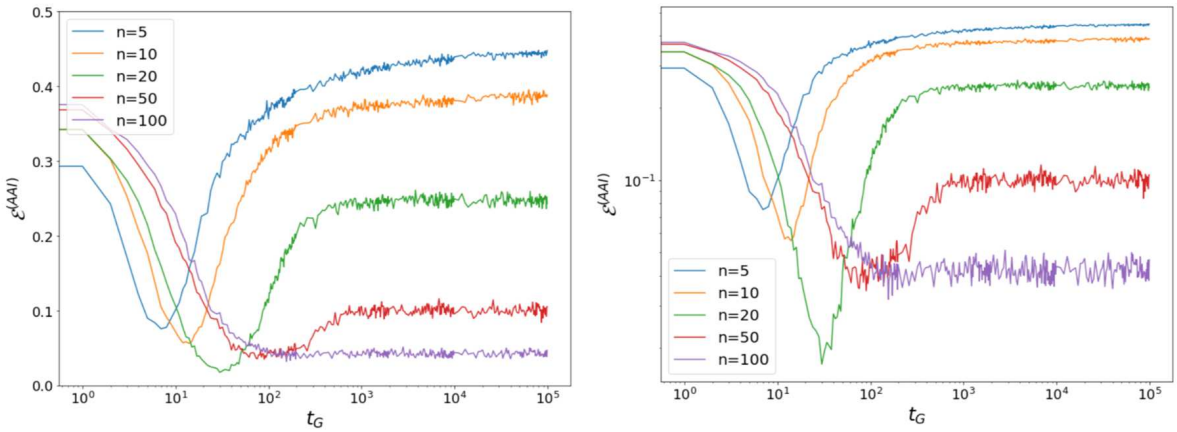

Figure 4.15: This image shows the average $\mathcal{E}^{(AAI)}$ vs t_G (sampling time) of samples generated from random noise of 20 different RBMs for each curve. The graph is shown on the left with a linear scale in $\mathcal{E}^{(AAI)}$, and on the right with a logarithmic scale. Each curve represents an Rdm-n with a different n described in the legend trained for 30 epochs.

Starting from the mixtures Rdm-10/Rdm-n with $N_{ep,n} \geq 5$ and observing the graphs in Fig. 4.16, we can see that the plateau decreases as n increases. On the other hand, if we observe the graphs for $N_{ep,n} < 5$, we can see that the plateaus between the mixtures Rdm-10/Rdm-50 and Rdm-10/Rdm-100 are approximately equal, while for the mixtures Rdm-10/Rdm-5 and Rdm-10/Rdm-20, the plateau heights decrease as the value of n increases.

Summary of all mixture graphs

Fig.4.17 shows all the individual graphs, each calculated on 20 RBM of the Rdm-10/Rdm-n mixtures (and 100 RBM for Rdm-10/Rdm-100) over $N_{ep} = 30$ total epochs with the divisions shown in Table4.1.

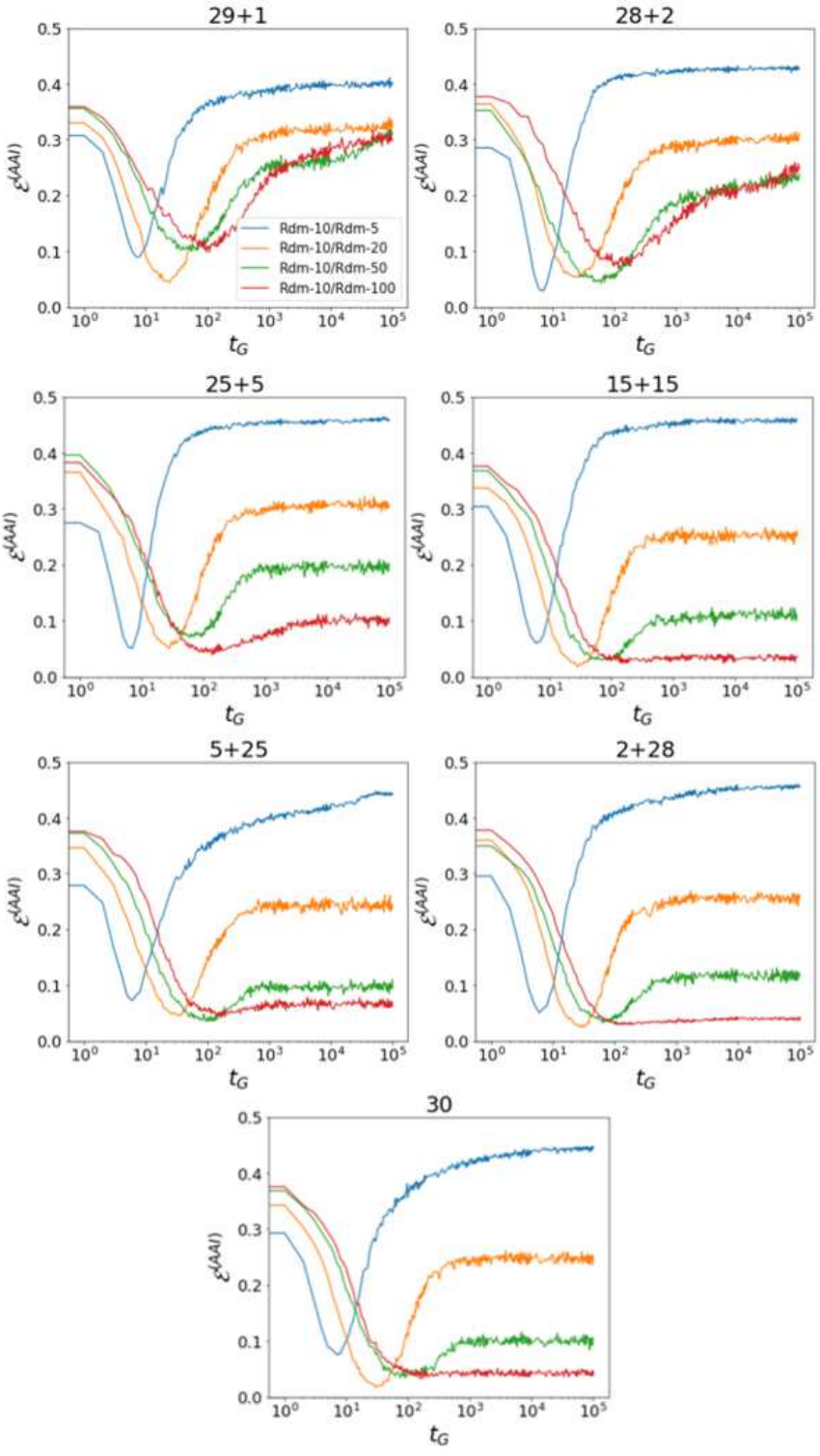

Figure 4.16: average $\mathcal{E}^{(AAI)}$ vs t_G (sampling time) of samples generated from random noise of 20 different RBMs. The mix is Rdm-10/Rdm-n, where the numbers above each graph indicate the combination of epochs also shown in table 4.1, where the first number is $N_{ep,10}$ and the second is $N_{ep,n}$. The figure legend is only shown on the first graph because it is the same for all of them and indicates the mixture of Rdm used to calculate the curve.

