Éléments sur les réseaux de J. Hopfield et les Machines de Boltzmann de G. Hinton

Jean-Eric Campagne
Université Paris-Saclay, CNRS/IN2P3, IJCLab, 91405 Orsay, France
jean-eric.campagne@ijclab.in2p3.fr

Résumé

Le prix Nobel de Physique 2024 [1] a été décerné à John J. Hopfield et Geoffrey E. Hinton pour leurs travaux pionniers dans les années 1980 à la base du développement marquant qu'il s'en est suivi dans le domaine de l'apprentissage automatique artificiel. Leurs deux contributions mentionnées par le comité Nobel à savoir les *réseaux de Hopfield* et *les Machines de Boltzmann* sont présentées avec quelques illustrations numériques [2].

***K*eywords** Hopfield network · Boltzmann Machine · Machine Learning

1 Introduction

Un florilège des architectures neuronales est donné sur la Figure 1, on peut y voir des réseaux de neurones de plus en plus complexes quand on parcours ce poster. Nous allons nous concentrer uniquement sur les architectures mises au point par John J. Hopfield (Hopfield, 1982) et Geoffrey E. Hinton (Hinton and Sejnowski, 1986) au début des années 1980. Par la suite, il est indéniable que les architectures des réseaux de neurones profonds ont gagné en popularité grâce à leur adaptabilité aux problèmes posés, et surtout grâce à leurs performances qui ont surpassées les méthodes antérieures, surtout dès que les *opérateurs convolutionnels* ont été mis en œuvre par Yann LeCun & Yoshua Bengio (Lecun and Bengio, 1995) [3] pour envisager la reconnaissance automatique des chiffres manuscrit. Et de nos jours, les modèles génératifs [4] sont le sujet de bien des discussions non seulement d'experts mais aussi de monsieur ou madame tout le monde.

Disons tout de suite que ces progrès ont été possible grâce à une collaboration de divers champs disciplinaires et technologiques. Le *end-to-end* est devenu un leitmotiv et d'usage tout azimut également. Cependant, la compréhension de la nature exacte de ces architectures est encore un domaine de recherche mathématiques. Les personnes désireuses de plus de profondeurs sur ces sujets pourront se référer par exemple au cours du Collège de France du Professeur S. Mallat [5].

Le document ne prétend être ni un traité de mathématiques ni un cours de physique statistique. Des renvois vers des articles et livres plus spécialisés sont donnés. Dans la suite, nous allons en premier faire un tour rapide de la "brique de base" que constitue un neurone artificiel (Section 2), puis suit naturellement l'introduction aux réseaux de Hopfield (Section 3) et Machines de Boltzmann (Section 4).

1. https://www.nobelprize.org/prizes/physics/2024/press-release/
2. Le site https://github.com/jecampagne/demo-hopfield-rbm-networks compagnon héberge des scripts Python permettant de refaire les simulations sur la plateforme Google Colab.
3. Voir aussi http://yann.lecun.com/exdb/lenet/index.html. Notons que Y. LeCun, Y. Bengio et G. Hinton ont reçu le Prix Turing 2019 https://awards.acm.org/turing/award-recipients
4. Par ex. Gemini https://gemini.google.com/, ChatGPT https://chatgpt.com/, Midourney https://www.midjourney.com/...
5. https://www.college-de-france.fr/fr/chaire/stephane-mallat-sciences-des-donnees-chaire-statutaire/events. Un site web compagnon qui regroupe les notes de cours en français et en anglais, ainsi que des scripts Python illustrant le cours sont également disponibles à l'url https://github.com/jecampagne/cours_mallat_cdf.

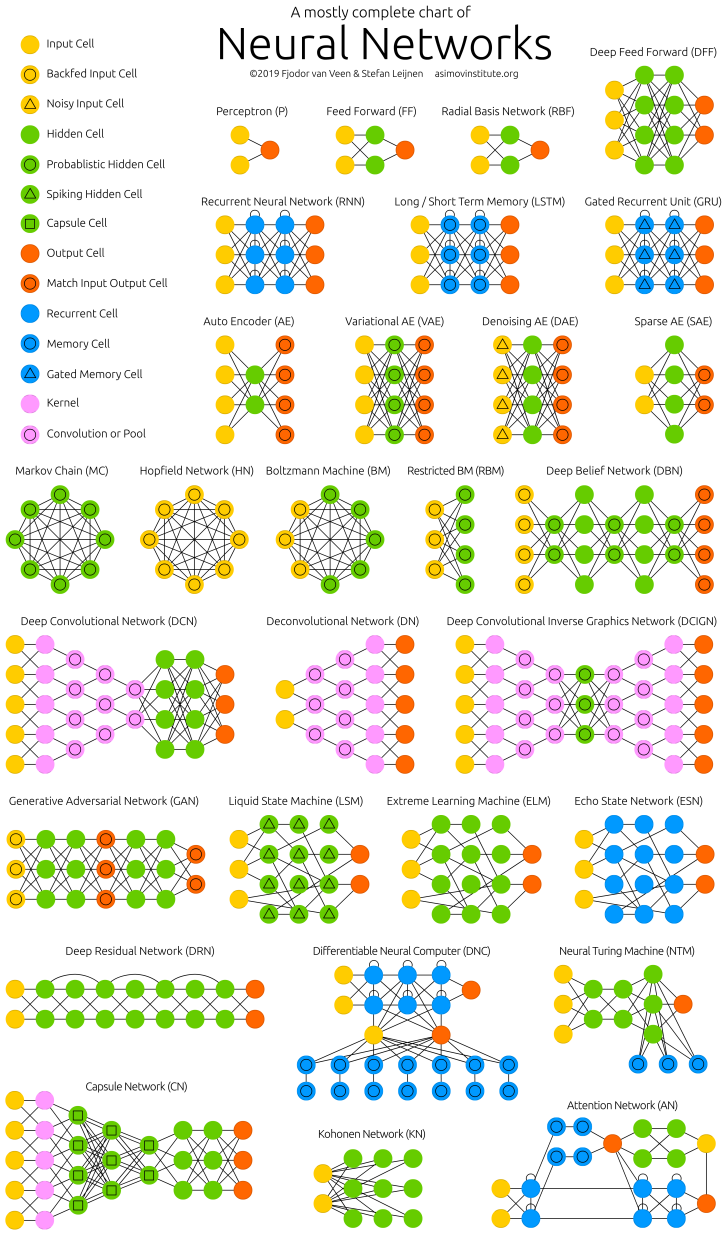

FIGURE 1 – Différentes architectures de réseaux de neurones (`https://www.asimovinstitute.org/neural-network-zoo/`).

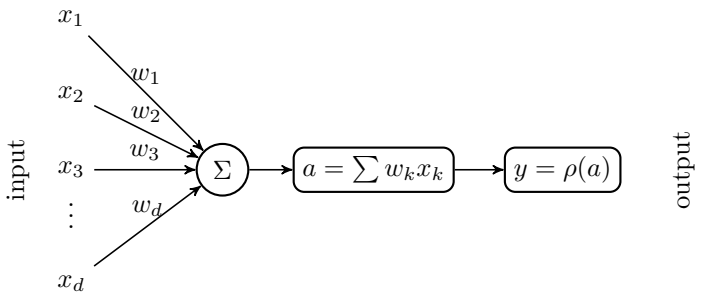

FIGURE 2 – Schéma simple de la fonction d'un neurone qui à partir des entrées $\{x_i\}_{i\leq d}$ et d'un vecteur de poids $\{w_i\}_{i\leq d}$ fournit une sortie y. La généralisation à un neurone délivrant plusieurs sorties est immédiate. La fonction ρ est une non-linéarité ponctuelle.

2 La brique de base : le modèle d'un unique neurone

Envisageons le cas d'un seul neurone étudié par Frank Rosenblatt dès 1957 (Rosenblatt, 1957) :

1. son *architecture* est donnée sur la Figure 2 ; il a en *entrée* un vecteur $x \in \mathbb{R}^d$ qu'il combine linéairement avec un vecteur de même taille $\boldsymbol{w}$. A cette combinaison, on peut y ajouter une constante ou bien l'inclure dans la définition de $\boldsymbol{w}$ avec $x(0) = 1$. On définit alors l'*activation* du neurone selon
$$a_i = \sum_j w_{ij} x_j \tag{1}$$

2. laquelle est suivit une procédure d'*actualisation* du neurone par une fonction ρ non-linéaire soit déterministe, ex. sigmoïde, tanh, ReLU, sinus, soit probabiliste ex. logistique. Cette *activation-actualisation* (appelée simplement activation la plus part du temps) constitue la *sortie* du neurone :
$$y_i = \rho(a_i) \tag{2}$$

Maintenant considérons des échantillons, $\{x_i, y_i\}_{i\leq n}$ dont on connaît pour l'entrée $x_i \in \mathbb{R}^d$ la sortie $y_i \in \{0, 1\}$. Une question peut se poser en ces termes : le neurone d'activation probabiliste paramétrée selon :
$$h_w(x) = P(y = 1|x; \boldsymbol{w}) = 1 - P(y = 0|x; \boldsymbol{w}) \tag{3}$$
est-il capable de stocker suffisamment d'*information* pour pouvoir associer y_i à x_i du lot d'échantillons. Il faut donc procéder à une sorte d'emprunte mémoire dans les valeurs des poids $\boldsymbol{w}$. Pour se faire, on l'entraîne comme un *classificateur binaire*.

La probabilité que y soit 0 ou 1 est donnée par
$$P(y|x; \boldsymbol{w}) = h_w(x)^y (1 - h_w(x))^{1-y} \tag{4}$$
et l'on forme le likelihood suivant en supposant que tous les échantillons sont indépendants
$$\mathcal{L}(\boldsymbol{w}) = \prod_{i=1}^{n} P(y_i|x_i; \boldsymbol{w}) \tag{5}$$
dont on cherche un maximum pour trouver la valeur de $\boldsymbol{w}$. Pour utiliser les algorithmes de minimisation, on utilise plutôt $-\log \mathcal{L}(\boldsymbol{w})$ que l'on normalise par le nombre d'échantillons n. Ainsi, on forme la fonction de coût (*loss*)
$$J(\boldsymbol{w}) = -\frac{1}{n} \sum_{i=1}^{n} \{y_i \log(h_w(x_i)) + (1 - y_i) \log(1 - h_w(x_i))\} \tag{6}$$
Il faut maintenant se donner une expression (une modèlisation) pour $h_w(x)$ (c'est-à-dire pour $P(y = 1|x; \boldsymbol{w})$). En principe on chercherait quelque chose comme [6]
$$h_w(x) = \rho(\langle x, \boldsymbol{w} \rangle) = \frac{1}{2}\Big(1 + \mathrm{sgn}(\langle x, \boldsymbol{w} \rangle)\Big) \tag{7}$$

6. le produit scalaire est noté $\langle ., . \rangle$.

