Université de Montréal

Échantillonnage dynamique de champs markoviens

par
Olivier Breuleux

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Mémoire présenté à la Faculté des arts et des sciences
en vue de l'obtention du grade de Maître ès sciences (M.Sc.)
en informatique

Novembre, 2009

Université de Montréal
Faculté des arts et des sciences

Ce mémoire intitulé:

Échantillonnage dynamique de champs markoviens

présenté par:

Olivier Breuleux

a été évalué par un jury composé des personnes suivantes:

Pascal Vincent
président-rapporteur

Yoshua Bengio
directeur de recherche

Fabian Bastin
membre du jury

RÉSUMÉ

L'un des modèles d'apprentissage non-supervisé générant le plus de recherche active est la machine de Boltzmann — en particulier la machine de Boltzmann restreinte, ou RBM. Un aspect important de l'entraînement ainsi que l'exploitation d'un tel modèle est la prise d'échantillons. Deux développements récents, la divergence contrastive persistante rapide (FPCD) et le *herding*, visent à améliorer cet aspect, se concentrant principalement sur le processus d'apprentissage en tant que tel. Notamment, le *herding* renonce à obtenir un estimé précis des paramètres de la RBM, définissant plutôt une distribution par un système dynamique guidé par les exemples d'entraînement. Nous généralisons ces idées afin d'obtenir des algorithmes permettant d'*exploiter* la distribution de probabilités définie par une RBM pré-entraînée, par tirage d'échantillons qui en sont représentatifs, et ce sans que l'ensemble d'entraînement ne soit nécessaire. Nous présentons trois méthodes : la pénalisation d'échantillon (basée sur une intuition théorique) ainsi que la FPCD et le *herding* utilisant des statistiques constantes pour la phase positive. Ces méthodes définissent des systèmes dynamiques produisant des échantillons ayant les statistiques voulues et nous les évaluons à l'aide d'une méthode d'estimation de densité non-paramétrique. Nous montrons que ces méthodes mixent substantiellement mieux que la méthode conventionnelle, l'échantillonnage de Gibbs.

Mots clés : Apprentissage machine, Champs Markoviens, Machine de Boltzmann, MCMC, Modèles probabilistes

ABSTRACT

One of the most active topics of research in unsupervised learning is the Boltzmann machine — particularly the Restricted Boltzmann Machine or RBM. In order to train, evaluate or exploit such models, one has to draw samples from it. Two recent algorithms, Fast Persistent Contrastive Divergence (FPCD) and Herding aim to improve sampling during training. In particular, herding gives up on obtaining a point estimate of the RBM's parameters, rather defining the model's distribution with a dynamical system guided by training samples. We generalize these ideas in order to obtain algorithms capable of *exploiting* the probability distribution defined by a pre-trained RBM, by sampling from it, without needing to make use of the training set. We present three methods: Sample Penalization, based on a theoretical argument as well as FPCD and Herding using constant statistics for their positive phases. These methods define dynamical systems producing samples with the right statistics and we evaluate them using non-parametric density estimation. We show that these methods mix substantially better than Gibbs sampling, which is the conventional sampling method used for RBMs.

Keywords: Machine Learning, Markov Random Fields, Boltzmann Machine, MCMC, Probabilistic models

TABLE DES MATIÈRES

LISTE DES FIGURES

LISTE DES SIGLES

AIS	*Annealed Importance Sampling*
BM	Machine de Boltzmann
CD	Divergence contrastive
FPCD	Divergence contrastive persistante rapide
PCD	Divergence contrastive persistante
RBM	Machine de Boltzmann restreinte
SP	Pénaliastion d'échantillon
StatFPCD	Divergence contrastive persistante rapide avec statistiques positives
StatHerd	*Herding* avec statistiques positives

NOTATION

$\mathcal{X}$	Un ensemble d'entraînement.
$\mathcal{X}_V$	Un ensemble de validation (appoint à l'entraînement).
$\mathcal{X}_T$	Un ensemble de test.
$\mathbb{B}$	L'ensemble binaire $\{0, 1\}$.
$\mathbb{E}$[expression]	L'espérance de l'expression expression, intégrant sur ses variables libres.
$\mathbb{I}$[condition]	1 si la condition est vraie, 0 sinon.
$x \sim \cdot$	x tiré de la distribution représentée par $\cdot$.
x	Un scalaire.
X	Un scalaire, une variable aléatoire, un vecteur de variables aléatoires ou un ensemble, selon le contexte.
$\mathbf{x}$	Un vecteur colonne.
x_i	Un élément d'un vecteur colonne.
$\mathbf{W}$	Une matrice.
W_{ij}	Un élément d'une matrice.
$\mathbf{W}'$	Une matrice transposée.
$\mathbf{I}$	La matrice identité $I_{ij} = \mathbb{I}[i = j]$. La dimension est implicite à l'usage.
$P(X = x)$ $\mathcal{P}(X = x)$ $\mathcal{P}(x)$	Probabilité que la variable aléatoire X prenne pour valeur x.
$P(X = x \mid Y = y)$ $\mathcal{P}(X = x \mid Y = y)$	Probabilité que la variable aléatoire X prenne pour valeur x si la variable aléatoire Y a pour valeur y.

REMERCIEMENTS

En premier lieu, j'aimerais remercier mon directeur, Yoshua Bengio, pour les précieux conseils qu'il m'a prodigués au fil de ma recherche et pour la patience qu'il a eue en me laissant les assimiler à mon rythme. La motivation et le support qu'il m'a insufflés durant les derniers mois ont été d'une valeur inestimable pour la complétion de cet ouvrage. J'aimerais également remercier tous les membres du LISA pour l'atmosphère sympathique et propice à la recherche qu'ils contribuent tous à créer ainsi que ma famille et amis pour leur patience et leur support de tous les jours.

CHAPITRE 1

INTRODUCTION

L'apprentissage machine est un champ d'étude dont la portée et les applications sont particulièrement larges. Fondée dans les mathématiques, inspirée par la biologie et mise en oeuvre par l'informatique, cette discipline a pour objectif avoué de reproduire fonctionnellement la capacité d'apprentissage et d'adaptation qu'on retrouve chez la plupart des espèces vivantes - culminant ultimement au Graal de la recherche en intelligence artificielle, la simulation de l'intelligence humaine (objectif qui est bien sûr loin d'être atteint).

Par opposition à d'autres approches à l'intelligence artificielle, tels les systèmes experts, l'apprentissage machine ne procède pas par ingénierie de règles logiques ou par calque de la méthodologie qui serait utilisée par un expert humain afin de résoudre un problème. Il s'intéresse plutôt à l'établissement de mécanismes *généraux* permettant, à partir d'*exemples* d'un concept ou d'une relation entre concepts, de découvrir une logique sous-jacente. Pour illustrer ce propos, supposons que nous désirons concevoir un système pouvant poser un diagnostic médical à partir d'une liste de symptômes. Une approche traditionnelle serait de programmer à la main un "système expert" basé sur un compendium de règles logiques telle "si on observe les symptômes X, Y et Z, alors le diagnostic est M", règles qu'un comité d'experts médecins déterminerait selon leur expérience. Par contraste, un système basé sur l'apprentissage pourrait recevoir une certaine quantité d'exemples tirés de dossiers médicaux où des symptômes sont associés à un diagnostic final, et à partir de cela il déterminerait la meilleure manière de "calculer" un diagnostic à partir de ces symptômes.

Les avantages de l'apprentissage machine, dans cette situation, sont la possibilité de dériver les règles automatiquement et potentiellement de découvrir des règles complexes qu'un être humain aurait de la difficulté à identifier. Les désavantages sont que le programme ainsi créé peut être ardu à comprendre, ce qui peut réduire

la confiance en la solution, et que l'éventail des tâches solvables est pour l'instant assez limité. Néanmoins, il existe de nombreuses applications où l'apprentissage machine est la seule option viable.

La tâche de diagnostic mentionnée ci-haut n'est pas représentative de toutes les tâches auxquelles il est pertinent d'appliquer l'apprentissage. Il existe plusieurs manières d'encadrer l'apprentissage : en particulier, deux modalités d'apprentissage très distinctes sont l'apprentissage *supervisé* et l'apprentissage *non-supervisé*. Nous nous intéresserons en particulier à la seconde modalité, mais il est utile à la compréhension de bien présenter les deux ainsi que les manières par lesquelles elles peuvent être complémentaires.

1.1 Apprentissage supervisé

De manière générale, l'apprentissage supervisé s'applique sur des exemples qui sont des paires (entrée, sortie) et a pour objectif de trouver une fonction qui, étant donnée l'entrée, parvienne à calculer une valeur qui corresponde, sinon exactement, du moins autant que possible, à la sortie désirée. Nous avons déjà vu un exemple en la tâche de diagnostic médical, où l'entrée est une liste de symptômes et la sortie est la ou les maladies ou conditions qui affligent probablement le patient. Un autre exemple serait la reconnaissance de la parole : on peut considérer une séquence d'exemples où l'entrée est un fichier audio et la sortie est un mot français duquel le fichier audio est une prononciation correcte. En finance, on peut considérer une séquence d'exemples où l'entrée est un portfolio d'investissements et la sortie est le rendement projeté à long terme.

Typiquement, on n'a accès qu'à un sous-ensemble de la relation qui nous intéresse : par exemple, quelques milliers d'enregistrements de mots prononcés par un petit nombre d'humains, en comparaison avec les centaines de milliards qui sont dits sur la planète chaque jour. Dans un autre ordre d'idées, on pourrait avoir à notre disposition plusieurs centaines de portfolios émis par le passé et ce qu'ils ont donné, mais aucune donnée sur le rendement futur des investissements présents.

L'objectif pratique de l'apprentissage est d'utiliser ce qu'on a afin de dériver une logique qui nous soit utile pour traiter de *nouvelles* données ou encore pour *prédire* le futur. Le concept de **généralisation** est donc très important. Pour cette raison, les données sont typiquement séparées en un **ensemble d'entraînement** qui servira directement à l'apprentissage et un **ensemble de test** sur lequel on pourra tester ce qui nous intéresse, c'est à dire la capacité de généralisation du modèle appris. En effet, on observe souvent un phénomène de "surapprentissage" où, une fois qu'une partie de la logique sous-jacente aux données est apprise, l'apprentissage se spécialise à outrance, nuisant par le fait même à la généralisation. Par exemple, on pourrait aisément apprendre par coeur qu'une donnée audio en particulier représente le mot "goyave", mais cela ne nous est d'aucune utilité réelle puisque l'intérêt véritable est de réussir à classifier correctement de nouvelles données audio. Il est possible de sous-diviser l'ensemble d'entraînement au gré des besoins, mais il reste toujours que l'ensemble de test ne doit en aucune façon influer sur l'apprentissage afin de pouvoir mesurer sans biais la capacité du modèle à prédire de nouvelles données. On appelle **ensemble de validation** un ensemble servant à évaluer un modèle dans un but de *sélection* parmi plusieurs variantes de modèles — bien que son utilisation soit similaire à celle d'un l'ensemble de test, la sélection demeure une forme primitive d'apprentissage et il est donc important de les distinguer.

En termes plus formels, dans le cadre de l'apprentissage supervisé, on fournit des paires (x, y) (chacune étant un **exemple**) et on désire, au terme de l'apprentissage, obtenir une fonction f telle que $f(x) \approx y$ tant pour les paires qui ont été fournies que pour les paires qui font partie de l'ensemble de test. Dans le contexte exposé ici, x est l'**entrée**. x est typiquement un vecteur de valeurs réelles ou binaires, mais peut également être un scalaire, une matrice, bref, un objet mathématique quelconque, tout dépendamment du problème. Lorsque y est une valeur continue, on parle de **régression** sur une **cible** y. Lorsque y est tiré d'un ensemble fini et non-ordonné, on parle de **classification** et chaque y possible est une **classe**.

Il existe une multitude d'algorithmes correspondant à l'apprentissage supervisé. Nous ne ferons que les survoler. Si l'on se restreint à la classification, il y a

par exemple les arbres de décision (voir Breiman et al. (1984)), qui segmentent l'espace des x de manière hiérarchique en zones qui correspondent chacun à un y possible ; les k-plus-proches voisins, qui classifient un exemple selon la classification des k exemples d'apprentissage les plus proches dans l'espace des x ; les machines à vecteurs de support (SVM) (Cortes et Vapnik, 1995), qui s'appliquent à séparer l'espace des x de sorte à faire le moins d'erreurs possible tout en maximisant la distance de la frontière entre différentes classes aux exemples qui lui sont le plus proches. Il y a également les réseaux de neurones, dont la version la plus simple est le perceptron (Rosenblatt, 1957), simple classifieur linéaire où la classe est déterminée par le signe du produit scalaire d'un vecteur de poids avec l'entrée. Le perceptron peut également être généralisé à plusieurs couches, chaque couche étant une représentation intermédiaire de l'entrée obtenue par une fonction non-linéaire des produits scalaires de vecteurs de poids avec la représentation de la couche précédente.

1.2 Apprentissage non-supervisé

Comme son nom le suggère, l'apprentissage non-supervisé est un type d'apprentissage où il n'y a pas de "sortie" correspondant à une entrée. En d'autres mots, les algorithmes de ce type s'appliquent sur l'entrée x en tant que telle, sans le y (qui n'existe peut-être pas). En général, l'apprentissage non-supervisé s'intéresse à exploiter la logique sous-jacente à la distribution des exemples eux-mêmes : leur probabilité, leur organisation, etc.

Des exemples intuitifs d'algorithmes non-supervisés sont les algorithmes de *partitionnement* : étant donné un ensemble d'exemples E, il s'agit de partitionner cet ensemble en un nombre fini et aussi petit que possible de sous-ensembles $E_1, E_2, ...$ tels que les exemples appartenant à un de ces sous-ensembles soient tous "proches". En d'autres termes, si l'on identifie chaque exemple x à une étoile, il s'agit d'identifier les galaxies. On peut considérer que dans le cadre du partitionnement, la cible de l'apprentissage est implicitement définie comme étant l'index de la "galaxie" à

laquelle l'"étoile" appartient. L'algorithme des k-moyennes (Lloyd, 1982) appartient à cette catégorie.

Certains algorithmes d'apprentissage non-supervisé sont utilisés afin d'apprendre la distribution de probabilités de l'ensemble d'entraînement. En d'autres mots, il s'agit de trouver une distribution de probabilités sur l'espace des x telle que la probabilité que l'ensemble d'entraînement soit issu de cette distribution soit aussi élevée que possible. Notons qu'il s'agit également de *généraliser* la distribution obtenue à des exemples nouveaux. Dans la mesure où l'on suppose que l'ensemble de test a été tiré de la même distribution que l'ensemble d'entraînement, on désire que sa probabilité soit également maximisée, même si ces exemples sont exclus du processus d'apprentissage.

Un aspect important de certains de ces algorithmes est la possibilité d'*échantillonner* des exemples : par exemple, idéalement, étant donnée la description d'une distribution de probabilités inférée d'un ensemble de lettres calligraphiées par une variété de personnes, on pourrait imaginer être capable de générer des lettres calligraphiées qui n'ont jamais fait partie de l'ensemble d'entraînement mais qui leur ressemblent à s'y méprendre. Dit simplement, l'algorithme pourrait générer des images comme celles qu'il a vues. Dans plusieurs cas, que nous visiterons dans les prochains chapitres, ce processus fait partie intégrante de l'apprentissage et son bon fonctionnement est donc crucial à leur réussite théorique. L'échantillonnage à partir d'un modèle déjà entraîné, de son côté, a plusieurs utilités potentielles : par exemple, il pourrait être utilisée pour complémenter l'ensemble d'entraînement avec de nouveaux exemples et donc fournir davantage de matériel d'apprentissage à d'autres algorithmes, améliorant par le fait même leurs propres performances. L'échantillonnage est aussi une forme de "créativité" ou de "contribution artistique" de la part de la machine — on pourrait envisager apprendre le concept de "beauté" à un algorithme d'estimation de densité en lui montrant quantité de belles images (choisies selon la subjectivité d'un être humain moyen) et espérer être en mesure de créer de nouvelles belles images par échantillonnage de la distribution de probabilités qui leur est associée. Bien sûr, pour ce faire, non seulement l'algorithme doit avoir bien appris et bien

généralisé, il nous faut également une bonne méthode d'échantillonnage, ce qui est loin d'être trivial. Ainsi, tout progrès dans la modélisation des distributions ainsi que dans les techniques d'échantillonnage est aussi un progrès dans cette direction.

Dans plusieurs situations, l'apprentissage non-supervisé peut être un pré-traitement utile à l'apprentissage supervisé. Par exemple, il est possible d'apprendre de manière non-supervisée une représentation des entrées qui soit plus informative de leurs caractéristiques importantes que leur représentation originale et donc contenant plus directement l'information nécessaire afin de prédire une cible (par exemple, on peut plus facilement identifier le tracé d'une lettre dans l'espace des traits que dans l'espace des pixels). Par ailleurs, comme il a été dit précédemment, si l'on est en mesure d'échantillonner efficacement des données synthétiques qui sont différentes des données d'entraînement mais représentatives de celles-ci, il est envisageable d'utiliser ces échantillons comme appoint aux données d'entraînement.

1.3 Description de la contribution du mémoire

Une classe d'algorithmes d'apprentissage non-supervisé générant récemment une certaine quantité de littérature sont les *champs de Markov aléatoires*, classe parmi laquelle on retrouve certains algorithmes tels la *machine de Boltzmann* (Hinton et al., 1984; Ackley et al., 1985), la machine de Boltzmann *restreinte* (RBM) (Smolensky, 1986) ainsi qu'une variation récente, le *herding* (Welling, 2009a,b), auquel nous référerons par le terme anglais en l'absence de traduction adéquate ou reconnue. Comme nous le verrons et sauf dans le cas du *herding*, où l'interprétation probabiliste est plus difficile, ces algorithmes décrivent une distribution de probabilités sur un espace ainsi qu'une procédure visant à adapter cette distribution à la distribution empirique représentée par l'ensemble d'entraînement.

Le processus d'échantillonage est à la fois partie intégrante de la procédure d'entraînement de ces algorithmes et une fin en soi, c'est à dire une manière par laquelle la distribution obtenue peut être exploitée. Plusieurs techniques ont été proposées récemment afin d'améliorer l'échantillonnage pendant l'entraînement de ces mo-

dèles, améliorant par le fait même la qualité des représentations qu'ils apprennent. Comparativement, toutefois, peu de recherche a été faite dans le but d'améliorer l'échantillonnage *après* l'apprentissage, bien que cela soit une tâche intéressante en soi.

Ce mémoire se veut la présentation, l'analyse et l'évaluation de plusieurs nouvelles techniques visant à tirer des échantillons de distributions définies par des RBM. Nous commencerons par définir chacun des algorithmes dont la compréhension est nécessaire. Les algorithmes déjà établis seront présentés au cours des chapitres 2 et 3. Le chapitre 4 sera consacré aux techniques d'échantillonnage à l'étude, dont nous présenterons les justifications théoriques. Au cours du chapitre 5, nous définirons une méthodologie appropriée pour évaluer l'amélioration de l'échantillonnage par rapport aux techniques conventionnelles utilisées. Les résultats de nos expériences seront présentés et discutés au chapitre 6. Le chapitre 7 conclura ce mémoire.

CHAPITRE 2

CHAMPS DE MARKOV ALÉATOIRES

2.1 Définition d'un champs de Markov aléatoire

Un **champs de Markov aléatoire** (nous référons le lecteur à Clifford (1990) pour tout détail manquant dans l'exposé qui suit) est un modèle constitué d'un ensemble de variables aléatoires $X = \{X_1, X_2, ...X_n\}$ dont les dépendances mutuelles peuvent être modélisées à l'aide d'un graphe non-dirigé $G = (V, E)$, où V est l'ensemble des indices $\{1, 2, ...n\}$ tels qu'il existe des X_i correspondants et E est un ensemble de paires $(i, j) \in V^2$ indiquant que les distributions des variables X_i et X_j sont conditionnellement dépendantes l'une de l'autre. Plus formellement :

$$X_a | X_{V \setminus \{X_a, X_b\}} \quad \perp \quad X_b \text{ si } (a, b) \notin E \tag{2.1}$$

Où $X \perp Y$ dénote l'indépendance des variables aléatoires X et Y. En d'autres mots, le *voisinage* immédiat de X_a, correspondant à toutes les variables X_b telles que $\{a, b\} \in E$, définit exactement la distribution de probabilités de X_a. Étant données les variables faisant partie de son voisinage, toute variable aléatoire de l'ensemble X est conditionnellement indépendante de toutes celles qui ne lui sont pas *directement* connectées.

Cette restriction sur les dépendances conditionnelles constitue une propriété de Markov au sens large, d'où le nom du modèle.

Une formulation commune pour les champs de Markov est la suivante :

$$p(x_1, x_2, ...x_n) \quad = \quad \frac{1}{\mathcal{Z}} \exp \left(\sum_{C \in \text{cliques}(G)} \theta_C g_C(x_C) \right) \tag{2.2}$$

Où $\mathcal{Z} = \sum_{\mathbf{x}} \exp \left(\sum_{C \in \text{cliques}(G)} \theta_C g_C(x_C) \right)$ est la **fonction de partition** de la distribution, c'est à dire le facteur de normalisation faisant en sorte que les

probabilités de tous les états somment à 1. Les g_C sont les **statistiques suffisantes** sur le graphe — chaque clique du graphe étant associée à l'une d'entre elles — et θ_C est le **paramètre** associé à la statistique g_C. Cette formulation est équivalente à une factorisation de la densité du champs de Markov aléatoire selon les cliques du graphe (il existe toutefois des champs de Markov aléatoires qui ne sont pas factorisables de cette manière, voir Moussouris (1974)).

2.2 Machine de Boltzmann

Une machine de Boltzmann (Hinton et al., 1984; Ackley et al., 1985) est un type de champs de Markov aléatoire sur des unités stochastiques binaires. La machine a un état interne $\mathbf{s} = (s_1, s_2, ...s_n), s_i \in \mathbb{B} = \{0, 1\}$ (ou $s_i \in \{-1, 1\}$[1]) sur lequel s'opère une dynamique décrite plus loin. À chaque configuration possible $\mathbf{s} \in \mathbb{B}^n$ est associée la probabilité suivante :

$$\mathcal{P}(\mathbf{s}) = \frac{\exp\left(\frac{-\mathrm{E}(\mathbf{s})}{T}\right)}{\sum_{\mathbf{y}\in\mathbb{B}^n} \exp\left(\frac{-\mathrm{E}(\mathbf{y})}{T}\right)} \tag{2.3}$$

$$= \frac{1}{\mathcal{Z}} \exp\left(\frac{-\mathrm{E}(\mathbf{s})}{T}\right) \tag{2.4}$$

Où $\mathcal{Z}$ est la fonction de partition (facteur de normalisation) du modèle, $\mathrm{E}(\mathbf{s})$ est l'**énergie** de la configuration $\mathbf{s}$ et T est la **température** du modèle.

L'énergie $\mathrm{E}(\mathbf{s})$ est une quantité réelle associée à chaque état pouvant être positive ou négative, correspondant (lorsque $T = 1$) au terme $-\sum_{C\in\text{cliques}(G)} \theta_C g_C(x_C)$ de l'équation 2.2. On peut facilement voir qu'une haute énergie résulte en une probabilité faible tandis qu'une basse énergie résulte en une probabilité élevée.

T est la *température* du modèle — plus T est grand, plus les énergies des dif-

[1]Pour simplifier, dans les développements et les explications qui suivent, nous supposerons qu'une unité prend comme valeur soit 0, soit 1. Toutefois, il est trivial d'adapter ces développements à la supposition que les unités du système prennent leurs valeurs dans l'ensemble {-1, 1}, ce qui résulte en une machine de Boltzmann tout à fait valide mais donnant des résultats légèrement différents

férents états sont compressées autour de zéro. Cela rapproche leurs probabilités et donc la distribution est plus uniforme. D'un autre côté, plus la température est basse, plus les différences en énergie de différents états sont magnifiées. Cette distribution est communément appelée distribution de Boltzmann et est présente notamment en thermodynamique, où elle décrit la distribution d'états microscopiques au sein d'un système macroscopique, par exemple la distribution des vélocités des particules constituant un gaz à l'équilibre, en fonction de sa température (Widom (2002), page 1).

Il existe, *a priori*, beaucoup de manières de définir l'énergie d'un état en fonction de ses composantes. Tout au long de ce mémoire, à moins de mention contraire, nous définirons l'énergie d'un état comme pour la machine de Boltzmann (Hinton et al., 1984; Ackley et al., 1985), c'est à dire comme une combinaison linéaire de ses composantes et de leurs corrélations :

$$\mathrm{E}(\mathbf{s}) = -\sum_i b_i s_i - \frac{1}{2}\sum_{i,j} W_{ij} s_i s_j \tag{2.5}$$

Les W_{ij} forment la **matrice de poids**, tandis que les b_i forment le **vecteur de biais**. Certaines restrictions s'appliquent : il ne doit y avoir aucun lien entre une unité et elle-même ($W_{ii} = 0$) et les liens doivent être symétriques ($W_{ij} = W_{ji}$). Comme indiqué précédemment, d'autres définitions d'énergie sont possibles. On pourrait notamment considérer les facteurs de troisième ordre $s_i s_j s_k$ (Sejnowski, 1986; Memisevic et Hinton, 2007).

On définit l'**activation** a_i de l'unité i comme étant la différence d'énergie obtenue en changeant l'état s_i de la valeur 1 à la valeur 0, en gardant les autres unités constantes.

$$a_i = \mathrm{E}(\mathbf{s}_{[s_i=1]}) - \mathrm{E}(\mathbf{s}_{[s_i=0]}) \tag{2.6}$$

$$= b_i + \sum_j W_{ij} s_j \tag{2.7}$$

Notons que le facteur $\frac{1}{2}$ disparaît puisque dans l'équation originale, on comptait

chaque lien deux fois pour $i < j$ et pour $j < i$, ce qui n'est pas le cas ici. Dans le cas où les états peuvent prendre les valeurs -1 ou 1 plutôt que 0 et 1, l'activation prend une valeur deux fois plus grande.

2.2.1 Unités cachées

On peut facilement considérer séparer les unités d'une machine de Boltzmann en deux groupes : les **unités visibles**, auxquelles on peut rattacher une signification concrète (par exemple, les pixels d'une image) et les **unités cachées**, dont les valeurs ne sont pas observées dans les données. Ces unités n'ont donc pas nécessairement de signification concrète mais peuvent augmenter la capacité de représentation du modèle en lui permettant de modéliser des relations plus complexes. Par exemple, le fait que trois unités i, j et k soient presque toujours dans le même état peut être représenté par de fortes valeurs pour W_{ij} et W_{jk}, mais aussi par de fortes valeurs pour W_{il}, W_{jl} et W_{kl} où l est une unité cachée, ce qui ajoute des degrés de liberté dans les poids entre i, j et k permettant de représenter des informations supplémentaires. La figure 2.1 schématise une machine de Boltzmann ayant des unités cachées.