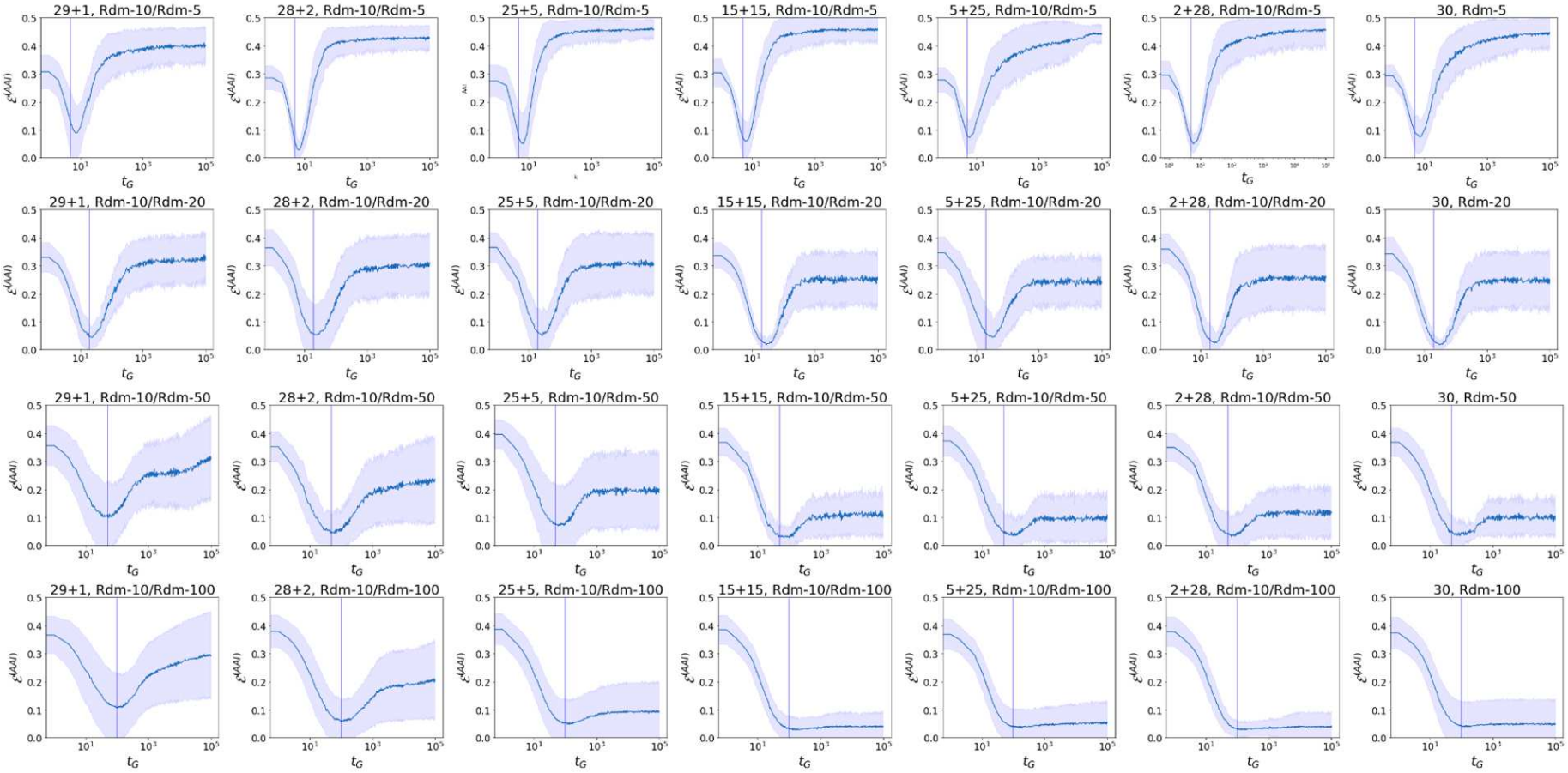

Figure 4.17: average $\mathcal{E}^{(AAI)}$ vs t_G (sampling time) of samples generated from random noise of 20 different RBMs for each curve. Only for the graphs with Rdm-10/Rdm-100 mixture and for pure Rdm-100, 100 RBM were used for each curve. The vertical line indicates where one would typically expect the minimum using the second Rdm of the mixture i.e. n of Rdm-n. The blue band indicates the variability of RBM and was calculated as the standard deviation on the individual RBM.

Increase of N_{ep} in Rdm-10/Rdm-100

We studied the Rdm-m/Rdm-n mixtures by trying to find a method to make the Adversarial Accuracy Indicator (AAI), as the number of samples being generated (t_G) increases, converge quickly and remain stable at equilibrium (like a PCD-10 that drops faster). Such behavior would provide stable goodness of the generated samples. It can be seen that a single epoch is enough to shift the minimum of the curve. However, the hope was that, by increasing the number of epochs and alternating the value of n, for example, every 15 epochs, the equilibrium regime would be quickly reached. In reality, this does not happen. Nevertheless, some interesting results are obtained on the transition from a regime out of equilibrium (OOE) to a regime at pseudo-equilibrium and vice versa.

Alternate training: we can see in Fig. 4.18 that as increasing the number of epochs, also shown in the literature [3], the mixing time (t_α) increases and if you keep the value of $\bar{n}$ balanced between 10 and 100 you switch back to the OOE regime after a certain number of epochs (in reality, this is also due to the low number of hidden units $N_h = 90$). This can be done for example by alternately sampling 15 epochs at $n = 10$ and 15 epochs at $n = 100$. Furthermore, it is important to note that as the number of epochs N_{ep} increases, the variability (blue band in the graphs in Fig.4.18) of the RBM in the minimum is significantly reduced, thus increasing the "accuracy" about the performance of the individual machine.

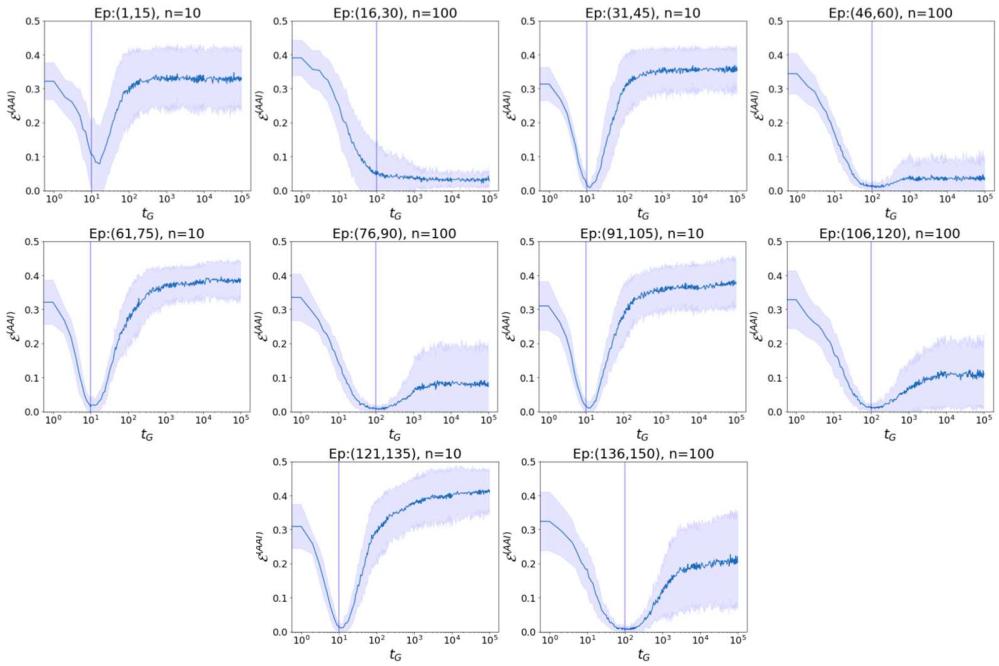

Figure 4.18: average $\mathcal{E}^{(AAI)}$ vs t_G (sampling time) of samples generated from random noise of 20 different RBMs for each curve. The mix is Rdm-10/Rdm-100, where the parenthesis above each figure indicates the range of epochs for which the n value of the Rdm-n algorithm was used. The vertical line indicates where one would typically expect the minimum using the Rdm-10 for training with $n = 10$ or using Rdm-100 for training with $n = 100$. The blue band indicates the variability of RBM and was calculated as the standard deviation on the individual RBM.