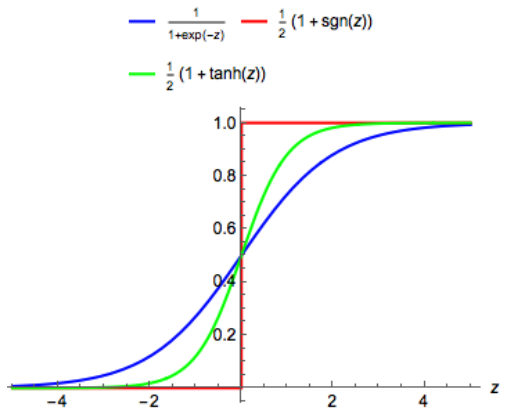

FIGURE 3 – Plusieurs fonctions $h_w(z) = \rho(\langle x, \boldsymbol{w}\rangle)$ pour réaliser la minimisation du likelihood.

où $\mathrm{sgn}(x)$ est la fonction "signe de x". En effet, si l'on considère une surface séparatrice entre les deux populations labellées "0" ou "1", le vecteur $\boldsymbol{w}$ correspond à la normale à cette hyper-surface et donc le signe est $+$ pour les 1 et $-$ pour les 0. Cependant, cette expression ne peut convenir comme argument du logarithme, c'est la raison pour laquelle David Cox en 1958 introduit [7] la *fonction logistique* (ou *sigmoïde*) (Figure 3) comme fonction d'activation dans ce type de problème

$$h_w(x) = \rho(\langle x, \boldsymbol{w}\rangle) = \frac{1}{1 + e^{-\langle x, \boldsymbol{w}\rangle}} \tag{8}$$

Cette formulation peut être justifiée pour ce qui nous concerne dans le cadre d'une distribution de Gibbs (voir Section 4). Afin de procéder à la *régression logistique*, à la fonction de coût (Equation 6) il est nécessaire d'ajouter une régularisation afin de corriger d'un possible surajustement (*overfitting*). Cela se fait par l'introduction par exemple d'une pénalité selon la norme L2 [8], c'est-à-dire que $J(\boldsymbol{w})$ devient alors

$$J_r(\boldsymbol{w}) = -\frac{1}{n}\sum_{i=1}^{n}\Big\{y_i \log(h_w(x_i)) + (1 - y_i)\log(1 - h_w(x_i))\Big\} + \lambda\|\boldsymbol{w}\|_2^2 \tag{9}$$

Plusieurs remarques peuvent être formulées à partir de là concernant une généralisation possible, le lien avec la problématique de Mécanique Statistique, et l'aspect du neurone vu comme une mémoire et non un classificateur.

1. Une première remarque concerne la généralisation de l'expression précédente [9] par

$$J_r(\boldsymbol{w}) = \frac{1}{2}\|\boldsymbol{w}\|_2^2 + C\sum_{i=1}^{n}\Big\{y_i \mathrm{cost}_1(\langle x_i, \boldsymbol{w}\rangle) + (1 - y_i)\mathrm{cost}_0(\langle x_i, \boldsymbol{w}\rangle)\Big\} \tag{10}$$

   Les fonctions $\mathrm{cost}_1(z)$ pour l'expression logistique et tanh sont montrées sur le graphe 4 en bleu et vert respectivement. Le cas de la courbe rouge exprime un effet de seuil, on alors de *hinge loss* associée à la *marge* des méthodes "Support Vector Machine" (SVM) développées par Vladimir Vapnik et collaborateurs d'abord en 1962-64 (Vapnik and Chervoneva, 1964) [10] puis dans les années 90's (Cortes and Vapnik, 1995)

$$\mathrm{cost}_1^{hinge}(\langle x_i, \boldsymbol{w}\rangle) = \begin{cases} 1 - \langle x_i, \boldsymbol{w}\rangle & \langle x_i, \boldsymbol{w}\rangle < 1 \\ 0 & \langle x_i, \boldsymbol{w}\rangle \geq 1 \end{cases} \tag{11}$$

   Les trois fonctions (Figure 4) sont des majorants [11] de la fonction "signe" présentée en noir. Elles permettent de rendre le problème convexe ce qui en facilite la minimisation.

7. Au passage D. Cox considérait que Joseph Berkson (1899-1982) était l'inventeur de la *régression logistique* en 1955. De même, on peut sans soute également citer le biologiste Raymond Pearl (1879-1940) qui l'utilisait dans les années 1920s pour décrire la courbe de croissance d'une population en fonction du temps (courbe logistique).

8. nb. La norme L2 est définie selon $\|\boldsymbol{x}\|_2 = \sqrt{\sum_{j=1}^{d}|x_i|^2}$.

9. Dans la littérature λ est sorti des $\{\}$ et on remplace $1/(2\lambda) = C$

10. nb. d'abord écrit en russe l'article fut par la suite traduit en anglais.

11. nb. on a mis à l'échelle les fonctions de coût logistic/tanh par $1/\log(2)$ pour ce faire, cela ne contribue pas à la minimisation.

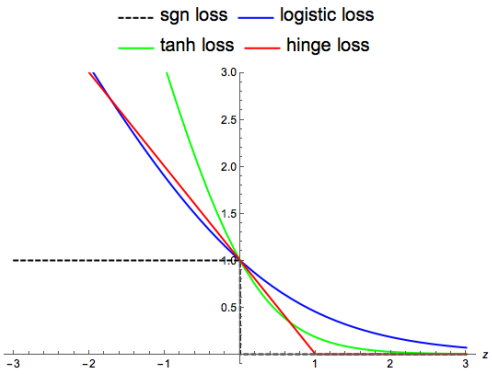

FIGURE 4 – Fonctions de risque/coût candidates.

Dans le cas de la minimisation de $J_r(\boldsymbol{w})$, pour interpréter l'origine du choix cost_1^{hinge}, prenons le cas où $C \gg 1$. Le problème devient alors une minimisation

$$\min_{\boldsymbol{w}} \left(\frac{1}{2} \|\boldsymbol{w}\|^2 \right) \tag{12}$$

sous *les contraintes* qui annulent le terme en C suivantes

$$\begin{cases} \langle x_i, \boldsymbol{w} \rangle \geq 1 & \text{si } y_i = 1 \\ \langle x_i, \boldsymbol{w} \rangle \leq -1 & \text{si } y_i = 0 \end{cases} \tag{13}$$

On retrouve le critère de marge des Support Vector Machines dans le cas $C \gg 1$.

2. Un seconde remarque concerne l'interprétation probabiliste de la fonction $J_r(\boldsymbol{w})$ (Equation 9) que l'on minimise pour en tirer le vecteur $\boldsymbol{w}$ optimal. On l'a réécrit selon (Equation 6) :

$$J_r(\boldsymbol{w}) = J(\boldsymbol{w}) + \lambda R(\boldsymbol{w}) \tag{14}$$

Or, $J(\boldsymbol{w})$ est le log-likelihood

$$P(\{y_i\}|\{x_i\}, \boldsymbol{w}) = \frac{1}{Z_J(\boldsymbol{w})} \exp\{-J(\boldsymbol{w})\} \tag{15}$$

alors la partie de régularisation $\lambda R(\boldsymbol{w})$ peut être interprétée comme le logarithme d'un *prior* :

$$P(\boldsymbol{w}|\lambda) = \frac{1}{Z_R(\lambda)} \exp\{-\lambda R(\boldsymbol{w})\} \tag{16}$$

Remarquons que dans le cas d'une régularisation en norme L2, ce *prior* correspond à une distribution gaussienne de $\sigma_w^2 = 1/\lambda$ donc la normalisation $Z_R(\lambda)$ est égal à $(2\pi/\lambda)^{d/2}$ (rappel : $\boldsymbol{w}$ est de dimension d). Donc, on peut construire la probabilité *a posteriori* du vecteur $\boldsymbol{w}$ connaissant le lot d'entraînement et la valeur de λ. Il vient

$$P(\boldsymbol{w}|D_{tr}) = \{x_i, y_i\}, \lambda) = \frac{P(\{y_i\}|\{x_i\}, \boldsymbol{w}) P(\boldsymbol{w}|\lambda)}{P(\{y_i\}, \lambda)} = \frac{1}{Z_{J_r}(\boldsymbol{w})} \exp\{-J_r(\boldsymbol{w})\} \tag{17}$$

Dans ce contexte, l'optimum $\boldsymbol{w}^*$ est la valeur la plus probable de $P(\boldsymbol{w}|D_{tr})$ qui se traduit par

$$\boldsymbol{w}^* = \underset{\boldsymbol{w}}{\text{argmax}}\ P(\boldsymbol{w}|D_{tr}) = \underset{\boldsymbol{w}}{\text{argmin}}\ (-\log P(\boldsymbol{w}|D_{tr})) = \underset{\boldsymbol{w}}{\text{argmin}}\ J_r(\boldsymbol{w}), \tag{18}$$

où $\boldsymbol{w}^*$ est également la valeur de $\boldsymbol{w}$ qui minimise la fonction de coût régularisée du domaine déterministe. La façon duale de considérer le cadre d'étude permet d'établi un pont entre la *Ridge Regression* et la *Bayesian Ridge Regression* de la littérature (voir par ex. Bishop, 2006). Notons la similitude de $P(\boldsymbol{w}|D_{tr})$ avec la fonction de Gibbs où $-J_r(\boldsymbol{w})$ serait considérée comme une énergie à minimiser, nous reviendrons sur ce point à propos des Machines de Boltzmann (Section. 4).

Finalement, dans le langage bayésien, si une nouvelle valeur x_{new} se présente à l'entrée du neurone, celui-ci produit alors la sortie $y_{new} = h_{w^*}(x_{new})$ d'une manière déterministe, tandis que la distribution

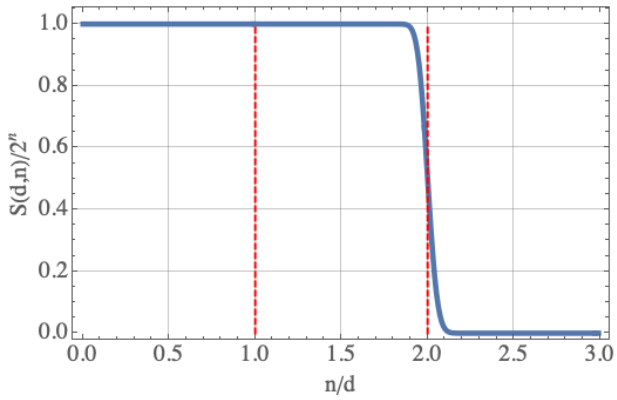

FIGURE 5 – Fonction $S(d,n)/2^n$ en fonction de n/d, pour $d = 1000$. Inspiré de Figure 40.9 du livre de MacKay (2003).

de probabilité de y_{new} conditionnée par la nouvelle valeur, le lot d'entraînement, et la valeur de l'hyper-paramètre λ est la suivante :

$$P(y_{new}|x_{new}, D_{tr}, \lambda) = \int P(y_{new}|x_{new}, \boldsymbol{w}, \lambda) P(\boldsymbol{w}|D_{tr}, \lambda)\ \mathrm{d}\boldsymbol{w} \tag{19}$$

L'intégrale pouvant être calculée par une technique de Monte Carlo dès lors que la dimensionalité devient grande.

3. Une troisième remarque concerne le point de vue considérant le neurone comme une *mémoire* à partir de laquelle on serait en mesure de reconstituer le lot d'entraînement D_{tr} constitué de n entrées distinctes. La question qui vient alors est : *quelle est la capacité d'un tel neurone ?* Somme toute comme $\boldsymbol{w} \in \mathbb{R}^d$, on s'attendrait à ce que le neurone ait une capacité infinie de stockage. Mais, force est de constater que personne n'a un accès direct aux poids $\boldsymbol{w}$, constituants internes du neurone : l'interaction s'effectue à travers l'entrée $\boldsymbol{x}$ uniquement. Donc, ayant n valeurs de $\boldsymbol{x}$ à notre disposition, et en sortie le neurones fournissant n bits (c'est-à-dire pour chaque entrée $\boldsymbol{x}_i$, la sortie est $y_i \in \{0,1\}$), la formulation de la question devient alors : *est-ce que l'on peut reproduire tous ces bits sans erreur* (2^n configurations) ? ou bien *quelle est la valeur du nombre n d'entrées maximale qui permet de n'avoir aucune erreur* ?