Dans la mesure où nous considérons que les unités cachées ne servent que d'auxiliaires au calcul et que nous nous intéressons en réalité à la distribution des unités visibles, il suffit de marginaliser sur les unités cachées. Si l'on dénote les unités visibles par $\mathbf{v}$ et les unités cachées par $\mathbf{h}$, de sorte que $\mathbf{s} = (\mathbf{v}, \mathbf{h})$, on obtient le résultat qui suit :

$$\begin{aligned}
\mathcal{P}(\mathbf{v}) &= \sum_{\mathbf{h}\in\mathbb{B}^{n_h}} \mathcal{P}(\mathbf{v},\mathbf{h}) && (2.8)\\
&= \frac{1}{\mathcal{Z}}\left(\sum_{\mathbf{h}\in\mathbb{B}^{n_h}} \exp\left(\frac{-\mathrm{E}(\mathbf{v},\mathbf{h})}{T}\right)\right) && (2.9)\\
&= \frac{1}{\mathcal{Z}}\exp\left(\log\left(\sum_{\mathbf{h}\in\mathbb{B}^{n_h}} \exp\left(\frac{-\mathrm{E}(\mathbf{v},\mathbf{h})}{T}\right)\right)\right) && (2.10)\\
&= \frac{1}{\mathcal{Z}}\exp\left(-\mathrm{E_v}(\mathbf{v},T)\right) && (2.11)
\end{aligned}$$

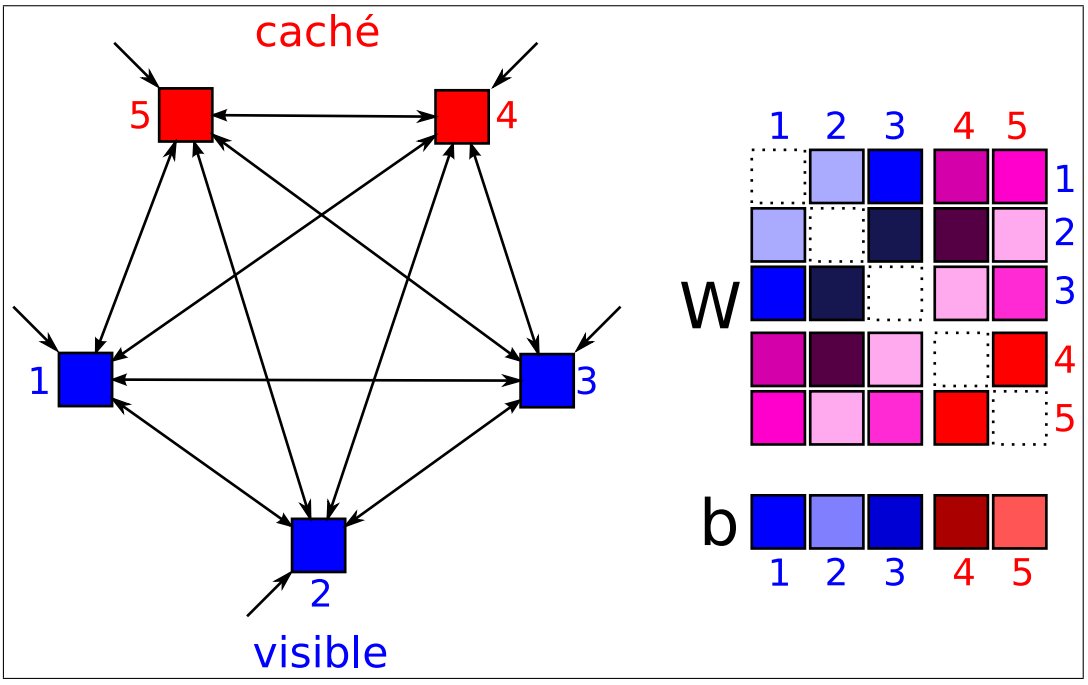


FIG. 2.1 – Visualisation d'une machine de Boltzmann à trois unités visibles (en bleu) et deux unités cachées (en rouge). Chaque unité est reliée à chaque autre, mais pas à elle-même — chaque unité reçoit également un biais qui lui est propre. Les paramètres du modèle sont montrés à droite — par exemple, la flèche de l'unité 1 à l'unité 4 est représentée par le poids en (1, 4). Plus un carré est clair, plus le poids est positif, plus il est foncé, plus le poids est négatif, tandis qu'un carré en pointillé a valeur nulle. Les paramètres sont colorés en bleu, rouge ou violet dépendamment de s'ils sont entre unités visibles, cachées ou entre unités des deux types. Notons que $\mathbf{W}$ est une matrice symétrique.

Où $\mathrm{E_v}(\mathbf{v}, T)$ est l'**énergie libre** de la configuration $\mathbf{v}$ à la température T, une quantité qui peut être mise en parallèle avec l'énergie d'un état complet (où les unités visibles et cachées sont fixées), à la différence près qu'elle ne varie pas linéairement en fonction de T (T ne peut donc pas être abstrait de la formulation de l'énergie libre).

2.2.2 Entraînement

Supposons un ensemble d'exemples d'entraînement $\mathcal{X} = \{\mathbf{x}^{(i)}\} \subseteq \mathbb{B}^{n_v}$ correspondant à des états possibles pour les unités visibles d'une machine de Boltzmann. L'objectif de la procédure d'entraînement est de trouver des paramètres $\theta = (\mathbf{W}, \mathbf{b})$ tels que la distribution définie par la machine de Boltzmann ainsi paramétrisée s'approche le plus de la distribution suggérée par les exemples. Par exemple, si l'on considère un millier d'images binaires de résolution 30x30 représentant des chiffres en vrac de zéro à neuf, la distribution *suggérée*, en termes de généralisation, serait un histogramme sur les 2^{900} images possibles où chaque image aurait une probabilité non-nulle si et seulement si elle représente un chiffre. Idéalement, on voudrait que la machine de Boltzmann parvienne à encoder cette distribution et à en tirer des échantillons[2].

En d'autres termes, d'une part, on veut faire en sorte que les exemples d'entraînement $\mathbf{x}^{(i)}$ aient une *basse énergie*, car une basse énergie résulte en une haute probabilité pour l'état correspondant. D'autre part, on veut faire en sorte que la plupart des états qui ne sont pas dans l'ensemble d'entraînement aient une *haute énergie*, c'est à dire une basse probabilité. Au terme de l'exercice, on espère que la machine aura appris la logique intrinsèque aux exemples d'entraînement, de sorte qu'elle soit capable de *généraliser* en donnant également une basse énergie, donc une haute probabilité, aux exemples qui "ressemblent" à ceux qu'elle a traités. Un ensemble de test distinct de l'ensemble d'entraînement devrait ainsi être réservé pour fins d'évaluation de la généralisation.

Dénotons $P^+(\mathbf{x})$ la probabilité de $\mathbf{x}$ dans l'ensemble d'entraînement et $P^-(\mathbf{x})$ sa probabilité dans la distribution définie par la machine de Boltzmann que l'on entraîne. Notons que $P^+(\mathbf{x})$ est souvent tout simplement $1/|\mathcal{X}|$, c'est à dire une distribution uniforme sur les exemples. $P^-(\mathbf{x})$, en supposant que le modèle comporte des unités cachées, est donné par $\mathcal{P}(\mathbf{x})$ dans l'équation 2.11.

[2]Notons qu'une machine de Boltzmann ne peut attribuer une probabilité nulle à un état, l'image de la fonction exponentielle sur les réels excluant zéro. Elle peut cependant lui attribuer une probabilité arbitrairement proche de zéro, ce qui est à toutes fins pratiques équivalent.

L'entraînement doit minimiser la "différence" entre les deux distributions. La mesure de différence que nous utiliserons est la divergence de Kullback-Leibler, définie comme suit entre $P^+(\mathbf{x})$ et $P^-(\mathbf{x})$:

$$C = D_{\mathrm{KL}}(P^+ \| P^-) = \sum_{\mathbf{x}\in\mathcal{X}} P^+(\mathbf{x}) \log\left(\frac{P^+(\mathbf{x})}{P^-(\mathbf{x})}\right) \quad (2.12)$$

$$= \sum_{\mathbf{x}\in\mathcal{X}} P^+(\mathbf{x}) \log(P^+(\mathbf{x})) - P^+(\mathbf{x}) \log(P^-(\mathbf{x})) \quad (2.13)$$

L'expression ci-haut représente le *coût* C à minimiser et vaut 0 lorsque les distributions $P^+(\mathbf{x})$ et $P^-(\mathbf{x})$ sont équivalentes. En supposant que comme mentionné plus haut $P^+(\mathbf{x}) = 1/|\mathcal{X}|$, $C = \frac{1}{|\mathcal{X}|}\left(-|\mathcal{X}|\log(|\mathcal{X}|) - \sum_{\mathbf{x}\in\mathcal{X}} \log(P^-(\mathbf{x}))\right)$, dont le minimum est identique à celui de la log-vraisemblance négative de $\mathcal{X}$, $-\sum_{\mathbf{x}\in\mathcal{X}} \log(P^-(\mathbf{x}))$.

Notons que étant données les limitations imposées par la fonction d'énergie sur la structure de la distribution de la machine de Boltzmann (par exemple, pour une fonction d'énergie finie, la probabilité d'un état ne peut être zéro), il n'est pas nécessairement possible d'imiter parfaitement la distribution cible. Qui plus est, même si cela était possible, cela ne serait probablement pas souhaitable : en effet, la distribution cible est typiquement la distribution empirique de l'ensemble d'entraînement, ce qui ne laisse pas de masse de probabilité aux exemples nouveaux ou à ceux de l'ensemble de test. En réalité, la capacité limitée de la machine de Boltzmann est un atout nous permettant d'espérer que la solution de coût minimal, à défaut de pouvoir calquer ou apprendre par coeur les exemples de l'ensemble d'entraînement, soit suffisamment générale pour être utile.

2.2.2.1 Gradient

La première étape vers la minimisation du coût C défini précédemment est de calculer son gradient par rapport aux paramètres du modèle θ :

$$\frac{\partial C}{\partial \theta} = \sum_{\mathbf{x}\in\mathcal{X}} -P^{+}(\mathbf{x})\frac{1}{P^{-}(\mathbf{x})}\frac{\partial P^{-}(\mathbf{x})}{\partial \theta} \tag{2.14}$$

$$= \sum_{\mathbf{x}\in\mathcal{X}} -\frac{P^{+}(\mathbf{x})}{P^{-}(\mathbf{x})}\frac{\partial \frac{\exp(-\mathrm{E_v}(\mathbf{x},T))}{\mathcal{Z}}}{\partial \theta} \tag{2.15}$$

$$= \sum_{\mathbf{x}\in\mathcal{X}} -\frac{P^{+}(\mathbf{x})}{P^{-}(\mathbf{x})}\left(\frac{1}{\mathcal{Z}}\frac{\partial \exp(-\mathrm{E_v}(\mathbf{x},T))}{\partial \theta} - \frac{\exp(-\mathrm{E_v}(\mathbf{x},T))}{\mathcal{Z}^2}\frac{\partial \mathcal{Z}}{\partial \theta}\right) \tag{2.16}$$

$$= \sum_{\mathbf{x}\in\mathcal{X}} -\frac{P^{+}(\mathbf{x})}{P^{-}(\mathbf{x})}\left(-\frac{\exp(-\mathrm{E_v}(\mathbf{x},T))}{\mathcal{Z}}\frac{\partial \mathrm{E_v}(\mathbf{x},T)}{\partial \theta} - \frac{\exp(-\mathrm{E_v}(\mathbf{x},T))}{\mathcal{Z}^2}\frac{\partial \mathcal{Z}}{\partial \theta}\right)$$

$$= \sum_{\mathbf{x}\in\mathcal{X}} P^{+}(\mathbf{x})\left(\frac{\partial \mathrm{E_v}(\mathbf{x},T)}{\partial \theta} + \frac{1}{\mathcal{Z}}\sum_{\mathbf{v}\in\mathbb{B}^{n_v}}\frac{\partial \exp(-\mathrm{E_v}(\mathbf{v},T))}{\partial \theta}\right) \tag{2.17}$$

La valeur de $\frac{\partial \mathrm{E_v}(\mathbf{v},T)}{\partial \theta}$ est ensuite dérivée comme suit :

$$\frac{\partial \mathrm{E_v}(\mathbf{v},T)}{\partial \theta} = -\frac{1}{T}\frac{\sum_{\mathbf{h}\in\mathbb{B}^{n_h}} \exp\left(\frac{-\mathrm{E}(\mathbf{v},\mathbf{h})}{T}\right)\frac{\partial \mathrm{E}(\mathbf{v},\mathbf{h})}{\partial \theta}}{\sum_{\mathbf{h}\in\mathbb{B}^{n_h}} \exp\left(\frac{-\mathrm{E}(\mathbf{v},\mathbf{h})}{T}\right)} \tag{2.18}$$

$$= \frac{-\mathbb{E}_{\mathbf{h}}\left[\frac{\partial \mathrm{E}(\mathbf{v},\mathbf{h})}{\partial \theta}\middle|\mathbf{v}\right]}{T} \tag{2.19}$$

En posant $g_\theta(\mathbf{v},\mathbf{h}) = -\frac{\partial \mathrm{E}(\mathbf{v},\mathbf{h})}{\partial \theta}$, on obtient que :

$$
\begin{aligned}
\frac{\partial C}{\partial \theta} &= \frac{1}{T}\sum_{\mathbf{x}\in\mathcal{X}} P^{+}(\mathbf{x})\left(-\mathbb{E}_{\mathbf{h}}[g_\theta(\mathbf{x},\mathbf{h})|\mathbf{x}] + \sum_{\mathbf{v}\in\mathbb{B}^{n_v}} \frac{\exp(-\mathrm{E_v}(\mathbf{v},T))}{\mathcal{Z}}\mathbb{E}_{\mathbf{h}}[g_\theta(\mathbf{v},\mathbf{h})|\mathbf{v}]\right) \\
&= \frac{1}{T}\sum_{\mathbf{x}\in\mathcal{X}} -P^{+}(\mathbf{x})\left(\mathbb{E}_{\mathbf{h}}[g_\theta(\mathbf{x},\mathbf{h})|\mathbf{x}] - \sum_{\mathbf{v}\in\mathbb{B}^{n_v}} P^{-}(\mathbf{v})\mathbb{E}_{\mathbf{h}}[g_\theta(\mathbf{v},\mathbf{h})|\mathbf{v}]\right) \qquad (2.20) \\
&= -\frac{1}{T}\left(\sum_{\mathbf{x}\in\mathcal{X}} P^{+}(\mathbf{x})\mathbb{E}_{\mathbf{h}}[g_\theta(\mathbf{x},\mathbf{h})|\mathbf{x}]\right) + \frac{1}{T}\left(\sum_{\mathbf{v}\in\mathbb{B}^{n_v}} P^{-}(\mathbf{v})\mathbb{E}_{\mathbf{h}}[g_\theta(\mathbf{v},\mathbf{h})|\mathbf{v}]\right) \qquad (2.21) \\
&= -\frac{1}{T}\left(\mathbb{E}^{+}[g_\theta(\mathbf{s})] - \mathbb{E}^{-}[g_\theta(\mathbf{s})]\right) \text{ (notation : } \mathbf{s} = (\mathbf{v},\mathbf{h})) \qquad (2.22) \\
&= -\frac{1}{T}\left(g_\theta^{+} - g_\theta^{-}\right) \text{ (autre notation)} \qquad (2.23)
\end{aligned}
$$

On peut donc toujours séparer le gradient en deux termes : $\mathbb{E}^{+}[g_\theta(\mathbf{s})]$, qu'on dénotera alternativement g_θ^{+} pour faire court, est appelée la **phase positive** du gradient. $\mathbb{E}^{-}[g_\theta(\mathbf{s})]$, ou g_θ^{-}, est appelée la **phase négative**. La phase positive correspond à l'espérance de la statistique paramétrisée par θ lorsque la distribution des unités visibles $\mathbf{x}$ est donnée par l'ensemble d'entraînement $\mathcal{X}$. Il est à noter que si certaines unités sont cachées et donc implicites, il est nécessaire d'inférer leurs valeurs à partir des unités visibles et des paramètres du modèle. Dans ce cas, la phase positive ne dépend pas uniquement de l'ensemble d'entraînement (ce qui serait le cas si toutes les unités étaient visibles). La phase négative, quant à elle, est entièrement définie par les paramètres du modèle.

Pour rendre le tout plus concret, supposons que la fonction d'énergie est telle que définie par l'équation 2.5 et que $\theta = (\mathbf{W},\mathbf{b})$. Le gradient de l'énergie par rapport au poids W_{ij} entre deux unités i et j, qui est dénoté par $g_{W_{ij}}(\mathbf{s})$, est tout simplement le produit $s_i s_j$ de leurs valeurs. Par rapport au biais d'une unité b_i, nous obtenons $g_{b_i}(\mathbf{s}) = s_i$. On peut donc dire que $\mathbf{W}$ paramétrise des statistiques qui sont les *corrélations* entre paires d'unités, tandis que $\mathbf{b}$ paramétrise leurs valeurs individuelles.

Lorsque toutes les unités sont visibles, le terme g_θ^{+} de la phase positive va de soi (les termes $\mathbb{E}_{\mathbf{h}}[g_\theta(\mathbf{v},\mathbf{h})|\mathbf{v}]$ à l'équation 2.21 ne dépendent plus du calcul de $\mathbf{h}$ ni

par ce fait de θ, et deviennent tout simplement $g(\mathbf{v})$) et est facile à calculer empiriquement. Lorsqu'une des unités, ou les deux, sont cachées, il suffit de supposer que pour chaque vecteur visible, les unités cachées sont fixées à l'espérance de leur valeur sous la contrainte que les unités visibles ne changent pas. Sous certaines contraintes de connectivité que nous verrons plus loin, cette espérance peut être calculée analytiquement et donc il est possible de calculer le terme de la phase positive exactement.

Si et seulement si les phases positive et négative sont égales, les deux contributions s'annulent et on se retrouve dans un point critique (en principe un minimum) de la fonction de coût C. Malheureusement, il n'existe pas de manière analytique ou efficace nous permettant de calculer les paramètres donnant lieu à cette situation. En fait, le terme de la phase négative, dans la mesure où l'on voudrait le calculer exactement, implique un nombre d'états à calculer qui croît exponentiellement avec le nombre d'unités. Par conséquent, il nous est impossible ne serait-ce que de calculer le gradient exact à un certain point dans l'espace des paramètres, sauf pour un nombre d'unités très petit.

2.2.2.2 Approximation stochastique

La stratégie utilisée afin d'entraîner une machine de Boltzmann, étant données les limitations listées ci-haut, consiste à calculer une *approximation* du gradient et à effectuer une *descente de gradient*, puisque l'objectif est de minimiser le coût, ce qui peut se faire en suivant une trajectoire qui le diminue progressivement. S'il n'y a pas d'unités cachées, le problème d'optimisation est convexe, c'est à dire qu'il n'y a qu'un seul minimum qu'on peut trouver à coup sûr par descente de gradient (note : en autant que celle-ci ne soit pas approximative). Dans le cas contraire, il existe des minima *locaux* dans lesquels l'optimisation peut rester coincée, mais qui peuvent tout de même être des solutions satisfaisantes.

La quantité g_θ^- posant problème, la solution typique est de l'approximer en tirant des *échantillons* $S = \{\mathbf{s}^{(1)}, \mathbf{s}^{(2)}, ..., \mathbf{s}^{(n)}\}$ de la distribution de la machine de Boltzmann et en compilant l'espérance empirique, sur ces échantillons, des statis-

tiques pondérées par θ (pour W_{ij}, la statistique correspondante sur un échantillon $\mathbf{s}$ serait $s_i s_j$). La quantité g_θ^+ peut également être approximée en sélectionnant à chaque mise à jour un sous-ensemble des exemples d'entraînement, ce qui peut réduire considérablement la quantité de calculs dans le cas où des unités cachées sont utilisées (LeCun et al., 1998b).

L'algorithme 1 présente une itération d'entraînement de la machine de Boltzmann, par descente de gradient approximée. Un entraînement complet nécessite normalement un grand nombre de répétitions de cette procédure.

Algorithm 1 Algorithme décrivant une machine de Boltzmann.

`ÉtapeMachineBoltzmann(`$\mathcal{X}$`, `θ`, `T`, `ϵ`)`
Une étape de l'entraînement d'une machine de Boltzmann. $\mathcal{X}$ est l'ensemble d'entraînement, θ sont les paramètres de la machine, T est la température et ϵ est le taux d'apprentissage.
Pour simplifier, un seul exemple positif et un seul exemple négatif sont générés et utilisés. Afin de stabiliser la procédure, plusieurs exemples peuvent être utilisés, en quel cas il suffit de moyenner leurs contributions (et non les exemples eux-mêmes, ce qui donnerait de mauvais résultats !)

$\mathbf{v}^+ \sim \mathcal{X}$ # Unités visibles de l'exemple positif
$\mathbf{h}^+ \sim \mathcal{P}(\mathbf{h}|\mathbf{v}^+; \theta, T)$ # Unités cachées de l'exemple positif
$(\mathbf{v}^-, \mathbf{h}^-) \sim \mathcal{P}(\mathbf{v}, \mathbf{h}; \theta, T)$ # Unités de l'exemple négatif
for i = 1 to $|\theta|$ **do**
  # Chaque θ_i est mis à jour selon sa statistique.
  $\theta_i \leftarrow \theta_i + \frac{\epsilon}{T}(g_{\theta_i}(\mathbf{v}^+, \mathbf{h}^+) - g_{\theta_i}(\mathbf{v}^-, \mathbf{h}^-))$
end for
return θ

Le paramètre ϵ est le **taux d'apprentissage** de la descente de gradient. C'est le facteur multiplicatif appliqué à la mise à jour des paramètres θ. Si l'on compare le processus d'apprentissage à une marche à travers les Rocheuses visant à marcher jusqu'au sommet d'une montagne, la mise à jour représente la direction dans laquelle on marche (typiquement celle qui monte le plus selon le terrain sur lequel on se trouve au moment de prendre la décision) et le taux d'apprentissage est la durée de temps pendant laquelle l'on marche dans cette direction. Ce paramètre est assez sensible : un taux trop petit ne permet pas aux paramètres de dévier suffisamment

de leur initialisation pour résoudre la tâche (on fait un petit pas à la fois et on se contente de monter sur la colline la plus proche), tandis qu'un taux trop grand les amène trop loin (on marche tellement longtemps qu'une fois rendu au sommet de la montagne, on marche jusqu'à la vallée de l'autre côté).

On peut imaginer qu'une bonne stratégie serait de commencer avec un pas relativement grand et de le diminuer au fur de l'apprentissage. En fait, afin de garantir la convergence de l'apprentissage à un minimum local, le taux d'apprentissage doit diminuer tel $\epsilon_t \approx \frac{1}{\epsilon_0 t}$, où t est le nombre de mises à jour jusqu'à ce point et ϵ_0 une constante (Bishop, 2006).

Une autre optimisation pouvant être appliquée lors de l'entraînement d'une machine de Boltzmann est la réduction de la température T à travers le temps (chaque mise à jour successive se faisant à une température plus basse). Comme il a été expliqué précédemment, une haute température a tendance à égaliser les probabilités, tandis qu'une basse température a tendance à faire ressortir les modes majeurs et à marginaliser les autres. L'avantage de commencer avec une haute température est que à une haute température tout l'espace des états a une probabilité significative et cette tendance est robuste aux mises à jour, ce qui permet à la procédure d'éviter de tomber dans un minimum local trop rapidement. Or, à mesure que l'apprentissage progresse, on est de plus en plus confiant que nous nous trouvons dans une "bonne" zone de l'espace et que les modes majeurs sont ceux qui nous intéressent. Cette méthode est une forme de *recuit simulé*. En pratique, les poids on tendance à augmenter d'eux-mêmes lors de l'apprentissage, ce qui est équivalent à une réduction de température. Il est possible d'apprendre convenablement sans avoir besoin de la changer, mais évidemment tout dépend des circonstances.

Toujours est-il qu'il apparaît clairement qu'il est crucial, pour bien entraîner une machine de Boltzmann, d'avoir une bonne technique d'échantillonnage qui nous permette de calculer les mises à jour successives de la descente de gradient de manière efficace et aussi peu biaisée que possible. Cet aspect sera couvert dans le prochain chapitre.

2.3 Machine de Boltzmann Restreinte

Une **machine de Boltzmann restreinte** (Smolensky, 1986), que nous abbrévierons RBM (pour le terme anglais *Restricted Boltzmann Machine*), est une machine de Boltzmann dont la connectivité entre les unités est limitée. Pour être plus précis, les unités sont séparées en unités visibles v_i et cachées h_j de sorte qu'il n'existe aucune connection entre unités du même groupe. C'est donc dire que la matrice de poids $\mathbf{W}$ est zéro pour toute paire d'indices $\{i, j\}$ pour laquelle s_i et s_j sont tous les deux visibles ou cachés. La figure 2.2 montre une machine de Boltzmann restreinte.

Étant donné l'indépendance entre les deux types d'unités, on peut simplifier un peu la représentation en les séparant explicitement et en réduisant la taille de matrice $\mathbf{W}$ de sorte qu'elle ne contienne que les poids non-nuls. On se retrouve alors avec des paramètres $\theta = (W, b, c)$ et l'énergie peut être exprimée de la manière suivante :

$$\mathrm{E}(\mathbf{v}, \mathbf{h}) \quad = \quad -\sum_i c_i v_i - \sum_j b_j h_j - \sum_{i,j} W_{ij} v_i h_j \tag{2.24}$$

Il n'existe aucun poids liant les $\mathbf{v}$ ou les $\mathbf{h}$ entre eux et on a deux vecteurs de biais, $\mathbf{b}$ et $\mathbf{c}$, qui agissent sur les unités cachées et visibles respectivement.

2.3.1 Avantages

La RBM possède plusieurs avantages par rapport à une machine de Boltzmann générale. Notamment, comme il n'existe aucune dépendance entre unités visibles et unités cachées dans leurs sous-groupes, il s'ensuit que $\mathcal{P}(\mathbf{v}|\mathbf{h})$ et $\mathcal{P}(\mathbf{h}|\mathbf{v})$ peuvent être calculés analytiquement. Notamment, le terme $\mathbb{E}[g_{W_{ij}}(\mathbf{s})|\mathbf{x}] = \mathbb{E}[s_i s_j|\mathbf{x}]$ qui se retrouve dans la contribution positive du gradient, lorsque i est une unité visible et j une unité cachée, est égal à $s_i \mathbb{E}[s_j|\mathbf{x}] = s_i \mathcal{P}(s_j|\mathbf{x})$ et peut donc être calculé aisément sans besoin d'avoir recours à l'échantillonnage (qui reste néanmoins nécessaire afin de calculer la contribution négative).

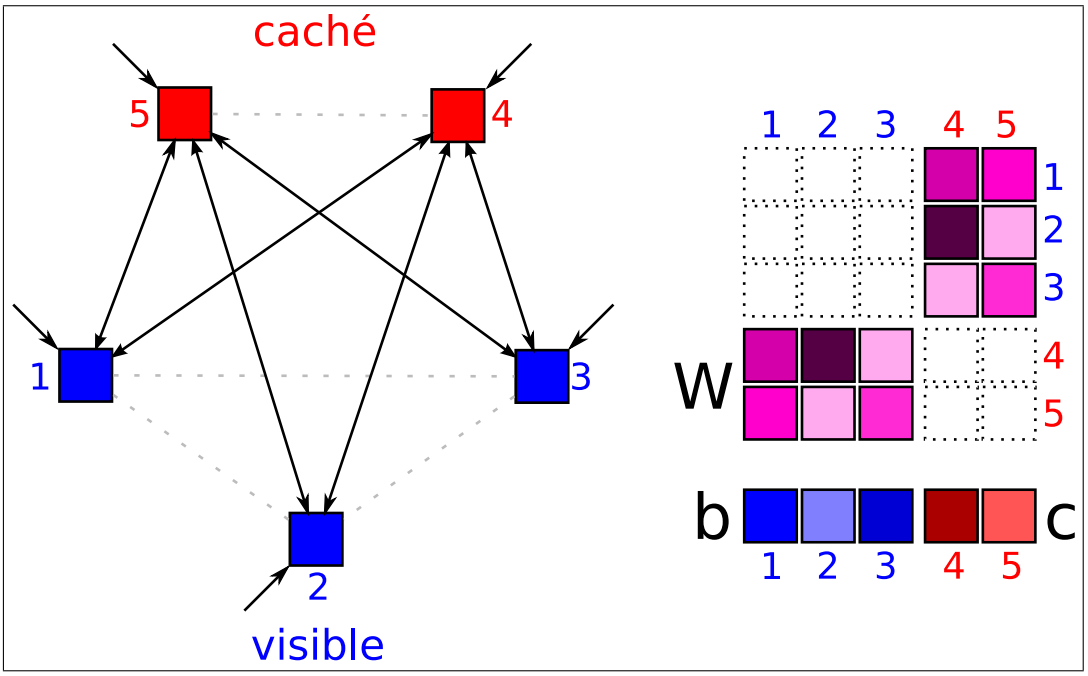


FIG. 2.2 – Visualisation d'une machine de Boltzmann restreinte à trois unités visibles (en bleu) et deux unités cachées (en rouge). Par rapport à la figure 2.1, notons que les poids entre unités visibles et les poids entre unités cachées ont disparu — les biais restent toutefois inchangés. Le léger changement de notation de la BM à la RBM est rendu explicitement ici : $\mathbf{W}$ est la matrice des connections entre unités visibles et cachées, $\mathbf{b}$ sont les biais des unités visibles seulement et $\mathbf{c}$ sont les biais des unités cachées seulement.

Il est possible d'entraîner une RBM de sorte à obtenir un modèle profond, c'est à dire avec plusieurs couches, de manière "gloutonne" (Hinton et al., 2006) : une fois une RBM entraînée, sa représentation cachée peut être calculée en échantillonnant de $\mathcal{P}(\mathbf{h}|\mathbf{v}) = \text{sigmoid}(\mathbf{W}\mathbf{v} + \mathbf{b})$. Cette représentation peut servir d'unités visibles à une autre RBM ayant ses propres unités cachées, dont on peut entraîner les paramètres séparément. En itérant cette procédure, on peut créer une hiérarchie de représentations qui est en fait structurellement similaire à un réseau de neurones.