A confirmation of what has been described is in the graph in Fig. 4.19 that collects all the step curves at $n = 100$ of RBM trained by alternating 15 epochs at $n = 10$ with 15 epochs at $n = 100$ in all for 150 epochs. In fact, in that graph, the entry into the OOE regime is evident, observing that as the epochs increase, the plateau of the average of $\mathcal{E}^{(AAI)}$ begins to rise again, redefining the minimum sharply. Furthermore, it can be seen that from the epoch interval $(46, 60)$, the minimum of the function for $n = 100$ stabilizes and does not decrease any further, and this is because the number of epochs is not sufficient to increase the mixing time to get $t_\alpha > \bar{n}$. Therefore, after this interval of epochs, the RBMs enter the OOE regime.

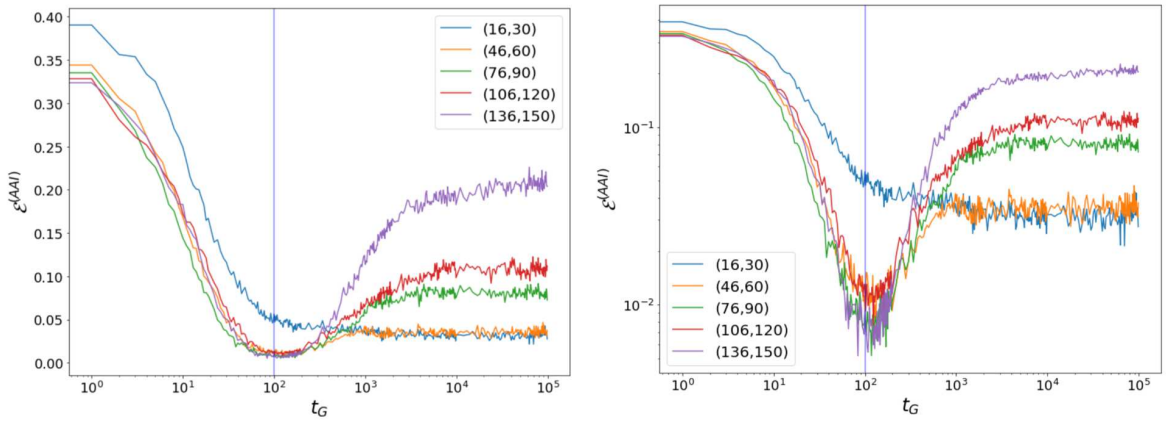

Figure 4.19: average $\mathcal{E}^{(AAI)}$ vs t_G (sampling time) of samples generated from random noise of 20 different RBMs for each curve. The graph is shown on the left with a linear scale in $\mathcal{E}^{(AAI)}$, and on the right with a logarithmic scale. The mix is Rdm-10/Rdm-100, where the legend indicates the range of epochs where $n = 100$. The vertical line indicates where one would typically expect the minimum using for Rdm-100.

An example of samples generated by an RBM trained in this way are shown in Fig. 4.20. For $t_G = 1000$, it can be seen that the regime is OOE, because "0" appears more and all the digits used in the training procedure do not appear.

Alternate training until $\mathbf{N_{ep}} = 45$: if we unbalance the value of $\bar{n}$ towards 100, i.e. by increasing the epochs in which the machine trains with Rdm-100, the mixing time does not exceed the value of $\bar{n}$ and therefore does not enter the OOE regime, or enters it only slightly (Fig. 4.21). To note this, it is sufficient to observe that the height of the plateau in logarithmic curves Fig. 4.22 does not rise to the starting level, but remains at a value approximately intermediate between the start and the minimum. This configuration is borderline between OOE and pseudo-equilibrium regime. In addition, it can be seen that in the epochs interval $(136, 150)$, in the logarithmic curve (Fig.4.22), the plateau is slightly higher than in all the others, and again this can be explained by the increase in mixing time (MT) as the number of epochs increases.

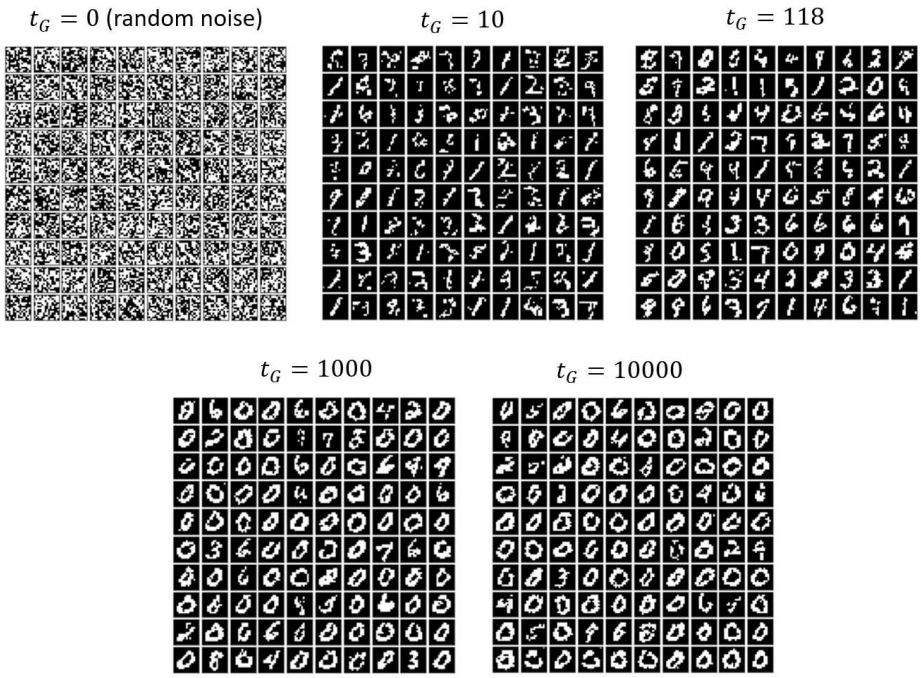

Figure 4.20: example of 100 samples generated starting from random noise ($t_G = 0$). The RBM is trained for 150 epochs with alternately steps of 15 epochs by Rdm-10 and Rdm-100. As can be seen, the best configuration is reached for $n = 118$, very close to $n = 100$ of the Rdm-100 algorithm. It can be seen that already by $t_G = 1000$ the similar numbers (digit "0") increase, consequently the RBM is entering in the OOE plateau which is consolidated also at $t_G = 10000$