   Le neurone initialement se comporte selon la fonction seuil de l'Équation 7 que l'on peut simplifier en ne gardant que la fonction "signe", c'est-à-dire la réponse est alors ± 1 que l'on peut recoder en $\{0,1\}$. Mettons qu'il n'y a pas de biais par la suite [12]. La question qui est posée revient à se demander *combien de fonctions distinctes de ce type peut-on simuler en utilisant un vecteur $\boldsymbol{w}$ à d dimensions, et en utilisant n entrées $\boldsymbol{x}$*. On note $S(d,n)$ ce nombre. Le résultat est le suivant (Cover (1965) et voir aussi MacKay (2003)) :

$$S(d,n) = \begin{cases} 2^n & d \geq n \\ 2\sum_{i=0}^{d-1} \binom{n-1}{i} & d \leq n \end{cases} \tag{20}$$

   Remarquons que le nombre maximal de fonctions binaires générées avec n entrées (2^n) est la valeur de $S(d,n)$ dès que $n \leq d$.

   Que dire lorsque $n = d$? cette valeur particulière définit le nombre maximal de données que l'on peut étiqueter par des bits sans erreur. On peut le dire aussi comme le nombre d'échantillons que le neurone peut classifier dans le pire des cas. On appelle cela la *dimension de Vapnik-Chervonenkis* [13] (d_{vc}). On en conclut, que munit d'une fonction d'activation binaire, sa dimension d_{vc} est égale à d. La recherche de la valeur (ou borne) de la dimension VC d'architectures et fonctions d'activations différentes est toujours un sujet d'actualité (Bartlett and Maass, 2003; Petersen and Sepliarskaia, 2024).

   Que nous dit la fonction $S(d,n)$ pour $n > d$ et d grand ? C'est-à-dire au delà de $d = d_{vc}$, est-ce que la capacité du neurone est vraiment dégradée ? En fait, il n'en est rien comme on peut le voir sur la Figure 5 : plus d augmente plus le rapport $S(d,n)/2^n \approx 1$ jusqu'à la valeur $n_{max} = 2d$ où apparaît un

12. nb. on peut toujours faire $d \to d+1$ à la fin si on inclus ce biais
13. De Vladimir N. Vapnik et Alexey Chervonenkis qui introduisent cette notion dans les années 70s.

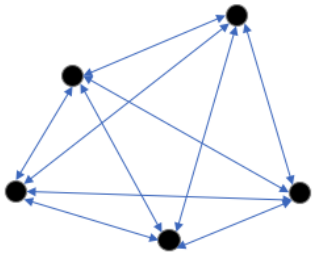

FIGURE 6 – Réseau de Hopfield : $w_{ij} = w_{ji}$ (flèche à double sens entre deux nœuds), $w_{ii} = 0$ (pas de flèche qui boucle sur le même noeud).

décrochage brutal (c'est un effet de sur-capacité en quelque sorte). Donc, on peut étiqueter (0/1) un nombre d'échantillons allant jusqu'à 2 fois la taille de l'entrée. Le nombre de bits maximal stockés (capacité de stockage) dans ce neurone à seuil linéaire est donc égal à $C(1, d) = 2d \times d = 2d^2$.

La généralisation pour une fonction à seuil polynomiale de degré p et pour de grandes valeurs de d donne la formule suivante de la capacité de stockage (Cover, 1965) :

$$C(p, d) = 2d\binom{p+d}{p} \approx 2\frac{d^{p+1}}{p!} \tag{21}$$

Ayant mis en place cette brique de base que représente un unique neurone, voyons à présent des architectures multi-neurones développées par J.J Hopfield et G.E Hinton.

3 Réseaux de Hopfield

Une architecture popularisée par John J. Hopfield en 1982 (Hopfield, 1982)correspond à la mise en interconnexion de neurones selon les principes suivant :

1. les connexions entre deux neurones (Figure 6) sont *symétriques* et *bidirectionnelles*, c'est-à-dire que l'on ne différencie pas l'action du neurone j sur le neurone i ($w(j \to i) = w_{ij}$) et l'action inverse donc $w_{ij} = w_{ji}$; il n'y a *pas de self-coupling* donc $w_{ii} = 0$. L'état d'un neurone est noté x_i. Si l'on veut inclure un biais dans le réseau, on introduit un neurone fictif x_0 d'état permanent 1. Toutes les sorties des neurones sont les entrées des autres neurones et vice-versa, et le réseau à N neurones. Il y a donc $N(N-1)/2$ poids différents.
2. la fonction d'activation $\rho(x)$ peut être par exemple une fonction à seuil de type $\text{sgn}(x)$ qui donne une valeur binaire, ou bien une fonction tanh qui donne une variable réelle continue. Ainsi, l'état d'un neurone est calculé selon le même procédé que pour un neurone unique. Par exemple, considérant le neurone i, on procède par une somme pondérée de son entrée, puis à son activation [14]

$$a_i(\boldsymbol{x}) = \sum_{j \neq i} w_{ij} x_j \qquad\qquad x'_i = \rho(a_i) \tag{22}$$

et x'_i devient la nouvelle valeur de l'état du neurone.

Cependant, il faut préciser la façon dont on met à jour le réseau à la suite de l'établissement d'un état initial pour tous les neurones. En quelque sorte, le temps est discrétisé et il faut préciser *les équations d'évolution du réseau.*

3. On peut considérer deux catégories de mise à jour :
   - *mode synchrone* : toutes les valeurs $\{x_i(t+1)\}_{i \leq N}$ sont calculées en même temps pour chaque neurones i à l'aide du set $\{x_i(t)\}_{i \leq N}$;
   - *mode asynchrone* : pour calculer $\{x_i(t+1)\}_{i \leq N}$ on passe en revu les N neurones soit dans un *ordre fixe séquentiel* ou soit de *manière aléatoire*, et l'on met à jour un neurone à la fois. Ainsi, mettons

14. nb. l'expression de a_i peut inclure un biais b_i qui peut être intégré dans la somme moyennant une redéfinition des poids $w_{i0} = bi$ et en rajoutant un nœud fixe $x_0 = 1$.

que l'on passe en revu le neurone 1 d'abord, puis le neurone 2 alors

$$x_1(t+1) = \rho\left(\sum_{j>1} w_{ij}x_j(t)\right)$$
$$\text{puis} \quad x_2(t+1) = \rho\left(w_{12}x_1(t+1) + \sum_{j>2} w_{ij}x_j(t)\right) \tag{23}$$

et ainsi de suite.

Pour les applications numériques qui suivent nous utilisons la méthode asynchrone aléatoire.

Les premiers travaux de J. Hopfield relatés en 1982 portent sur l'émergence de propriétés collectives des réseaux de neurones artificiels sur des bases de neurobiologie. Ils ont relancé l'intérêt pour les réseaux de neurones artificiels dont l'étude avait été délaissée après le livre de Marvin Minsky et Seymour Papert de 1969 (Minsky and Papert, 1969).

Par exemple, on peut se poser la question : comment stocker $\boldsymbol{x} = \{x_i\}_{i\leq N}$ bits (codés ± 1) d'une manière stable ? C'est-à-dire quelles sont les conditions pour que $\boldsymbol{x}$ soit invariant par l'action d'une itération d'évolution :

$$\forall i \leq N, \quad \rho(a_i(\boldsymbol{x})) = x_i \tag{24}$$

En somme *on cherche des points fixes* de la transformation d'évolution du système.

Par analogie, on peut concevoir l'évolution du réseau de Hopfield comme une relaxation d'un système physique vers un état de minimum local de l'énergie :

$$E(x) = -\frac{1}{2}\sum_{i,j} w_{ij}x_i x_j \tag{25}$$

(nb. le 1/2 tient compte de la symétrie de w_{ij}). On veux cependant pouvoir stoker P patterns ($\{x_i^{(p)}\}_{i\leq N, p\leq P}$), c'est-à-dire trouver *des états stables* que l'on peut reproduire à partir d'états légèrement différents. Ces patterns sont donc des *attracteurs* de la transformation d'évolution. L'image en terme de paysage de la fonction énergie, c'est de pouvoir créer des bassins attracteurs en forme d'entonnoirs qui soient profonds et le plus large possible.

En fait, les réseaux d'Hopfield ont des propriétés de *mémorisation* que l'on peut qualifier d'étonnantes et suscitent toujours des recherches et des mises en application (voir par ex. Ramsauer et al., 2020; Wen et al., 2009). Outre la mémorisation de patterns on peut citer pêle-mêle la reconnaissance de formes, la restauration d'images dégradées, l'optimisation combinatoire (ex. le problème du voyageur de commerce), la recherche d'informations dans des bases de données, le contrôle optimal de systèmes complexes (ex. gestion du trafic), en neurosciences le fonctionnement du cerveau, en physique statistique l'étude de verres de spins, etc. Ne pouvant pas être exhaustif dans ce court article, nous allons donc mettre en avant quelques propriétés de ces réseaux, concernant la mémorisation par exemple.

Il nous faut donc pouvoir trouver les poids w_{ij} à partir des P patterns donnés *a priori*. C'est un problème d'apprentissage supervisé.

3.1 Apprentissage de Hebb

La première méthode d'apprentissage qui a été utilisée est due à Donald Hebb qui propose en 1949 (Hebb, 1949) une loi d'accroissement de l'efficacité synaptique entre neurones lors d'une stimulation répétée et persistante [15]. Cette loi dans le contexte des réseaux d'Hopfield s'écrit [16]

$$w_{ij} = \frac{1}{N}\sum_{p=1}^{P} x_i^{(p)}x_j^{(p)} \tag{26}$$

15. nb. La persistance synaptique de long terme dans l'hippocampe de lapins a été mise en évidence au tournant des années 1970.

16. nb. la normalisation $1/N$ n'est pas importante en soit pour l'optimisation.

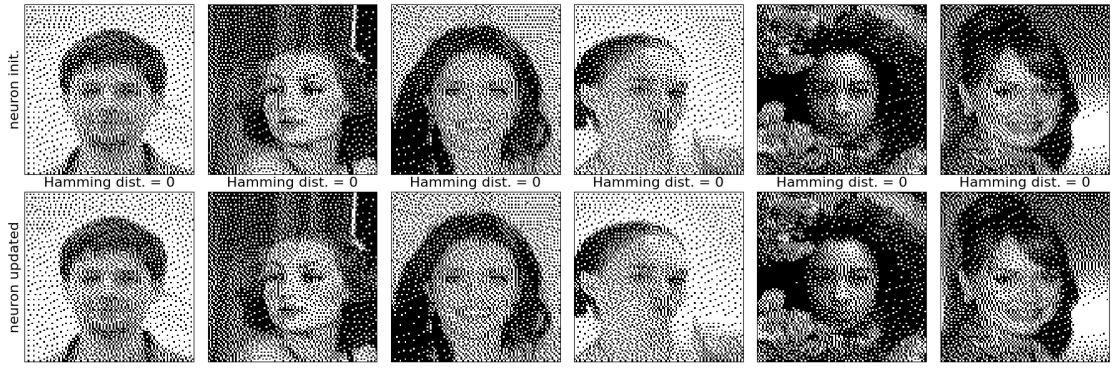

FIGURE 7 – Haut : Images binarisées (± 1 code blanc/noir) de 100×100 pixels ayant servies à calculer les poids par la méthode de Hebb d'un réseau de Hopfield de 10,000 neurones. Bas : Après une mise à jour asynchrone aléatoire des neurones, on constate que les 8 patterns sont stables. La distance de Hamming compte le nombre de pixels différents entre les images de la même colonne.

avec N le nombre de neurones du réseau, P le nombre de patterns, et tout en satisfaisant $w_{ii} = 0$ pour tout $i \leq P$. Notons que si l'on considère un premier pattern $x^{(1)}$ de dimension $N \times 1$, alors la matrice des poids peut se mettre sous la forme

$$\boldsymbol{w}^{(1)} = \frac{1}{N}\left(x^{(1)}x^{(1)T} - I_{N\times N}\right) \tag{27}$$

et l'énergie d'un état x est égale à

$$E(x) = -\frac{1}{2}x^T\boldsymbol{w}^{(1)}x = -\frac{1}{2N}(x^Tx^{(1)}x^{(1)T}x - x^Tx) \tag{28}$$

Or $x^Tx = \|x\|^2 = N$ car les composantes des vecteurs sont égales à ± 1, donc

$$E(x) = -\frac{1}{2N}|\langle x, x^{(1)}\rangle|^2 + \frac{1}{2} \tag{29}$$

Donc, l'énergie est minimale pour $x = x^{(1)}$, $x^{(1)}$ est un état stable et son énergie est égale à $-(N-1)/2$. La question qui vient alors est de connaître le nombre de patterns que l'on peut stocker de manière à ce qu'ils soient des minimums stables de l'énergie ?