La RBM peut ainsi servir de pré-entraînement à un modèle supervisé de structure similaire. On peut en effet considérer les paramètres d'une RBM à n couches entraînées par la procédure sus-mentionnée comme étant les paramètres initiaux de

n couches d'un réseau de neurones dont l'activation est une sigmoide. En rajoutant une couche purement supervisée dont la sortie correspond à une valeur sur laquelle on veut faire de la régression, ou une classe pour fins de classification, on peut utiliser des techniques standard de descente de gradient sur un coût de régression ou de classification. Il a été montré que des résultats supérieurs peuvent être obtenus en initialisant les paramètres d'un réseau de neurones à n couches avec les paramètres d'un modèle non-supervisé tel une RBM plutôt qu'aléatoirement (Hinton et al., 2006; Larochelle et al., 2007).

CHAPITRE 3

ÉCHANTILLONNAGE

Les techniques énumérées au fil de la section précédente dépendent de manière critique de l'obtention d'une certaine quantité d'échantillons provenant des modèles décrits et donc du choix d'une méthode d'échantillonnage. Les techniques d'échantillonnage que nous verrons appartiennent toutes à une catégorie assez vaste, c'est à dire les techniques de chaînes de Markov Monte Carlo, que l'on abbréviera MCMC pour *Markov Chain Monte Carlo*. Les chaînes de Markov et une technique de MCMC, l'échantillonnage de Gibbs, seront présentées dans les sections qui suivent. Pour plus d'informations nous référons le lecteur à Hernández-Lerma et Lasserre (2003) et Andrieu et al. (2003).

3.1 Chaînes de Markov

Soit un système dynamique stochastique défini par une séquence de variables aléatoires $X_1, X_2, ...X_t$, pour x_i dans un domaine quelconque Ω. On dira de ce système qu'il est une *chaîne de Markov* s'il a la propriété de Markov, c'est à dire que :

$$P(X_t = x_t | X_1 = x_1, X_2 = x_2, ...X_{t-1} = x_{t-1}) \quad = \quad P(X_t = x_t | X_{t-1} = x_{t-1}) \tag{3.1}$$

Ou, de manière équivalente :

$$X_t | X_{t-1} \quad \perp \quad X_1, X_2, ...X_{t-2} \tag{3.2}$$

C'est à dire que le t-ième état de la chaîne ne dépend que de l'état qui le précède. On peut considérer une restriction supplémentaire à la chaîne de Markov qui en

fait une chaîne **homogène**, c'est à dire sans dépendance temporelle :

$$P(X_i = x_i | X_{i-1} = x_{i-1}) \quad = \quad P(X_j = x_j | X_{j-1} = x_{j-1}) \forall i, j \tag{3.3}$$

Cela signifie que tout état dépend de l'état qui le précède suivant la même distribution de probabilités pour tous les points dans la chaîne, par contraste à une chaîne de Markov où chaque état serait par exemple calculé de manière unique par rapport à l'état précédent (induisant une dépendance à la valeur l'indice temporel t).

3.1.1 Matrice de transition

Dans le cas où l'on a une chaîne de Markov dont le nombre d'états pouvant être pris par les variables aléatoires X_i est dénombrable, on peut définir sur cet espace d'états des matrices de transition $\mathbf{T}_n$ où $\mathbf{T}_{nij}$ correspond à $P(X_n = j | X_{n-1} = i)$ (chaque rangée de la matrice $\mathbf{T}_n$ sommant donc à 1).

Si l'on pose :

$$R_{ij} = T_{n-1,ij} \quad = \quad P(X_{n-1} = j | X_{n-2} = i) \tag{3.4}$$

$$S_{ij} = T_{n,ij} \quad = \quad P(X_n = j | X_{n-1} = i) \tag{3.5}$$

Il s'ensuit que :

$$P(X_n = j | X_{n-2} = i) \quad = \quad \sum_k P(X_n = j, X_{n-1} = k | X_{n-2} = i) \tag{3.6}$$

$$= \quad \sum_k P(X_{n-1} = k | X_{n-2} = i) P(X_n = j | X_{n-1} = k) \tag{3.7}$$

$$= \quad \sum_k R_{ik} S_{kj} \tag{3.8}$$

$$= \quad (\mathbf{RS})_{ij} \tag{3.9}$$

Ce qui se généralise par simple induction à l'équation suivante :

$$P(X_n = j | X_0 = i) \quad = \quad \left(\prod_k \mathbf{T}_k \right)_{ij} \tag{3.10}$$

Si la chaîne de Markov est homogène, alors $\mathbf{T}_i = \mathbf{T}_j = \mathbf{T}\ \forall i, j$. Il s'ensuit que la matrice de transition régissant la transition de X_0 à X_n est $\mathbf{T}^n$.

3.1.2 Ergodicité

Une chaîne de Markov est dite *ergodique* si et seulement si il existe un entier k tel que $P(X_n = i) > 0\ \forall n > k\ et\ \forall i \in \Omega$ et ce peu importe la valeur de l'état initial x_0. Pour une chaîne de Markov ayant un nombre d'états fini et homogène, cela est équivalent à dire que $\mathbf{T}^n > 0\ \forall n > k$.

Intuitivement, une chaîne de Markov ergodique est un processus stochastique tel que chaque état peut être atteint éventuellement à partir de n'importe quel autre état avec une probabilité non-nulle et ce de manière apériodique, c'est à dire que pour n'importe quels états de départ et d'arrivée i et j, il existe un entier k tel que j peut être atteint en partant de i en k étapes ou en $k+1$ étapes. Cette définition exclut donc toute chaîne de Markov comportant un ou plusieurs états pour lesquels toute transition vers un état de leur complément (non-vide) a une probabilité nulle. Elle exclut également une chaîne de Markov à deux états alternant entre les états 0 et 1 avec probabilité 1 : l'état 0, par exemple, s'il était également l'état initial, ne pourrait exister qu'à toutes les étapes de rang pair, alors que l'ergodicité nécessite la possibilité qu'il arrive à n'importe quelle étape à partir d'un certain temps.

3.1.3 Distribution d'équilibre

La **distribution d'équilibre** d'une chaîne de Markov dont la matrice de transition est $\mathbf{T}$ est un vecteur π tel que $\pi\mathbf{T} = \pi$. Le vecteur de probabilités π est donc le vecteur propre de $\mathbf{T}$ de valeur propre 1.

Si une chaîne de Markov est ergodique, π existe et il est unique. De plus,

$(\lim_{n\to\infty} \mathbf{T}^n)_{ij} = \pi_j$, c'est à dire que l'application de la dynamique de la chaîne de Markov, pour un nombre d'étapes suffisamment grand, tendra à produire l'état j avec probabilité π_j. Ces résultats peuvent être démontrés en utilisant le théorème de Perron-Frobenius (Minc, 1988) sur la matrice stochastique $\mathbf{T}$ lorsqu'elle est positive.

Notons bien que pour une initialisation donnée d'une chaîne de Markov ergodique, la séquence d'états au début du processus n'est pas nécessairement représentative de la distribution d'équilibre, à plus forte raison si l'état initial est extrêmement improbable — et l'improbabilité de l'état initial est d'autant plus attendue que le nombre d'états possible est grand. Il est donc à-propos d'itérer pour un certain temps avant d'être raisonnablement certain d'obtenir un échantillon de la distribution π, une période dite de *burn-in*. Ceci dit, si l'état initial est le moindrement probable, la chaîne de Markov y reviendra éventuellement et donc on s'attend à ce que le *burn-in* se répète périodiquement — en fait, chaque état a un *burn-in* qui lui est associé. La signification du terme doit en fait être interprétée comme le laps de temps nécessaire pour que la chaîne de Markov "oublie" l'état initial (s'affranchisse de la "sélection" qui a été faite au départ) de sorte que l'échantillon tiré soit véritablement indépendant.

3.1.4 Mixage

Le **mixage** est une propriété qui, sans avoir de définition formelle, est une quantification de la *vitesse* à laquelle une chaîne de Markov est capable de visiter les modes correspondants à sa distribution d'équilibre. Informellement, le mixage d'une chaîne de Markov pourrait être défini comme le nombre minimal d'échantillons successifs qu'il est nécessaire de tirer d'une chaîne de Markov pour que la distribution de ces échantillons corresponde à sa distribution d'équilibre avec une incertitude donnée (autrement dit, le mixage est l'espérance du *burn-in*). C'est donc que pour n'importe quelle statistique g, le mixage est la vitesse à laquelle la moyenne empirique de cette statistique sur les échantillons tirés se rapproche de son espérance. La figure 3.1 illustre le concept de mixage pour certaines matrices

de transition.

Posons la matrice de transition $\mathbf{T}$ suivante :

$$\mathbf{T} = \begin{pmatrix} 0.999 & 0.001 \\ 0.001 & 0.999 \end{pmatrix} \tag{3.11}$$

La distribution d'équilibre correspondante est $\pi = (0.5, 0.5)$, c'est à dire que si l'on itère la chaîne de Markov, disons, un milliard de fois, on s'attend à obtenir l'état 1 à peu près aussi souvent que l'état 2, ce qui n'est pas surprenant étant donné la symétrie de la matrice de transition.

Toutefois, si à une étape donnée nous sommes à l'état 1, l'espérance du nombre d'itérations nécessaires avant d'effectuer une transition vers l'état 2 est $0.001^{-1} = 1000$. Si l'on regarde la séquence $X_0, X_1, X_2...$ des observations successives sur cette chaîne, elle sera très probablement composée de longues séquences où l'état 1 transitionne vers lui-même pour environ un millier d'itérations, puis l'état 2 pour tout aussi longtemps, et ainsi de suite. Cela signifie qu'une séquence de dix ou cent échantillons sera toujours insuffisante pour discerner clairement la distribution d'équilibre. Ce n'est donc qu'avec plusieurs milliers d'échantillons que l'équiprobabilité des deux états pourra être vue, ce qui est un indice de mauvais mixage. Par opposition, avec une chaîne de Markov où chaque entrée serait équiprobable, de courtes séquences suffiraient et on dirait donc que cette chaîne mixe très bien.

Le mixage pour des chaînes non-triviales est souvent difficile à évaluer, mais nous présenterons des techniques permettant de l'estimer.

3.2 Échantillonnage de Gibbs

Les techniques dites de *Markov Chain Monte Carlo* visent à construire une chaîne de Markov dont la distribution d'équilibre est la distribution recherchée. Il devient donc possible d'obtenir des échantillons de la distribution en itérant la chaîne pour un nombre suffisant d'étapes. L'échantillonnage de Gibbs (Geman et Geman, 1984) est l'une de ces techniques.

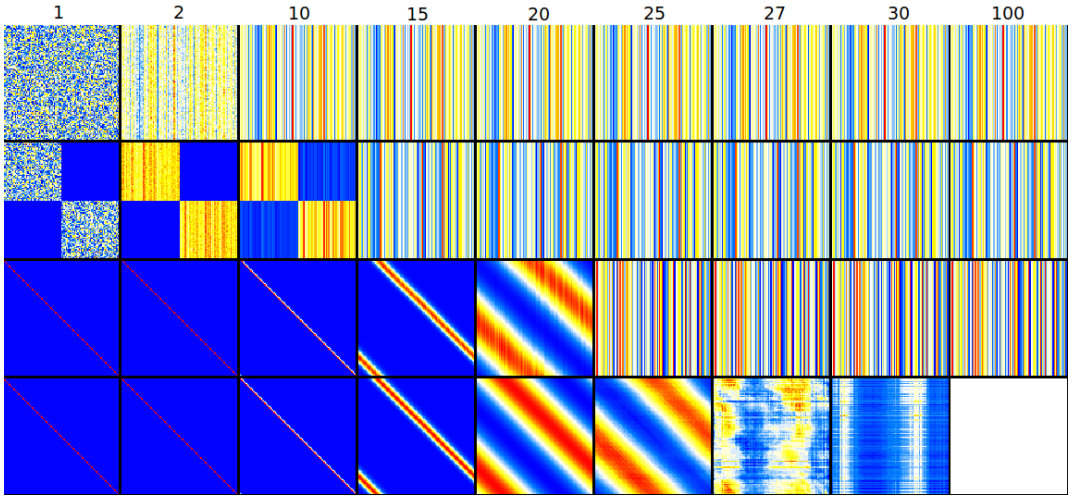


FIG. 3.1 – Visualisation du mixage de certaines chaînes de Markov. Chaque rangée représente une matrice de transition $\mathbf{T}$ différente. La colonne étiquetée i, pour une rangée donnée, représente $\mathbf{T}^{2^{i-1}}$ (donc la dernière colonne est la matrice de transition mise à la puissance 2^{99}). Chaque image est normalisée de sorte que sa plus petite valeur soit bleue, sa plus grande valeur, rouge, sa valeur médiane, blanche. La distribution d'équilibre est atteinte lorsque la matrice de transition n'est que lignes verticales. Pour la première rangée, $\mathbf{T}$ est aléatoire et mixe rapidement. Pour la deuxième rangée, $\mathbf{T}$ est séparée en quatre quadrants, deux ayant une très faible probabilité — le mixage est considérablement plus lent. Pour la troisième rangée, $T_{i,i} \gg T_{i,i+1}$ et $T_{ij} = 0\ \forall i > j$ et $\forall |i-j| > 1$ — le mixage est encore plus lent. De même pour la quatrième rangée, mais sa distribution d'équilibre est équiprobable. Notons que la différence entre bleu et rouge peut être extrêmement petite : les dernières images de la quatrième rangée représentent un intervalle $< 10^{-6}$!

Étant donné un ensemble de variables aléatoires $Z = \{Z_1, Z_2, ...Z_n\}$ (qui peuvent avoir des distributions arbitraires), un ensemble $\mathcal{K}$ d'ensembles d'indices correspondants à ces variables (par exemple, une partition des indices, mais pas nécessairement) et une méthode efficace permettant le calcul des probabilités conditionnelles $\mathcal{P}(Z|Z_K), K \in \mathcal{K}$, l'échantillonnage de Gibbs vise à produire des échantillons provenant de la distribution jointe $\mathcal{P}(Z)$, en supposant que cette distribution est difficile et/ou particulièrement coûteuse à estimer.

L'échantillonnage de Gibbs définit une chaîne de Markov dont la matrice de

transition $\mathbf{T}$ peut s'exprimer en fonction de probabilités conditionnelles — en supposant que ces probabilités sont faciles à calculer, $\mathbf{T}$ l'est également. On dénote $\mathbf{z}^{(i)}$ la i-ème assignation possible que peut prendre la variable aléatoire Z et $\mathbf{z}_K^{(i)}$ le sous-vecteur de $\mathbf{z}$ correspondant à l'ensemble d'indices K, c'est à dire $\mathbf{z}_K^{(i)} = (z_{K_1}^{(i)}, z_{K_2}^{(i)}, ...)$.

$$T_{ij} = \frac{1}{|\mathcal{K}|} \sum_{K \in \mathcal{K}} \mathcal{P}(Z = \mathbf{z}^{(j)} | Z_K = \mathbf{z}_K^{(i)}) \tag{3.12}$$

La matrice de transition comporte une ligne pour chaque valeur possible de Z où chaque entrée définit la probabilité de passer à chaque autre valeur possible. Si l'on considère Z comme un ensemble de dix variables aléatoires binaires, $\mathbf{T}$ est donc une matrice de dimensions $2^{10} \times 2^{10} = 1024 \times 1024$. Cette taille importe toutefois peu puisqu'il est inutile de calculer $\mathbf{T}$ en entier : en supposant qu'on soit dans l'état $\mathbf{z}^{(j)}$, la prochaine itération de la chaîne peut être effectuée en choisissant $K \in \mathcal{K}$ équiprobablement[1] et en tirant un échantillon de $\mathcal{P}(Z|Z_K = x_K^{(j)})$ (opération que l'on suppose efficace). Notons que le K choisi représente l'ensemble des éléments de l'état courant qui ne changeront pas à la prochaine itération de la chaîne : le prochain état est en effet choisi parmi tous les états possibles qui ont ces éléments en commun. Il est souvent facile de calculer la probabilité conditionnelle d'un seul élément de Z étant donnés tous les autres et dans ce cas, il est courant de choisir $\mathcal{K}$ tel que chaque $K \in \mathcal{K}$ est l'ensemble de tous les indices sauf un. De cette manière, à chaque itération, une seule valeur peut changer.

On peut aisément vérifier que la chaîne définie plus haut a pour distribution stationnaire la distribution jointe $\mathcal{P}(Z)$. En effet, si la distribution stationnaire π de la chaîne est définie comme le vecteur propre de la matrice de transition $\mathbf{T}$ qui correspond à la valeur propre 1 et donc satisfait l'équation :

$$\pi_j = \sum_i \pi_i T_{ij} \tag{3.13}$$

Et en définissant $p_j = \mathcal{P}(Z = \mathbf{z}^{(j)})$, la probabilité jointe de la configuration $\mathbf{z}^{(j)}$

[1] En pratique, il est courant de simplement itérer sur les K de manière cyclique.

sous la distribution que l'on désire approximer par Gibbs, on peut tout simplement vérifier que le vecteur propre π de la matrice de transition $\mathbf{T}$ est égal au vecteur $\mathbf{p}$:

$$\begin{aligned}
\pi_j &= \sum_i \pi_i T_{ij} && (3.14)\\
&= \sum_i \pi_i \frac{1}{|\mathcal{K}|} \sum_{K\in\mathcal{K}} \mathcal{P}(Z = \mathbf{z}^{(j)} | Z_K = \mathbf{z}_K^{(i)}) && (3.15)\\
&= \frac{1}{|\mathcal{K}|} \sum_{K\in\mathcal{K}} \sum_i \pi_i \mathcal{P}(Z = \mathbf{z}^{(j)} | Z_K = \mathbf{z}_K^{(i)}) && (3.16)\\
&= \frac{1}{|\mathcal{K}|} \sum_{K\in\mathcal{K}} \sum_i \frac{\pi_i \mathcal{P}(Z_K = \mathbf{z}_K^{(i)} | Z = \mathbf{z}^{(j)}) \mathcal{P}(Z = \mathbf{z}^{(j)})}{\mathcal{P}(Z_K = \mathbf{z}_K^{(i)})} && (3.17)\\
&= \mathcal{P}(Z = \mathbf{z}^{(j)}) \frac{1}{|\mathcal{K}|} \sum_{K\in\mathcal{K}} \sum_i \frac{\pi_i \mathbb{I}[\mathbf{z}_K^{(i)} = \mathbf{z}_K^{(j)}]}{\mathcal{P}(Z_K = \mathbf{z}_K^{(i)})} && (3.18)\\
&= p_j \frac{1}{|\mathcal{K}|} \sum_{K\in\mathcal{K}} \frac{\sum_i \pi_i \mathbb{I}[\mathbf{z}_K^{(i)} = \mathbf{z}_K^{(j)}]}{\mathcal{P}(Z_K = \mathbf{z}_K^{(j)})} && (3.19)\\
p_j + \epsilon_j &= p_j \frac{1}{|\mathcal{K}|} \sum_{K\in\mathcal{K}} \frac{\sum_i (p_i + \epsilon_i) \mathbb{I}[\mathbf{z}_K^{(i)} = \mathbf{z}_K^{(j)}]}{\mathcal{P}(Z_K = \mathbf{z}_K^{(j)})} && (3.20)\\
p_j &= p_j - \epsilon_j + \frac{1}{|\mathcal{K}|} \sum_{K\in\mathcal{K}} \frac{\sum_i \epsilon_i \mathbb{I}[\mathbf{z}_K^{(i)} = \mathbf{z}_K^{(j)}]}{\mathcal{P}(Z_K = \mathbf{z}_K^{(j)})} && (3.21)
\end{aligned}$$

Où $\pi_j - p_j = \epsilon_j$ et la dernière étape est obtenue en utilisant le fait que $\sum_i p_i \mathbb{I}[\mathbf{z}_K^{(i)} = \mathbf{z}_K^{(j)}] = \mathcal{P}(Z_K = \mathbf{z}_K^{(j)})$. Il existe une solution évidente et consistante lorsque $\epsilon = 0$ et donc $\pi_j = p_j$, la distribution d'équilibre étant exactement la distribution cible. En supposant que $\mathbf{T} > 0$, la chaîne est ergodique et on sait que cette solution est la seule qui existe et que l'itération de la chaîne y converge.

3.2.1 Limitations

L'échantillonnage de Gibbs a quelques limitations connues. La plus évidente se produit lorsque les probabilités conditionnelles utilisées comme base à la définition de la chaîne de Markov peuvent être nulles (notons que la preuve de convergence à

l'équation 3.21 suppose que ce n'est pas le cas). Posons par exemple un problème à deux variables binaires $Y = (y_1, y_2)$ tel que $P((0,0)) = P((1,1)) = 0.5$. Si l'on partitionne selon les deux variables, c'est à dire en prenant $\mathcal{K} = \{\{1\}, \{2\}\}$, on remarque que la probabilité de transition du mode $(0,0)$ vers lui-même est égal à $\frac{1}{2}(P(y = (0,0)|y_1 = 0) + P(y = (0,0)|y_2 = 0) = \frac{1}{2}(1+1) = 1$. Cela implique trivialement que toute transition vers un autre état arrive avec probabilité nulle et donc qu'il est impossible d'en sortir lorsqu'on s'y trouve. De même pour le mode $(1,1)$. Il est donc évident que la chaîne de Markov ne visitera jamais les deux modes de la distribution à la fois, puisque la probabilité de transition hors de chacun d'eux est nulle. Dans cette situation, la chaîne n'est pas ergodique et l'échantillonnage de Gibbs ne fonctionne pas.

Une version plus plausible et moins extrême de ce problème est que la chaîne de Markov définie par l'échantillonnage de Gibbs démontre souvent un pauvre mixage. Un exemple simple où cet effet peut se produire serait un cas similaire à l'exemple précédent mais où $P((0,0)) = P((1,1)) = 0.498$ et $P((0,1)) = P((1,0)) = 0.001$. Les règles de l'échantillonnage de Gibbs font en sorte que d'une itération à l'autre, soit y_1 garde sa valeur, soit y_2 reste identique. Il est donc impossible de transitionner directement de l'état $(0,0)$ à l'état $(1,1)$ et vice-versa. Or, la probabilité de transition de l'état $(0,0)$ vers l'état $(0,1)$ est $\frac{1}{2}(P(y = (0,1)|y_1 = 0) + P(y = (0,1)|y_2 = 0) = \frac{1}{2}(\frac{0.001}{0.498+0.001} + 0) \approx 0.001$ et de même pour aller vers $(1,0)$. Si l'on veut passer de l'état $(0,0)$ à l'état $(1,1)$, deux chemins sont possibles : $(0,0) \rightarrow (0,1) \rightarrow (1,1)$ (probabilité d'approximativement 0.001×0.5) ou bien $(0,0) \rightarrow (1,0) \rightarrow (1,1)$, même probabilité. Il s'ensuit que l'espérance du nombre d'itérations nécessaires pour passer de l'un des deux modes vers l'autre est environ 1000. La distribution d'équilibre est correcte, mais le mixage est aussi mauvais que pour la chaîne de Markov de l'équation 3.11.

Un mauvais mixage peut également se produire lorsque la distribution cible comporte un seul mode ayant une probabilité significative, le reste de la masse de probabilités étant répartie uniformément parmi les autres configurations. Supposons que $Y = (y_1, y_2)$ tel que $0 \leq y_i \leq 1000$ (il y a donc $1000 \times 1000 = 10^6$ états

possibles), $P((0,0)) = 0.5$ et $P((i,j)) = \frac{1}{2(10^6-1)}$ pour $(i,j) \neq (0,0)$. La probabilité de transition de l'état $(0,0)$ vers lui-même est $\frac{1}{2}(P(y = (0,0)|y_1 = 0) + P(y = (0,0)|y_2 = 0) \approx \frac{1}{2}(0.999 + 0.999) = 0.999$, tandis que la probabilité de transition de n'importe quel autre état vers $(0,0)$ requiert d'atteindre un état de la forme $(i,0)$ ou $(0,i)$, ce qui prendra en moyenne un millier d'itérations de la chaîne. Par conséquent, dans cette situation, l'échantillonnage de Gibbs va alterner entre de longues séquences de l'état $(0,0)$ et de longues séquences d'états autres. Il y a bien distribution d'équilibre car il existe un chemin de n'importe quel état à n'importe quel autre de manière apériodique, mais la probabilité de l'état $(0,0)$ ne peut être correctement estimée par échantillonnage que moyennant plusieurs milliers d'échantillons.

Il va sans dire que le problème du mixage peut être considérablement amplifié lorsque l'espace d'états est considérable. L'échantillonnage de Gibbs n'est pas très robuste à ce niveau : quelques états pour lesquels la distribution conditionnelle est dégénérée sont autant de "goulots d'étranglement" pouvant détruire le mixage. Si, au lieu de considérer deux variables ayant un millier d'états chacune, on considère plusieurs centaines de variables même binaires, un nombre de plus en plus élevé de ces goulots d'étranglement peuvent se manifester. L'échantionnage de Gibbs pourrait alors vraisemblablement (mais pas nécessairement) rester coincé dans un mode (ou un nombre limité de modes) pour une période de temps si grande qu'elle serait à toutes fins pratique infinie.

3.3 Application de l'échantillonnage de Gibbs à la machine de Boltzmann

L'échantillonnage de Gibbs est la méthode la plus couramment utilisée afin de produire des échantillons de la distribution d'une machine de Boltzmann susceptibles d'être utilisés dans la procédure de mise à jour. La variable aléatoire Z_i, dans ce contexte, a pour distribution de probabilités celle de la i-ème unité de la machine. Afin d'échantillonner de Z, il faudrait, au minimum, calculer la fonction de

partition $\mathcal{Z}$, qui requiert un calcul pour chaque configuration possible. Par contre, étant donné l'état courant $\mathbf{z}$ de la machine de Boltzmann, on peut aisément calculer la probabilité $q_i(\mathbf{z})$ qu'une unité soit active étant donné l'état de toutes les autres unités (un sous-état de $\mathbf{z}$ que nous dénoterons $\mathbf{z}_{\neg i}$ — nous dénoterons de plus $\mathbf{z}_{[z_i=0]}$ l'état $\mathbf{z}$ où z_i est mis à 0 et les autres éléments du vecteur gardent leur valeur et $\mathbf{z}_{[z_i=1]}$ l'état $\mathbf{z}$ où z_i est mis à 1).

$$
\begin{aligned}
q_i(\mathbf{z}) &= P(Z_i = 1 | Z_{\neg i} = \mathbf{z}_{\neg_i}) && (3.22)\\
&= \frac{P(Z_i = 1 \wedge Z_{\neg i} = \mathbf{z}_{\neg_i})}{P(Z_i = 0 \wedge Z_{\neg i} = \mathbf{z}_{\neg_i}) + P(Z_i = 1 \wedge Z_{\neg i} = \mathbf{z}_{\neg i})} && (3.23)\\
&= \frac{P(Z = \mathbf{z}_{[z_i=1]})}{P(Z = \mathbf{z}_{[z_i=0]}) + P(Z = \mathbf{z}_{[z_i=1]})} && (3.24)\\
&= \frac{\exp\left(\frac{-\mathrm{E}(\mathbf{z}_{[z_i=1]})}{T}\right)}{\exp\left(\frac{-\mathrm{E}(\mathbf{z}_{[z_i=0]})}{T}\right) + \exp\left(\frac{-\mathrm{E}(\mathbf{z}_{[z_i=1]})}{T}\right)} && (3.25)\\
&= \frac{\exp\left(\frac{-\mathrm{E}(\mathbf{z}_{[z_i=1]})}{T}\right)}{\exp\left(\frac{-\mathrm{E}(\mathbf{z}_{[z_i=1]})+a_i}{T}\right) + \exp\left(\frac{-\mathrm{E}(z_{[i=1]})}{T}\right)} && (3.26)\\
&= \frac{\exp\left(\frac{-\mathrm{E}(\mathbf{z}_{[z_i=1]})}{T}\right)}{\exp\left(\frac{-\mathrm{E}(\mathbf{z}_{[z_i=1]})}{T}\right)\left(1 + \exp\left(\frac{-a_i}{T}\right)\right)} && (3.27)\\
&= \frac{1}{1 + \exp\left(\frac{-a_i}{T}\right)} && (3.28)\\
&= \mathrm{sigmoid}\left(\frac{a_i}{T}\right) && (3.29)
\end{aligned}
$$

Rappelons que $a_i = \mathrm{E}(\mathbf{z}_{[z_i=1]}) - \mathrm{E}(\mathbf{z}_{[z_i=0]}) = b_i + \sum_i W_{ij} z_j$ où $\mathbf{b}$ et $\mathbf{W}$ sont les paramètres de biais et de poids du modèle.