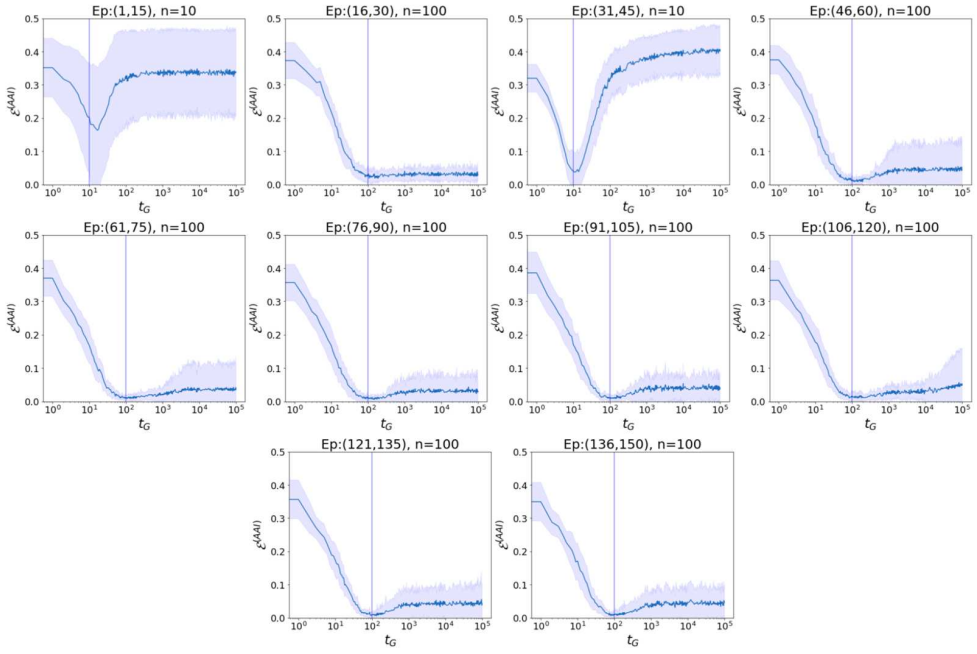

Figure 4.21: average $\mathcal{E}^{(AAI)}$ vs t_G (sampling time) of samples generated from random noise of 20 different RBMs for each curve. The mix is Rdm-10/Rdm-100, where the parenthesis above each figure indicates the range of epochs for which the n value of the Rdm-n algorithm was used. The vertical line indicates where one would typically expect the minimum using the Rdm-10 for training with $n = 10$ or using Rdm-100 for training with $n = 100$. The blue band indicates the variability of RBM and was calculated as the standard deviation on the individual RBM. Note that after 15 epochs the minimum at $t_G = 10$ is still too early to give reliable values.

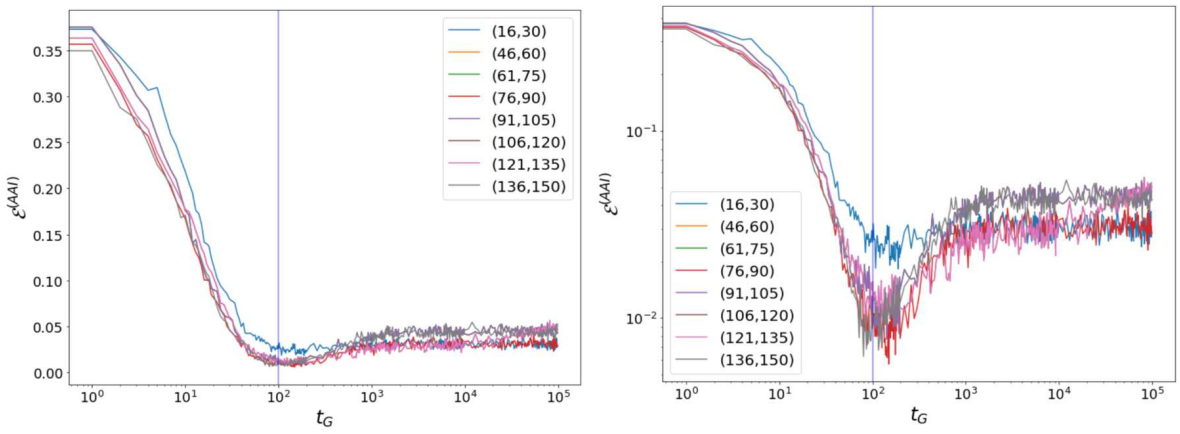

Figure 4.22: average $\mathcal{E}^{(AAI)}$ vs t_G (sampling time) of samples generated from random noise of 20 different RBMs for each curve. The graph is shown on the left with a linear scale in $\mathcal{E}^{(AAI)}$, and on the right with a logarithmic scale. The mix is Rdm-10/Rdm-100, where the legend indicates the range of epochs where $n = 100$. The vertical line indicates where one would typically expect the minimum using for Rdm-100.

All epochs with n = 100: if one were to keep $n = 100$ for all 150 epochs, the regime change does not take place or one can say that there is an even smaller hint of a minimum than the previous mixed part (Fig. 4.23).

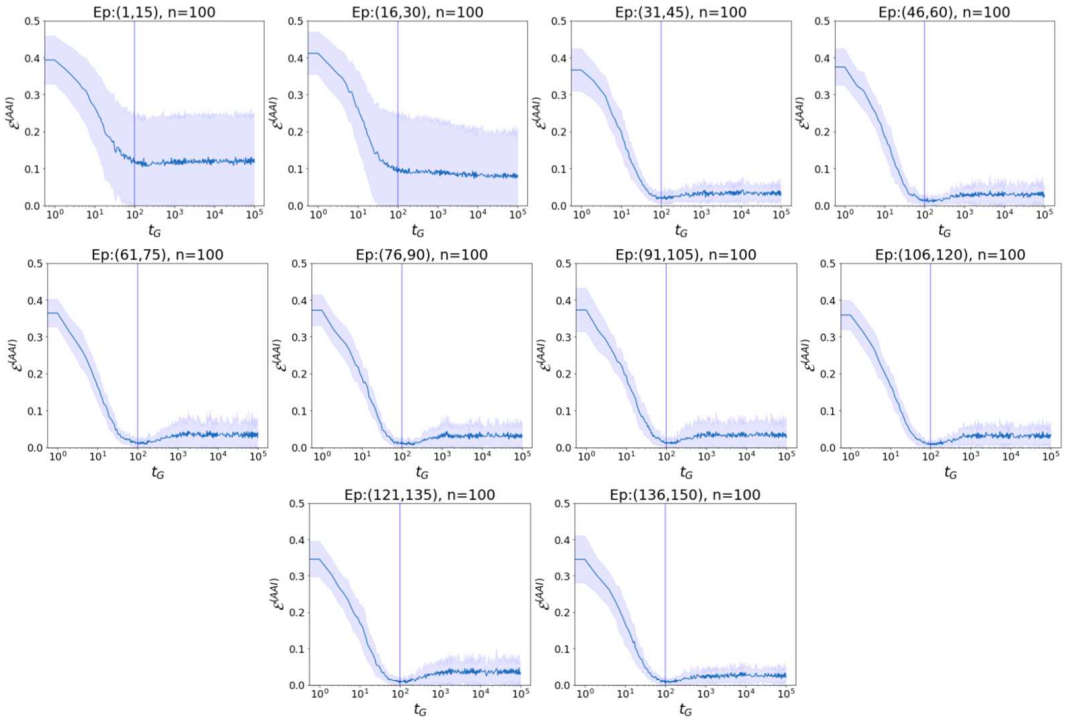

Figure 4.23: average $\mathcal{E}^{(AAI)}$ vs t_G (sampling time) of samples generated from random noise of 20 different RBMs for each curve. Rdm-100 was used, where the parenthesis above each figure indicates the range of epochs for which value of the $n = 100$ was used. The vertical line indicates where one would typically expect the minimum using the Rdm-100. The blue band indicates the variability of RBM and was calculated as the standard deviation on the individual RBM. Note that after 30 epochs the minimum at $t_G = 10$ is still too early to give reliable values, despite having done so on other occasions.