Mémorisation : robustesse. Si on conçoit un réseau de $N = 100^2$ neurones, on peut stocker en fixant les poids par la loi de Hebb (Equation 26), par exemple 8 images binarisées de 100×100 pixels de visages issus du lot "CelebA" (Liu et al., 2015) (Figure 7). Il est tout à fait remarquable que l'on peut annuler 50% des poids w_{ij} (déconnecter des relations entre neurones), tout en gardant la matrice des poids symétrique, et pour autant les 8 patterns stockés sont toujours stables.

Maintenant, gardons le réseau initial de Hopfield dont les poids sont calculés à partir des 8 images binarisées qui sont des patterns stables. Prenons en une parmi ces 8 et annulons aléatoirement 99.7% de ses pixels, et initialisons les neurones du réseau avec cette version très dégradée. Procédons ensuite à une unique itération de mise à jour des états des neurones. Le résultat est donné sur la Figure 8 : on retrouve bien l'image initiale. Un autre test de robustesse est présenté sur la Figure 9. Partant de la même image initial (pattern stable), on se permet de basculer l'état de 48.0% de ses pixels tirés aléatoirement. Puis on procède comme précédemment à une unique itération de mise à jour des états du réseau : de nouveau on retrouve l'image initiale.

Ces tests numériques illustrent parfaitement la résilience d'un tel dispositif de mémorisation et le pouvoir attracteur des patterns mémorisés.

Mémorisation : capacité. Cependant, dès que l'on tente de rajouter de nouveau patterns au-delà des 8 initiaux, le nombre de patterns qui deviennent instables, c'est-à-dire qu'ils ne sont pas des points fixe de l'opération de mise à jour des états des neurones, augmente également comme la Figure 10 l'illustre pour le réseau utilisé précédemment. Aux environs de 30 patterns, tous sont instables. Cependant, si on répète cet exercice avec des patterns aléatoires (distribution de Rademacher), le réseau présente sa première instabilité au bout d'environ ≈ 400 patterns stockés (Figure 11). On note donc une limite capacitaire, cependant plus

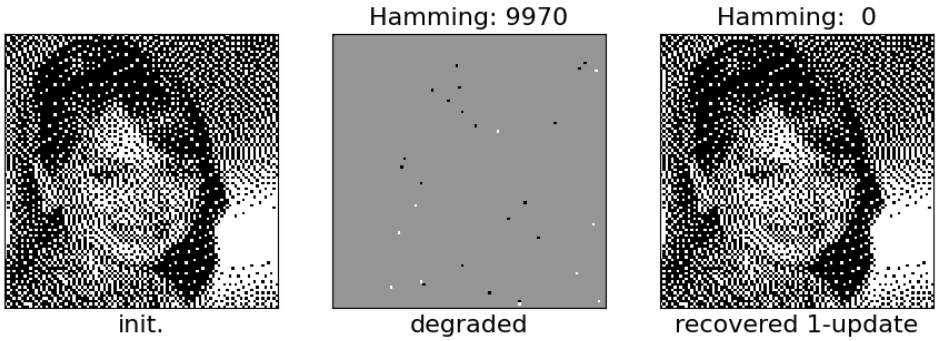

FIGURE 8 – A gauche : image originale. Au centre : image dégradée après avoir mis à 0 aléatoirement 99.7% des pixels. Cette image sert d'entrée au réseau de Hopfield dont les poids ont été calculée par la méthode de Hebb (Figure 7). Image de droite : état des neurones après une itération d'évolution du réseau : on retrouve l'image du pattern inaltéré, c'est une illustration du pouvoir "attracteur" de ces patterns stockés.

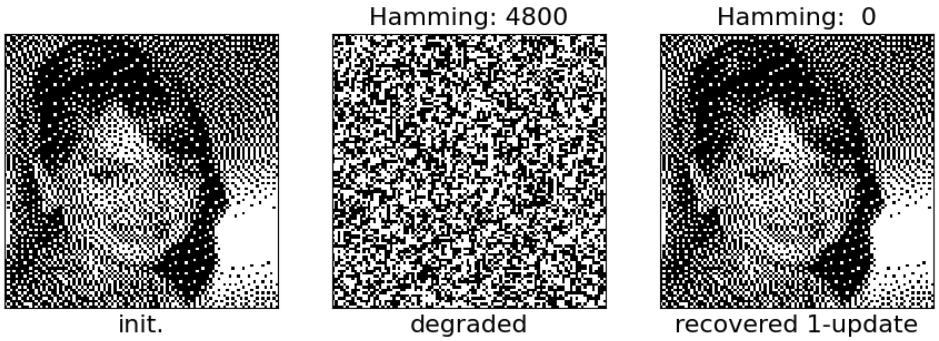

FIGURE 9 – Même légende que la Figure 8 sauf que cette fois-ci, on a mis à zéro 48% de l'état des neurones avant leur mise à jour.

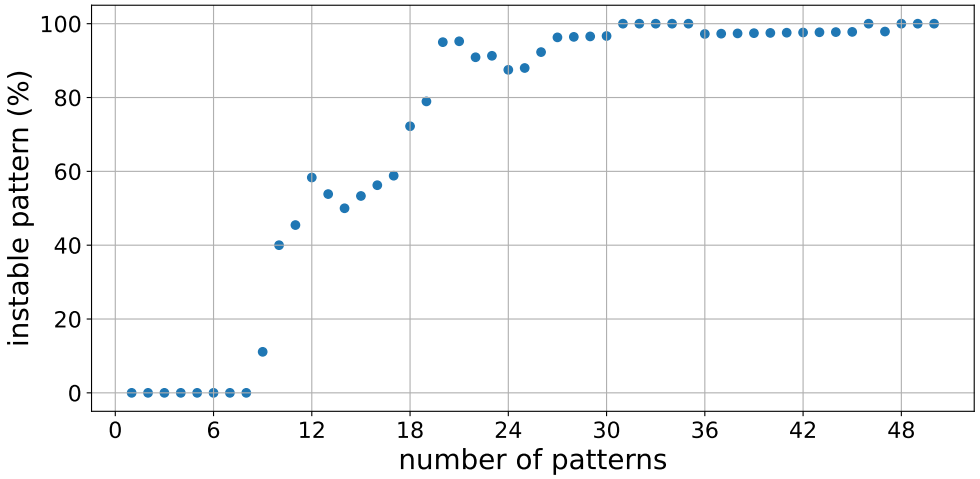

FIGURE 10 – Fraction de patterns (visages) instables en fonction du nombre de patterns que l'on tente d'intégrer dans le réseau à 10,000 neurones utilisé pour les Figures 7 à 9. A partir de 8 patterns, on constate que de plus en plus le réseau à de moins en moins de capacité à les mémoriser, c'est-à-dire qu'il y a de plus en plus de patterns instables par une itération de mise à jour des états des neurones.

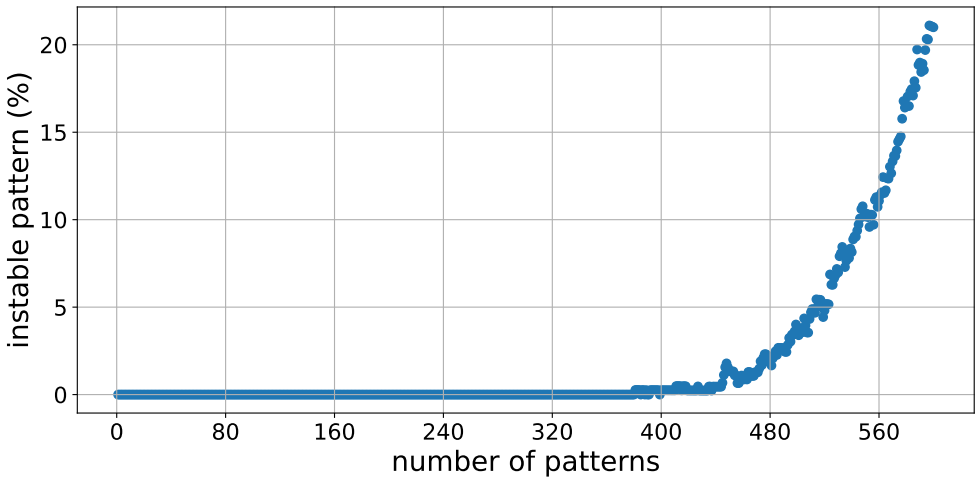

FIGURE 11 – Légende identique à la celle de la Figure 10 mais cette fois les patterns sont générés aléatoirement suivant la loi de Rademacher (loi binomiale de paramètre $p = 0.5$ avec des valeurs ± 1).

importante pour ces patterns aléatoires quelle ne l'est pour des visages. La différence principale réside dans l'existence de corrélations spatiales à différentes échelles pour ce qui concerne les visages, tandis qu'elles sont absentes dans les patterns aléatoires. C'est un effet étudié dans la littérature (voir par ex. Löwe, 1998; De Marzo and Iannelli, 2023). Notons cependant que beaucoup de propriétés des réseaux d'Hopfield sont estimées à partir des patterns aléatoires.

On peut tenter d'estimer le nombre maximal P_{max} de patterns que l'on peut stocker en requérant que la probabilité de bascule - après une itération et avoir initialisé le réseau avec n'importe quel pattern stocké - de n'importe quel bit de n'importe quel pattern soit inférieure à ε petit. On génère les patterns à N bits aléatoirement. Soit donc $\boldsymbol{x}^{(p)}$ un pattern parmi les P patterns stockés à cette étape, on initialise les états des N neurones avec ce pattern et l'on procède à une itération. Il vient alors pour le neurone i quelconque du réseau :

$$a_i = \sum_{j \neq i} w_{ij} x_j^{(p)} \tag{30}$$

Or, les poids w_{ij} sont obtenus à partir de la loi de Hebb, et on peut isoler la contribution du pattern p :

$$w_{ij} = \frac{1}{N} x_i^{(p)} x_j^{(p)} + \frac{1}{N} \sum_{k \neq p} x_i^{(k)} x_j^{(k)} \tag{31}$$

Donc

$$a_i = \underbrace{\frac{1}{N} \sum_{j \neq i} x_i^{(p)} (x_j^{(p)})^2}_{T_1} + \underbrace{\frac{1}{N} \sum_{j \neq i} \sum_{k \neq p} x_i^{(k)} x_j^{(k)} x_j^{(p)}}_{T_2} \tag{32}$$

Le premier terme se simplifie en faisant remarquer que $(x_j^{(p)})^2 = 1$, on a donc

$$T_1 = \frac{N-1}{N} x_i^{(p)} \approx x_i^{(p)} \tag{33}$$

Remarquons que si le second terme (T_2) n'était pas présent, le neurone i ne changerait pas d'état. Ainsi, la propension à changer d'état vient de ce second terme. Or, le produit de trois nombres aléatoires indépendants suivant une loi Rademacher suit également une loi de Rademacher. La somme partielle de n nombres générés par une loi de Rademacher n'est autre qu'une simple marche aléatoire sur $\mathbb{Z}$, donc asymptotiquement elle suit une loi normale $\mathcal{N}(0, \sigma = \sqrt{n})$. Ainsi, a_i suit une loi normale (si N et P sont grands) [17]

$$a_i \sim \mathcal{N}\left(x_i, \sigma = \sqrt{\frac{P}{N}}\right) \tag{34}$$

17. nb. $T_2 \approx z/N$ avec $z \sim \mathcal{N}(0, \sigma = \sqrt{NP})$, donc $T_2 \sim \mathcal{N}(0, \sigma = \sqrt{P/N})$.