L'échantillonnage de Gibbs sur une machine de Boltzmann procède donc comme suit : en premier lieu, l'état $\mathbf{z}$ de la machine de Boltzmann est initialisée à un certain état $\mathbf{z}^{(0)}$. On itère ensuite un processus où z_i devient 1 avec probabilité $q_i(\mathbf{z})$ (et 0 sinon) pour i choisi au hasard ou séquentiellement. Après un certain temps passé à atteindre la distribution stationnaire — le pari de la technique étant que ce temps se compare avantageusement au coût exponentiel d'un calcul exact — on obtient

un échantillon de la distribution de Boltzmann.

Le problème principal de cette technique, telle qu'appliquée sur les machines de Boltzmann générales, est que bien qu'elle se compare effectivement avantageusement à toute méthode requiérant le calcul de la fonction de partition (et pour cause, celui-ci devenant complètement infaisable à partir d'un certain nombre d'unités), son mixage est typiquement assez mauvais et donc il faut itérer très longtemps avant d'espérer avoir un échantillon représentatif. Rappelons que cet échantillon, qui assigne des valeurs aux unités visibles et cachées, (voir le terme $(\mathbf{v}^-, \mathbf{h}^-)$ dans l'algorithme 1) est partie intégrante de l'approximation du gradient utilisée dans la mise à jour des paramètres et que la fidélité de cette approximation dépend de la qualité de la couverture que les échantillons font de la véritable distribution pendant l'apprentissage, dont la durée est limitée par des considérations pratiques. La lenteur de l'échantillonnage est également une motivation pratique à tirer moins d'échantillons pour chaque mise à jour des paramètres afin de se déplacer plus rapidement vers une zone intéressante de l'espace paramétrique. Or, la stabilité de l'apprentissage dépend du nombre d'échantillons tirés (moins on en a, plus l'apprentissage est bruité). Il n'est pas trivial de déterminer le compromis idéal, mais il appert que l'apprentissage d'une machine de Boltzmann générale comportant un nombre important d'unités n'est pas viable sans faire appel à des techniques plus élaborées.

Dans cette perspective, un compromis très intéressant est offert par la machine de Boltzmann restreinte. Étant donné que, au sein de la RBM, les unités cachées sont indépendantes entre elles et les unités visibles également, il s'ensuit qu'il est possible d'échantillonner toutes les unités cachées *simultanément* à partir des unités visibles et vice versa. Ainsi, les unités sont partitionnées en deux groupes seulement. ce qui accélère significativement le processus d'échantillonnage et le rend également plus robuste étant donné que les dépendances entre variables sont plus faibles et donc moins susceptibles d'engendrer une dynamique complexe. De plus, l'espérance des unités cachées étant données les unités visibles (calcul qui est nécessaire pour la mise à jour de la machine) peut être calculée exactement et efficacement, ce qui

réduit la marge d'erreur. Pour toutes ces raisons, la RBM est l'un des sous-modèles de la machine de Boltzmann qui connaît le plus de succès (de fait, une performance à l'état de l'art sur plusieurs tâches) et celui sur lequel la recherche récente s'est le plus concentrée. Pour les besoins de ce mémoire, nous adopterons donc la structure de la RBM et évaluerons plusieurs techniques d'échantillonnage par rapport à leur performance sur cette structure.

3.3.1 Divergence contrastive

L'application de l'échantillonnage de Gibbs sur une machine de Boltzmann restreinte admet, en pratique, plusieurs heuristiques concernant notamment l'initialisation, à chaque mise à jour, de l'état de la chaîne de Markov, ainsi que le nombre d'itérations effectuées afin d'obtenir un échantillon. On suppose un certain nombre n de chaînes de Markov qui seront initialisées différemment et itérées en parallèle (on les appelle également "particules négatives"). Le processus produit ainsi n échantillons, sur lesquels on peut calculer l'espérance des statistiques suffisantes de la RBM.

Dans le cadre de la **divergence contrastive** (Hinton, 1999, 2002) (que nous abbrévierons CD pour *Contrastive Divergence*), l'état de chaque chaîne de Markov est initialisé avec un exemple d'entraînement — pour être plus précis, on utilise typiquement les mêmes exemples qui sont utilisés dans la phase positive. On effectue sur chaque chaîne k itérations de Gibbs : les unités cachées sont d'abord échantillonnées à partir des unités visibles suivant la distribution conditionnelle $P(\mathbf{h}|\mathbf{v})$, puis les unités visibles sont ré-échantillonnées à partir de ces unités cachées en utilisant $P(\mathbf{v}|\mathbf{h})$, et ainsi de suite, le paramètre k correspondant au nombre de fois que ce processus de va-et-vient est effectué afin d'obtenir un échantillon des unités visibles. La figure 3.2 montre les calculs requis afin d'obtenir les échantillons positif et négatif pour $k = 1$. $k = 1$ fonctionne convenablement en pratique (Hinton, 2002; Hinton et al., 2006; Larochelle et al., 2007), mais on obtient de meilleurs résultats avec k plus grand au coût d'un entraînement plus lent.

Tout ceci nous permet de définir un algorithme précis afin de faire une mise à

jour — voir l'algorithme 2. On suppose qu'on ne voit qu'un seul exemple d'entraînement à la fois et que la température **T** est toujours égale à 1.

Algorithm 2 Algorithme décrivant la divergence contrastive

`EchantillonCD(x, W, b, c, k)`
Échantillonnage de ($\mathbf{x}_e$, $\mathbf{h}_e$) à partir de l'état courant $\mathbf{x}$ de la chaîne de Markov et des paramètres $\mathbf{W}$, $\mathbf{b}$ et $\mathbf{c}$, pour k itérations de Gibbs. Notons que $\mathbf{h}$ et non $\mathbf{h}_e$ est retourné, puisque, en pratique, la mise à jour marche mieux avec $\mathbf{h}$ qu'avec $\mathbf{h}_e$.

$\mathbf{h} \leftarrow \text{sigmoid}(\mathbf{W}\mathbf{x} + \mathbf{b})$
for i=1 to k **do**
  $\mathbf{h}_e \sim \mathbf{h}$ # Note : ceci veut dire que $h_{e_j} \leftarrow 1$ avec probabilité h_j
  $\mathbf{x} \leftarrow \text{sigmoid}(\mathbf{W}'\mathbf{h}_e + \mathbf{c})$
  $\mathbf{x}_e \sim \mathbf{x}$
  $\mathbf{h} \leftarrow \text{sigmoid}(\mathbf{W}\mathbf{x}_e + \mathbf{b})$
end for
return $\mathbf{x}_e$, $\mathbf{h}$

`MiseAJourCD(x, θ, ϵ, k)`
Mise à jour des paramètres θ d'une RBM avec un exemple $\mathbf{x}$ observé, k itérations de Gibbs et un pas d'apprentissage ϵ.

$(\mathbf{W}, \mathbf{b}, \mathbf{c}) \leftarrow \theta$
$\mathbf{h} \leftarrow \text{sigmoid}(\mathbf{W}\mathbf{x} + \mathbf{b})$
$(\mathbf{x}^-, \mathbf{h}^-) \leftarrow \text{EchantillonCD}(\mathbf{x}, \mathbf{W}, \mathbf{b}, \mathbf{c}, k)$
$\mathbf{W} \leftarrow \mathbf{W} + \epsilon(\mathbf{x}'\mathbf{h} - \mathbf{x}^{-'}\mathbf{h}^-)$
$\mathbf{b} \leftarrow \mathbf{b} + \epsilon(\mathbf{h} - \mathbf{h}^-)$
$\mathbf{c} \leftarrow \mathbf{c} + \epsilon(\mathbf{x} - \mathbf{x}^-)$
$\theta \leftarrow (\mathbf{W}, \mathbf{b}, \mathbf{c})$
return θ

3.3.2 Divergence contrastive persistante

La **divergence contrastive persistante** (Tieleman, 2008) (que nous abbrévierons PCD pour *Persistent Contrastive Divergence*) est une variation de l'algorithme présenté plus haut où l'initialisation des n chaînes de Markov se fait différemment : en fait, l'état de chacune de ces chaînes est *persistant*, ce qui signifie que les échantillons tirés après chaque mise à jour servent d'initialisation aux chaînes pour la mise à jour suivante. On peut concevoir ce mécanisme comme une chaîne de Markov itérée continuellement, mais périodiquement "interrompue" par une mise à jour

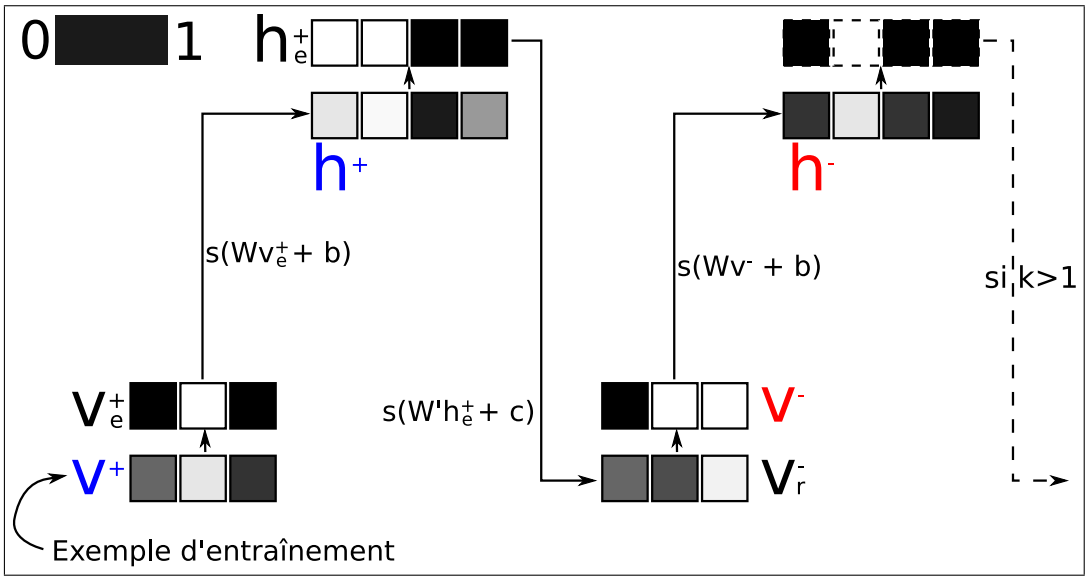


FIG. 3.2 – Visualisation de la CD lorsque $k = 1$. $s(\cdot)$ est la fonction sigmoide. On commence par calculer les unités cachées à partir des unités visibles, puis on recalcule les unités visibles à partir des unités cachées, et ainsi de suite. À chaque étape, des valeurs continues sont produites par la sigmoide mais sont binarisées par échantillonnage pour l'étape suivante. Lorsque présents, l'indice e indique un vecteur échantillonné et l'indice r indique un vecteur représentation (continu). $\mathbf{v}^+$ et $\mathbf{h}^+$, en bleu, sont les composantes de l'état utilisé pour la phase positive, tandis que $\mathbf{v}^-$ et $\mathbf{h}^-$, en rouge, sont les composantes de l'état utilisé pour la phase négative. On peut imaginer choisir ces états différemment, mais les choix montrés par la figure 3.2 sont ceux qui marchent le mieux expérimentalement. Si $k > 1$, il suffit de répéter la figure vers la droite de sorte à ce qu'il y ait k flèches descendantes.

du modèle et donc de la matrice de transition. Voir l'algorithme 3 et la figure 3.3 pour davantage de précisions.

Le principe derrière cette variation est que en autant que les mises à jour du modèle sous-jacent à la machine de Boltzmann soient faibles et progressives, le processus d'échantillonnage est en mesure de "suivre" les changements dans la surface de probabilité à mesure qu'ils se produisent. En d'autres mots, étant donnée la distribution stationnaire d'une machine de Boltzmann à un temps t et la distribution stationnaire de cette même machine de Boltzmann après une mise à jour, c'est à dire au temps $t + 1$, il est raisonnable de penser que ces deux distributions

sont très similaires (il est en fait possible de les rendre arbitrairement similaires en diminuant le pas, c'est à dire le facteur par lequel la mise à jour est multipliée). Par conséquent, un échantillon provenant de la première peut être considéré comme un échantillon assez représentatif de la deuxième et donc un bon point de départ pour une chaîne de Markov qui, après un nombre limité d'itérations, deviendra plus fidèle à la nouvelle distribution et un bon point de départ en vue de la prochaine mise à jour (voir analyse par Younes (1998)).

L'avantage de la divergence contrastive persistante est que les échantillons, en n'étant pas "calqués" sur les exemples d'apprentissage qui servent à initialiser des chaînes de Markov dans la version originale de l'algorithme, représentent mieux la véritable distribution de la machine de Boltzmann sur l'espace de tous les états plutôt que sur l'espace des états proches des exemples d'apprentissage. En effet, si l'on considère que le *burn-in* d'une chaîne de Markov est le nombre d'étapes nécessaires pour que son état devienne indépendant de l'état initial, la CD-k est nécessairement biaisée si le *burn-in* est supérieur à k — les particules négatives sont alors restreintes à un "voisinage" autour des exemples d'entraînement et le comportement du modèle hors de ce voisinage n'est pas contrôlé. Un exposé empirique sur le biais induit par la CD-k et dans quelle mesure les points fixes de la CD-k sont différents de ceux du véritable gradient est donné dans Carreira-Perpiñan et Hinton (2005) — la conclusion étant que le biais est existant mais faible dans leurs expériences. La PCD apporte une solution à ce problème en permettant aux chaînes de Markov de parcourir plus librement l'espace d'états.

L'inconvénient de la PCD, cependant, est que l'échantillonnage peut vraisemblablement rester coincé dans un recoin de l'espace des états pour suffisamment de temps pour dérailler l'apprentissage (un cas de mauvais mixage). En pratique, les avantages prennent généralement le pas sur les inconvénients et des résultats supérieurs peuvent être obtenus sur certains jeux de données grâce à la persistance (Tieleman, 2008).

Algorithm 3 Algorithme décrivant la divergence contrastive persistante.

`MiseAJourPCD(`$\mathbf{x}$`,` θ`,` ϵ`,` k`,` $\mathbf{x}^-$`)`

Mise à jour des paramètres θ d'une RBM avec un exemple $\mathbf{x}$ observé, k itérations de Gibbs, un pas d'apprentissage ϵ ainsi que l'état courant $\mathbf{x}^-$ de la chaîne de Markov. La seule différence entre la CD et la PCD est que $\mathbf{x}^-$ est maintenant un état persistant du système, modifié à chaque mise à jour, tout comme θ. Voir l'algorithme 2 pour la définition de la procédure EchantillonCD.

$(\mathbf{W}, \mathbf{b}, \mathbf{c}) \leftarrow \theta$
$\mathbf{h} \leftarrow \text{sigmoid}(\mathbf{W}\mathbf{x} + \mathbf{b})$
$(\mathbf{x}^-, \mathbf{h}^-) \leftarrow \text{EchantillonCD}(\mathbf{x}^-, \mathbf{W}, \mathbf{b}, \mathbf{c}, k)$
$\mathbf{W} \leftarrow \mathbf{W} + \epsilon(\mathbf{x}'\mathbf{h} - \mathbf{x}^{-\prime}\mathbf{h}^-)$
$\mathbf{b} \leftarrow \mathbf{b} + \epsilon(\mathbf{h} - \mathbf{h}^-)$
$\mathbf{c} \leftarrow \mathbf{c} + \epsilon(\mathbf{x} - \mathbf{x}^-)$
$\theta \leftarrow (\mathbf{W}, \mathbf{b}, \mathbf{c})$
return θ, $\mathbf{x}^-$

3.3.3 Divergence contrastive persistante rapide

La **divergence contrastive persistante rapide** (Tieleman et Hinton, 2009) (que nous abbrévierons FPCD pour *Fast Persistent Contrastive Divergence*) est une variation sur le thème de la PCD qui utilise deux ensembles de paramètres. D'une part, les paramètres "normaux" de la RBM $\theta = (\mathbf{W}, \mathbf{b}, \mathbf{c})$. D'autre part, un nouvel ensemble de paramètres dits "rapides" $\theta_F = (\mathbf{W}_F, \mathbf{b}_F, \mathbf{c}_F)$. Les paramètres utilisés dans le calcul de la phase positive du gradient sont θ, alors que les paramètres utilisés dans le calcul de la phase négative sont la somme $\theta + \theta_F$. θ_F est mis à jour de la même manière que θ, à la différence près que le taux d'apprentissage pour θ_F est potentiellement différent et que θ_F est multiplié par un facteur $\alpha \in [0, 1]$. L'algorithme 4 décrit l'algorithme de manière plus précise.

Le principe de la FPCD part de l'observation que l'apprentissage aide au mixage, dans la mesure où si, dans la phase négative de chaque mise à jour, on pénalise un terme relié à l'échantillon qu'on vient de tirer, la masse de probabilité associée à celui-ci sera plus faible et donc il est peu probable que la chaîne de Markov de la PCD demeure à cet endroit.

La FPCD exacerbe cet effet en utilisant un modèle différent pour les phases

Algorithm 4 Algorithme décrivant la divergence contrastive persistante rapide.

`MiseAJourFPCD(`$\mathbf{x}$`,` θ`,` θ_F`,` ϵ`,` ϵ_F`,` α`,` k`,` $\mathbf{x}^-$`)`

Mise à jour des paramètres normaux et rapides θ et θ_F d'une RBM avec un exemple x observé, k itérations de Gibbs, des pas d'apprentissage ϵ et ϵ_F pour les deux groupes de paramètres, un facteur α de réduction des paramètres rapides ainsi que l'état courant $\mathbf{x}^-$ de la chaîne de Markov.

Notons que $\mathbf{h}$ et $\mathbf{h}^-$ sont calculés à partir de paramètres différents et que θ_F est poussé vers 0 par le multiplicateur α, $0 < \alpha < 1$.

Voir l'algorithme 2 pour la définition de la procédure EchantillonCD.

$(\mathbf{W}, \mathbf{b}, \mathbf{c}) \leftarrow \theta$
$(\mathbf{W}_F, \mathbf{b}_F, \mathbf{c}_F) \leftarrow \theta_F$
$\mathbf{h} \leftarrow \text{sigmoid}(\mathbf{W}\mathbf{x} + b)$
$(\mathbf{x}^-, \mathbf{h}^-) \leftarrow \text{EchantillonCD}(x, (\mathbf{W} + \mathbf{W}_F), (\mathbf{b} + \mathbf{b}_F), (\mathbf{c} + \mathbf{c}_F), k)$
$\mathbf{W} \leftarrow \mathbf{W} + \epsilon(\mathbf{x}'\mathbf{h} - \mathbf{x}^{-\prime}\mathbf{h}^-)$
$\mathbf{b} \leftarrow \mathbf{b} + \epsilon(\mathbf{h} - \mathbf{h}^-)$
$\mathbf{c} \leftarrow \mathbf{c} + \epsilon(\mathbf{x} - \mathbf{x}^-)$
$\mathbf{W}_F \leftarrow \alpha\mathbf{W}_F + \epsilon_F(\mathbf{x}'\mathbf{h} - \mathbf{x}^{-\prime}\mathbf{h}^-)$
$\mathbf{b}_F \leftarrow \alpha\mathbf{b}_F + \epsilon_F(\mathbf{h} - \mathbf{h}^-)$
$\mathbf{c}_F \leftarrow \alpha\mathbf{c}_F + \epsilon_F(\mathbf{x} - \mathbf{x}^-)$
$\theta \leftarrow (\mathbf{W}, \mathbf{b}, \mathbf{c})$
$\theta_F \leftarrow (\mathbf{W}_F, \mathbf{b}_F, \mathbf{c}_F)$
return θ, θ_F, $\mathbf{x}^-$

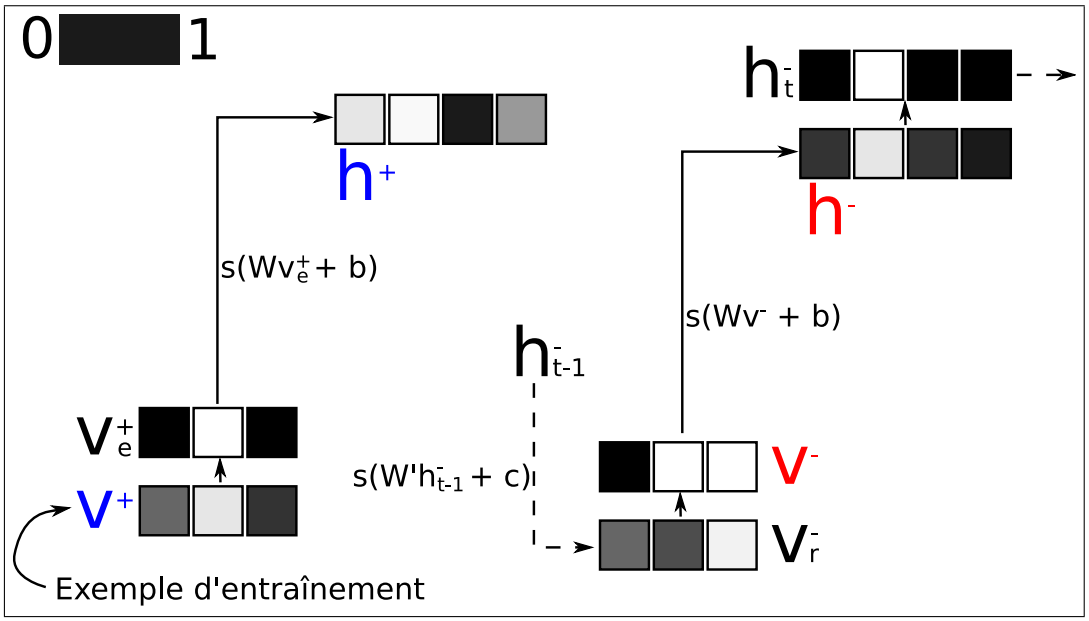


FIG. 3.3 – Visualisation de la t-ième itération de la PCD. On remarque immédiatement la différence avec la CD : le calcul de la particule négative est dissocié du calcul de la particule positive. Il dépend en fait de la particule négative produite à l'étape précédente (note : l'état persistant peut être $\mathbf{v}^-$ comme dans l'algorithme ou $\mathbf{h}^-$ comme sur la figure, cela n'a pas vraiment d'importance). La FPCD utilise le même principe, mais a ceci de particulier que $\mathbf{W}$ et $\mathbf{b}$, dans la partie négative, sont différents. Le *herding* utilise, lui aussi, le même principe, à ceci près que $s(\cdot)$ est remplacé par $\mathbb{I}[\cdot > 0]$ et que la chaîne négative se poursuit vers la droite jusqu'à l'atteinte d'un point fixe.

positive et négative : alors que la phase positive se déroule normalement, la phase négative utilise des paramètres qui répondent plus fortement à l'apprentissage et donc pénalisent beaucoup plus les modes courants, forçant l'échantillonnage à aller voir ailleurs. Typiquement, le pas ϵ_F est plus grand que le pas ϵ et donc θ_F aura tendance à bouger plus rapidement.

Les paramètres rapides θ_F induisent un biais dans l'apprentissage, dans la mesure où le principe de la RBM requiert que les phases positive et négative réfèrent à la même distribution sous-jacente et donc aux mêmes paramètres. Le paramètre α (si $0 < \alpha < 1$), en pénalisant multiplicativement les paramètres θ_F, les empêche

de dévier trop loin de zéro en bornant l'espérance de la différence $||\theta - \theta_F||$. En pratique, la technique fonctionne bien et permet d'améliorer la performance des RBMs. Un aspect où la FPCD a un avantage substantiel sur la PCD est lorsque le pas ϵ est diminué au fil de l'apprentissage, ce qui est nécessaire afin de garantir la convergence des paramètres. En effet, plus ce pas est petit, moins les paramètres changent et il devient plus probable que les échantillons négatifs $\mathbf{x}^-$ produits par la PCD demeurent "coincés" dans un mode local, biaisant le processus d'apprentissage. Avec la FPCD, ϵ peut être diminué (garantissant la convergence) alors que ϵ_F reste constant (permettant à la chaîne de continuer à bien mixer).

3.4 Herding

Le *herding* (Welling, 2009a,b) est un algorithme récent préconisant une approche nouvelle aux RBMs où l'échantillonnage et le modèle sont intrinsèquement reliés, l'un n'ayant aucun sens sans l'autre. Prenons la limite de la probabilité d'un état lorsque la température T tend vers zéro :

$$\lim_{T\to 0} \mathcal{P}(\mathbf{x}) = \lim_{T\to 0} \frac{\exp\left(\frac{-\mathrm{E}(\mathbf{x})}{T}\right)}{\sum_{\mathbf{y}} \exp\left(\frac{-\mathrm{E}(\mathbf{y})}{T}\right)} \tag{3.30}$$

$$= \mathbb{I}[\mathbf{x} = \mathrm{argmin}_{\mathbf{y}}\mathrm{E}(\mathbf{y})] \tag{3.31}$$

En d'autres mots, à température 0, toute la masse de probabilité est concentrée sur un seul point, celui dont l'énergie est minimale (car son exponentielle domine à la limite de $T \to 0$). L'échantillonnage, dans cette situation, revient donc à une simple minimisation de l'énergie. La distribution de probabilités qui est ainsi définie ne présente évidemment pas grand intérêt. Or, si on considère un modèle qui *apprend*, c'est à dire dont les paramètres sont constamment mis à jour, cette distribution change continuellement. On peut alors s'intéresser à la *séquence* des $\mathbf{x}_t^-$ tels que $\mathbf{x}_t^- = \mathrm{argmin}_{\mathbf{y}}\mathrm{E}(\mathbf{y};\theta_t)$ et considérer les propriétés de la distribution définie par cette séquence.

Oublions pour un instant la température du modèle. En abbréviant la contribu-

tion des phases positives et négatives au temps t (qui correspondent respectivement à l'espérance des statistiques suffisantes de l'ensemble d'entraînement et du modèle courant) par $\hat{g}_t^+$ et $\hat{g}_t^-$, les paramètres θ au temps $t+1$ d'une machine de Boltzmann sont définis comme suit :

$$\begin{aligned}
\theta_{t+1} &= \theta_t + \epsilon(\hat{g}_t^+ - \hat{g}_t^-) && (3.32)\\
&= \theta_0 + \epsilon\left(\sum_{i=1}^{t} \hat{g}_t^+ - \hat{g}_i^-\right) && (3.33)\\
&= \theta_0 + \epsilon\sum_{i=1}^{t} \hat{g}_i^+ - \epsilon\sum_{i=1}^{t} \hat{g}_i^- && (3.34)
\end{aligned}$$

Supposons maintenant que $\lim_{t\to\infty} \frac{\theta_t}{t} = 0$. Comme le montre Welling (2009a) :

$$\begin{aligned}
0 &= \lim_{t\to\infty} \frac{\theta_t}{t} && (3.35)\\
&= \lim_{t\to\infty} \frac{1}{t}\left(\theta_0 + \epsilon\sum_{i=1}^{t} \hat{g}_i^+ - \epsilon\sum_{i=1}^{t} \hat{g}_i^-\right) && (3.36)\\
&= \lim_{t\to\infty} \frac{1}{t}\theta_0 + \frac{1}{t}\left(\epsilon\sum_{i=1}^{t} \hat{g}_i^+ - \epsilon\sum_{i=1}^{t} \hat{g}_i^-\right) && (3.37)\\
&= \lim_{t\to\infty} \frac{1}{t}\left(\epsilon\sum_{i=1}^{t} \hat{g}_i^+ - \epsilon\sum_{i=1}^{t} \hat{g}_i^-\right) && (3.38)\\
\lim_{t\to\infty} \frac{1}{t}\sum_{i=1}^{t} \hat{g}_i^+ &= \lim_{t\to\infty} \frac{1}{t}\sum_{i=1}^{t} \hat{g}_i^- && (3.39)\\
g^+ &= g^- && (3.40)
\end{aligned}$$

En d'autres termes, si les paramètres du modèle croissent plus lentement que linéairement, les statistiques suffisantes g^- de la séquence d'échantillons tirés du modèle (à la limite de l'infini) sont identiques aux statistiques suffisantes g^+ des données d'entraînement (et ce peu importe le pas ϵ). Dans une machine de Boltzmann restreinte, ces statistiques suffisantes sont l'espérance de la valeur de chaque unité visible ou cachée ainsi que l'espérance de leur produits deux à deux, mais le résultat se généralise à d'autres types de statistiques.

Welling (2009a) prouve que les paramètres du modèle seront bornés[2] dans un contexte idéalisé où il est possible de trouver, à chaque étape, l'état pour lequel la fonction d'énergie est à son minimum global, ainsi que dans une situation plus réaliste où une méthode est utilisée pour trouver un minimum local et la fonction d'énergie n'est pas dégénérée (plate). Ceci dit, la croissance des paramètres du modèle peut aisément être vérifiée à faible coût, en calculant $||\theta||$ après chaque mise à jour (ou un certain nombre mises à jour).