Looking at the minima as a whole (Fig.4.24), one can see that the last epoch has the best quality values and the best stabilization at equilibrium or plateau (lowest of all). The reason for this is the imbalance of the value of $\bar{n}$ towards 100.

The minimum for Rdm-100 at 150 epochs is hinted more then the Rdm-100 trained at 30 epochs (i.e., the plateau is higher). The reason why this happens is that as the number of epochs increases, the Mixing Time increases and consequently the value of n is supposed to be very similar to the Mixing Time, but still lower. Despite this, the regime is still considered to be pseudo-equilibrium and this can be seen by observing generated samples (Fig.4.25)

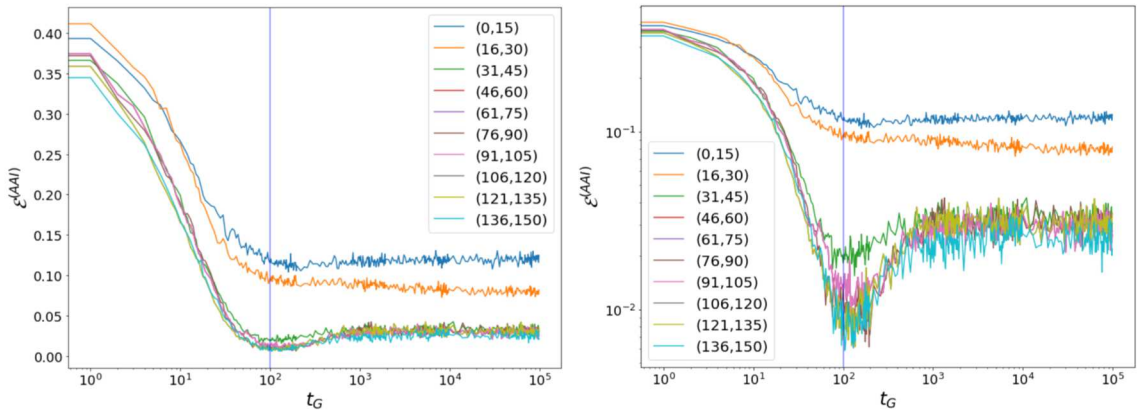

Figure 4.24: average $\mathcal{E}^{(AAI)}$ vs t_G (sampling time) of samples generated from random noise of 20 different RBMs for each curve. The graph is shown on the left with a linear scale in $\mathcal{E}^{(AAI)}$, and on the right with a logarithmic scale. The algorithm is Rdm-100, where the legend indicates the range of epochs where $n = 100$. The vertical line indicates where one would typically expect the minimum using for Rdm-100.

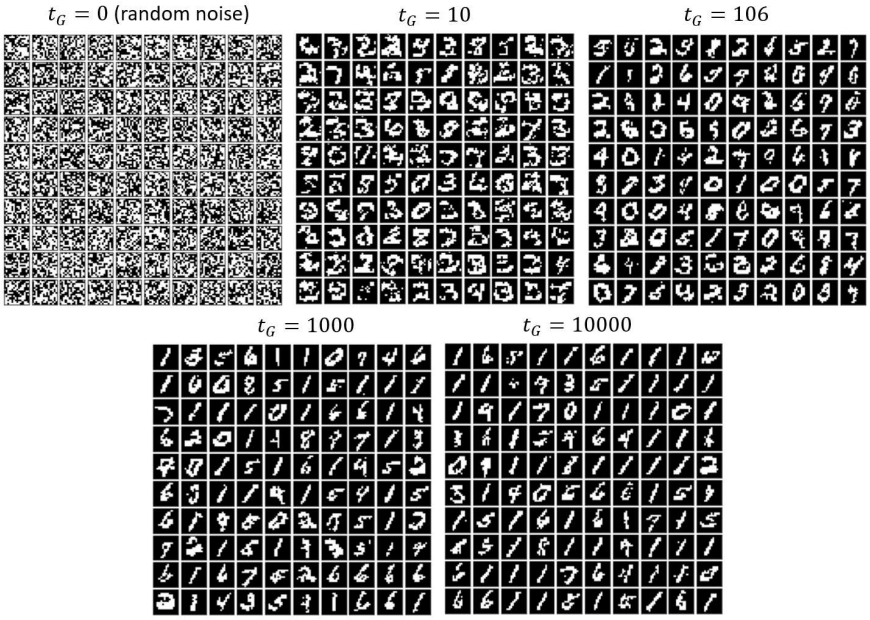

Figure 4.25: example of 100 samples generated starting from random noise. The RBM is trained for 150 epochs with Rdm-100. The best configuration is reached for $n = 106$, very close to $n = 100$ of the Rdm-100 algorithm. It can be seen that already by $t_G = 1000$ the similar numbers (digit "1") increase, but even for $t_G = 10000$ all digits are present, which confirms belonging in the pseudo-equilibrium regime.

Final consideration: Fig.4.26 shows the minima of the last epoch interval $(136, 150)$ for the three types of Rdm-10/Rdm-100 mixtures (to be precise, the last one is pure Rdm-100). It can be seen that the pseudo-equilibrium regime is better when $n = 100$ is used for all intervals (lower plateau then the alternate cases), because for the same total epochs N_{ep}, as the number of steps with $n = 100$ increases, average $\bar{n}$ gets closer to the mixing time (MT). It is not the purpose of this thesis to demonstrate this, but actual calculations of the mixing time (t_α) for each machine can shed light on this question.

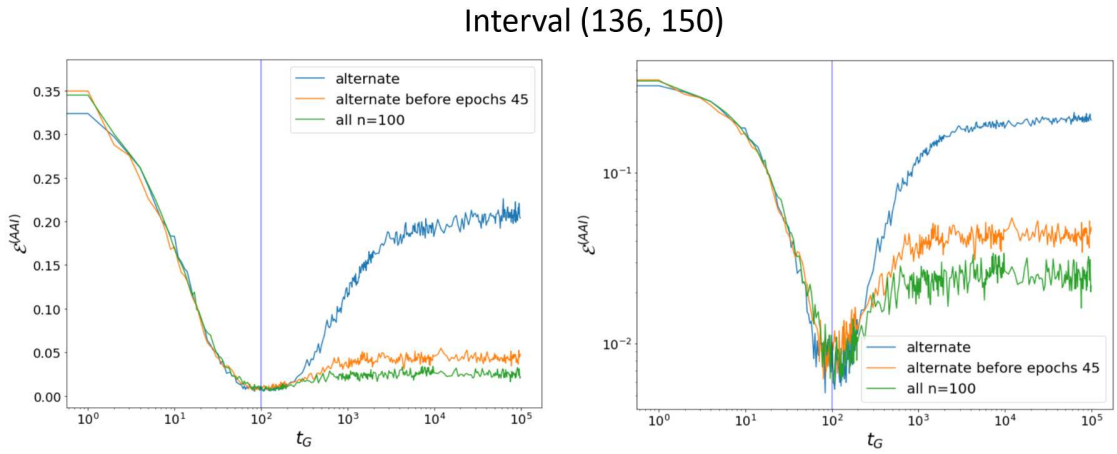

Figure 4.26: average $\mathcal{E}^{(AAI)}$ vs t_G (sampling time) of samples generated from random noise of 20 different RBMs for each curve taken in the range of epochs $(136, 150)$ of the three types of training seen in the previous figure. The graph is shown on the left with a linear scale in $\mathcal{E}^{(AAI)}$, and on the right with a logarithmic scale. The algorithm is mixture Rdm-10/Rdm-100, where the legend indicates the type of training. The vertical line indicates where one would typically expect the minimum using for Rdm-100.