Imaginons que $x_i = 1$ (le cas $x_i = -1$ est identique), alors la probabilité qu'il devienne négatif et induit un flip du neurone vers la valeur -1, est égale à

$$P(a_i < 0) = \int_{-\infty}^{0} \exp\left(-\frac{1}{2}\left(\frac{x-1}{\sigma}\right)^2\right) \frac{dx}{\sigma\sqrt{2\pi}} = \frac{1}{2}\text{erfc}\left(\sqrt{\frac{N}{2P}}\right) \tag{35}$$

Cela exprime que si on fixe à 1% la probabilité qu'un neurone particulier bascule lors de première itération, alors le nombre de patterns doit approximativement correspondre environ à 18% du nombre de neurones du réseau ($P \approx 0.18N$). Quand on essaye d'inclure plus de patterns, il y a des effets collectifs qui font basculer plusieurs neurones.

On peut maintenant utiliser une forme asymptotique de la probabilité [18] qu'il y ait au moins un neurone de P patterns qui bascule soit égale à ε, ce qui fixe la valeur du nombre maximal de patterns admissibles P_{max}

$$\frac{P_{max}}{N} \approx \frac{1}{4\log N + 2\log(1/\varepsilon) + 3\log(P_{max}/N)} \tag{36}$$

Pour $\varepsilon = 1\%$ et $N = 10^4$, on obtient alors $P_{max} \approx 0.03N \approx 300$ ce que l'on constate en effet dans notre expérimentation numérique [19]. Donc, on fait bien le constat que le réseau mémorise beaucoup plus de patterns randomisés que des patterns structurés comme les visages. D' On montre que les patterns qui sont les mieux à même d'être stockés doivent être indépendants et identiquement distribués selon la loi de Rademacher.

Asymptotiquement, le nombre maximum de patterns randomisés stables dans un réseau de N neurones (selon notre définition) est (voir Ramsauer et al., 2020, et références incluses)

$$P_{max} \approx O(\frac{N}{\log N}) \tag{37}$$

L'étude complète des réseaux d'Hopfield discrets mène à se poser la question par exemple de l'existence d'états stables avec quelques bits différents des patterns voulus *(états de verres de spins)*, ou bien de l'existence d'états stables supplémentaires, l'existence d'états stables résultats de combinaisons linéaires de patterns voulus etc. Cela met en évidence une classification d'états selon la valeur de P/N. On montre (voir par ex. Crisanti et al., 1986; Hertz, 2018) que si on accepte un peu d'erreur, il y a un seuil assez abrupte pour $P \approx 0.138N$ au-delà duquel le réseau ne mémorise pas. Comme chaque pattern à N bits et qu'il y a environ $N^2/2$ poids, on peut dire qu'*un réseau de Hopfield "classique" stocke de l'ordre de* 0.28 *bits/poids.*

3.2 Apprentissage à-la-perceptron

Ce qui précède est en partie basé sur l'apprentissage de Hebb. Cependant, envisageons un neurone pris isolément. Il a $N-1$ entrées et 1 sortie avec une fonction d'activation, et sa tâche est la suivante : considérant un pattern $x^{(k)}$ que l'on veut fixer, si les $N-1$ entrées du neurones sont les vraies valeurs $\{x_j^{(k)}\}_{j\neq i, j\leq N}$ alors on aimerait que la sortie du neurone soit $x_i^{(k)}$. Ainsi, pour chaque neurone, il possède des entrées et une cible parfaitement définies qui peuvent donc servir à un apprentissage supervisé. Il est alors naturel d'utiliser la fonction de coût d'un perceptron (Section 2) légèrement remaniée

$$J_r(w) = -\sum_{i=1}^{N}\sum_{k=1}^{P}\left\{y_i^{(k)}\log\Big(h_w(x_i^{(k)})\Big) + (1-y_i^{(k)})\log\Big(1-h_w(x_i^{(k)})\Big)\right\} + \frac{\alpha}{2}\|w\|^2 \tag{38}$$

avec

$$y_i^{(k)} = \begin{cases} 1 & \text{si} \quad x_i^{(k)} = +1 \\ 0 & \text{si} \quad x_i^{(k)} = -1 \end{cases} \tag{39}$$

et

$$h_w(x_i^{(k)}) = \frac{1}{1 + e^{-\beta\sum_{j=1}^{N} w_{ij}x_j^{(k)}}} \tag{40}$$

où l'on a introduit un paramètre β qui n'est pas sans évoquer le facteur $1/kT$ de Physique Statistique et qui régule la pente à l'origine de la sigmoïde. Pour minimiser la fonction de coût J_r, on utilise une méthode de

18. $\text{erfc}(x) \underset{x\to\infty}{\approx} e^{-x^2}/(x\sqrt{x})$.

19. L'approximation (Equation 36) donne $P_{max} = 283$ tandis que la solution "exacte" donne $P_{max} = 298$.

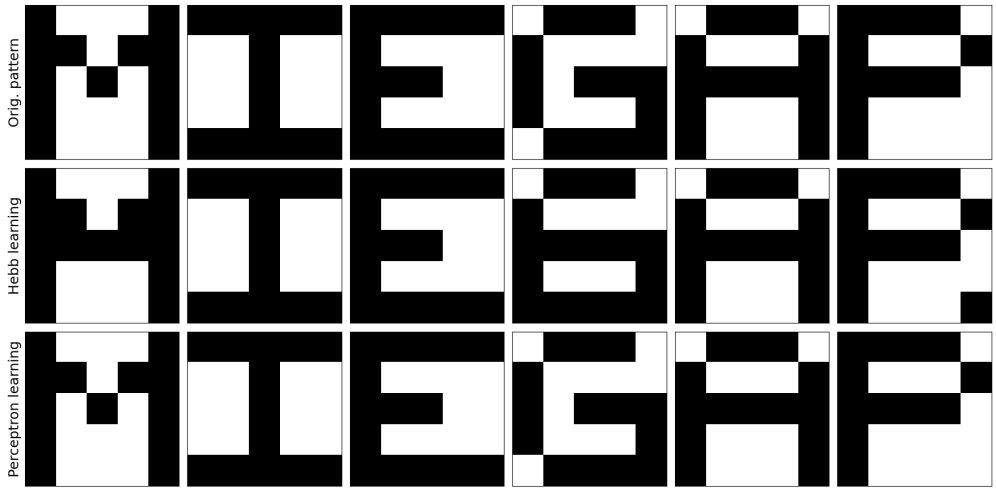

FIGURE 12 – Illustration de la capacité supérieure d'un réseau de Hopfield à 25 neurones en utilisant l'apprentissage "à-la-perceptron" comparé à l'apprentissage de Hebb. En haut : les 6 patterns que l'on veut fixer. Au milieu : résultat après un apprentissage de Hebb, on remarque que seuls 3 patterns enregistrés sont 100% stable. En bas : le résultat après un apprentissage "à-la-perceptron" avec $\beta = 1.4$, un "learning rate" de 0.01, et un "weight decay" de 0.01. Figure inspirée du livre de MacKay (2003).

descente de gradient avec un "learning rate" (ℓ_r). Ainsi pour un pas de minimisation, nous avons l'équation d'évolution suivante dans le cas de la sigmoïde [20] :

$$\begin{aligned} w(t+1) &= w(t) - \ell_r \nabla_w J_r(w) \\ \nabla_w J_r(w) &= -\beta \sum_{i,k} x_i^{(k)} (y_i^{(k)} - h_w(x_i^{(k)})) + \alpha w \end{aligned} \tag{41}$$

D'un point de vue pratique notons que l'on peut absorber le terme β dans les définitions de ℓ_r et α.

Un test simple est illustré sur la Figure 12 avec un petit réseau de 25 neurones qui mémorise des patterns 5×5 qui ne peut mémoriser plus de 4 patterns par la méthode de Hebb classique. Par contre, par la méthode de minimisation de J_r, on arrive à lui faire mémoriser 6 patterns soit un gain assez net. Pour ce qui concerne le réseau à $N = 10,000$ neurones utilisés précédemment, une expérimentation numérique montre que l'on peut stocker un peu plus de 3,000 patterns aléatoires 100×100 soit un gain d'un facteur 10 par rapport à l'apprentissage de Hebb. Donc, on voit clairement que la capacité des réseaux peut être augmentée en changeant l'apprentissage des poids.

Dans la littérature, il y a d'autres méthodes d'apprentissage explorant également l'effet des biais et des termes de self-coupling ou encore des connexions non-symétriques (Tolmachev and Manton, 2020).

3.3 Autres fonctions d'activation

Dans la section précédente, la méthode de Hebb d'apprentissage des poids est changée pour une version "à-la-perceptron", par contre la fonction d'activation du neurone i est celle d'origine à savoir

$$a_i(\boldsymbol{x}) = \rho(\sum_j w_{ij} x_j) \tag{42}$$

où $\rho(x)$ est essentiellement la fonction "signe" ou des formes différentiables, afin de donner la dynamique du système une fois les poids optimisés. Les versions dites "modernes" des réseaux d'Hopfield clairement utilisent d'autres fonctions d'activations, et cela leur confère des capacités augmentées.

20. Le terme en α est souvent appelé "weight decay" dans la littérature (Bishop, 2006)

Par exemple, en 2016 Dmitry Krotov et John J Hopfield (Krotov and Hopfield, 2016) étudient un nouveau type de mémoire associative, *"dense associative memory"* (ou DAM), en prenant la fonction suivante

$$a_i(\boldsymbol{x}) = \text{sgn}\left(\sum_{p=1}^{P}[F(1.x_i^{(p)} + \sum_{j\neq i} x_j^{(p)}x_j)] - F((-1).x_i^{(p)} + \sum_{j\neq i} x_j^{(p)}x_j)]\right) \tag{43}$$

où $\{\boldsymbol{x}^p\}_{p\leq P}$ sont les patterns que l'on veut stocker, et F est une fonction "smooth" de $\mathbb{R}$ dans $\mathbb{R}$. Notons que si $F(x) = x^2$, alors

$$a_i(\boldsymbol{x}) = \text{sgn}\left[\sum_{p=1}^{P}\sum_{j\neq i} x_i^{(p)}x_j^{(p)}x_j\right] \tag{44}$$

qui n'est autre que la version classique et ne change donc pas les propriétés. Le point est que ce passe-t'il si maintenant on utilise une forme polynomiale $F(x) = x^n$? Il vient

(Krotov & Hopfield (2016)) *Dans le cas où la fonction d'activation Equation 43 est gouvernée par une fonction $F(x) = x^n$, alors considérant des patterns dont les bits suivent une loi de Rademacher, nous avons les résultats suivants concernant la probabilité P_{error} qu'un seul neurone soit instable (K et N grand)*

$$P_{error} \approx \sqrt{\frac{(2n-3)!!}{2\pi}\frac{P}{N^{n-1}}}\exp\left(\frac{N^{n-1}}{2P(2n-3)!!}\right) \tag{45}$$

et le nombre maximal P_{max} de pattern mémorisé peut être mis sous la forme $P_{max} = \alpha_n(P_{error})N^{n-1}$. Notamment, Si $P_{error} = 0.5\%$ et $n = 2$ alors on retrouve $P_{max} \approx 0.14N$ du réseau standard, et si $P_{error} < 1/N$ qui correspond à une mémoire pratiquement sans erreur alors

$$P_{max} \approx \frac{1}{2(2n-3)!!}\frac{N^{n-1}}{\log N} \tag{46}$$

Cette borne est à comparer à la formule obtenue classiquement (Equation 37). A titre d'exemple si $N = 10^4$ et $n = 3$, alors $P_{max} \approx 200\ N$ qui montre l'amélioration possible de la capacité de stockage.