Il appert que la perspective du *herding* est très différente de ce que nous avons vu jusqu'ici : plutôt que d'apprendre des paramètres précis θ correspondant à la distribution cible, on essaie d'apprendre une dynamique sur les paramètres qui produise une séquence d'échantillons ayant les statistiques voulues.

En prenant la limite $T \to 0$, le résultat général de l'équation 3.40 tient toujours, mais on gagne quelques avantages : d'une part, puisque toute la probabilité est concentrée dans l'état d'énergie minimale, la magnitude de cette énergie n'importe pas et donc $\forall \lambda > 0, \mathcal{P}(x;\theta) = \mathcal{P}(x;\lambda\theta)$. Cela suggère que la magnitude des paramètres initiaux θ_0 ainsi que le pas d'apprentissage ϵ sont superflus — du moins, en observant que $\lambda\theta + \lambda\epsilon z = \lambda(\theta + \epsilon z)$ (où z est la mise à jour des paramètres, qui ne dépend pas de λ), on peut voir que la multiplication des paramètres initiaux et du taux d'apprentissage par un même facteur fait en sorte que la séquence de paramètres produite au cours de l'apprentissage est également multipliée par ce facteur (voir aussi (Welling, 2009a)). D'autre part, le processus d'apprentissage devient déterministe. Finalement, comme nous le verrons à la section suivante, il existe une procédure simple de descente de gradient sur l'énergie qui n'implique que des opérations arithmétiques simples (alors que les techniques exposées précédemment (CD, PCD et FPCD) requierent un certain nombre d'opérations coûteuses telles l'exponentiation).

[2] C'est à dire que $\lim_{t\to\infty} \frac{\theta_t}{t} = 0$

3.4.1 Échantillonnage

À température zéro, nous désirons obtenir, sinon l'état d'énergie minimale, l'état dont l'énergie est la plus basse possible. Un "échantillon" peut donc être obtenu en itérant une procédure de descente de gradient sur l'énergie (supposons ici une machine de Boltzmann générale dont les paramètres sont $\theta = (\mathbf{W}, \mathbf{b})$). L'énergie peut s'exprimer comme suit :

$$\mathrm{E}(\mathbf{s}) = -\sum_i b_i s_i - \frac{1}{2}\sum_{i,j} W_{ij} s_i s_j \tag{3.41}$$

$$\mathrm{E}(\mathbf{s}) = \sum_i s_i \left(-b_i - \frac{1}{2}\sum_j W_{ij} s_j \right) \tag{3.42}$$

$$\tag{3.43}$$

Le gradient par rapport à une unité s_i en particulier est alors :

$$\frac{\partial \mathrm{E}(\mathbf{s})}{\partial s_i} \quad = \quad -b_i - \frac{1}{2}\sum_j W_{ij} s_j \tag{3.44}$$

On remarque que le gradient de s_i ne dépend pas de s_i (note : on suppose que $W_{ii} = 0$ comme pour une machine de Boltzmann) — en fait, il représente la contribution linéaire de cette unité à l'énergie. Il s'ensuit que si le gradient est positif, on veut que s_i soit le plus proche possible de $-\infty$, car chaque fois que 1 est soustrait à s_i, l'énergie descend d'un facteur égal à la valeur du gradient. À l'opposé, si le gradient est négatif, on veut que s_i soit le plus proche possible de $+\infty$. Comme s_i n'a que deux valeurs possibles (0 ou 1) la procédure à suivre est très simple.

Posons $z_i = \mathrm{signe}(-\frac{\partial \mathrm{E}(\mathbf{s})}{\partial s_i})$ le signe inverse du gradient. Si $z_i = -1$, la valeur de s_i qui minimise l'énergie (toutes les autres unités restant constantes) est $s_i = 0$. Donc, dans cette situation, on applique le changement $s_i \leftarrow 0$. Par le même principe, si $z_i = 1$, alors $s_i \leftarrow 1$. Si $z_i = 0$, on en conclut que la valeur de s_i importe peu et on

peut tout simplement décider de ne pas changer s_i. Tel que nous l'avons défini, z_i indique dans quelle direction l'état de la $i$ème unité doit être changé.

En suivant cette procédure pour un s_i quelconque, tout changement doit strictement diminuer l'énergie du système et donc il est impossible de passer deux fois par le même état sauf dans le cas facilement détectable où il n'y a aucun changement. Étant donné que le nombre d'états possibles est fini, il s'ensuit que cette procédure converge nécessairement vers un point fixe où toutes les unités s_i sont satisfaites de leur valeur (et en pratique, cela se produit assez rapidement). Notons bien que ce point fixe n'est pas nécessairement le point d'énergie *minimale*, mais peut très bien n'être qu'un minimum local. Par exemple, la situation où tous les biais b_i et toutes les unités ont valeur zéro est un point fixe peu importe les paramètres W_{ij} du modèle, bien qu'on puisse imaginer que d'autres états aient une énergie moindre, par exemple le cas où tous les paramètres sont négatifs et les unités ont toutes valeur un. Notons que cette procédure est identique à celle qui est utilisée dans un réseau de Hopfield (Hopfield, 1982).

Le *herding* et la procédure d'"échantillonnage" se généralisent trivialement au cas où les unités peuvent prendre les valeurs -1 ou 1 (en fait, la procédure est encore plus simple puisqu'il suffit de faire $s_i \leftarrow z_i$). Ici encore, il existe un certain nombre de minima locaux, bien qu'il soit un peu plus compliqué d'en fournir un bon exemple. En prenant $\mathbf{W} = \begin{pmatrix} 0 & 2 \\ 2 & 0 \end{pmatrix}$ et $\mathbf{b} = \begin{pmatrix} -1 & -1 \end{pmatrix}$, $\frac{\partial \mathrm{E}(\mathbf{s})}{\partial s_i} = 0$ lorsque $\mathbf{s} = \begin{pmatrix} 1 & 1 \end{pmatrix}$. Or, l'énergie à ce point est nulle tandis que pour $\mathbf{s} = \begin{pmatrix} -1 & -1 \end{pmatrix}$, elle est plus basse (négative).

Dans le cas où on utilise un modèle de type RBM, toutes les unités cachées peuvent être mises à jour en même temps à partir des unités visibles et vice versa. Cela est illustré par l'algorithme 5 décrivant le *herding*.

Algorithm 5 Algorithme décrivant le *herding*.

`EchantillonHerd(x, W, b, c)`

Échantillonnage de $(\mathbf{x}_e, \mathbf{h}_e)$ à partir de l'état courant $\mathbf{x}$ de la chaîne de Markov et des paramètres $\mathbf{W}$, $\mathbf{b}$ et $\mathbf{c}$. La boucle while itère jusqu'à ce que $\mathbf{x}$ ne change plus.

$\mathbf{h} \leftarrow \mathbb{I}[\mathbf{W}\mathbf{x} + \mathbf{b} > 0]$
while $\mathbb{I}[\mathbf{W}'\mathbf{h} + \mathbf{c} > 0] \neq \mathbf{x}$ **do**
  $\mathbf{x} \leftarrow \mathbb{I}[\mathbf{W}'\mathbf{h} + \mathbf{c} > 0]$
  $\mathbf{h} \leftarrow \mathbb{I}[\mathbf{W}\mathbf{x} + \mathbf{b} > 0]$
end while
return $\mathbf{x}$, $\mathbf{h}$

`MiseAJourHerd(x, θ, ϵ, x⁻)`

Mise à jour des paramètres θ d'une RBM entraînée par *herding* avec un exemple $\mathbf{x}$ observé, un pas d'apprentissage ϵ (Welling (2009a) utilise $\epsilon = 1$) ainsi que l'état courant $\mathbf{x}^-$.

La seule différence entre la PCD et le *herding* est la manière par laquelle $\mathbf{h}$, $\mathbf{x}^-$ et $\mathbf{h}^-$ sont calculés.

$(\mathbf{W}, \mathbf{b}, \mathbf{c}) \leftarrow \theta$
$\mathbf{h} \leftarrow \mathbb{I}[W\mathbf{x} + \mathbf{b} > 0]$
$(\mathbf{x}^-, \mathbf{h}^-) \leftarrow \text{EchantillonHerd}(\mathbf{x}, \mathbf{W}, \mathbf{b}, \mathbf{c}, k)$
$\mathbf{W} \leftarrow \mathbf{W} + \epsilon(\mathbf{x}'\mathbf{h} - \mathbf{x}^{-\prime}\mathbf{h}^-)$
$\mathbf{b} \leftarrow \mathbf{b} + \epsilon(\mathbf{h} - \mathbf{h}^-)$
$\mathbf{c} \leftarrow \mathbf{c} + \epsilon(\mathbf{x} - \mathbf{x}^-)$
$\theta \leftarrow (\mathbf{W}, \mathbf{b}, \mathbf{c})$
return θ, $\mathbf{x}^-$

CHAPITRE 4

ÉCHANTILLONNAGE DYNAMIQUE

Au cours du chapitre précédent, nous avons défini l'échantillonnage de Gibbs et l'avons placé en contexte dans le cadre de l'apprentissage d'une machine de Boltzmann. Nous avons également défini le concept de mixage, c'est à dire la vitesse à laquelle une chaîne de Markov, dont celles définies par l'échantillonnage de Gibbs, est en mesure de cycler à travers ses différents modes.

Il s'avère en réalité que Gibbs sur une machine de Boltzmann pré-entraînée, en l'absence donc de mises à jour sur ses paramètres, a un très mauvais mixage. L'échantillonnage de Gibbs peut en effet nous fournir des échantillons autour d'un même mode pour des centaines d'itérations de suite — par exemple, si l'on entraîne le modèle à reconnaître des images de chiffres manuscrits, l'échantillonnage pourrait nous fournir une centaine d'images de zéros d'affilée. Cela se produit même dans des situations où d'autres modes sont équiprobables à celui qui est visité (ce qui se peut se vérifier aisément : si l'énergie associée au chiffre 2 est comparable à l'énergie associée au chiffre 0, on devrait observer autant de 0 que de 2, et ainsi de suite pour d'autres modes que l'on connaît *a priori*). Bref, l'échantillonnage de Gibbs ne parvient pas à bien représenter la distribution encodée par la RBM et donc il est difficile d'évaluer à quel point celle-ci est fidèle à la distribution cible, sans compter le fait que même si on pouvait le montrer, on ne pourrait l'exploiter.

A priori, le mauvais mixage semble être un problème majeur étant donné le rôle prépondérant des échantillons dans le calcul de facteurs importants dans l'apprentissage. En effet, un mauvais mixage devrait significativement biaiser les estimés puisque la chaîne de Markov devrait être itérée extrêmement longtemps pour qu'on obtienne un échantillon de la distribution stationnaire plutôt qu'un échantillon "proche" de l'état initial de la chaîne. En fait, le mixage de Gibbs semble si mauvais qu'il ne devrait même pas y avoir apprentissage. Or, malgré tout, l'apprentissage fonctionne, ce qu'on peut constater de diverses manières : en analysant

la transformation appliquée par les paramètres sur l'entrée (ce qui peut se faire visuellement si l'entrée est une image, car le paramètre de poids peut être vu comme un ensemble de filtres sur les images fournies en entrée), ou en utilisant le modèle entraîné comme initialisation à un réseau de neurone ordinaire et en constatant que le résultat d'un apprentissage supervisé sur une certaine tâche est meilleur que si l'initialisation du réseau avait été aléatoire (donc du moins, quelque chose d'utile a été extrait) Hinton et al. (2006); Bengio et al. (2007). Dans le cas de la CD, la chaîne de Markov est réinitialisée à chaque mise à jour, donc chaque échantillon est tiré du mode le plus proche d'un exemple d'entraînement, ce qui dans un certain sens contourne le problème. Dans le cas de la PCD, où les échantillons sont tirés d'une chaîne persistante, cela est plus intrigant.

Il y a donc une différence significative entre l'échantillonnage de Gibbs sur un modèle entraîné et sur un modèle (de type PCD, FPCD ou *herding*) en cours d'apprentissage (la séquence des $\mathbf{x}^-$ utilisés dans la phase négative). En fait, on peut observer que le fait même de mettre à jour les paramètres du modèle aide au mixage, et cela même lorsque l'apprentissage est à toutes fins pratiques terminé (soit après quelques centaines de milliers de mises à jour sur les jeux de données que nous utiliserons). Le taux de mixage observé semble aussi être proportionnel à la magnitude de la mise à jour, contrôlée par le taux d'apprentissage ϵ.

Il s'avère donc qu'il est utile, afin de mieux comprendre le processus d'apprentissage de la RBM comme décrit dans le chapitre précédent, de considérer que l'échantillonnage se fait sur une chaîne de Markov *dynamique*, périodiquement modifiée par le processus de mise à jour des paramètres. Intuitivement, chaque mise à jour réduit l'énergie près d'un ou plusieurs exemples d'entraînement et l'augmente près d'un ou plusieurs échantillons. Par conséquent, les prochains échantillons auront une probabilité plus faible de revenir là où ils se trouvaient précédemment, ce qui favorise le mixage. La FPCD est en grande partie basée sur cette observation et l'exploite en augmentant spécifiquement la magnitude des changements sur les paramètres utilisés pour l'échantillonnage tout en permettant aux paramètres normaux de converger vers un point fixe. Le *herding*, de son côté, rejette la néces-

sité d'obtenir un estimé précis des paramètres du modèle : si l'on considère que notre objectif est tout simplement de produire des échantillons de la distribution cible, la preuve 3.35–3.40 nous garantit que, sous une condition facile à vérifier, les échantillons obtenus pendant l'entraînement auront les mêmes statistiques que celles des exemples d'entraînement, pour un certain nombre de statistiques choisies au préalable. En autant que ces statistiques soient suffisamment représentatives de la distribution cible, ce qui n'est pas nécessairement le cas, surtout pour des distributions complexes, on peut alors espérer obtenir des échantillons convenables.

Ceci étant dit, il n'existe pas de technique d'échantillonnage connue qui puisse opérer sur un estimé ponctuel des paramètres avec un bon mixage sans utiliser l'ensemble d'entraînement et sans modifier cet estimé. Or, en s'inspirant des principes mis de l'avant par la FPCD et le *herding*, une telle technique peut être envisagée.

4.1 Un exemple simple

Comme les chaînes de Markov sous-jacentes à l'apprentissage d'une machine de Boltzmann comportent typiquement un nombre d'états exponentiel en le nombre d'unités, il est difficile de les analyser directement. Aussi, pour commencer, il convient de tester quelques théories de base sur des modèles plus simples : étant donné une chaîne de Markov simple dont le mixage est mauvais, existe-t-il une ou plusieurs méthodes simples et efficaces pouvant accélérer le rhythme auquel la chaîne parcourt ses différents modes ?

Il s'avère que oui, dans certains cas. Posons une chaîne de Markov dont la matrice de transition est $\mathbf{T}$ et la distribution à l'équilibre est π. Alors :

$$\pi\mathbf{T} = \pi \tag{4.1}$$

Posons $\widetilde{\mathbf{T}} = \frac{\mathbf{T}-\lambda\mathbf{I}}{1-\lambda}$ où $0 < \lambda \leq \min_i T_{ii}$. On peut facilement vérifier que $\widetilde{\mathbf{T}}$ est

une matrice de transition valide, chaque ligne sommant à 1. Or,

$$\begin{aligned}
\pi\widetilde{\mathbf{T}} &= \pi\frac{\mathbf{T}-\lambda\mathbf{I}}{1-\lambda} && (4.2)\\
&= \frac{\pi\mathbf{T}}{1-\lambda}-\frac{\lambda\pi}{1-\lambda} && (4.3)\\
&= \frac{\pi}{1-\lambda}-\frac{\lambda\pi}{1-\lambda} && (4.4)\\
&= \frac{\pi-\lambda\pi}{1-\lambda} && (4.5)\\
&= \pi && (4.6)
\end{aligned}$$

Par conséquent, si π est la distribution stationnaire de $\mathbf{T}$, elle est également la distribution stationnaire de $\widetilde{\mathbf{T}}$. Or, $\widetilde{\mathbf{T}}$ a potentiellement un bien meilleur mixage que $\mathbf{T}$. Par exemple :

$$\mathbf{T} = \begin{pmatrix} 0.995 & 0.005 \\ 0.001 & 0.999 \end{pmatrix} \quad (4.7)$$

En prenant $\lambda = 0.99$. on obtient :

$$\widetilde{\mathbf{T}} = \begin{pmatrix} 0.5 & 0.5 \\ 0.1 & 0.9 \end{pmatrix} \quad (4.8)$$

Alors que $\mathbf{T}$ passera en moyenne 200 à 1000 itérations d'affilée dans les modes 1 et 2, $\widetilde{\mathbf{T}}$ passera en moyenne 2 et 10 itérations d'affilée dans ces mêmes modes, une grande amélioration. Cela montre qu'il est possible, dans certains cas, d'améliorer significativement le mixage en modifiant la matrice de transition. Plus particulièrement, cela montre que lorsqu'une chaîne a tendance à rester coincée longtemps dans un même mode, une solution viable est de pénaliser la transition d'un mode vers lui-même. Notons que la technique ne peut améliorer le mixage de toute chaîne.

Prenons par exemple la chaîne suivante :

$$\mathbf{T} = \begin{pmatrix} 0.500 & 0.498 & 0.001 & 0.001 \\ 0.498 & 0.500 & 0.001 & 0.001 \\ 0.001 & 0.001 & 0.500 & 0.498 \\ 0.001 & 0.001 & 0.498 & 0.500 \end{pmatrix} \tag{4.9}$$

La distribution stationnaire de cette chaîne est $\pi = (0.5, 0.5, 0.5, 0.5)$, c'est à dire que chaque état est équiprobable. Toutefois, cette chaîne aura tendance à osciller entre les modes 1 et 2 pour 500 itérations d'affilée avant de passer au mode 3 ou au mode 4, à partir duquel on aura une oscillation entre ceux-ci pour 500 itérations en moyenne, et ainsi de suite. On a donc deux "groupes" de modes, le mixage à l'intérieur d'un groupe étant bon, mais le mixage entre les deux groupes étant mauvais. En soustrayant $0.5\mathbf{I}$ et en renormalisant, on obtient :

$$\widetilde{\mathbf{T}} = \begin{pmatrix} 0.000 & 0.996 & 0.002 & 0.002 \\ 0.996 & 0.000 & 0.002 & 0.002 \\ 0.002 & 0.002 & 0.000 & 0.996 \\ 0.002 & 0.002 & 0.996 & 0.000 \end{pmatrix} \tag{4.10}$$

Le mixage de la nouvelle chaîne est un peu supérieur, mais marginalement. On se retrouve toujours avec une alternance entre les modes 1 et 2 pour en moyenne 250 itérations (quoique cette alternance est plus régulière qu'avant étant donné l'impossibilité de passer d'un mode à lui-même), après quoi on alterne entre les modes 3 et 4, et ainsi de suite. On ne se retrouve pas avec la situation voulue, qui serait une alternance continuelle entre les quatre modes.

Cette situation est en fait semblable à celle que l'on retrouve dans une RBM : la RBM reste coincée dans des "modes" qui sont en fait des groupes d'états proches les uns des autres. Il nous faut donc user d'une technique qui pénaliserait la transition d'un mode vers lui-même ainsi que vers les modes qui lui sont "proches". Dans le contexte de l'exemple précédent, on pourrait considérer soustraire une matrice

diagonale par blocs, c'est à dire :

$$\begin{pmatrix} \lambda & \lambda & 0 & 0 \\ \lambda & \lambda & 0 & 0 \\ 0 & 0 & \lambda & \lambda \\ 0 & 0 & \lambda & \lambda \end{pmatrix} \tag{4.11}$$

Cela fonctionnerait sur la matrice $\mathbf{T}$, mais c'est en raison de sa symétrie. En général, cette modification changerait la distribution d'équilibre (il suffit de regarder ce qui se passe lorsqu'un tel bloc couvre tous les modes et un terme λ est soustrait à chaque entrée de la matrice — à peu près toute matrice de transition est un contre-exemple !). Néanmoins, il apparaît clair qu'une méthode d'échantillonnage permettant d'améliorer le mixage d'une RBM doit, d'une part, pénaliser la probabilité de passer d'un mode à lui-même et à ses proches voisins et d'autre part, d'assurer de minimiser la déviation à la distribution cible que cette pénalisation induit.

4.2 Pénalisation d'échantillons

À partir de l'intuition selon laquelle la pénalisation de la probabilité de passer d'un mode à lui-même est une manière viable d'améliorer le mixage, on peut imaginer un algorithme d'échantillonnage sur une RBM, qu'on appellera pénalisation d'échantillons (abbrévié SP pour *Sample Penalization*) qui, à chaque itération de Gibbs, soustrait un terme aux paramètres correspondant au dernier échantillon. En d'autres mots, en supposant les paramètres $\theta = (\mathbf{W}, \mathbf{b}, \mathbf{c})$ d'une RBM et l'échantillon courant $(\mathbf{x}^-, \mathbf{h}^-)$, le prochain échantillon serait tiré de la distribution $\theta^- = \theta - g_\theta(\mathbf{x}^-, \mathbf{h}^-) = (\mathbf{W} - \epsilon \mathbf{x}^{-\prime}\mathbf{h}^-, \mathbf{b} - \epsilon \mathbf{h}^-, \mathbf{c} - \epsilon \mathbf{x}^-)$.

Autre que l'expérimentation, il n'y a pas de critère clair nous permettant de montrer que cet algorithme fonctionne. Étant donné l'influence intriquée des paramètres sur la distribution qu'ils représentent et le fait que l'espérance de θ^- est $\theta - \mathbb{E}[g_\theta(\mathbf{x}^-, \mathbf{h}^-)]$, ce qui est typiquement différent pour la majorité des jeux de don-

nées, il y a lieu de penser que cette méthode d'échantillonnage tire des échantillons d'une distribution différente, mais possiblement proche.

4.3 Statistiques positives

L'absence de justification théorique solide en faveur de l'algorithme SP ainsi que le fait que l'espérance des paramètres d'échantillonnage θ^- est différente des paramètres originaux θ sont autant de facteurs motivant l'élaboration d'une meilleure approche à l'amélioration du mixage.

L'équation 3.40, de par sa généralité, suggère une manière d'échantillonner un modèle déjà entraîné. En effet, sous une croissance sous-linéaire des paramètres, les statistiques des échantillons négatifs produits vont tendre à être les mêmes que les statistiques de la contribution positive. Lors de l'entraînement, on veut ainsi diriger les paramètres de sorte à ce qu'ils définissent une distribution dont les statistiques correspondent aux statistiques positives calculées à partir de l'ensemble d'entraînement. Lors de l'échantillonnage, l'on veut conserver les statistiques de la distribution du modèle, tout en laissant les paramètres "bouger" suffisamment rapidement pour que le mixage soit bon. Dans cette situation, il est donc indiqué d'utiliser, comme statistiques positives d'une seconde phase d'"entraînement", les statistiques *négatives* du modèle original.

En remplaçant ainsi la contribution positive par les statistiques $\sum_{\mathbf{v}\in\mathbb{B}^n} \mathcal{P}(\mathbf{v})g_\theta(\mathbf{v})$ de la distribution de laquelle on veut échantillonner, on obtient une invariance, chaque mise à jour s'appliquant à diriger le modèle vers ce qu'il était au début de la procédure. En d'autres termes, l'on "apprend" la distribution même du modèle, alors qu'on s'y trouve déjà. Or, ce second "apprentissage", qui ne tire qu'un nombre limité d'échantillons à chaque mise à jour, est stochastique et bruité, ce qui est bénéfique au mixage. Chaque itération du processus d'échantillonnage ainsi défini consiste en une contribution positive constante (les statistiques du modèle) et une contribution négative très bruitée (une pour chaque échantillon), mais en espérance, on sait qu'on tourne en rond.

Le problème principal, c'est d'estimer les statistiques du modèle — si on savait le faire, il nous serait possible de calculer la valeur exacte du gradient de la log-vraisemblance. Nous tricherons donc, suivant une idée mise de l'avant par Welling (2009a) : en supposant qu'une RBM ait été bien entraînée, il est raisonnable de supposer que $g_\theta^- = \sum_{\mathbf{s}\in\mathbb{B}^n} \mathcal{P}(\mathbf{s}) g_\theta(\mathbf{s}) = \frac{1}{|\mathcal{X}|} \sum_{\mathbf{x}\in\mathcal{X}} \mathbb{E}_{\mathbf{h}}[g_\theta(\mathbf{x},\mathbf{h})] = g_\theta^+$. En effet, tout minimum local de la log-vraisemblance négative, que l'on cherche à minimiser, est tel que ces deux termes sont égaux (voir équation 2.23) — en supposant que le gradient est bien estimé et que la descente de gradient converge, on s'attendrait donc à ce que ce soit le cas. L'idée est donc ici de supposer, de bonne foi, que les statistiques de la phase positive (que nous dénoterons $\widetilde{\theta}$), moyennées sur l'ensemble des données d'entraînement, correspondent aux statistiques de la phase négative qu'il nous est impossible d'estimer directement — ni indirectement par échantillonnage, puisque c'est la tâche qui nous occupe[1]. Notons que pour un apprentissage parfait, notre supposition serait effectivement correcte, mais qu'elle l'est également pour un apprentissage restant coincé dans un minimum local.

Il est à noter que si la machine de Boltzmann n'a aucune unité cachée, l'échantillonnage n'est alors qu'une continuation directe de l'apprentissage avec un taux d'apprentissage suffisamment grand pour favoriser le mixage. En effet, en l'absence d'unités cachées, les statistiques positives ne dépendent que des données d'entraînement et sont constantes tout au long de celui-ci. En approximant les statistiques négatives par les statistiques positives, puis en utilisant cette approximation comme statistiques positives lors de l'échantillonnage, il n'y a en fait aucun changement dans la procédure.

Si, en revanche, la machine a des unités cachées, l'échantillonnage a ceci de différent que la relation entre unités visibles et cachées est *fixée*. Au cours de l'apprentissage, le terme de la contribution positive g_t^+ à un moment t de l'entraînement dépend des paramètres θ, car ces paramètres régissent la relation entre les unités

[1] L'on pourrait considérer estimer les statistiques négatives par une technique d'échantillonnage différente, mais à part celles que nous développons dans cette section, nous n'en connaissons aucune qui puisse nous permettre de faire cette estimation en un temps raisonnable.

visibles et cachées, une relation qui change au cours de l'apprentissage. Chaque contribution positive implique donc un recalcul à chaque mise à jour à l'aide de l'ensemble d'entraînement ou d'un sous-ensemble de celui-ci, lié au fait que la relation entre unités visibles et cachées a changé.

Étant donné un modèle déjà entraîné, toutefois, l'évolution de la relation entre unités visibles et cachées n'est plus désirable — on suppose que l'apprentissage a déjà convergé vers quelque chose de solide. En fait, si l'échantillonnage doit se faire avec un taux d'apprentissage élevé afin de favoriser le mixage, cette relation pourrait même être dégradée. Cela nous permet de considérer, afin d'échantillonner *après* apprentissage, que cette relation, non plus ajustable, est devenue une *cible*. Les statistiques $\widetilde{\theta}$ s'inscrivent dans cette perspective : chaque exemple de l'ensemble d'entraînement se voit couplé avec une représentation précise pour les unités cachées et les statistiques des échantillons produits doivent être consistantes avec ces représentations pré-calculées. En somme, lors de l'échantillonnage, la machine de Boltzmann devient complètement observée, c'est à dire que toutes les unités sont visibles. Ce gel de la relation entre unités visibles et cachées a aussi pour avantage de rendre convexe l'optimisation des paramètres[2] — le minimum local vers lequel l'apprentissage a convergé devient un minimum global autour duquel l'on peut graviter.