The fact that our mixing time is lower then the literature [3] is because the number of hidden units considered in this thesis work is $N_h = 90$ and this was done to increase the speed of computation, whereas in the literature [3] a higher value ($N_h = 500$) was used. In addition, the algorithms used are also different, so the comparison stems solely from the fact that both are trained as RBM with the Rdm-n algorithm. In following section, the change of regime as the number of hidden units is discussed, which may partly justify the differences with what are found in the literature [3].

4.6 Different number of hidden unit

Another possible reason for the regime change can be deduced from the study of the graph of $\mathcal{E}^{(AAI)}$ vs t_G while the number of hidden units N_h changes, working with $N_{ep} = 30$ epochs over 20 RBM, maintaining the Rdm-100 algorithm for training (Fig.4.27).

What can be seen immediately is that $N_h = 90$, used for the calculations in the previous cases, is a specific number of hidden units leading to a pseudo-equilibrium regime. Indeed, with N_h higher or lower than 90, the RBM is in the OOE regime. Furthermore, it can be seen that for the number of epochs chosen ($N_{ep} = 30$), the number of hidden units $N_h = 90$ gives the best

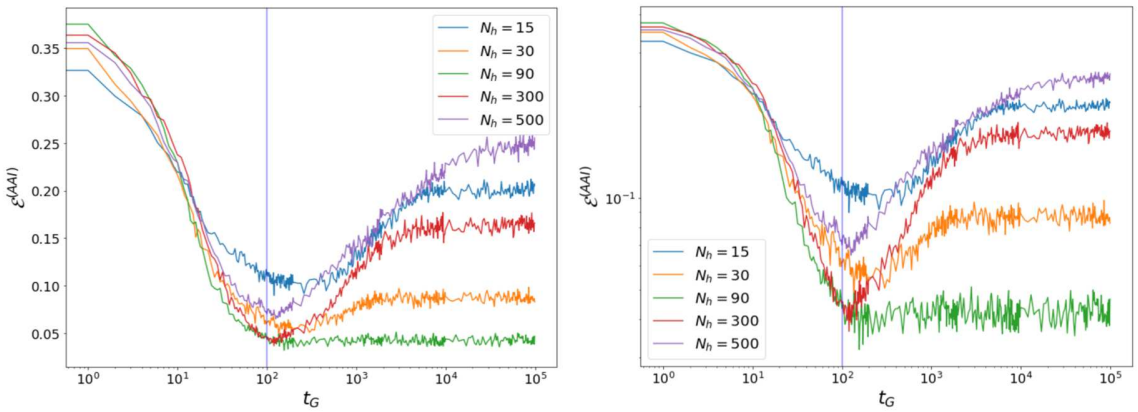

Figure 4.27: average $\mathcal{E}^{(AAI)}$ vs t_G (sampling time) of samples generated from random noise of 20 different RBMs for each curve. The graph is shown on the left with a linear scale in $\mathcal{E}^{(AAI)}$, and on the right with a logarithmic scale. These curves were calculated by training the machine with Rdm-100 at $N_{ep} = 30$ by varying the number of hidden units N_h. The vertical line indicates where one would typically expect the minimum using Rdm-100.

quality of the samples generated among those studied.

Another graph that can be observed in Fig. 4.28 is on the height of the last sampled point (and thus the final height of the plateau) as a function of the number of hidden units, i.e., $\mathcal{E}^{(AAI)}(t_G = 10^5)$ vs. N_h. The value of $N_h = 90$ chosen for this thesis work provides a kind of minimum in the height of the plateau. Since there is a high standard deviation indicating the variability of the RBM, this graph is only a hint for more in-depth studies that can be done by increasing the number of epochs, improving the performance of the algorithms, and thus reducing such variability.

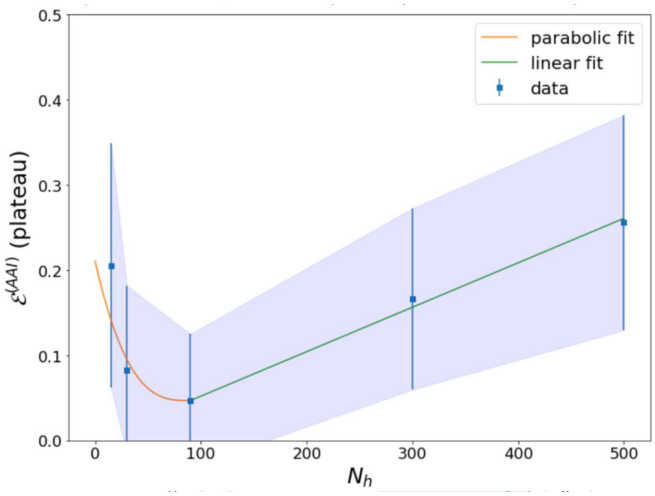

Figure 4.28: average $\mathcal{E}^{(AAI)}(t_G = 10^5)$ vs N_h of samples generated from random noise of 20 different RBMs for each point. These points were calculated by training the machine with Rdm-100 at $N_{ep} = 30$ by varying the number of hidden units N_h.The blue curve and error bars in the figure indicate the standard deviations on the individual RBM. Parabolic and linear fits were performed on the points to derive some trend from which to further the analysis. In any case, one can see the presence of a minimum in the plateau height for $N_h = 90$. However, the points are few, so a study with more points could put more clarity.

Finally, these graphs provide clues to the fact that the mixing time may be correlated with the number of hidden units, and this is due to the obvious regime change when only the number of hidden units N_h varies.

4.7 Python Codes

This section summarizes some key parts of the code used for this thesis.

Before starting to describe the algorithms used we show in the following code the library for the various calculations:

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import pyplot
```

4.7.1 Rdm-training

Activation Function

Before showing the code used for training, the calculation of the activation function, useful for the Gibbs sampling block, is shown:

```
def activate(v_in, wei, bias):
  act = np.dot(v_in, wei) + bias #scalar product + bias
  prob = 1. / (1. + np.exp(-act)) #calulation of sigmoid function
  n = np.shape(act)
  v_out = np.zeros(n)
  #activation of 1 if probability array prob is bigger then
  #an array of random samples of the same dimension
  v_out[np.random.random_sample(n) < prob] = 1
```

Training Algorithm

The following code describes how training is done in the case of the Rdm-10 algorithm:

```
#TRAINING Rdm-10
#N=10000 is the number of image taken of MNIST
#number of visibile and hidden units
L=14**2
M=90

#Initalization of parameter
sigma = np.sqrt(0.1 / float(L+M))
w = sigma * np.random.rand(L,M)
a = sigma * np.random.rand(L)
b = np.zeros(M)

#Initalization of variable
v = np.array(X) #X contain MNIST images
h = np.zeros((N,M))
```

```
vm = np.zeros((N,L))
hm = np.zeros((N,M))
vi = np.random.random_sample((N,L))
vi = (vi > 0.5) * 1.0

#learning rate
l_rate=0.008

#minibatch dimension
mini = 50
Nbatch=int(N/mini)