Fort de ce résultat une question se pose alors : que se passe-t'il quand $n \to \infty$ voire même quand la fonction de $F = e^x$? Mete Demircigil et collaborateurs en 2017 ont montré le théorème suivant (Demircigil et al., 2017) :

(Demircigil et al, (2017)) *Soit le cas où la fonction d'activation Equation 43 est gouvernée par la fonction $F(x) = e^x$ Considérons $0 < \alpha < \log(2)/2$, et $\{\boldsymbol{x}^{(p)}\}_{p\leq P}$ tel que $P = e^{\alpha N} + 1$ (N est le nombre de neurones du réseau), des patterns i.i.d de Rademacher, et $\rho \in [0, 1/2[$. Alors $\forall p \leq P$ et $\forall \tilde{\boldsymbol{x}}^{(p)} \in \mathcal{S}(\boldsymbol{x}^{(p)}, \rho N)$, la sphère de Hamming centrée sur $\boldsymbol{x}^{(p)}$ et de rayon supposé entier ρN, on a*

$$\mathbb{P}(\exists q\ \exists j,\ \text{tq. } a_j(\tilde{\boldsymbol{x}}^{(p)}) \neq \boldsymbol{x}_j^{(p)}) \to 0 \tag{47}$$

si α est choisi tel que

$$\alpha < \frac{I(1-2\rho)}{2} \tag{48}$$

avec [a]

$$I(x) = \frac{1}{2}\{(1+x)\log(1+x) + (1-x)\log(1-x)\} \tag{49}$$

a. On reconnaît à un facteur 1/2 près l'entropie de Shannon d'un processus de Bernoulli de probabilité x pour l'une des ceux valeurs. $I(1) = \log(2)$, $I(0) = 0$.

La *sphère de Hamming* du nom de Richard W. Hamming est basée sur la distance éponyme qui nous a servie à calculer la différence en nombre de bits entre deux patterns (Section 3). Cette notion se généralise à toute sorte d'alphabet. C'est réellement une distance, et elle intervient en théorie des codes correcteurs. En l'occurrence, les réseaux de Hopfield "corrigent" les patterns qui diffèrent légèrement d'un pattern stable.

Le théorème implique que pour de grandes valeurs de N, alors presque sûrement tous les patterns sont des points fixes de la dynamique du réseau et le nombre maximal de patterns que l'on peut stocker est de l'ordre

de ($\alpha < I(1)/2$)

$$P_{max} \approx 2^{N/2} \tag{50}$$

ce qui est un accroissement considérable par rapport au modèle de Hopfield classique. A titre d'exemple avec le réseau que nous avons utilisé pour stocker des images binarisées 100×100, $N = 10^4$ et donc $P_{max} \approx 1.4\ 10^{1505}$. En comparaison, le nombre de protons dans l'Univers que l'on estime à $\approx 10^{80}$ devient un nombre très modeste. Cependant, en contrepartie de ce nombre colossal d'états stables, le rayon de la sphère de Hamming centrée sur chaque pattern, tend vers 0, ce qui en fait donc des attracteurs très localisés.

En passant, la démonstration du théorème fait appel à des résultats de la Théorie des Grandes Déviations introduite en 1966 par S. R. S. Varadhan (Varadhan, 2008). Elle concerne le comportement asymptotique de queues de distributions. En particulier, concernant n variables $(X_i)_{i \leq n}$ i.i.d suivant la loi de Rademacher, si l'on forme leur somme $S_n = \sum_i X_i$, alors

$$\forall x \in]0, 1[, \quad \mathbb{P}(S_n \geq nx) \leq e^{-nI(x)} \tag{51}$$

avec $I(x)$ la fonction donnée dans le théorème. Ce type d'inégalité quantifie la déviation de variables aléatoires par rapport à une valeur fixe (ex. leur espérance), elles sont qualifiées *"d'inégalités de concentrations"*.

3.4 Réseaux de Hopfield continus : analogie avec le modèle d'Ising

Un aspect qui a vite émergé après l'introduction des réseaux discrets de Hopfield tient au fait que la fonction "sgn" (signe) n'est pas différentiable. Ce point peut être levé si l'on confère au neurone la possibilité de sortir un réel dans l'intervalle $[-1, 1]$, et non juste les valeurs ± 1. Nous avons déjà abordé en fait ce point en utilisant la fonction $h_w(x)$ (Figure 3) lors de l'apprentissage "à-la-perceptron" (Section 3.2). Dans la littérature, il s'agit de *réseaux de Hopfield continus*, notion introduite par J.J Hopfield en 1984 (Hopfield, 1984). En particulier, il y a un lien naturel avec le modèle d'Ising [21].

C'est en 1974 que William A. Little a montré l'apparition d'états persistants en lien avec des effets à longue distance dans le modèle d'Ising (Little, 1974). En fait, l'étude des verres de spins par leur liens avec les réseaux de neurones à gagner en popularité après les travaux de J.J Hopfield. Dans ce cadre, la fonction non linéaire d'activation est

$$x_i = \tanh(\beta a_i) = \rho_\beta(a_i) = \rho(\beta a_i) \tag{52}$$

avec $\beta = 1/T$ ($k = 1$) où T peut être vue comme la température du système. Si $T \to 0$ alors la fonction d'activation à pour limite la fonction $\text{sgn}(x)$ des réseaux précédents. La dynamique des états des neurones du système est modélisée par une équation différentielle

$$\frac{dx_i}{dt} = -x_i(t) + \rho_\beta(a_i(t)) \tag{53}$$

Quand le système atteint un point stable pour tous les neurones ($\forall i,\ dx_i/dt = 0$) alors $x_i(t) = \rho_\beta(\sum_j w_{ij} x_i(t)) = \rho_\beta(a_i(t))$. Il se trouve que ce système dynamique possède une fonction de Liapounov (pourvu que la matrice des poids w_{ij} soit symétrique) dont l'expression est (voir par ex. Šíma and Orponen, 2003)

$$E = -\frac{1}{2} \sum_{ij} w_{ij} x_i x_j + \frac{1}{\beta} \sum_i \int_0^{x_i} \rho^{-1}(x)\ dx \tag{54}$$

$$= -\frac{1}{2} \sum_{ij} w_{ij} x_i x_j + \frac{1}{\beta} \sum_i I(x_i) \tag{55}$$

avec $I(x)$ la fonction du théorème 2.

En fait, les relations d'activation et d'actualisation de manière asynchrone des états des neurones sont tout à fait équivalentes à celles obtenues dans la méthode variationnelle de spins indépendants appliquée au modèle d'Ising, et l'expression ci-dessus n'est autre alors que l'énergie libre de Gibbs (Jain et al., 2018).

21. En bref, le "modèle d'Ising" dans sa version originale, et un modèle de spins en interaction introduit par Wilhelm Lenz en 1920 et que Ernest Ising (son étudient) a résolu en 1D uniquement et a pu montrer l'absence de transition de phase. La résolution exacte en dimension 2 de Lars Onsager en 1944 a permis quant à elle primo de prouver l'existence d'une transition de phase, par exemple celle du (para-ferro)magnétisme, avec les moyens de la Physique Statistique, secundo de comprendre pourquoi en 1D il ne pouvait y en avoir, et tertio à ouvert de nouvelles approches pour l'étude des exposants critiques dans les années 70.

Nous n'allons pas poursuivre les nombreuses études des réseaux continus d'Hopfield à la fois par exemple en physique statistique, ou pour des applications techniques et problèmes d'optimisations. L'article de H. Ramsauer et al. déjà mentionné (Ramsauer et al., 2020) utilise une nouvelle fonction d'activation qui n'est autre que le *mécanisme d'attention* du transformer `BERT` (Devlin, 2018; Kenton and Toutanova, 2019). Cela permet de concevoir des nouvelles couches de réseaux de neurones profonds, telles (sic) *"qu'elles permettent de nouvelles approches de l'apprentissage profond, au-delà des réseaux entièrement connectés, convolutionnels ou récurrents, et offrent des mécanismes de regroupement, de mémoire, d'association et d'attention."*. Donc, on constate que les réseaux de J.J Hopfield on développé tout un champ disciplinaire.

Nous allons à présent nous tourner vers des réseaux qui constitue une continuation des ces développements en implémentant directement une distribution de Gibbs.

4 Les Machines de Boltzmann

Nous avons vu dans la Section 3.4 la version des réseaux d'Hopfield où l'on considère la fonction $x_i = \tanh(a_i)$ pour actualiser l'état d'un neurone (x_i) à la suite de son activation $a_i = \sum_j w_{ij} x_j$. Ce type de modèle a une conception qui rappelle celle du modèle d'Ising [22] où l'on cherche à minimiser l'énergie d'un système de spins *inter-dépendants* [23]

$$E(\boldsymbol{x}) = -\frac{1}{2}\boldsymbol{x}^T \boldsymbol{W}\boldsymbol{x} = -\frac{1}{2}\sum_{i,j} w_{ij} x_i x_j \tag{56}$$

Le réseau d'Hopfield permet de mettre en œuvre une solution approchée optimisant une solution où les spins sont *indépendants*. Cependant, il est légitime de se demander s'il n'existe pas un moyen de résoudre le problème original à l'aide d'un réseau de neurones qui satisferait directement la distribution de Boltzmann-Gibbs [24]

$$P(\boldsymbol{x}|\boldsymbol{W}) = \frac{1}{Z(\boldsymbol{W})} e^{-E(\boldsymbol{x};\boldsymbol{W})} = \frac{1}{Z(\boldsymbol{W})} \exp\left\{\frac{1}{2}\boldsymbol{x}^T \boldsymbol{W}\boldsymbol{x}\right\} \tag{57}$$

avec la fonction de partition s'écrivant formellement

$$Z(\boldsymbol{W}) = \sum_{\boldsymbol{x}} \exp\left\{\frac{1}{2}\boldsymbol{x}^T \boldsymbol{W}\boldsymbol{x}\right\} \tag{58}$$

où la somme sur $\boldsymbol{x}$ considère toutes les configurations de bits possibles. Pour ce faire, un nouveau type d'architecture neuronale a été introduite pas D. Ackley, G. Hinton et T. Sejnowski en 1985 (Ackley et al., 1985), pourtant dans un contexte traitant d'architectures parallèles. Notons que les auteurs répondent également à l'article de Minsky & Paper de 1969 (Minsky and Papert, 1969) et motiva la réédition amendée de 1988 (Minsky and Papert, 1988). Il y a différentes variantes de ces architectures, cependant ce sont avant tout des réseaux de neurones inter-connectés pour lesquels :

- l'activation d'un neurone i est identique à celle d'un réseau de Hopfield

$$a_i(\boldsymbol{x}) = \sum_j w_{ij} x_j \tag{59}$$

  avec en général $w_{ii} = 0$ et $w_{ij} = w_{ji}$, et rappelons que l'on peut toujours mettre des biais w_{i0} par l'intermédiaire d'un neurone d'activité immuable $x_0 = 1$;
- l'actualisation du neurone est stochastique, et pour des états binaires 0/1, nous avons les probabilités suivantes [25]

$$P(x_i = +1) = \frac{1}{1 + e^{-a_i}} \qquad P(x_i = 0) = 1 - P(x_i = +1) \tag{60}$$

Notons au passage que la différence d'énergie nécessaire pour faire basculer d'un état $+1$ à un état 0 pour le neurone i est donnée d'une part selon

$$\Delta E_i = E(x_i = 0) - E(x_i = +1) = a_i \tag{61}$$

22. voir note 21
23. Concernant le modèle d'Ising, il est d'usage que les poids w_{ij} soient notés J_{ij} et ils sont interprétés comme des interactions entre spins i et j, lesquels sont notés σ_i au lieu de x_i.
24. Dans la plus part des cas, le facteur β est omis.
25. nb. Nous optons pour un modèle d'état "on/off" d'un neurone représenté par 1-bit ($x_i \in \{0, 1\}$). Si nous avions pour $x_i = \pm 1$, il y aurait un facteur 2 dans le poids de accordé à a_i.

et d'autre part d'après la distribution de Gibbs

$$\Delta E_i = \log\left(\frac{p_+}{1-p_+}\right) \qquad \text{avec} \quad p_+ = P(x_i = +1) \tag{62}$$

ce qui redonne fonction logistique (Equation 60) qui découle donc directement de la distribution de Gibbs.