Pour une RBM, nous calculons les "statistiques" positives $\widetilde{\theta} = (\widetilde{\mathbf{W}}, \widetilde{\mathbf{b}}, \widetilde{\mathbf{c}})$ qui

[2]Ceci peut être vérifié en prenant la dérivée par rapport à θ de l'équation 2.21 — étant donné qu'il n'y a pas d'unités cachées, $\mathbb{E}_{\mathbf{h}}[g_\theta(\mathbf{v}, \mathbf{h})|\mathbf{v}]$ devient $g(\mathbf{v})$, qui ne dépend pas de θ. On obtient que $\frac{\partial^2 C}{\partial \theta^2} = \sum_{\mathbf{v}_1} P^-(\mathbf{v}_1) g(\mathbf{v}_1) g(\mathbf{v}_1)' - \sum_{\mathbf{v}_1} P^-(\mathbf{v}_1) g(\mathbf{v}_1) \sum_{\mathbf{v}_2} P^-(\mathbf{v}_2) g(\mathbf{v}_2)' = \mathbb{E}[g(\mathbf{v})g(\mathbf{v})'] - \mathbb{E}[g(\mathbf{v})]\mathbb{E}[g(\mathbf{v})'] = \mathrm{Cov}[g(\mathbf{v})]$, la covariance étant une quantité positive définie. Lorsque applicable, cela signifie que C est convexe en θ et n'a qu'un seul minimum.

suivent :

$$\widetilde{\mathbf{W}} = \frac{1}{|\mathcal{X}|} \sum_{\mathbf{x} \in \mathcal{X}} \mathbf{x}' \mathcal{P}(\mathbf{h}|\mathbf{x}) \tag{4.12}$$

$$\widetilde{\mathbf{b}} = \frac{1}{|\mathcal{X}|} \sum_{\mathbf{x} \in \mathcal{X}} \mathcal{P}(\mathbf{h}|\mathbf{x}) \tag{4.13}$$

$$\widetilde{\mathbf{c}} = \frac{1}{|\mathcal{X}|} \sum_{\mathbf{x} \in \mathcal{X}} \mathbf{x} \tag{4.14}$$

Dans Welling (2009a), $\widetilde{\theta}$ est utilisé dans le contexte du herding en tant qu'approximation, dans un cas où les données d'entraînement ne sont pas présentes, et est calculé au cours de l'apprentissage par la moyenne des contributions positives à chaque étape, c'est à dire comme $\widetilde{\theta} = \frac{1}{t} \sum_t g_t^+$. Notre calcul diffère en ce qu'il ne dépend pas de l'historique de l'apprentissage mais seulement et directement de l'estimé des paramètres de la RBM de laquelle nous voulons échantillonner. Comme le herding n'a pas pour objectif d'obtenir un estimé précis des paramètres mais plutôt d'exploiter leur dynamique, il semble raisonnable (mais non nécessaire) de moyenner les statistiques du moins à court terme. Dans le cadre où la distribution est définie précisément, l'historique n'a pas d'importance. Nous avons également généralisé la technique à tout modèle de type RBM et avons montré expérimentalement qu'elle produit des échantillons ayant les statistiques voulues.

Notons finalement qu'en posant $\widetilde{\theta} = 0$, on obtient une procédure semblable à l'algorithme SP présenté à la section 4.2. Effectivement, la mise à jour des paramètres avec SP ne fait que leur soustraire les termes de la phase négative, sans additionner les termes de la phase positive, ce qui revient au cas où ces derniers sont nuls. Il est à noter que au vu de l'équation 3.40, un tel processus d'échantillonnage produirait à long terme des échantillons dont les statistiques suffisantes seraient les mêmes que les statistiques positives, c'est à dire globalement nulles, ce qui est généralement indésirable. Toutefois, cela n'est vrai que si les contributions sont accumulées et que les paramètres sont libres de converger vers un attracteur ayant cette propriété. SP, tel que nous l'avons défini, pénalise les paramètres *originaux*

en fonction du dernier échantillon seulement, ce qui borne leur déviation.

SP reste donc une possibilité intéressante dans le cas où la déviation des paramètres d'échantillonnage par rapport aux paramètres originaux est controlée. Pour ce faire, nous pouvons considérer que les paramètres d'échantillonnage sont l'addition des paramètres originaux avec une déviation ou paramètres "rapides" tendant vers zéro à un certain rythme.

4.4 Algorithme

À partir des sections précédentes, il nous est possible d'élaborer un algorithme d'échantillonnage suffisamment général pour que chacune des techniques sus-présentées soit un cas spécial de celui-ci. En s'inspirant de la FPCD, nous ajoutons un ensemble de paramètres "rapides", θ_F dans lesquels toute modification devra se faire. De cette manière les paramètres originaux θ ne changent jamais et cela nous permet également d'expérimenter avec un taux multiplicateur α. Plus α est petit, plus la déviation des paramètres d'échantillonnage par rapport aux paramètres originaux sera faible et locale dans le temps, ne prenant en compte que les derniers échantillons générés. Par contraste, α grand permettra aux poids d'échantillonnage de dévier davantage des poids originaux, fournissant un meilleur mixage au prix d'un biais (peut-être) plus grand.

Nous étudierons les quatre variations suivantes (exprimées en fonction de l'algorithme 6) :

1. **Gibbs** : C'est la technique d'échantillonnage standard. Il suffit de poser $\epsilon_F = 0$, ce qui nullifie la dynamique.
2. **SP** : Pénalisation d'échantillon, voir section 4.2. Il suffit de poser $\tilde{\theta} = 0$ et $\alpha < 1$.
3. **StatFPCD** : Algorithme où $\widetilde{\theta}$ est choisi tel que décrit dans la section 4.3, en supposant qu'on échantillonne par Gibbs entre chaque mise à jour des paramètres θ. Le paramètres α donne une flexibilité supplémentaire en nous permettant de limiter la croissance des θ_F.

4. **StatHerd** : Algorithme tel que décrit précédemment, en supposant qu'on "échantillonne", comme dans le herding ou les réseaux de Hopfield, en itérant jusqu'à équilibre une procédure gloutonne de descente de gradient sur l'énergie.

Algorithm 6 Algorithme d'échantillonnage.

`EchantillonGeneral(`θ`,` θ_F`,` $\widetilde{\theta}$`,` ϵ_F`,` α`, Echantillonneur,` $\mathbf{x}^-$`)`
Échantillonnage d'une RBM de paramètres θ, avec paramètres d'ajustement θ_F et statistiques positives $\widetilde{\theta}$. Également dans la liste d'arguments, un pas d'apprentissage ϵ_F et un multiplicateur α pour les paramètres θ_F. $\mathbf{x}^-$ est le dernier échantillon produit et Echantillonneur est ou bien EchantillonCD (voir algorithme 2) ou bien EchantillonHerd (voir algorithme 5).
θ_F est initialisé à zéro.

$(\mathbf{W}, \mathbf{b}, \mathbf{c}) \leftarrow \theta$
$(\mathbf{W}_F, \mathbf{b}_F, \mathbf{c}_F) \leftarrow \theta_F$
$(\widetilde{\mathbf{W}}, \widetilde{\mathbf{b}}, \widetilde{\mathbf{c}}) \leftarrow \widetilde{\theta}$
$(\mathbf{x}^-, \mathbf{h}^-) \leftarrow \text{Echantillonneur}(\mathbf{x}^-, \mathbf{W} + \mathbf{W}_F, \mathbf{b} + \mathbf{b}_F, \mathbf{c} + \mathbf{c}_F)$
$\mathbf{W}_F \leftarrow \alpha \mathbf{W}_F + \epsilon_F(\widetilde{\mathbf{W}} - \mathbf{x}^{-\prime}\mathbf{h}^-)$
$\mathbf{b}_F \leftarrow \alpha \mathbf{b}_F + \epsilon_F(\widetilde{\mathbf{b}} - \mathbf{h}^-)$
$\mathbf{c}_F \leftarrow \alpha \mathbf{c}_F + \epsilon_F(\widetilde{\mathbf{c}} - \mathbf{x}^-)$
$\theta_F \leftarrow (\mathbf{W}_F, \mathbf{b}_F, \mathbf{c}_F)$
return θ_F, $\mathbf{x}^-$

CHAPITRE 5

EXPÉRIENCES

Au cours des sections précédentes, plusieurs algorithmes d'apprentissage ont été décrits, certains étant bien connus dans la littérature (CD, PCD) et d'autres faisant état de recherches plus récentes (FPCD, *herding*) et chacun (sauf pour le *herding*) définissant une distribution précise au terme de l'apprentissage à travers leurs paramètres. D'autre part, plusieurs algorithmes ont été présentés afin de tirer des échantillons de ces distributions : une technique classique (Gibbs), une variante d'un algorithme présenté dans Welling (2009a) (StatHerd) ainsi que deux nouveautés (SP et StatFPCD), la seconde étant inspirée de concepts déjà présentés dans Tieleman et Hinton (2009); Welling (2009a). Ce mémoire se veut une évaluation objective de la qualité des échantillons produits par ces techniques.

Pour ce faire, en premier lieu, des modèles seront entraînés sur plusieurs jeux de données à l'aide des algorithmes CD, PCD, FPCD et *herding*. Pour chacun de ces modèles pré-entraînés, des échantillons seront tirés en utilisant plusieurs techniques et la qualité de ces échantillons sera évaluée en les utilisant en tant qu'ensemble d'entraînement pour un modèle de densité non-paramétrique évalué sur un ensemble de test.

5.1 Jeux de données

Nous évaluerons les algorithmes sur les deux jeux de données qui suivent.

5.1.1 USPS

Le jeu de données USPS comporte 9298 exemples. Chaque exemple est une image de taille 16x16 en niveaux de gris représentant un chiffre allant de zéro à neuf, ce qui correspond à un vecteur d'entrée de 256 nombres à virgule flottante entre 0 (noir) et 1 (blanc). Ces données ont été publiées par le *US Postal Service*,

le service postal américain.

USPS est normalement séparé en un ensemble d'entraînement de 7291 exemples et un ensemble de test de 2007 exemples et nous utiliserons ce partitionnement, à ceci près que nous séparerons également l'ensemble d'entraînement en un ensemble de 6291 exemples pour l'entraînement en tant que tel et un ensemble de 1000 exemples de validation.

L'ensemble USPS a quelques particularités qu'il convient de mentionner : chaque chiffre n'est pas représenté équitablement, la classe zéro et la classe un étant significativement plus importantes que les huit autres et certaines disparités existant entre les ensembles d'entraînement et de test également. Ces considérations sont toutefois d'intérêt mineur (les modèles à l'étude devraient tout simplement modéliser ces particularités).

5.1.2 MNIST

Le jeu de données MNIST comporte 70000 exemples. Chaque exemple est une image de taille 28x28 en niveaux de gris représentant un chiffre allant de zéro à neuf, ce qui correspond à un vecteur d'entrée de 784 nombres à virgule flottante entre 0 (noir) et 1 (blanc).

MNIST est normalement séparé en un ensemble d'entraînement de 50000 et des ensembles de validation et de test comportant 10000 exemples chacun. Nous utiliserons ce partitionnement, sauf dans le cadre de l'estimation de densité, où nous utiliserons 18000 exemples pour la validation et 2000 pour le test.

5.1.3 Binarisation

Bien qu'il soit techniquement possible d'entraîner les modèles CD, PCD et FPCD sur des unités visibles continues en entrée (ces modèles sont capables d'exploiter l'information supplémentaire), ce n'est pas le cas du modèle *herding*. Plusieurs expériences seront donc effectuées sur des données strictement binaires. USPS et MNIST seront binarisés avec un seuil à 0.4, soit $x_i \leftarrow \mathbb{I}[x_i > 0.4]$. Ce seuil a été

(a) MNIST (b) USPS

FIG. 5.1 – Cent premiers exemples des ensembles d'entraînements de MNIST et USPS, binarisés avec un seuil à 0.4.

choisi à l'oeil afin de préserver visuellement l'identité des caractères.

5.2 Entraînement non-supervisé

Quatre algorithmes d'apprentissage seront utilisés pour obtenir des modèles pré-entraînés desquels il est possible de tirer des échantillons : la divergence contrastive (CD), la divergence contrastive persistante (PCD), la divergence contrastive persistante avec poids rapides (FPCD) ainsi que le *herding* (herd). Chacun de ces algorithmes est sensible à un certain nombre d'hyper-paramètres, certains étant communs et d'autres plus spécifiques.

5.2.1 Type d'unités

Tous les algorithmes à l'étude peuvent être entraînés avec des unités binaires pouvant prendre soit les valeurs 0 ou 1, soit les valeurs -1 ou 1. Ces deux scénarios mènent à une dynamique très différente et donc à un comportement très différent dans certains cas.

5.2.2 Nombre d'unités cachées

Plus il y a d'unités cachées, plus le modèle a de "capacité" d'apprentissage, c'est à dire qu'il est en mesure de représenter des distributions plus compliquées et qui capturent mieux les nuances dans les données. Ce paramètre est donc utile afin d'évaluer comment chaque algorithme répond à une augmentation de la capacité disponible. Sur USPS, nous utiliserons 16, 200 et 500 unités cachées. Sur MNIST, nous entraînerons chaque modèle sur 16, 200 et 1000 unités cachées.

5.2.3 Initialisation

Les paramètres $\mathbf{b}$ de chaque modèle seront initialisés à zéro. Les paramètres $\mathbf{W}$ de chaque modèle seront initialisés aléatoirement selon un certain germe. Afin de minimiser les variations dues à cette initialisation, plusieurs germes devront être utilisés. Notons que d'initialiser les poids $\mathbf{W}$ à zéro n'est pas acceptable car cette configuration mène à un modèle dégénéré avec l'implantation des algorithmes que nous utiliserons[1].

L'initialisation en tant que telle se fera avec une distribution uniforme dans un certain intervalle. Pour les modèles CD, PCD et FPCD, il est utile que la variance des poids associés à une unité cachée soit à peu près égale à 1 afin d'éviter de les saturer dès le départ (cette idée s'applique également aux réseaux de neurones, voir LeCun et al. (1998a)). En effet, la probabilité conditionnelle qu'une unité cachée soit activée est représentée par une courbe sigmoide sur la pondération de l'entrée par les poids lui étant associés et la portion quasi-linéaire de cette courbe, qui est la plus sensible aux changements, s'étend de -1 à 1. Il est donc logique de faire

[1] Le calcul de la mise à jour des paramètres dans la CD, PCD et FPCD requiert l'estimation de l'espérance des corrélations entre les unités visibles et cachées, or l'espérance des unités cachées peut être inférée exactement à partir des unités visibles — si l'on se réfère aux figures 3.2 et 3.3, il est facile de voir que lorsque $\forall i, j\ \mathbf{W}_i = \mathbf{W}_j$ et $\forall i, j\ b_i = b_j$, alors $\forall i, j\ h_i = \text{sigmoid}(b_i + \sum_k W_{ik} v_k) = \text{sigmoid}(b_j + \sum_k W_{jk} v_k) = h_j$. Il s'ensuit que $\epsilon(v_k^+ h_i^+ - v_k^- h_i^-) = \epsilon(v_k^+ h_j^+ - v_k^- h_j^-)$ et $\epsilon(h_i^+ - h_i^-) = \epsilon(h_j^+ - h_j^-)$. Par conséquent, chaque ligne de $\mathbf{W}$ et chaque élément de $\mathbf{b}$ seront mis à jour de la même manière que chaque autre, perpétuant la condition initiale. $h_i = h_j$ sera donc toujours vrai, chaque unité cachée fera toujours exactement la même chose et la capacité du modèle s'effondre. L'apprentissage requiert donc une brisure de symétrie.

en sorte que le produit scalaire des poids avec l'entrée soit typiquement entre ces valeurs et on peut aisément calculer que d'initialiser les poids uniformément dans l'intervalle $\left[-\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}}\right]$ où n est le nombre d'unités visibles satisfait cette contrainte.

En ce qui a trait au *herding*, seul le signe du produit scalaire compte. Par conséquent, l'intervalle d'initialisation n'a pas véritablement d'importance. Pour être plus précis, l'apprentissage avec un intervalle d'initialisation $[-u, u]$ et un pas ϵ est équivalent à l'apprentissage avec un intervalle d'initialisation $[-\lambda u, \lambda u]$ et un pas $\lambda\epsilon$ (Welling, 2009a) et donc il suffit d'ajuster un seul de ces paramètres.

5.2.4 Taux d'apprentissage

Le pas ou taux d'apprentissage ϵ est le facteur multiplicatif appliqué à la mise à jour des paramètres (voir la section 2.2.2.2). Afin d'améliorer la probabilité que l'apprentissage converge, nous réduirons ϵ au fur et à mesure de celui-ci. En particulier, nous ajusterons le calcul de ϵ_t de sorte à ce qu'il ait une valeur de 10^{-4} à la toute fin de l'apprentissage, peu importe sa valeur de départ, que nous choisirons parmi $10^{\{-1,-2,-3,-4\}}$. Pour n mises à jour au total pendant l'apprentissage, cela correspond à $\epsilon_t = \frac{\epsilon_0}{\max(1, \epsilon_0 \frac{10^4 t}{n})}$.

Une politique où le pas demeure constant tout au cours de l'apprentissage a également été considérée, mais la réduction proposée du pas donne également de bons résultats tout en améliorant la convergence.

5.2.5 Taille de groupement

Chaque mise à jour sera calculée sur un sous-ensemble de l'ensemble d'entraînement. On peut observer empiriquement que l'efficacité de l'apprentissage semble optimale pour une taille de sous-ensemble qui est supérieure à 1 mais bien inférieure à l'ensemble complet (LeCun et al., 1998b). Ce paramètre n'est pas très sensible et nous entraînerons tous les modèles avec 8 exemples par mise à jour.

5.2.6 Nombre de chaînes

Chaque mise à jour sera également calculée à partir d'un certain nombre d'échantillons, chacun d'entre eux étant produit par une chaîne de Markov distincte. Pour la CD, ces chaînes sont calquées sur les exemples d'entraînement et donc il est commun d'en prendre exactement autant. Pour la PCD, la FPCD et le *herding*, il est possible de varier. La performance de ces algorithmes est plus grande lorsque le nombre d'échantillons augmente car un plus grand nombre d'échantillons permet une meilleure estimation des moments de la distribution. Nous prendrons un nombre de chaînes égal au nombre d'exemples présentés à chaque mise à jour, c'est à dire 8.

5.2.7 Nombre d'itérations de Gibbs

En ce qui a trait à la CD, chaque mise à jour est effectuée en calculant la divergence contrastive entre plusieurs exemples d'entraînement et l'état d'autant de chaînes de Markov initialisées avec ces mêmes exemples après un certain nombre d'itérations de Gibbs. Ce nombre (appelons-le k, voir l'algorithme 2) est souvent pris comme 1, mais cela tend à produire des échantillons trop proches des exemples d'entraînement et donc la performance de l'algorithme augmente en fonction de k — quoique k est rarement supérieur à 10. Comme l'objectif de nos travaux n'est pas de comparer rigoureusement la CD aux alternatives (un sujet qui est déjà bien couvert dans la littérature (Tieleman, 2008; Tieleman et Hinton, 2009)), nous utiliserons $k = 1$.

5.2.8 Paramètres rapides

L'algorithme de la FPCD fonctionne à l'aide de poids et de biais supplémentaires qui ne sont utilisés que pour tirer des échantillons au cours de la phase négative. Ils sont mis à jour de manière similaire aux paramètres normaux, mais avec un pas qui leur est propre. Ils sont également multipliés après chaque mise à jour par un facteur multiplicatif global α (voir algorithme 4).

Le pas ϵ_F des paramètres rapides sera choisi en conformité avec la méthode de (Tieleman et Hinton, 2009), c'est à dire qu'il sera initialisé à la même valeur que le pas des paramètres normaux, mais ne sera jamais réduit. Au cours de l'apprentissage, le pas des paramètres rapides sera donc de plus en plus grand proportionnellement au pas des paramètres normaux.

Tieleman et Hinton (2009) suggère également un facteur multiplicatif $\alpha = 0.95$ afin d'obtenir de bons résultats. Seront également essayés, à titre d'essai, des facteurs de 0 et de 1, correspondant respectivement à des paramètres rapides qui ne se rappellent que de la dernière mise à jour et des paramètres rapides qui sont libres de diverger des paramètres normaux.

5.3 Échantillonnage

Pour chaque modèle entraîné comme décrit précédemment, plusieurs séquences d'échantillons seront tirées selon les méthodes d'échantillonnage suivantes : échantillonnage de Gibbs (Gibbs), échantillonnage avec pénalisation des échantillons (SP), échantillonnage avec statistiques positives en utilisant l'échantillonnage de type FPCD (StatFPCD) et échantillonnage avec statistiques positives en utilisant l'échantillonnage de type *herding* (StatHerd). Chacun de ces algorithmes a un certain nombre d'hyper-paramètres décrits ci-bas :

5.3.1 Température

Cet hyper-paramètre sera appliqué à l'échantillonnage de Gibbs à titre d'essai. L'augmentation de la température du modèle réduit les paramètres $\mathbf{W}$, $\mathbf{b}$ et $\mathbf{c}$ de sorte à obtenir une distribution différente où les modes sont moins prononcés. Bien que la distribution soit différente, le mixage est possiblement meilleur et on peut considérer une température plus élevée comme offrant un bon compromis entre fidélité et mixage.

5.3.2 Taux d'apprentissage

Cet hyper-paramètre (ϵ_F dans l'algorithme 6) est utilisé dans les techniques SP, StatFPCD et StatHerd afin de déterminer par quel facteur la mise à jour des paramètres associée à chacune est multipliée. Nous essaierons des valeurs allant de 10^{-5} à 10^{0}. Notons bien que ce taux d'"apprentissage" n'est pas nécessairement le même que celui utilisé durant l'apprentissage en tant que tel.

5.3.3 Réduction des paramètres rapides

L'implémentation des techniques d'échantillonnage SP, StatFPCD et StatHerd procède en gardant un ensemble de paramètres $(\mathbf{W}_F, \mathbf{b}_F, \mathbf{c}_F)$ qui sont mis à jour alors que les paramètres initiaux $(\mathbf{W}, \mathbf{b}, \mathbf{c})$ restent inchangés. L'échantillonnage utilise les paramètres $(\mathbf{W}^-, \mathbf{b}^-, \mathbf{c}^-) = (\mathbf{W} + \mathbf{W}_F, \mathbf{b} + \mathbf{b}_F, \mathbf{c} + \mathbf{c}_F)$ Les paramètres $(\mathbf{W}_F, \mathbf{b}_F, \mathbf{c}_F)$ peuvent être multipliés par une constante (α dans l'algorithme 6) à chaque mise à jour afin de les réduire, ce qui nous permet de contrôler la déviation des paramètres d'échantillonnage par rapport aux paramètres originaux. Ceci semble nécessaire dans le cas de l'algorithme SP mais non pour les deux autres, où la fidélité de la procédure à la distribution voulue est montrée à la section 4.3.

5.3.4 État initial

Chaque échantillon de chacune des techniques à l'étude est le résultat d'un calcul sur l'échantillon précédent — une sorte de marche aléatoire sur un état persistant. Il y a plusieurs manières de choisir l'état initial : par un état constant, aléatoire ou bien avec un exemple de l'ensemble d'entraînement. Nous nous contenterons de la dernière option, qui est la plus généreuse pour tout le monde.

5.3.5 Binarisation

Nous évaluerons l'échantillonnage sur des échantillons binaires. Or, certaines techniques sont en mesure de produire des représentations continues. En supposant que ces représentations soient normalisées entre 0 et 1, deux techniques peuvent être

utilisées afin de les binariser. La première est d'interpréter chaque unité continue comme une probabilité (ce qui est effectivement la manière correcte de l'interpréter, il faut bien le préciser) et de tirer 1 avec cette probabilité, 0 sinon. La deuxième est de procéder par seuil : toute unité dont la valeur est supérieure à $\frac{1}{2}$ est binarisée à 1, les autres à 0 (donc $x_i \leftarrow \mathbb{I}[x_i > 0.5]$.

Il s'avère que le choix de la technique de binarisation dépend du jeu de données utilisé. Les jeux de données représentant des images comportent typiquement une certaine cohérence spatiale, c'est à dire que les pixels qui les composent ne sont pas indépendants les uns des autres : deux pixels qui se touchent, par exemple, ont davantage de chances d'être identiques que deux pixels pris au hasard. Les modèles capables de générer des échantillons continus, même s'ils ont été entraînés sur des exemples binarisés, vont implicitement respecter cette cohérence. Si deux pixels ont la même valeur continue, disons $\frac{1}{2}$, et qu'ils se touchent, il est plausible qu'ils soient binarisés de la même manière[2]. Pour cette raison, nous binariserons en utilisant un seuil.

5.4 Évaluation par densité

L'échantillonnage sera évalué sur une tâche d'estimation de densité, l'estimation étant effectuée par la méthode des fenêtres de Parzen[3]. Étant donné un ensemble $\mathcal{E}$ d'exemples et un exemple x dont on veut estimer la densité :

$$p(\mathbf{x}) = \frac{1}{|\mathcal{E}|} \sum_{\mathbf{y} \in \mathcal{E}} \mathrm{K}(\mathbf{x}, \mathbf{y}) \tag{5.1}$$

Où $\mathrm{K}(\mathbf{x}, \mathbf{y})$ est un **noyau** tel que $\int_{-\infty}^{\infty} \mathrm{K}(\mathbf{x}, \mathbf{y}) d\mathbf{x} = 1$. Notons que étant donnée la dimensionnalité élevée des problèmes que nous étudierons, les densités que nous obtiendrons seront de magnitudes extrêmement faibles et dans ce cas il est

[2]Ceci pourrait être modélisé en ajoutant des paramètres afin de paramétriser les relations entre unités visibles — une machine de Boltzmann semi-restreinte. Osindero et Hinton (2008) ont exploré cette option afin de modéliser des distributions d'images naturelles.

[3]Voir travaux similaires par Diggle et Gratton (1984).

préférable de travailler dans le domaine logarithmique.

Nous construirons un ensemble $\mathcal{E}$ pour chaque séquence d'exemples ou d'échantillons que nous désirons évaluer et calculerons la contribution type à la probabilité jointe des exemples de test $\mathcal{X}_T$ du jeu de données approprié sous la densité définie par $\mathcal{E}$ ($\sqrt[|\mathcal{X}_T|]{\prod_{\mathbf{x}\in\mathcal{X}_T} p(\mathbf{x})}$ ou $\frac{1}{|\mathcal{X}_T|}\sum_{\mathbf{x}\in\mathcal{X}_T}\log(p(\mathbf{x}))$ à l'échelle logarithmique). Cette mesure correspond à la log-probabilité moyenne d'un exemple de test et nous désirons la maximiser. Nous abbrévierons cette mesure $m_{\mathcal{X}_T}(\mathcal{E})$.

Pour chaque séquence d'échantillons S tirés de nos modèles, étant donné l'ensemble d'entraînement $\mathcal{X}$ et un ensemble de validation $\mathcal{X}_V$ de même taille, nous pouvons construire les $\mathcal{E}$ suivants :

1. $\mathcal{E}_1 = \mathcal{X}$: pour évaluation de la densité de l'ensemble de test sous l'ensemble d'entraînement.
2. $\mathcal{E}_2 = \mathcal{X}_V$: pour évaluation de la densité de l'ensemble de test sous l'ensemble de validation.
3. $\mathcal{E}_3 = S$: pour évaluation de la densité de l'ensemble de test sous les échantillons tirés du modèle seulement.
4. $\mathcal{E}_4 = S \cup \mathcal{X}$: pour évaluation de la densité de l'ensemble de test sous l'union de l'ensemble d'entraînement et des échantillons tirés du modèle.
5. $\mathcal{E}_5 = S \cup \mathcal{X}_V$: pour évaluation de la densité de l'ensemble de test sous l'union de l'ensemble de validation et des échantillons tirés du modèle.

Nous nous concentrerons principalement sur $\mathcal{E}_3$, considérant que $\mathcal{E}_3$ nous donne une indication plus facile à interpréter du mixage du modèle, c'est à dire la progression de la mesure de densité à mesure que des échantillons sont ajoutés. Les autres mesures sont toutefois intéressantes dans le but d'évaluer le surapprentissage du modèle. Si un modèle surapprend, on s'attendrait à ce qu'il produise des échantillons quasi identiques aux exemples d'entraînement. Par conséquent, si l'on estime la densité de l'ensemble de test en utilisant l'ensemble d'entraînement et que l'on rajoute ces échantillons, ils n'apporteront pas grand chose de nouveau : on ne fait qu'ajouter des "copies" et donc l'estimation ne devrait pas être bien meilleure.

Par contre, si on les ajoute à l'ensemble de validation, qui n'a pas été vu lors de l'entraînement, l'amélioration devrait être plus marquée. On s'attendrait donc à ce que le surapprentissage mène à ce que $m(\mathcal{E}_4) < m(\mathcal{E}_5)$ si $m(\mathcal{E}_1) = m(\mathcal{E}_2 e)$ — ou du moins, à ce que les mesures 4 et 5 convergent vers des asymptotes différentes lorsque $|S| \to \infty$.

Étant donné que la quantité $\sum_{\mathbf{x} \in \mathcal{X}} \mathrm{K}(\mathbf{x}, \mathbf{y})$ retrouvée dans le calcul de la densité peut facilement être accumulée au fur et à mesure de l'échantillonnage, nous pouvons facilement tracer comment la mesure $m_{\mathcal{X}_T}(\mathcal{E})$ évolue en fonction de $|\mathcal{E}|$. En autant que les échantillons générés soient proches d'au moins un exemple de test, ce tracé peut nous permettre notamment d'évaluer le mixage : plus la pente de la courbe de $m_{\mathcal{X}_T}(\mathcal{E})$ en fonction de $|\mathcal{E}|$ est marquée, plus le mixage est bon.