#number of epochs
max_epoch=30

#Initialization of variation of parameters
dw=0
da=0
db=0

#********Centring Trick Parameters************
v_data_mu=np.zeros(L)
mu=np.zeros(L)
lam=np.zeros(M)

for i in range(N):
  v_data_mu += v[i]
mu=v_data_mu/N

xi=0.01
mu_b=np.zeros((Nbatch, L))
lam_b=np.zeros((Nbatch, M))

#*********************************************
#Number of step training Rdm-n
Nrdm=10 #Nrdm is equal n of Rdm-n

#******************Start of the algorithm**********************
for epoch in range(1, 1+max_epoch):
  #Initialization of the first step of Gibbs Sampling
  for i in range(N):
    hm[i] = activate(vi[i]-mu, w, b)
    h[i] = activate(v[i]-mu, w, b)
  p=0
  for k in range(Nbatch):
    #Initialization of parameters for gradient
    v_data, h_data = np.zeros(L), np.zeros(M)
    v_model, h_model = np.zeros(L), np.zeros(M)
    vh_data, vh_model = np.zeros((L,M)), np.zeros((L,M))

    #Initialization of parameters for ADAM
```

```
    st_w = np.zeros((L,M))
    st_a = np.zeros(L)
    st_b = np.zeros(M)
    m_w = np.zeros((L,M))
    m_a = np.zeros(L)
    m_b = np.zeros(M)

    for l in range(mini):
      #Block Gibbs Sampling
      for i in range(Nrdm):
        #generation of fantasy variable vm and hm for the mini-batc
        vm[p]=activate(hm[p]-lam, w.T, a)
        hm[p] = activate(vm[p]-mu, w, b)
      p=p+1

    #calculation of the value for the gradient of LogLIkelihood (LL)
    for i in range(mini):
      v_data += v[i+k*mini]
      h_data += h[i+k*mini]
      v_model += vm[i+k*mini]
      h_model += hm[i+k*mini]
      vh_data += np.outer((v[i+k*mini].T-mu.T), (h[i+k*mini]-lam))
      vh_model += np.outer((vm[i+k*mini].T-mu.T), (hm[i+k*mini]-lam))

    #calculation of mu_batches and lambda_batches of k-th batches
    mu_b[k]=v_data/mini
    lam_b[k]=h_data/mini

    #update of parametrs as a function of new offset
    a=a+xi*np.dot(w,(lam_b[k].T-lam.T))
    b=b+xi*np.dot(w.T,(mu_b[k].T-mu.T))

    #mean of Centring Trick calculation
    mu=(1-xi)*mu+xi*mu_b[k]
    lam=(1-xi)*lam+xi*lam_b[k]

    #SGD calculation with ADAM
    b1=0.9
    b2=0.99
    epsilon=10**(-10)
    gt_w=(vh_data-vh_model)/mini
    gt_a=(v_data-v_model)/mini
    gt_b=(h_data-h_model)/mini
    m_w=b1*m_w+(1-b1)*gt_w
    m_a=b1*m_a+(1-b1)*gt_a
    m_b=b1*m_b+(1-b1)*gt_b
    st_w=b2*st_w+(1-b2)*gt_w*gt_w
    st_a=b2*st_a+(1-b2)*gt_a*gt_a
    st_b=b2*st_b+(1-b2)*gt_b*gt_b
    mt_w=m_w/(1-b1)
    mt_a=m_a/(1-b1)
```

```
    mt_b=m_b/(1-b1)
    stt_w=st_w/(1-b2)
    stt_a=st_a/(1-b2)
    stt_b=st_b/(1-b2)
    dw=l_rate*np.divide(mt_w,np.sqrt(stt_w + epsilon))
    da=l_rate*np.divide(mt_a,np.sqrt(stt_a + epsilon))
    db=l_rate*np.divide(mt_b,np.sqrt(stt_b + epsilon))

    #Update of parameters at the end of a mini-batch
    w=w+dw
    a=a+da
    b=b+db

  print('complete epoch:', epoch, '/', max_epoch)
  #suffling of variable of Dataset and initial variable
  np.random.shuffle(v)
  np.random.shuffle(vi)

#******************End of the algorithm**********************
```

4.7.2 Generating Sample

Calculation of $\mathcal{A}_T$ and $\mathcal{A}_S$

During sample generation, the function $\mathcal{E}^{(AAI)}$ is calculated, so we first define the functions useful for calculating this index:

```
Nsamp=100 #number of sample generated
#Function for calculating Euclidean Distance
def Euclide_m(v1, v2, Nsamp, ind, same):
  diff=np.zeros((Nsamp,L))
  dist=np.zeros(Nsamp)
  for i in range(Nsamp):
    diff[i]=(v1[ind]-v2[i])**2
  #the distance between equal indexes in the case of dss or dtt is excluded
  if same==1:
    diff[ind]=np.max(diff)
  dist=np.sqrt(np.sum(diff, 1))
  dist=np.min(dist)
  return dist

#Function for calculating As
def As(Nsamp, v, vi):
  sum=0
  dss=np.zeros(Nsamp)
  dst=np.zeros(Nsamp)
  for i in range(Nsamp):
    dss[i]=Euclide_m(v[:Nsamp], v[:Nsamp], Nsamp, i, 1)
    dst[i]=Euclide_m(v[:Nsamp], vi, Nsamp, i, 0)
  for m in range(Nsamp):
    if dss[m] < dst[m]:
```

```
      sum += 1
  As=sum/Nsamp
  return As

#Function for calculating At
def At(Nsamp, v, vi):
  sum=0
  dtt=np.zeros(Nsamp)
  dts=np.zeros(Nsamp)
  for i in range(Nsamp):
    dtt[i]=Euclide_m(vi, vi, Nsamp, i, 1)
    dts[i]=Euclide_m(vi, v[:Nsamp], Nsamp, i, 0)
  for m in range(Nsamp):
    if dtt[m] < dts[m]:
      sum += 1
  At=sum/Nsamp
  return At
```

Code for generating sample and calculation of $\mathcal{E}^{(AAI)}$

The following code describes how generating samples and calculation of $\mathcal{E}^{(AAI)}$ are done for every RBM:

```
Nsamp=100 #number of samples generated
Ncamp=10000 #number of Gibbs Samplings

#Generation of Nsamp random samples
vi = np.random.random_sample((100,Lpx*Lpx))
vi = (vi > 0.5) * 1.0

app=0 #supporting variable

#Initialization of EAA and x(or t_G)
EAA=np.zeros(int(Ncamp))
x=np.zeros(int(Ncamp))

#****************Start of Sampling algorithm**********************
for k_i in range(Ncamp):
  #Calulcation of E(AAI)
  if k_i < 20:
    EAA[app]=(As(Nsamp, v, vi)-0.5)**2 + (At(Nsamp, v, vi)-0.5)**2
    x[app]=k_i
    app += 1
  if k_i > 20 and k_i< 200 and k_i % 2 == 0:
    EAA[app]=(As(Nsamp, v, vi)-0.5)**2 + (At(Nsamp, v, vi)-0.5)**2
    x[app]=k_i
    app += 1
  if k_i > 200 and k_i< 2000 and k_i % 20 == 0:
    EAA[app]=(As(Nsamp, v, vi)-0.5)**2 + (At(Nsamp, v, vi)-0.5)**2
    x[app]=k_i
    app += 1
```

```
  if k_i > 2000 and k_i < 10000 and k_i % 100 == 0:
    EAA[app]=(As(Nsamp, v, vi)-0.5)**2 + (At(Nsamp, v, vi)-0.5)**2
    x[app]=k_i
    app += 1