Si l'on dispose d'un lot d'entraînement qui se présente sous forme de patterns indépendants et identiquement distribués à N bits comme pour le réseau d'Hopfield, $\mathcal{D}_{tr} = \{\boldsymbol{x}^{(p)}\}_{p\leq P}$, on peut utiliser la formulation bayésienne pour obtenir la distribution *a posteriori* des poids afin de les optimiser. Il vient alors

$$P(\boldsymbol{W}|\mathcal{D}_{tr}) = \frac{P(\mathcal{D}_{tr}|\boldsymbol{W}) \times P(\boldsymbol{W})}{P(\mathcal{D}_{tr})} \tag{63}$$

avec $P(\boldsymbol{W})$ un *prior* sur les poids et $P(\mathcal{D}_{tr}|\boldsymbol{W})$ est le likelihood que l'on cherche à maximiser. Il fait intervenir la fonction de partition $Z(\boldsymbol{W})$ selon

$$\log P(\mathcal{D}_{tr}|\boldsymbol{W}) = \log \prod_{p=1}^{P} P(\boldsymbol{x}^{(p)}|\boldsymbol{W}) = \sum_p \left(\frac{1}{2}\boldsymbol{x}^{(p)T}\boldsymbol{W}\boldsymbol{x}^{(p)} - \log Z(\boldsymbol{W})\right) \tag{64}$$

Or, comme la matrice $\boldsymbol{W}$ est symétrique, on montre facilement que [26]

$$\frac{\partial \log Z(\boldsymbol{W})}{\partial w_{ij}} = \sum_{\boldsymbol{x}} x_i x_j P(\boldsymbol{x}|\boldsymbol{W}) = \mathbb{E}_{\boldsymbol{x}\sim P(\boldsymbol{x}|\boldsymbol{W})}[x_i x_j] = \langle x_i x_j \rangle_{P(\boldsymbol{x}|\boldsymbol{W})} \tag{65}$$

Donc, le gradient du log-likelihood s'écrit alors

$$\frac{1}{P}\frac{\partial \log P(\mathcal{D}_{tr}|\boldsymbol{W})}{\partial w_{ij}} = \frac{1}{P}\sum_p \left(\boldsymbol{x}_i^{(p)}\boldsymbol{x}_j^{(p)} - \langle x_i x_j\rangle_{P(\boldsymbol{x}|\boldsymbol{W})}\right) = \langle x_i x_j\rangle_{\mathcal{D}_{tr}} - \langle x_i x_j\rangle_{P(\boldsymbol{x}|\boldsymbol{W})} \tag{66}$$

où l'on a noté

$$\langle x_i x_j\rangle_{\mathcal{D}_{tr}} = \frac{1}{P}\sum_p \boldsymbol{x}_i^{(p)}\boldsymbol{x}_j^{(p)} \tag{67}$$

L'actualisation des poids par montée de gradient prend donc la forme suivante

$$w_{ij}^{t+1} = w_{ij}^t + \ell_r\left(\langle x_i x_j\rangle_{\mathcal{D}_{tr}} - \langle x_i x_j\rangle_{P(\boldsymbol{x}|\boldsymbol{W})}\right) \tag{68}$$

Le gradient du log-likelihood est la différence entre la corrélation $\langle x_i x_j\rangle$ estimée à partir du lot d'entraînement, et celle estimée à partir du modèle en court d'optimisation. La première corrélation est similaire à la méthode de Hebb, et correspond exactement à ce qui est fait après une itération avec un *prior* $W = 0$. Cependant, par la suite le second terme prend de l'importance et son estimation se fait par Monte Carlo pour échantillonner la densité de probabilité de Gibbs (Equation 57). Notons que le gradient du log-likelihood devient nul quand la corrélation observée dans le modèle coïncide avec celle observée sur le lot d'entraînement.

Remarquons que cette méthode de maximum de vraisemblance peut être formulée autrement. En requérant que la distribution du modèle ($P(\boldsymbol{x}|\boldsymbol{W})$) tende vers la distribution empirique des échantillons du lot d'entraînement, $P(\boldsymbol{x}|\mathcal{D}_{tr}) = \frac{1}{P}\sum_p \delta(\boldsymbol{x} - \boldsymbol{x}^{(p)})$, on minimise la divergence de Kullback-Leibler [27] notée D_{KL} :

$$\begin{aligned} D_{KL}(P(\boldsymbol{x}|\mathcal{D}_{tr})||P(\boldsymbol{x}|\boldsymbol{W})) &= \mathbb{E}_{\boldsymbol{x}\sim P(\boldsymbol{x}|\mathcal{D}_{tr})}\left[\log\frac{P(\boldsymbol{x}|\mathcal{D}_{tr})}{P(\boldsymbol{x}|\boldsymbol{W})}\right] \\ &= \mathbb{E}_{\boldsymbol{x}\sim P(\boldsymbol{x}|\mathcal{D}_{tr})}[\log P(\boldsymbol{x}|\mathcal{D}_{tr})] - \mathbb{E}_{\boldsymbol{x}\sim P(\boldsymbol{x}|\mathcal{D}_{tr})}[\log P(\boldsymbol{x}|\boldsymbol{W})] \\ &= -\frac{1}{P}\sum_p \left(\frac{1}{2}\boldsymbol{x}^{(p)T}\boldsymbol{W}\boldsymbol{x}^{(p)} - \log Z(\boldsymbol{W})\right) + C \end{aligned} \tag{69}$$

avec C une constante indépendante de $\boldsymbol{W}$ qui n'intervient pas dans l'optimisation de cette divergence, laquelle donne les mêmes poids $\boldsymbol{W}$ que l'optimisation du likelihood.

26. nb. on note $\langle x\rangle_{p(x)}$ la moyenne de la variable aléatoire x selon sa distribution de probabilité $p(x)$.

27. Voir par ex. Jean-Eric Campagne. Notes et commentaires au sujet des conférences de S. Mallat du Collège de France (2024). Cours de Master. Apprentissage et génération par échantillonnage aléatoire, `https://hal.science/hal-04549633v2`

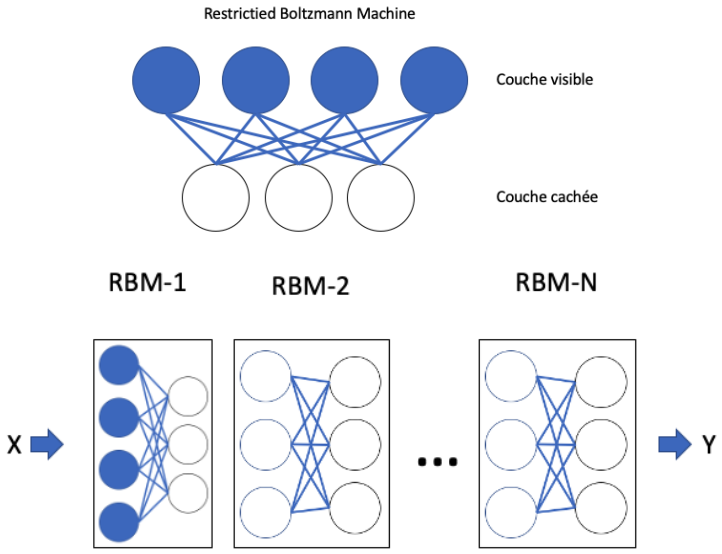

FIGURE 13 – Haut : Illustration d'une machine de Boltzmann à 1 couche cachée où ne sont considérées que des connections entre les neurones "visibles" et les neurones "cachées". C'est ce qu'on appelle une Restricted Boltzmann Machine ou RBM. Bas : une collection de RBM qui sont entraînées l'une après l'autre. C'est une technique d'apprentissage progressif des "grands" réseaux de neurones utilisant les fonctions d'activation de type tanh.

4.1 Introduction des neurones cachés

Le développement des machines de Boltzmann ne serait pas une (r)évolution majeure par rapport aux réseaux de Hopfield si on se contentait de la description de la statistique à deux points dans les données (soit $\langle x_i x_j \rangle$). La connaissance de cette statistique est uniquement suffisante pour des processus gaussiens. Mais qu'en est-il des statistiques pour décrire des images ? L'apport majeur de G. Hinton et al. est d'avoir utilisé des neurones cachés (*hidden neurons*) pour décrire les corrélations d'ordre supérieur (Ackley et al., 1985).

L'activation des neurones cachés est identique à celle des neurones que nous avons utilisé jusqu'à présent. La seule différence vient du fait que seuls les neurones "non-cachés" (ou *visibles*) sont en relation avec le "monde extérieur". En particulier, quand on donne un lot d'entraînement pour les neurones *visibles* ($\boldsymbol{x}$), $\mathcal{D}_{tr} = \{\boldsymbol{v}^{(p)}\}_{p \leq P}$, les états des neurones cachés ne sont pas spécifiés, on les note alors $\boldsymbol{h}$. Et pour simplifier les notations, l'état complet du réseau est donné par $\boldsymbol{z} = (\boldsymbol{v}, \boldsymbol{h})$.

Pour optimiser les poids, il nous faut tout d'abord une expression du likelihood, et en particulier pour un pattern $\boldsymbol{v}^{(p)}$, nous avons

$$P(\boldsymbol{v}^{(p)}|\boldsymbol{W}) = \sum_{\boldsymbol{h}} P(\boldsymbol{v}^{(p)}, \boldsymbol{h}|\boldsymbol{W}) = \frac{\sum_{\boldsymbol{h}} \exp\left\{\frac{1}{2}\boldsymbol{z}^T \boldsymbol{W} \boldsymbol{z}\right\}}{\sum_{\boldsymbol{v},\boldsymbol{h}} \exp\left\{\frac{1}{2}\boldsymbol{z}^T \boldsymbol{W} \boldsymbol{z}\right\}} \tag{70}$$

Le gradient par rapport à w_{ij} du log-likelihood global se calcule comme précédemment et l'on obtient à la place de l'Équation 66 l'expression suivante :

$$\frac{\partial \log P(\mathcal{D}_{tr}|\boldsymbol{W})}{\partial w_{ij}} = \sum_{p} \left(\langle z_i z_j \rangle_{P(\boldsymbol{h}|\boldsymbol{v}^{(p)}, \boldsymbol{W})} - \langle z_i z_j \rangle_{P(\boldsymbol{v},\boldsymbol{h}|\boldsymbol{W})} \right) \tag{71}$$

On retrouve remanié les deux termes dont le premier fixe les états $\boldsymbol{x}$ aux valeurs du lot d'entraînement laissant les états cachés "libres", et le second terme où les états de tous les neurones sont issus de la distribution jointe du modèle courant.