5.4.1 Noyau binomial

Le noyau binomial sera utilisé afin de faire une estimation de densité sur des paramètres binaires :

$$\mathrm{K}(\mathbf{x}, \mathbf{y}) \quad = \quad \beta^{|\{i|x_i = y_i\}|}(1-\beta)^{|\{i|x_i \neq y_i\}|} \tag{5.2}$$

Où β est un paramètre représentant la masse de probabilités à distribuer à côté des exemples d'entraînement. Ce paramètre peut être optimisé en choisissant sa valeur qui maximise $\mathrm{K}(\mathcal{E}, \mathcal{X}_V)$. En pratique, β ne semble pas très sensible, c'est à dire que l'on obtient des résultats qualitativement similaires pour des β similaires et que le β optimal varie peu en fonction de $|\mathcal{E}|$.

β a comme exposant le nombre de bits de $\mathbf{x}$ et $\mathbf{y}$ qui sont identiques et inversement pour le terme $(1-\beta)$. Le logarithme du noyau est également simple à calculer :

$$\log(\mathrm{K}(\mathbf{x}, \mathbf{y})) \quad = \quad \log(\beta)|\{i|x_i = y_i\}| + \log(1-\beta)|\{i|x_i \neq y_i\}| \tag{5.3}$$

CHAPITRE 6

RÉSULTATS

6.1 Estimation de la densité

Pour une RBM donnée, il est en principe possible de calculer la log-vraisemblance exacte d'un ensemble de test par rapport à la distribution qu'elle définit. Ce calcul est la véritable mesure de qualité du modèle, mais son coût est prohibitif : $O(2^{\min(n_v,n_h)})$, où n_v est le nombre d'unités visibles et n_h le nombre d'unités cachées[1]. Couplée à une séquence d'échantillons, la technique d'estimation de densité présentée à la section 5.4 se veut une estimation peu coûteuse de cette quantité. Il n'est toutefois pas garanti qu'elle soit correcte. De mauvais résultats pourraient être obtenus dans trois cas :

1. Lorsque la méthode d'échantillonnage est infidèle, c'est à dire que les échantillons proviennent de la mauvaise distribution.
2. Lorsque la méthode d'échantillonnage est trop lente, donc le calcul demeure inefficace. Le produit du coût du tirage d'un échantillon et du nombre d'échantillons à tirer devrait demeurer inférieur au coût de la méthode analytique. Par exemple, une méthode ayant un mauvais mixage pourrait nécessiter un nombre d'échantillons exponentiel en le nombre d'unités du modèle.
3. Lorsque la distribution cible est non-locale. Notre technique d'estimation est basée sur un ensemble d'échantillons, chacun augmentant la densité là où il se trouve ainsi que celle de ses voisins, proportionnellement au nombre d'unités qui diffèrent de l'un à l'autre (ou, dit autrement, le nombre minimal d'arêtes à parcourir pour aller d'un coin à un autre sur un hypercube). Or, si, par exemple, la distribution cible couvre l'ensemble des vecteurs binaires de

[1] Si $n_v < n_h$, on peut utiliser l'énergie libre des configurations visibles telle que définie à l'équation 2.11, sinon on peut, à la place, dériver l'énergie libre des configurations cachées. L'énergie libre est facile à calculer dans une RBM.

somme paire, les voisins immédiats de tout exemple ont une somme impaire et devraient donc avoir une densité nulle (mais leurs voisins à eux devraient avoir une densité importante !). Notre estimateur ne peut rendre justice à ce genre de distribution.

Le troisième cas ne correspondant pas à nos jeux de données, nous pouvons l'ignorer. Restent les points un et deux. Afin de vérifier la pertinence de notre évaluation, nous avons entraîné une certaine quantité de modèles avec seulement 16 unités cachées. $2^{16} = 65536$, ce qui est suffisamment peu pour être capable de calculer la véritable log-vraisemblance des modèles. Nous avons ensuite comparé celle-ci avec la log-vraisemblance obtenue par estimation de densité. Des résultats sont présentés à la figure 6.1.

La *valeur* de la log-vraisemblance analytique d'un modèle et celle de la log-vraisemblance obtenue par fenêtres de Parzen ne sont pas identiques et elles ne sont pas proches non plus, mais il convient de souligner que nous essayons ici de faire une estimation empirique d'une distribution sur 2^{n_v} exemples possibles à l'aide d'un échantillon de taille très limitée : 100 000. La log-vraisemblance tend à augmenter avec le nombre d'échantillons et il est difficile de vérifier la valeur de l'asymptote. L'estimateur de Parzen pourrait également surestimer la log-vraisemblance analytique en exploitant la localité pour fournir de la densité à des endroits proches des exemples de test où le modèle n'en mettrait pas.

Bref, ceci étant dit, il nous suffit que les deux mesures soient *corrélées* et nous observons en effet des corrélations claire pour tous les algorithmes sur la figure 6.1. Ces corrélations sont intéressantes : en effet, elles nous permettent potentiellement de sélectionner le meilleur modèle à coût modique. Chacun des algorithmes d'échantillonnage semble avoir des avantages et des inconvénients. Systématiquement, SP et Gibbs produisent de moins bons échantillons (ce qu'on peut voir sur la figure 6.1 par le fait que la log-vraisemblance négative est plus élevée pour ces méthodes), mais sur la forme et l'ordonnancement des courbes, semblent être plus fidèles en général.

Une anomalie notable est le modèle entraîné par CD avec des unités 0/1 et échantillonné par StatFPCD ou StatHerd (figures 6.1(b) et 6.1(c)). Alors que sa log-vraisemblance analytique semble être bien pire que celle des modèles entraînés par PCD ou FPCD, sa log-vraisemblance estimée est similaire. Notons que SP et Gibbs (figures 6.1(d) et 6.1(e)) réflètent ce que la log-vraisemblance analytique indique et que le problème ne se produit que pour des unités 0/1. Une explication peut être avancée pour la pauvre log-vraisemblance analytique : la CD échantillonne systématiquement près des exemples d'entraînement, ce qui peut contribuer à "creuser" un fossé d'énergie près des ensembles d'entraînement sans se préoccuper du reste de l'espace. Des modes spurieux pourraient ainsi se former ailleurs dans l'espace d'états, accumulant une masse de probabilité non-négligeable sans jamais être pénalisés dans les mises à jour. La PCD et la FPCD évitent potentiellement ce problème en ayant une chaîne persistante. Il semble toutefois que StatFPCD et StatHerd arrivent à obtenir de bons échantillons. Une raison possible à cela serait que les statistiques positives et négatives n'ont pas convergé vers l'égalité lors de l'entraînement et que notre procédure d'échantillonnage corrige le tir. Si c'est le cas, cette correction est effectuée assez rapidement, car nos expérimentations ne montrent pas une amélioration marquée de l'échantillonnage au fil du temps. Il se pourrait également que StatFPCD et StatHerd nuisent au mixage entre les modes voulus et les modes spurieux — cela serait curieux, à notre avis, mais possible — nous n'avons pas testé le comportement de ces algorithmes lorsque initialisés avec l'un de ces modes indésirables.

Il est toutefois plus difficile d'expliquer pour quelle raison ces phénomènes ne se produisent pas avec des unités -1/1. Il est possible que la dynamique des unités 0/1 ait certaines caractéristiques qui rendent le mixage de la CD particulièrement médiocre par rapport à ce qui est le cas avec des unités -1/1. Pour le *herding* spécifiquement, la section 6.4 justifie la supériorité du système d'unités -1/1 sur le système 0/1 — bien que ce résultat ne se généralise pas à la situation qui nous occupe, on peut imaginer que le choix du type d'unités ait une incidence complexe sur le comportement de tous les algorithmes étudiés ici. En tous les cas, il semble

que l'échantillonnage par StatFPCD ou StatHerd est en mesure de "corriger" l'entraînement de certains modèles.

6.2 Évaluation qualitative des échantillons

Une manière simple d'évaluer l'échantillonnage est tout simplement la visualisation, pour des jeux de données qui s'y prêtent bien. Son utilité est limitée : l'évaluation est qualitative et susceptible d'être trompée par le surapprentissage. En effet, un modèle ayant surappris pourrait tout simplement générer des copies conformes des exemples d'entraînement. Le résultat serait plaisant à l'oeil, mais peu utile en pratique — l'équivalent d'un plagiat. Toutefois, la visualisation reste le meilleur moyen de vérifier que les échantillons sont bien ce à quoi l'on s'attend et de faire le lien entre le fonctionnement réel et perçu du modèle.

La figure 6.2 présente et décrit les cent premiers échantillons tirés par chacune des quatre techniques d'échantillonnage à l'étude sur un même modèle entraîné sur MNIST. Ce sont parmi les meilleurs échantillons que nous avons obtenus sur ces données. Les observations qualitatives sur cet échantillonnage très limité semblent également corroborées quantitativement, comme nous le verrons ci-après. Visuellement, les échantillons tirés en utilisant StatFPCD et StatHerd semblent beaucoup plus variés et mieux formés que ceux tirés par SP ou Gibbs.

6.3 Évaluation quantitative des méthodes d'échantillonnage

Nos expériences préliminaires nous ont permis d'identifier que les méthodes StatHerd et StatFPCD étaient de loin les meilleures, suivies par SP, puis par Gibbs. Nous avons donc utilisé StatHerd afin de sélectionner les meilleurs modèles sur USPS avec 200 unités cachées et sur MNIST avec 1000 unités cachées, modèles sur lesquels nous avons pu faire une exploration plus exhaustive de la paramétrisation de l'échantillonnage. Les figures 6.3, 6.4 et 6.5 montrent les résultats dans plusieurs situations.

Ces figures montrent clairement que StatFPCD est la technique qui marche

le mieux, suivie de près par StatHerd, de loin par SP et de très loin par Gibbs. Les trajectoires saccadées suivies par Gibbs correspondent à son pauvre mixage : chaque bond en avant est un bond d'un mode majeur vers un autre n'ayant pas été visité précédemment, ce qui procure un gain important que de meilleures techniques obtiennent presque immédiatement car elles visitent tous les modes majeurs très tôt. SP fait systématiquement mieux, mais l'irrégularité de la progression demeure. Par contraste, la progression de StatFPCD et StatHerd semble très "lisse", suggérant que tous les modes majeurs ont effectivement été visités au moins une fois au début. Ces résultats sont consistants avec l'impression visuelle donnée par les échantillons produits par chaque technique.

6.3.1 Paramétrisation

Nous avons observé que le pas d'"apprentissage" ϵ_F contrôle en grande partie le mixage : plus ϵ_F est grand, plus la transition d'un mode majeur à un autre semble rapide. Cela est conséquent avec l'idée que le mixage est amélioré en pénalisant le mode dans lequel on se trouve : plus ϵ_F est grand, plus cette pénalisation est grande et donc plus la motivation à aller ailleurs est grande. Cette amélioration a toutefois un prix : les transitions d'un mode majeur à un autre tendent souvent à passer par des modes intermédiaires spurieux, peu intéressants, qui dans les faits nuisent à la qualité de l'échantillonnage. Dans une certaine mesure, on peut avancer qu'un pas d'apprentissage trop grand pénalise trop les modes majeurs, de sorte qu'on a une situation qu'on pourrait appeler "anti-mixage" : la séquence d'échantillons se voit surimposée par un "bruit blanc" qui la dégrade sans changer les statistiques de manière durable et serait surtout visible lors des transitions d'un mode vers un autre. Un ϵ_F suffisamment grand pourrait également faire croître les poids linéairement de sorte à ce que le processus d'échantillonnage diverge, mais cela ne semble pas être le cas ici.

Nous avons observé que le pas ϵ_F optimal pour StatFPCD a tendance à être plus petit que le pas optimal pour StatHerd d'un facteur variant entre 10 et 100 (note : dans les figures 6.2(b) et 6.2(c), le pas est plus petit pour StatFPCD que

pour StatHerd, ce qui explique en partie pourquoi le premier semble moins bien mixer). Cette observation semble montrer que StatHerd est plus robuste à un pas d'apprentissage plus élevé. Ceci semble logique si on considère la méthode par laquelle chaque échantillon est obtenu, par descente de gradient sur l'énergie : chaque échantillon aura tendance à avoir une énergie plus faible qu'un échantillon obtenu par StatFPCD (qui peut, aléatoirement, aller dans des zones de haute énergie). Par conséquent, il est logique de penser que les échantillons produits par StatHerd se trouvent dans des "cuvettes" sur la surface d'énergie dont les voisins ont également une énergie basse. Il est donc moins probable que le "bruit blanc" mentionné précédemment déplace cette cuvette suffisamment loin pour qu'on y trouve un exemple spurieux, alors que StatFPCD, qui visite des états plus proches de la "frontière", sera davantage affecté. Toutefois, cet "avantage" de StatHerd est tout relatif, puisque à taux d'apprentissage égal, StatFPCD domine quand même — d'autres problèmes semblent mitiger sa performance.

Pour StatFPCD, le nombre d'itérations k de Gibbs entre chaque mise à jour est un facteur important. Cet effet est surtout visible sur MNIST, comme la figure 6.5(a) le montre, et tend à saturer à partir d'un certain k. Un k "effectif" a été calculé pour StatHerd, correspondant au nombre d'itérations moyen jusqu'à équilibre. Pour un k comparable (et même inférieur), StatFPCD tend à offrir une meilleure qualité. Cela suggère encore une fois des limitations inhérentes à StatHerd, puisque l'algorithme utilisé ici ne peut itérer plus loin que l'équilibre. Il est intéressant de voir que le k effectif de StatHerd est dans la zone de saturation de StatFPCD, ou proche, et l'un pourrait donc servir d'estimation *a priori* à l'autre.

Le paramètre α optimal pour StatFPCD semble, *a priori*, être 1, bien qu'une investigation plus poussée pourrait être faite. C'est à dire que l'échantillonnage marche mieux si les paramètres θ_F sont laissés à eux-mêmes. Cela peut s'expliquer à la lumière du fait que l'équation 3.40 nous garantit, sous certaines conditions faciles à vérifier, que les échantillons proviendront de la bonne distribution lorsque $\alpha = 1$. Un α plus faible aurait pour effet de pénaliser la déviation aux paramètres originaux θ — il est évident qu'une valeur très faible défairait la pénalisation des derniers

modes visités et nuirait significativement au mixage, mais d'un autre côté une valeur légèrement inférieure à 1 pourrait réduire le bruit blanc dû à la variance de θ_F. Il n'y a pas lieu de penser que θ_F ait une variance très significative, cependant, et $\alpha = 0.95$ donne des résultats significativement inférieurs à $\alpha = 1$. Il convient de préciser que le paramètre α utilisé dans l'algorithme de la FPCD (algorithme 4), qui a un rôle similaire, mène à un comportement différent et qu'une valeur de 1 fonctionne mal dans cette situation.

6.3.2 Comparaison avec l'ensemble d'entraînement

Pour certains modèles et certaines techniques d'échantillonnage bien paramétrisées, il est possible de générer une séquence d'échantillons qui produise, sinon un meilleur estimé de la densité de l'ensemble de test que l'ensemble d'entraînement, un estimé qui n'en dévie pas de manière statistiquement significative. Pour le jeu de données USPS, l'ensemble d'entraînement de 6,291 exemples donne une log-vraisemblance de -80.25 ± 1.31, un peu moins bon que la valeur de -76.14 ± 1.31 que donnent 100 000 échantillons tirés d'un modèle[2]. Pour le jeu de données MNIST, la différence est plus grande et significative : -157.74 ± 2.07 pour l'ensemble d'entraînement de 50 000 exemples et -149.09 ± 2.08 pour 200 000 échantillons, une amélioration de près de 5.5%.

Une observation surprenante sur MNIST est que certaines séquences d'échantillons offrent une performance marginalement supérieure à l'ensemble d'entraînement pour un nombre *égal* et significatif d'exemples, comme le montre la figure 6.5(b). Ainsi, les 50 000 premiers échantillons produits sur le meilleur modèle entraîné par FPCD par la meilleure méthode d'échantillonnage que nous avons essayée donnent un estimé de densité supérieur aux 50 000 exemples d'entraînement. Cela suggère que les premiers sont de qualité égale ou supérieure aux seconds, du

[2]L'incertitude est sur la moyenne de la log-vraisemblance des exemples de test et donc dépend principalement de la taille de celui-ci — un estimé plus précis pourrait être obtenu avec davantage d'exemples de test. Notons que les barres d'erreur sur les figures représentent autre chose, c'est à dire l'erreur sur différents germes pour l'initialisation des modèles avant l'entraînement, ce qui est plus pertinent dans le contexte.

moins comme estimateur de densité. Bien que cela puisse paraître surprenant, ce n'est pas impossible étant donné certaines particularités de la méthode d'évaluation. Si l'on désire estimer une distribution à partir d'un seul exemple, le meilleur choix serait non pas un exemple d'entraînement quelconque mais la moyenne de tous les exemples d'entraînement : bien que cette moyenne ne fasse probablement pas partie de la distribution, elle est le point le plus proche de tous les exemples. De même, si le nombre d'exemples avec lesquels définir une densité de type fenêtres de Parzen est limité, un avantage substantiel pourrait être obtenu en utilisant des exemples "prototypiques", chacun étant proche de plusieurs modes mineurs. Ce genre de comportement est attendu d'un modèle de capacité limitée et on s'attendrait à ce que l'avantage disparaisse après un certain nombre de tirages. La plupart des séquences d'échantillonnage que nous avons essayées ont effectivement cette propriété : elles font mieux pour plusieurs milliers d'exemples avant de se dégrader, l'ensemble d'entraînement reprenant le dessus par la suite (les deux courbes du haut sur la figure 6.5(a) sont un exemple de ce phénomène).

6.3.3 Temps de calcul

La production de 100 000 échantillons par StatFPCD avec $k = 15$ sur MNIST, pour un modèle ayant 1 000 unités cachées, prend aux alentours de vingt heures, ce qui est beaucoup si la méthode est appliquée afin de déterminer quel est le meilleur modèle entraîné. Ceci étant dit, un k plus grand améliore certes la qualité des échantillons, mais si le modèle M_1 est meilleur que le modèle M_2, en autant que la méthode d'échantillonnage mixe suffisamment rapidement, l'estimation de la log-vraisemblance devrait donner M_1 gagnant peu importe k. $k = 1$ étant quinze fois plus rapide que $k = 15$, l'avantage en temps de calcul est substantiel.

Il appert également que l'échantillonnage est un processus très facilement parallélisable : un nombre arbitraire de chaînes sur un modèle pré-entraîné peuvent être itérées simultanément à partir d'initialisations différentes avec un minimum (voire une absence complète) de communication inter-processus. L'échantillonnage peut donc être accéléré très facilement d'un facteur égal au nombre de processeurs

disponibles (répartis sur plusieurs machines s'il le faut).

6.3.4 Surapprentissage

Le fait que, comme expliqué à la section précédente, certaines séquences d'échantillons parviennent à "dominer" la trajectoire de l'ensemble d'entraînement suggère que les modèles arrivent à généraliser : en effet, le surapprentissage est défini comme une dégradation de la généralisation due à une fidélité trop grande à l'ensemble d'entraînement en particulier. Par conséquent, la performance de l'ensemble d'entraînement constitue une borne supérieure à la performance d'un modèle qui surapprendrait - une différence de performance favorisant le modèle ne peut être expliquée que par la synthèse, de la part du modèle, de certains principes généraux.

Afin d'évaluer plus objectivement la capacité de généralisation des modèles, nous avons calculé la log-vraisemblance de l'ensemble de test sous les cinq modalités présentées à la section 5.4. La figure 6.6 montre trois courbes correspondant à la progression de $m(\mathcal{E}_{S\cup\mathcal{X}_1})$, $m(\mathcal{E}_{S\cup\mathcal{X}_2})$ et $m(\mathcal{E}_{S\cup\mathcal{X}_V})$ où $\mathcal{X}_1$, $\mathcal{X}_2$ et $\mathcal{X}_V$ contiennent chacun 18 000 exemples et les deux premiers sont des sous-ensembles de l'ensemble d'entraînement complet $\mathcal{X}$ (qui contient 50 000 exemples), et S est l'ensemble des échantillons tirés jusqu'à ce point. À mesure que le nombre d'échantillons tirés du modèle augmente, ceux-ci en viennent naturellement à dominer le mélange. Il semble que l'apport de $\mathcal{X}_1$, $\mathcal{X}_2$ et $\mathcal{X}_V$ est approximativement le même, la différence initiale entre les deux courbes (où aucun échantillon n'a été ajouté) s'amenuisant à mesure que des échantillons sont tirés.

6.4 Apprentissage par herding

En raison du mécanisme particulier par lequel le *herding* fonctionne et sous la supposition que la température est zéro, il n'est pas possible de définir la log-vraisemblance analytique d'un modèle entraîné par cette méthode. La présentation que Welling (2009a) fait de cet algorithme et de la qualité de son échantillonnage est donc principalement qualitative, quoique certains résultats objectifs sont présentés

sous la forme d'une méthode de classification basée sur le *herding*. Néanmoins, le rapport de forces entre la magnitude des poids de départ et le taux d'apprentissage ϵ n'est pas exploré. Qui plus est, les résultats sont présentés avec des unités -1/1 pour la simple raison que le *herding* avec des unités 0/1 ne semblait pas fonctionner. En faisant usage des méthodes d'évaluation de l'échantillonnage que nous avons présentées et validées, nous avons essayé plusieurs valeurs pour l'hyper-paramètre ϵ, pour les deux types d'unités, dans le but de mieux comprendre le comportement du *herding* et ses particularités.

La figure 6.7 montre la performance du *herding* en fonction du taux ϵ pour les deux types d'unités, sur le jeu de données USPS, qui est également utilisé dans Welling (2009a). Nous pouvons constater que la dynamique des unités 0/1 dérape lorsque le pas est d'un ordre de grandeur comparable à l'initialisation des poids, alors que la dynamique des unités -1/1 semble beaucoup moins sensible. Nous observons que le problème arrive très tôt durant l'apprentissage : avec $\epsilon = 1$ et les paramètres étant initialisés uniformément selon $W_{ij} \sim U[-1, 1]$, les paramètres deviennent fortement négatifs très rapidement, après quelques mises à jour, notamment les biais des unités cachées et les poids. Or, cela pousse les unités cachées à être systématiquement zéro et les unités visibles à ne dépendre que de leurs propres biais. Tout apprentissage véritable est nullifié à partir de ce point et le résultat est bien montré à la figure 6.7(a). Nous pouvons expliquer ce phénomène de la manière suivante.

Supposons un vecteur binaire $\mathbf{x}$ de taille n_v et une matrice $\mathbf{W}$ de taille $n_h \times n_v$ où $W_{ij} \sim U[-1, 1]$ et le processus suivant :

$$\mathbf{x} \leftarrow \mathbf{W}'\mathbf{W}\mathbf{x} \tag{6.1}$$

$$\tag{6.2}$$

Cela implique que :

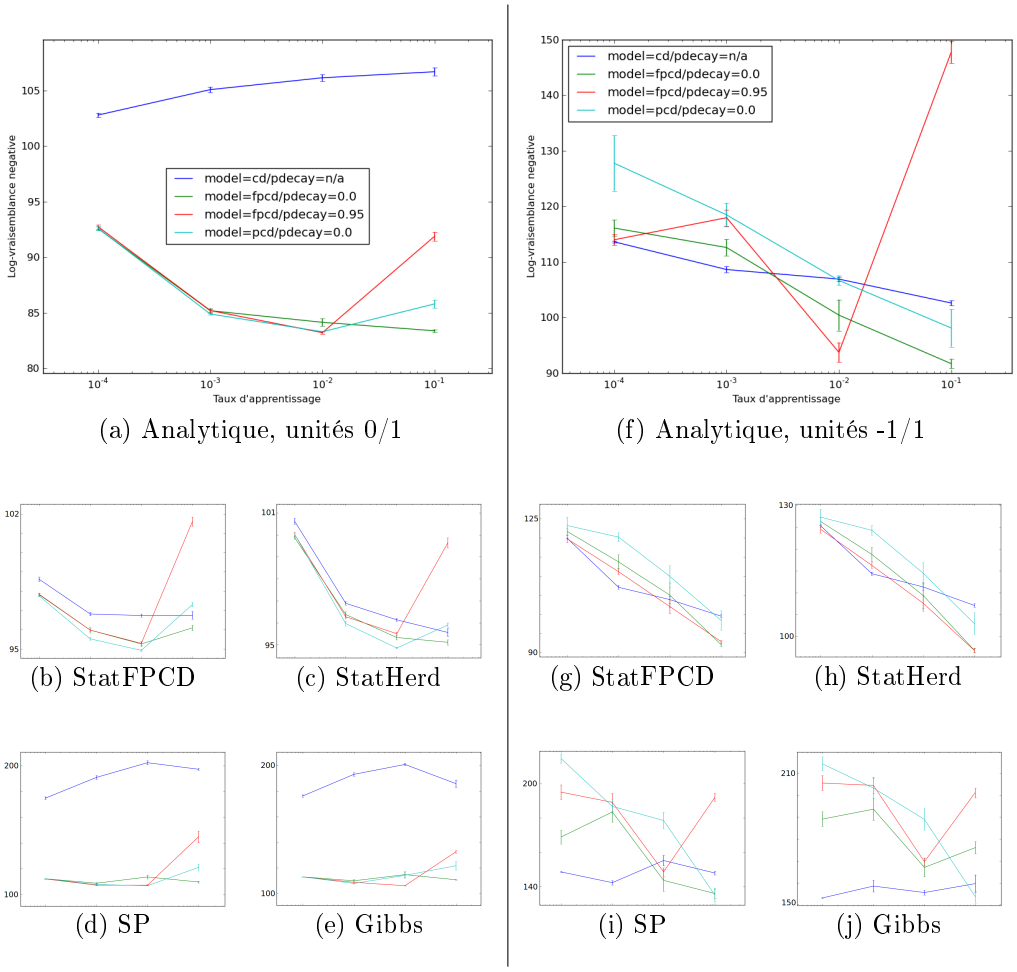
$$x_i \leftarrow \sum_j W_{ij} \sum_k W_{kj} x_k \tag{6.3}$$

$$= x_i \left(\sum_j W_{ji}^2 \right) + \left(\sum_j \sum_{k \neq i} W_{ji} W_{jk} x_k \right) \tag{6.4}$$

Étant données les propriétés statistiques de $\mathbf{W}$, l'espérance de la somme de gauche est strictement positive et l'espérance de la somme de droite est nulle - lorsque n_v et n_h sont suffisamment grands, la loi de limite centrale s'applique et les deux termes sont distribués normalement. Lorsque x_i est zéro, le premier terme disparaît et il ne reste que le second - la nouvelle valeur de x_i sera donc équiprobablement positive ou négative. Or, lorsque x_i est un, l'espérance de sa nouvelle valeur est positive avec une probabilité plus élevée. Si l'on applique un seuil à $\mathbf{x}$ tel que toute valeur négative devient zéro et toute valeur positive devient un, il s'ensuit qu'un 1 a une probabilité plus grande d'être préservé par la transformation qu'un 0 (probabilité que l'on pourrait estimer par une loi normale pour n_v et n_h suffisamment grands). Si x_i prend les valeurs -1 et 1 au lieu de 0 et 1, par contre, chaque valeur a en moyenne autant de chances d'être préservée qu'une autre.

La transformation exprimée ci-haut est très semblable à celle qui est appliquée par le *herding*, à la seule différence qu'un seuil supplémentaire est appliqué (mais on peut vérifier empiriquement que le même comportement demeure). En bref, pour des unités binaires, la dynamique du *herding* appliquée sur des paramètres aléatoires fait en sorte que les exemples négatifs contiennent un nombre disproportionné de 1. Lorsque pris en compte dans la mise à jour, cela résulte en une forte contribution négative. Lorsque ϵ est grand, cette contribution peut faire basculer le système dans un état d'où il ne peut plus sortir, ce que l'on constate expérimentalement. Si ϵ est plus petit, par contre, l'apprentissage peut suivre son cours et contrecarrer cette tendance avant qu'il ne soit trop tard. Avec des unités -1/1, il n'y a pas de "biais" vers une valeur en particulier et le système se comporte convenablement dès le début. Néanmoins, dans tous les cas, le choix de ϵ semble

importer, ne serait-ce qu'au niveau du taux de mixage.