  #Sampling Phase
  for k in range(Nsamp):
    hf = activate(vi[k]-mu, w, b)
    vi[k] = activate(hf-lam, w.T, a)
#****************End of Sampling algorithm**********************
```

Chapter 5

Conclusion

The main aims of this thesis were to use a simplified Restricted Boltzmann Machine programmed from scratch to replicate some of the results obtained in the literature [3] and to study some new behavior that had never been dealt with before, like the study of variable training, where the number of Gibbs samples changes within training.

As a first objective, the results were successfully replicated as in literature [3], but wanting to be precise, the graphs constructed always consider an average of at least 10 RBMs, because working with single RBMs rarely leads to the creation of RBMs that generate anomalous samples. However, it was noted that when training at 150 epochs of 20 RBMs using 100 Gibbs sampling, with training that starts with random variables (Rdm), this behavior was greatly attenuated. Despite this, considering a higher number of epochs for each RBM trained, would have led to a computation time too high for the number of RBMs considered and for the computing power available.

Training RBMs with variable Gibbs sampling is done by training the machine for a certain number of epochs ($N_{ep,m}$) with m Gibbs samplings and for the remaining epochs ($N_{ep,n}$) using n samplings. All this is done while keeping the total number of epochs (N_{ep}) constant. From the literature [3], it is known that in the out of equilibrium (OOE) regime, the minimum in the Adversarial Accuracy Indicator (AAI) plot, which indicates the goodness of the generated samples, as a function of the number of samplings of generation (t_G), is reached for t_G, approximately equal to the number of samplings used in the training.

The goal of studying the behavior of RBMs trained at variable Gibbs samplings was successfully achieved. First, we discovered the minimum shift in the AAI plot with respect to the generation sampling time. In fact, a single training epoch (the time for the training procedure to go through the entire data set once) is needed to shift the minimum of the AAI from that found by training the RBM at m samplings to that at n samplings. Consequently, if we make the training at intervals of epochs $N_{ep,m} = 29$ and $N_{ep,n} = 1$, the final minimum in AAI is at the generation sampling time equal to n. However, the minimum in the AAI has a high value and does not guarantee good quality samples. The second thing we have done is the study of the transition

from the OOE regime to the pseudo-equilibrium regime as the number of epochs $N_{ep,m}$ and $N_{ep,n}$ change. By training with Gibbs sampling, using for the same machine, $m = 10$, $n = 50$ or $m = 10$, $n = 100$, a lowering of the plateau was observed until the transition to a pseudo-equilibrium regime, with plateau height very close to that of the minimum at $t_G \simeq n$. Next, we increased the number of total epochs, alternating blocks of 15 epochs with $n = 10$ and $n = 100$ for 150 epochs to observe the changes from the OOE regime to the pseudo-equilibrium regime. In addition, we lowered the number of steps with $n = 10$, until we studied the case where all epochs have $n = 100$. The regime changes are justified by the change in mixing time (MT) as the number of epochs increases and also by the change in the average value (weighted by epochs) of the number of samplings in the training. Finally, the number of hidden units was varied, showing how the case of $N_h = 90$ hidden units is a special case of the pseudo-equilibrium regime for the RBM constructed in this thesis.

This thesis leaves new ideas for several new studies to be conducted on RBM training with variable Gibbs sampling. First, in our case only RBMs with $m = 10$ (in first part of training) were studied. So, starting training with $m \neq 10$, could lead to new results. Moreover, in this thesis, the number of m and n was very limited due to the calculation time and algorithm used. Therefore, a complete analysis of mixtures with $m, n > 100$ could lead to more complete results. Considering 90 hidden units was also a choice due to saving computation time, so further study with 500 hidden variables, could provide different and interesting results. To improve the quality of training, the total number of epochs should be increased N_{ep}. Finally, the study of mixing time, to prove the regime in which a certain RBM is (OOE, pseudo-equilibrium or equilibrium), was not done, although the plateau provides an important clue.

Bibliography

[1] Pankaj Mehta, Marin Bukov, Ching-Hao Wang, Alexandre G.R. Day, Clint Richardson, Charles K. Fisher, and David J. Schwab. A high-bias, low-variance introduction to machine learning for physicists. *Physics Reports*, 810:1–124, 2019. A high-bias, low-variance introduction to Machine Learning for physicists.

[2] Andrew Yale, Saloni Dash, Ritik Dutta, Isabelle Guyon, Adrien Pavao, and Kristin Bennett. Generation and evaluation of privacy preserving synthetic health data. *Neurocomputing*, 416, 04 2020.

[3] Aurélien Decelle, Cyril Furtlehner, and Beatriz Seoane. Equilibrium and non-equilibrium regimes in the learning of restricted boltzmann machines. *CoRR*, abs/2105.13889, 2021.

[4] Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[5] Asja Fischer and Christian Igel. Training restricted boltzmann machines: An introduction. *Pattern Recognition*, 47(1):25–39, 2014.

[6] Peter Clifford. Markov random fields in statistics. *Disorder in physical systems: A volume in honour of John M. Hammersley*, pages 19–32, 1990.

[7] Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*, volume 4. Springer, 2006.

[8] Pierre Brémaud. *Markov chains: Gibbs fields, Monte Carlo simulation, and queues*, volume 31. Springer Science & Business Media, 2001.

[9] Jovan Tanevski, Sašo Džeroski, and Ljupco Kocarev. Approximate bayesian parameter inference for dynamical systems in systems biology. *Prilozi. Oddelenie za matematičko-tehnički nauki / Makedonska akademija na naukite i umetnostite = Contributions. Section of Mathematical and Technical Sciences / Macedonian Academy of Sciences and Arts*, 31:73–98, 01 2010.

[10] Léon Bottou. *Stochastic Gradient Descent Tricks*, pages 421–436. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[11] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*, 2016.

[12] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[13] Geoffrey E Hinton, Terrence J Sejnowski, et al. Learning and relearning in boltzmann machines. *Parallel distributed processing: Explorations in the microstructure of cognition*, 1(282-317):2, 1986.

[14] Matteo Figliuzzi, Pierre Barrat-Charlaix, and Martin Weigt. How pairwise coevolutionary models capture the collective residue variability in proteins? *Molecular biology and evolution*, 35(4):1018–1027, 2018.

[15] Geoffrey E Hinton. Training products of experts by minimizing contrastive divergence. *Neural computation*, 14(8):1771–1800, 2002.

[16] Tijmen Tieleman and Geoffrey Hinton. Using fast weights to improve persistent contrastive divergence. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, page 1033–1040, New York, NY, USA, 2009. Association for Computing Machinery.

[17] Heping Li, Yunxia Qin, Xiaohu Xiao, and Chaorong Tang. Screening of valid reference genes for real-time rt-pcr data normalization in hevea brasiliensis and expression validation of a sucrose transporter gene hbsut3. *Plant Science*, 181(2):132–139, 2011.

[18] Grégoire Montavon and Klaus-Robert Müller. Deep boltzmann machines and the centering trick. *Neural Networks: Tricks of the Trade: Second Edition*, pages 621–637, 2012.

[19] Jan Melchior, Asja Fischer, and Laurenz Wiskott. How to center deep boltzmann machines. *The Journal of Machine Learning Research*, 17(1):3387–3447, 2016.

[20] Geoffrey E Hinton. A practical guide to training restricted boltzmann machines. *Neural Networks: Tricks of the Trade: Second Edition*, pages 599–619, 2012.

[21] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.

[22] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.