(a) Analytique, unités 0/1 (f) Analytique, unités -1/1

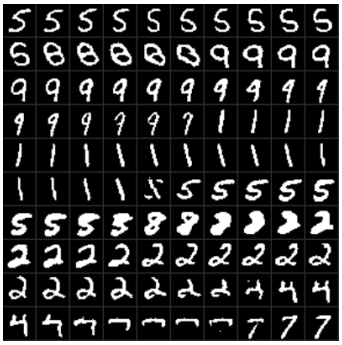
(b) StatFPCD (c) StatHerd (g) StatFPCD (h) StatHerd

(d) SP (e) Gibbs (i) SP (j) Gibbs

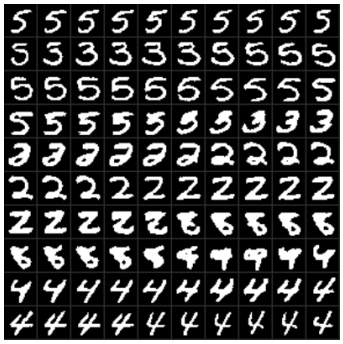
FIG. 6.1 – Comparaisons visuelles entre la log-vraisemblance analytique (en haut) et la log-vraisemblance estimée sur 100 000 échantillons tirés selon plusieurs méthodes (en bas). Sur chaque figure, la courbe bleue indique la log-vraisemblance (moyennée sur cinq germes) d'un modèle entraîné par CD selon le taux d'apprentissage durant l'entraînement. La courbe verte indique la même chose pour des modèles entraînés par FPCD avec oubli complet des poids rapides ($\alpha = 0$). En rouge, pour $\alpha = 0.95$. En cyan, sur des modèles entraînés par PCD. Sur les figures de gauche, les unités prennent les valeurs binaires 0 et 1, à droite, les valeurs -1 et 1. Le jeu de données est USPS et le nombre d'unités cachées est 16. Les barres d'erreur sont sur cinq germes pour l'entraînement (donc cinq modèles ayant les mêmes paramètres mais une initialisation aléatoire différente).

(a) MNIST

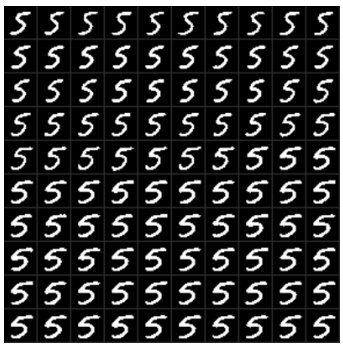
(b) StatHerd



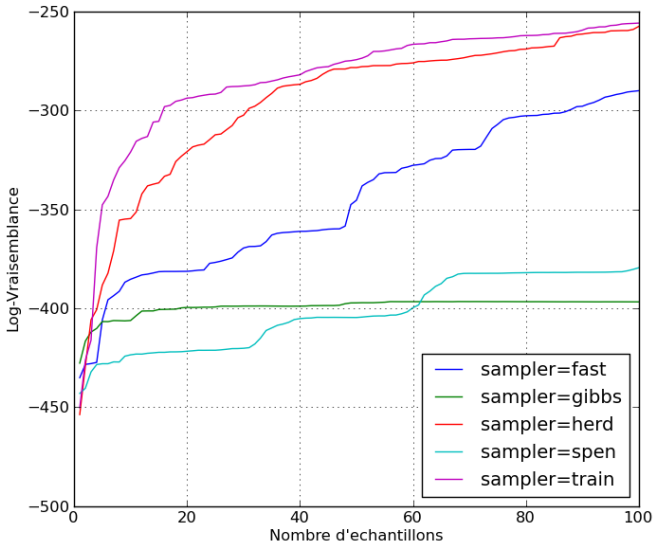
(c) StatFPCD



(d) SP

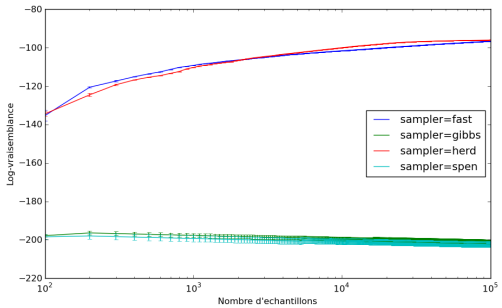


(e) Gibbs

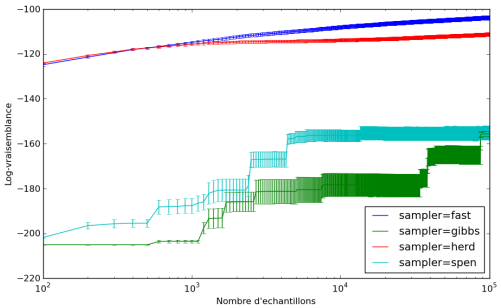


(f) Estimation de la log-vraisemblance en fonction du nombre d'échantillons pour chacun des groupes d'échantillons montrés. En magenta, les exemples de la figure 6.2(a) ; rouge : 6.2(b) ; bleu : 6.2(c) ; cyan : 6.2(d) ; vert : 6.2(e).

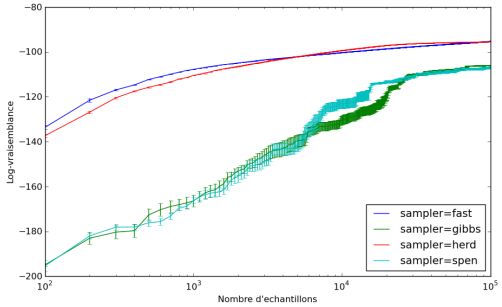
FIG. 6.2 – Cent premiers échantillons produits par plusieurs méthodes d'échantillonnage sur le même modèle (entraîné sur MNIST avec la méthode FPCD et 1000 unités cachées -1/1) (sauf 6.2(a) qui représente les 100 premiers échantillons de l'ensemble d'entraînement). Chaque méthode est initialisée avec le même exemple de l'ensemble d'entraînement. $k = 15$ dans chaque cas, si applicable.



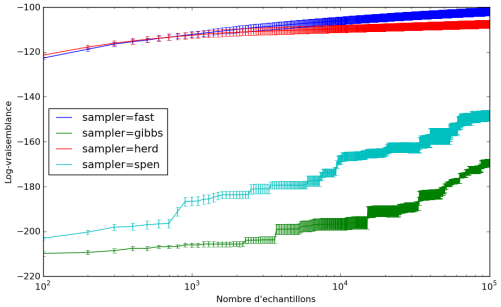
(a) Pré-entraînement CD, 16 unités 0/1



(b) Pré-entraînement CD, 16 unités -1/1

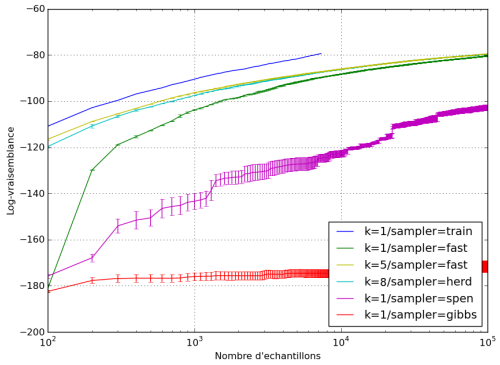


(c) Pré-entraînement FPCD, 16 unités 0/1

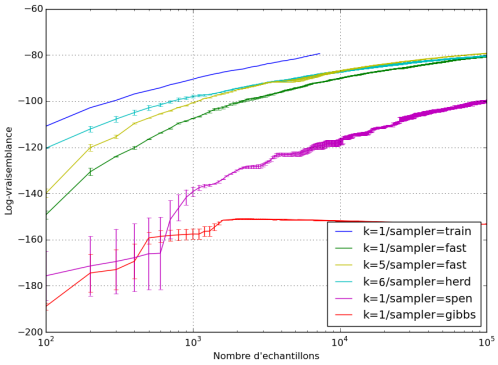


(d) Pré-entraînement FPCD, 16 unités -1/1

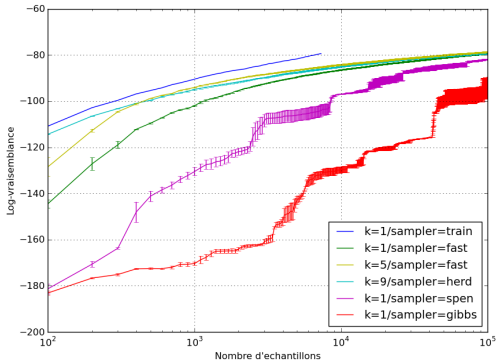
FIG. 6.3 – Progression de la log-vraisemblance en fonction du nombre d'échantillons tirés à partir de différents modèles pré-entraînés, pour plusieurs techniques d'échantillonnage. Sur toutes les figures, en bleu, échantillonnage par StatFPCD, en rouge, par StatHerd, en vert, par Gibbs et en cyan, SP. Le jeu de données utilisé est USPS. Les barres d'erreur sont sur cinq germes pour l'entraînement.



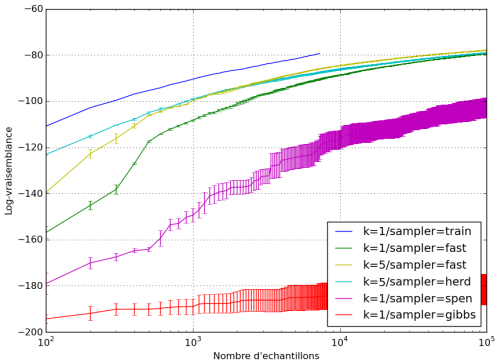
(a) Pré-entraînement CD, 200 unités 0/1



(b) Pré-entraînement CD, 200 unités -1/1

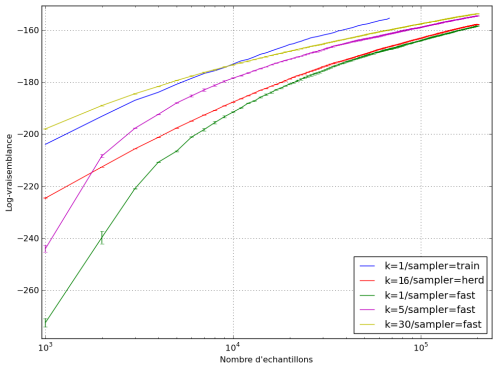


(c) Pré-entraînement FPCD, 200 unités 0/1

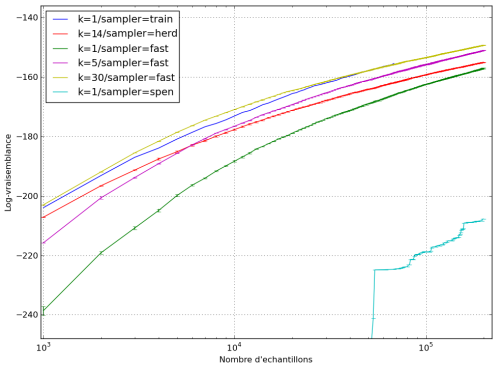


(d) Pré-entraînement FPCD, 200 unités -1/1

FIG. 6.4 – Progression de la log-vraisemblance en fonction du nombre d'échantillons tirés à partir de différents modèles pré-entraînés, pour plusieurs techniques d'échantillonnage. Sur toutes les figures, en bleu, la performance de l'ensemble d'entraînement, en vert et jaune échantillonnage par StatFPCD pour $k = 1$ et $k = 5$, en cyan, échantillonnage par StatHerd, en magenta par SP, en rouge par Gibbs. Le jeu de données utilisé est USPS. Les barres d'erreur sont sur deux germes pour l'entraînement.



(a) Pré-entraînement FPCD, 1000 unités 0/1



(b) Pré-entraînement FPCD, 1000 unités -1/1

FIG. 6.5 – Progression de la log-vraisemblance en fonction du nombre d'échantillons tirés à partir de différents modèles pré-entraînés, pour plusieurs techniques d'échantillonnage. Sur toutes les figures, en bleu, la performance de l'ensemble d'entraînement, en rouge, échantillonnage par StatHerd, en vert, magenta et jaune, échantillonnage par StatFPCD pour $k = 1, 5, 30$. En cyan, échantillonnage par SP. La performance de Gibbs est trop faible pour être montrée, de même pour la performance de SP sur la figure 6.5(a). Le jeu de données utilisé est MNIST. Les barres d'erreur sont sur deux germes pour l'entraînement.

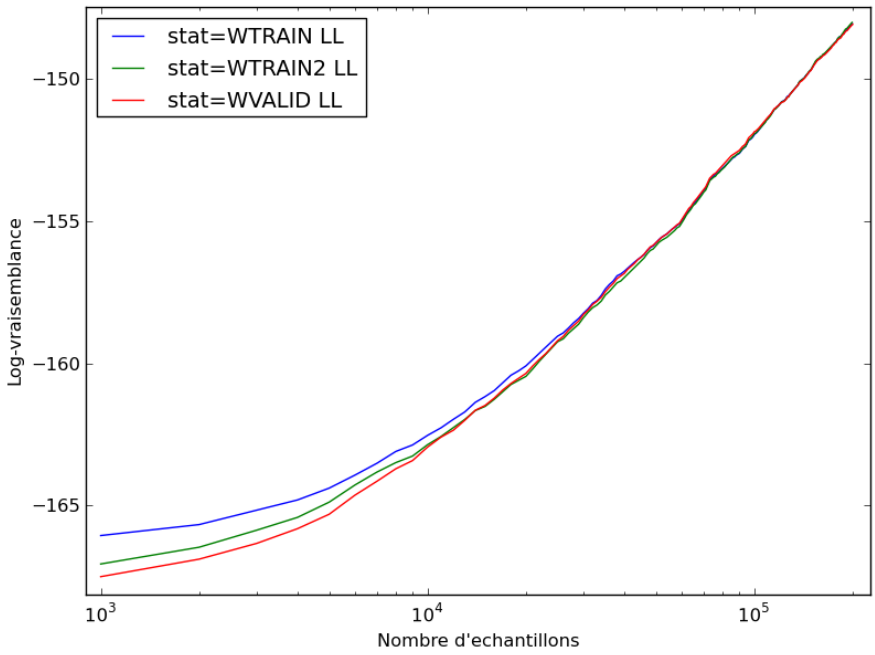
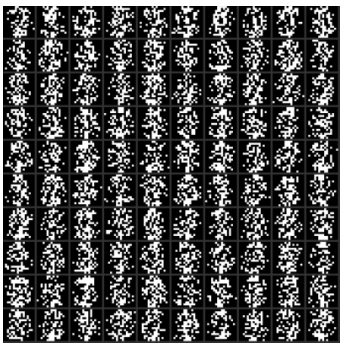
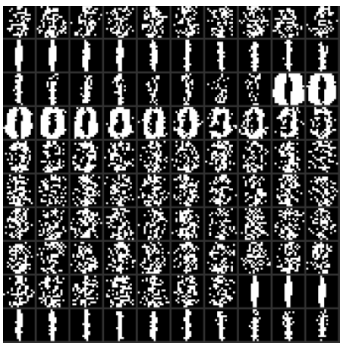


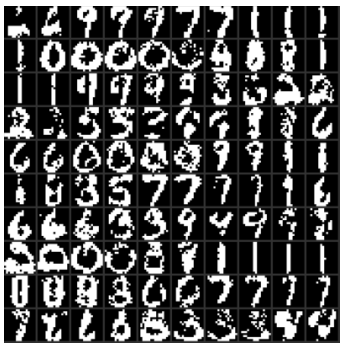
FIG. 6.6 – Progression de la log-vraisemblance en fonction du nombre d'échantillons tirés. Les deux courbes du haut représentent deux ensembles distincts de 18 000 exemples d'entraînement plus un nombre croissant d'échantillons (en abscisse). La courbe du bas représente 18 000 exemples de validation plus un nombre croissant d'échantillons.



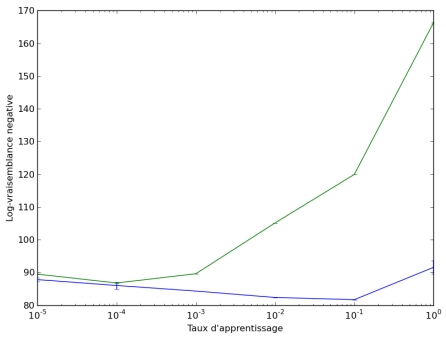
(a) Unités 0/1, $\epsilon = 1.0$



(b) Unités 0/1, $\epsilon = 0.1$



(c) Unités 0/1, $\epsilon = 0.001$



(d) Log-vraisemblance estimée par 100 000 échantillons en fonction du taux d'apprentissage ϵ pendant l'entraînement

(e) Unités -1/1, $\epsilon = 1.0$

FIG. 6.7 – Échantillons produits sur des modèles pré-entraînés par *herding* sur le jeu de données USPS, pour différentes valeurs du taux d'apprentissage ϵ durant l'entraînement. En haut (vert), pour des unités 0/1. En bas (bleu), pour des unités -1/1.

CHAPITRE 7

CONCLUSION

Ce mémoire se veut une contribution incrémentale à l'apprentissage non-supervisé. L'apprentissage au sein d'une machine de Boltzmann restreinte, un modèle bien établi, produit une distribution de probabilités qui est difficile à exploiter en raison de problèmes de mixage. En faisant la synthèse de concepts tirés de techniques récentes d'apprentissage sur une RBM, c'est à dire la divergence contrastive persistante rapide et le herding, nous sommes parvenus à améliorer la qualité de l'échantillonnage de la distribution d'une RBM pré-entraînée.

7.1 Revue de la littérature

Nous avons commencé par présenter, au cours des chapitres 2 et 3, les concepts fondamentaux nécessaires à la compréhension de ce mémoire. La machine de Boltzmann a été décrite comme un système paramétrique pouvant représenter une distribution de probabilités sur l'ensemble des vecteurs binaires d'une certaine taille à partir d'une fonction d'*énergie* sur ceux-ci. À partir de ces définitions, nous avons dérivé le mécanisme fondamental d'apprentissage d'une machine de Boltzmann, c'est à dire la descente de gradient sur la log-vraisemblance négative des données d'apprentissage, et mis en évidence les difficultés inhérentes à cette procédure. Nous avons identifié le processus d'échantillonnage comme étant la tâche la plus problématique parmi celles qui sont requises afin d'estimer le gradient.

Nous avons par la suite décrit le concept de chaîne de Markov, central à une classe de méthodes heuristiques d'échantillonnages appelées *Markov Chain Monte Carlo*. En particulier, l'échantillonnage de Gibbs permet d'obtenir des échantillons de la distribution jointe de plusieurs variables aléatoires à partir de leurs distributions conditionnelles. Cette méthode est celle qui est utilisée conventionnellement dans l'entraînement de la machine de Boltzmann. Le "mixage" d'une chaîne et *a*

fortiori d'une méthode de MCMC est la vitesse à laquelle elle visite les modes de sa distribution d'équilibre et le mauvais mixage est le problème principal dans l'entraînement d'une RBM. Nous avons présenté deux algorithmes récents dans la littérature, la divergence contrastive rapide et le herding dont l'objectif avoué est d'améliorer le mixage pendant l'entraînement.

7.2 Nouvelles approches proposées et expérimentations

Au commencement du chapitre 4, nous avons exploré de manière théorique quelques idées pour améliorer le mixage : notamment, la soustraction d'un multiple de l'identité de la matrice de transition d'une chaîne de Markov (suivie d'une renormalisation) peut améliorer son mixage sans changer sa distribution d'équilibre. Notant bien que cette technique est inapplicable directement à une RBM, nous avons toutefois émis l'hypothèse que le mixage d'une RBM pourrait être amélioré en pénalisant les échantillons récents, d'où l'algorithme de pénalisation d'échantillon (SP).

À partir d'un résultat important de Welling (2009a) démontrant sous certaines conditions faciles à vérifier qu'une chaîne de Markov périodiquement interrompue par une mise à jour de ses paramètres a une distribution d'équilibre ayant les statistiques voulues, nous avons formalisé deux algorithmes, StatHerd et StatFPCD qui utilisent cette propriété directement afin de tirer des échantillons d'un modèle pré-entraîné précis - le premier d'entre eux ayant été utilisé sous une forme légèrement différente par Welling (2009a).

Finalement, nous avons défini une procédure permettant d'évaluer de manière objective la qualité d'une séquence d'échantillons par rapport à un ensemble de test. La méthode des fenêtres de Parzen permet de définir une densité non-paramétrique à partir d'un ensemble d'exemples et fonctionne de manière incrémentale, ce qui nous permet de tracer l'évolution de notre mesure à mesure que le nombre d'échantillons augmente. En combinant cette mesure avec un estimateur calculé à partir des exemples d'entraînement et de validation, nous pouvons également déterminer

si le modèle surapprend.

7.3 Résultats

Selon nos expériences, notre estimateur de Parzen produit une mesure de qualité qui corrèle bien avec la log-vraisemblance véritable là où la comparaison est possible. Cela nous donne confiance en la pertinence de l'estimateur ainsi qu'en la fidélité et le bon fonctionnement des techniques d'échantillonnage que nous avons évaluées. Nous avons montré que les techniques StatFPCD et StatHerd mixent beaucoup mieux que l'échantillonnage de Gibbs conventionnel. SP marche également systématiquement mieux que Gibbs, mais par une marge beaucoup plus faible et au prix d'un biais dans la distribution.

Étant donné le peu de recherche qui a été fait jusqu'ici concernant le herding et la difficulté de l'évaluer objectivement autrement que par l'échantillonnage, nous avons étudié son comportement par rapport au pas d'apprentissage. Nous avons montré notamment que la difficulté apparente d'entraîner le herding avec des unités binaires 0/1 résulte d'un pas d'apprentissage trop grand et d'une dynamique asymétrique qui contribuent ensemble à plonger le modèle dans un état dégénéré dès les premières mises à jour.

7.4 Travaux futurs

La grande majorité des modèles utilisés dans le cadre de l'apprentissage machine, incluant bien entendu la machine de Boltzmann, sont des modèles approximatifs et heuristiques à presque tous les niveaux. Une machine de Boltzmann conventionnelle est paramétrisée par sa connectivité, par le nombre d'unités cachées et leur type (0/1 ou -1/1), par son pas d'apprentissage, par la magnitude de ses paramètres initiaux. La FPCD ajoute un deuxième pas d'apprentissage et un taux d'oubli. Il en va de même pour les techniques d'apprentissage. Qui plus est, la performance d'un algorithme est fortement liée au jeu de données sur lequel il est évalué. Tout ceci tend à montrer que de déterminer l'orientation de travaux futurs est en soi

un problème difficile et que toute recherche et tout résultat risquent d'être assez circonstantiels.

Ainsi, les faiblesses de ce mémoire sont autant de pistes pour des travaux futurs dont l'objectif principal serait de solidifier et promulguer son message. L'estimation de la log-vraisemblance par échantillonnage et Parzen pourrait être comparée à d'autres techniques d'estimation telles l'*Annealing Importance Sampling* (AIS) (Neal, 2001). Il serait approprié, voire important de reproduire nos résultats sur d'autres jeux de données et de les généraliser à des jeux de données comportant des valeurs continues. Finalement, dans les situations où la RBM capture bien la distribution cible, les échantillons en tant que tel pourraient être utiles dans de nombreuses situations et il serait intéressant de voir comment l'établissement de banques d'échantillons pourrait enrichir les jeux de données existants.

BIBLIOGRAPHIE

D. H. Ackley, G. E. Hinton et T. J. Sejnowski. A learning algorithm for Boltzmann machines. *Cognitive Science*, 9:147–169, 1985.

Christophe Andrieu, Nando de Freitas, Arnaud Doucet et Michael I. Jordan. An introduction to MCMC for machine learning. *Machine Learning*, 50(1-2):5–43, 2003.

Y. Bengio, P. Lamblin, D. Popovici et H. Larochelle. Greedy layer-wise training of deep networks. Dans *Advances in NIPS 19*, 2007.

Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

L. Breiman, J. H. Friedman, R. A. Olshen et C. J. Stone. *Classification and Regression Trees*. Wadsworth International Group, Belmont, CA, 1984.

Miguel A. Carreira-Perpiñan et Geoffrey E. Hinton. On contrastive divergence learning. Dans Robert G. Cowell et Zoubin Ghahramani, éditeurs, *Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics (AISTATS'05)*, pages 33–40. Society for Artificial Intelligence and Statistics, 2005.

Peter Clifford. Markov random fields in statistics. Dans Geoffrey Grimmett et Dominic Welsh, éditeurs, *Disorder in Physical Systems : A Volume in Honour of John M. Hammersley*, pages 19–32. Oxford University Press, 1990.

Corinna Cortes et Vladimir Vapnik. Support vector networks. *Machine Learning*, 20:273–297, 1995.

P. Diggle et R. Gratton. Monte carlo methods of inference for implicit statistical models. *Journal of the Royal Statistical Society. Series B (Methodological)*, 46 (2):193–227, 1984.

Stuart Geman et Donald Geman. Stochastic relaxation, gibbs distributions, and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6:721–741, November 1984. URL `http ://dx.doi.org/10.1080/02664769300000058`.

Onésimo Hernández-Lerma et Jean Bernard Lasserre. *Markov Chains and Invariant Probabilities*. Birkhäuser Verlag, 2003.

G. E. Hinton, T. J. Sejnowski et D. H. Ackley. Boltzmann machines : Constraint satisfaction networks that learn. Rapport technique TR-CMU-CS-84-119, Carnegie-Mellon University, Dept. of Computer Science, 1984.

Geoffrey E. Hinton. Products of experts. Dans *Proceedings of the Ninth International Conference on Artificial Neural Networks (ICANN)*, volume 1, pages 1–6, Edinburgh, Scotland, 1999. IEE.

Geoffrey E. Hinton. Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14:1771–1800, 2002.

Goeffrey E. Hinton, Simon Osindero et Yee Whye Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18:1527–1554, 2006.

John J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences, USA*, 79, 1982.

H. Larochelle, D. Erhan, A. Courville, J. Bergstra et Y. Bengio. An empirical evaluation of deep architectures on problems with many factors of variation. Dans *ICML 2007*, 2007.

Y. LeCun, L. Bottou, Y. Bengio et P. Haffner. Gradient based learning applied to document recognition. *IEEE*, 86(11):2278–2324, novembre 1998a.

Y. LeCun, L. Bottou, G. B. Orr et K.-R. Müller. Efficient BackProp. Dans G. B. Orr et K.-R. Müller, éditeurs, *Neural Networks : Tricks of the Trade*, pages 9–50. Springer, 1998b.

Stuart P. Lloyd. Least squares quantization in pcm. *Information Theory, IEEE Transactions on*, 28(2):129–137, 1982.

Roland Memisevic et Geoffrey E. Hinton. Unsupervised learning of image transformations. Dans *Proceedings of the Computer Vision and Pattern Recognition Conference (CVPR'07)*, 2007.

H. Minc. *Nonnegative Matrices*. John Wiley & Sons, New York, 1988.

John Moussouris. Gibbs and markov random systems with constraints. *Journal of Statistical Physics*, 10(1):11–33, 1974.

Radford M. Neal. Annealed importance sampling. *Statistics and Computing*, 11(2): 125–139, April 2001. URL `http ://dx.doi.org/10.1023/A :1008923215028`.

Simon Osindero et Geoffrey E. Hinton. Modeling image patches with a directed hierarchy of markov random field. Dans J.C. Platt, D. Koller, Y. Singer et S. Roweis, éditeurs, *Advances in Neural Information Processing Systems 20 (NIPS'07)*, pages 1121–1128, Cambridge, MA, 2008. MIT Press.

Frank Rosenblatt. The perceptron — a perceiving and recognizing automaton. Rapport technique 85-460-1, Cornell Aeronautical Laboratory, Ithaca, N.Y., 1957.

T. Sejnowski. Higher-order boltzmann machines. Dans *AIP Conference Proceedings*, volume 151, pages 398–403, 1986.

Paul Smolensky. Information processing in dynamical systems : Foundations of harmony theory. Dans D. E. Rumelhart et J. L. McClelland, éditeurs, *Parallel Distributed Processing*, volume 1, chaptitre 6, pages 194–281. MIT Press, Cambridge, 1986.

Tijmen Tieleman. Training restricted boltzmann machines using approximations to the likelihood gradient. Dans William W. Cohen, Andrew McCallum et Sam T. Roweis, éditeurs, *Proceedings of the Twenty-fifth International Conference on Machine Learning (ICML'08)*, pages 1064–1071. ACM, 2008.

Tijmen Tieleman et Geoffrey Hinton. Using fast weights to improve persistent contrastive divergence. Dans Léon Bottou et Michael Littman, éditeurs, *Proceedings of the Twenty-sixth International Conference on Machine Learning (ICML'09)*, pages 1033–1040, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-516-1.

Max Welling. Herding dynamic weights for partially observed random field models. Dans *Proceedings of the 25th Conference in Uncertainty in Artificial Intelligence (UAI'09)*. Morgan Kaufmann, 2009a.

Max Welling. Herding dynamic weights to learn. Dans Léon Bottou et Michael Littman, éditeurs, *Proceedings of the Twenty-sixth International Conference on Machine Learning (ICML'09)*. ACM, 2009b.

B. Widom. *Statistical Mechanics : a concise introduction for chemists*. Cambridge University Press, 2002.

Laurent Younes. On the convergence of markovian stochastic algorithms with rapidly decreasing ergodicity rates. Dans *Stochastics and Stochastics Models*, pages 177–228, 1998.