L'apprentissage des machines de Boltzmann est très consommateur de ressources ainsi par la suite, on distingue deux types de machines de Boltzmann : les RBM ("R" pour *restricted* ou *restreinte*) (Figure 13 haut) où l'on ne considère uniquement que les connections entre les neurones "visibles" et les neurones "cachés", et les DBM ("D" pour *Deep* ou *profond*) qui sont un enchaînement de RBM (Figure 13 bas). Historiquement, les

DBM ont joué un rôle important à la suite des RBM, dans le renouveau de l'usage "moderne" des réseaux de neurones après que G. Hinton, S. Osindero et Y. Teh aient publié en 2006 (Hinton et al., 2006) un algorithme rapide d'entraînement ressemblant à l'algorithme de back-propagation (voir par ex. Rumelhart et al., 1986; Lecun, 1988) qui a permis l'élaboration d'architectures profondes. Nous nous contenterons d'une étude sommaire avec les RBM.

4.2 Restricted Boltzmann Machine

Dans les conditions d'une RBM, l'énergie se met sous la forme

$$E(\boldsymbol{v}, \boldsymbol{h}, \boldsymbol{W}) = -\boldsymbol{v}^T \boldsymbol{W} \boldsymbol{h} \tag{72}$$

avec $\boldsymbol{W}$ une matrice réelle [28], de dimension $N \times H$ (H neurones "cachés", et on garde la notation N pour le nombre de neurones "visibles" [29]). Comme on peut le remarquer, l'énergie se simplifie grandement car il n'y a plus les termes quadratiques xx et hh. La minimisation de $-\log P(\mathcal{D}_{tr}|\boldsymbol{W})$ par une descente de gradient donne une étape de mise à jour du poids w_{ij} s'écrivant selon la méthode traditionnelle (Equation 69)

$$w_{ij}^{t+1} = w_{ij}^t + \ell_r \left(\langle h_i v_j \rangle_{P(\boldsymbol{h}|\boldsymbol{v}^{(p)}, \boldsymbol{W})} - \langle h_i v_j \rangle_{P(\boldsymbol{v}, \boldsymbol{h}|\boldsymbol{W})} \right) \tag{73}$$

Comme déjà mentionné, il faut recourir à des techniques de type Monte Carlo coûteuses pour estimer le second terme. G. Hinton développa alors une alternative qu'il baptise la méthode de *divergence de contraste* (*"contrastive divergence"*) (Hinton, 2002). L'idée est d'utiliser une chaîne de Markov qui débute avec les données d'entraînement et de l'arrêter au bout de n étapes afin de faire évoluer la distribution $P(\boldsymbol{v}, \boldsymbol{h}|\boldsymbol{W}^n)$ vers la distribution empirique. Dans le cas des RBM, une bonne approximation de mise à jour des poids a été élaborée est s'écrit simplement

$$w_{ij}^{t+1} = w_{ij}^t + \ell_r \left(\langle h_i v_j \rangle_{P(\boldsymbol{h}|\boldsymbol{v}^{(p)}, \boldsymbol{W})} - \langle h_i v_j \rangle_{P(\boldsymbol{v}, \boldsymbol{h}|\boldsymbol{W}^n)} \right) \tag{74}$$

De plus dans la plus part des cas, une seule étape ($n = 1$) suffit, on la note CD-1. On espère par une telle formulation de la mise à jour des poids que la direction du gradient soit la bonne. L'étude des conditions de convergence ont été élaborées par la suite (voir par ex. Sutskever and Tieleman, 2010).

L'algorithme d'optimisation par la méthode CD-1 peut se décliner ainsi (voir par ex. Hinton, 2012) :

1. A l'étape t, soit le lot d'entraînement $\mathcal{D}_{tr} = \{\boldsymbol{v}^{(p)}\}_{p \leq P}$ où chaque entrée est de dimension $1 \times N$
2. tirage d'un lot de $\{\boldsymbol{h}\}_{\leq P}$ (chacun de dimension $1 \times H$) selon $P(h|\{\boldsymbol{v}^{(p)}\}, \boldsymbol{W}^t)$;
3. calcule de $\delta W_+ = \boldsymbol{v}^{(p)^T} \boldsymbol{h}$;
4. tirage d'un lot de "faux" $\{\tilde{\boldsymbol{v}}\}_{\leq P}$ selon $P(v|\{\boldsymbol{h}\}, \boldsymbol{W}^t)$;
5. tirage d'un lot de nouveaux $\{\tilde{\boldsymbol{h}}\}_{\leq P}$ selon $P(h|\{\tilde{\boldsymbol{v}}\}, \boldsymbol{W}^t)$;
6. calcule de $\delta W_- = \tilde{\boldsymbol{v}}^T \tilde{\boldsymbol{h}}$;
7. mise à jour les poids $\boldsymbol{W}^{t+1} = \boldsymbol{W}^t + \ell_r(\delta W_+ - \delta W_-)$ et on boucle à partir de l'étape 1 jusqu'à atteindre le nombre désiré d'itérations (*epochs*).

Notons que si l'on veut une approximation d'ordre CD-n, il suffit de boucler n-fois sur les opérations 4 et 5, en changeant $\boldsymbol{h}$ par $\tilde{\boldsymbol{h}}$ à chaque itération.

D'un point de vue pratique, on peut utiliser une mise à jour de $\boldsymbol{W}$ incluant un effet de *momentum* et une régularisation L2 (*weight decay*), et durant l'entraînement on peut suivre l'évolution de l'erreur de génération $\sum_{i \leq P} \|\boldsymbol{v}^{(i)} - \tilde{\boldsymbol{v}}_i\|^2$. Notons qu'une approche moderne viserait à optimiser directement cette erreur quadratique. Il reste à savoir comment effectuer les différents tirages des différentes étapes de l'algorithme. C'est là où le découplage entre les neurones "visibles" d'une part et les neurones "cachés" d'autre part simplifie la tâche. Si l'on veut calculer la probabilité conditionnelle des neurones "cachées" sachant le "reste" (neurones "visibles" et les poids $\boldsymbol{W}$), il vient

$$P(\boldsymbol{h}|\boldsymbol{v}, \boldsymbol{W}) = \frac{e^{\boldsymbol{v}^T \boldsymbol{W} \boldsymbol{h}}}{\sum_{\boldsymbol{h}} e^{\boldsymbol{v}^T \boldsymbol{W} \boldsymbol{h}}} \tag{75}$$

28. Attention : la matrice qui est symétrique et de diagonale nulle, concerne l'ensemble des neurones $\boldsymbol{z}$. Par contre, dans l'expression mentionnée $\boldsymbol{W}$ ne concerne que les interactions entre les deux types de neurones.
29. nb. les vecteurs de type "h" (resp. "v") sont de dimension $1 \times H$ (resp. $1 \times N$)

Or, la notation $\sum_{\boldsymbol{h}}$ signifie que l'on somme sur toutes les configurations des bits h_i :

$$\sum_{\boldsymbol{h}} := \sum_{h_1\in\{0,1\}} \sum_{h_2\in\{0,1\}} \cdots \sum_{h_H\in\{0,1\}} \tag{76}$$

ainsi

$$\sum_{\boldsymbol{h}} e^{\boldsymbol{v}^T\boldsymbol{W}\boldsymbol{h}} = \sum_{\boldsymbol{h}} \prod_{j=1}^{H} e^{\boldsymbol{v}^T\boldsymbol{W}_{\bullet_j}h_j} = \prod_{j=1}^{H} \sum_{h_j\in\in\{0,1\}} e^{\boldsymbol{v}^T\boldsymbol{W}_{\bullet_j}h_j} = \prod_{j=1}^{H} \left(1 + e^{\boldsymbol{v}^T\boldsymbol{W}_{\bullet_j}}\right) \tag{77}$$

Donc,

$$P(\boldsymbol{h}|\boldsymbol{v},\boldsymbol{W}) = \prod_{j=1}^{H} \frac{e^{\boldsymbol{v}^T\boldsymbol{W}_{\bullet_j}h_j}}{1 + e^{\boldsymbol{v}^T\boldsymbol{W}_{\bullet_j}}} = \prod_{j=1}^{H} P(h_j|\boldsymbol{v},\boldsymbol{W}) \tag{78}$$

On a alors une factorisation des probabilités conditionnelles qui rend la génération des bits h_i simple et de plus ($\sigma(x)$ correspond à la fonction sigmoïde)

$$P(h_j = 1|\boldsymbol{v},\boldsymbol{W}) = \sigma(\boldsymbol{v}^T\boldsymbol{W}_{\bullet_j}) = 1 - P(h_j = 0|\boldsymbol{v},\boldsymbol{W}) \tag{79}$$

Ainsi, il suffit de faire un tirage d'un nombre selon la loi uniforme sur $[0,1]$ et de le comparer à $\sigma(\boldsymbol{v}^T\boldsymbol{W}_{\bullet_j})$ pour déclarer le neurone h_i actif (ou non). Il en va de même pour les neurones visibles, on a

$$P(\boldsymbol{v}|\boldsymbol{h},\boldsymbol{W}) = \prod_{i=1}^{N} P(v_i|\boldsymbol{h},\boldsymbol{W}) \tag{80}$$

avec

$$P(v_i = 1|\boldsymbol{h},\boldsymbol{W}) = \sigma(\boldsymbol{W}_{i\bullet}\boldsymbol{h}) = 1 - P(v_i = 0|\boldsymbol{h},\boldsymbol{W}) \tag{81}$$

Illustrons l'algorithme de minimisation avec un lot d'entraînement iconique.

4.3 Exemples d'usages avec le lot "MNIST"

Pour montrer quelques aspects numériques, nous allons utiliser comme "patterns" les imagettes produites par Yann LeCun en 1998 : ce sont de petites images représentant des digits écrits manuellement et étiquetés (*labels*), de 28×28 pixels dont les valeurs sont codées en 255 niveaux de gris sur $[0,1]$ (60,000 de training et 10,000 de test.). Ce lot d'imagettes (Modified National Institute of Standards and Technology, MNIST) a permis de mettre au point beaucoup d'algorithmes au cours de challenges [30] et reste encore un lot d'images pour des approches plutôt à caractères pédagogiques [31].

Optimisation. Dans un premier temps, le lot d'entraînement nous sert pour optimiser une RPM ayant $N = 28^2 = 784$ neurones visibles et $H = 140$ neurones cachés [32]. On utilise des batchs de 64 images et l'algorithme décrit dans la précédente section est déroulé en itérant 100 fois ("epochs") sur l'ensemble du lot. On utilise l'approximation CD-1. L'évolution de l'optimisation est montrée sur la Figure 14. Une fois l'optimisation effectuée, on peut alors extraire les éléments de la matrice $\boldsymbol{W}^*$ selon 140 imagettes de dimension 28×28 (Figure 15).

Classification linéaire des features. Maintenant que la matrice $\boldsymbol{W}^*$ est fixée, on peut utiliser $\boldsymbol{h} = \sigma(\boldsymbol{v}.\boldsymbol{W}^*)$ afin d'obtenir pour chaque image du lot de test ($\boldsymbol{v}$) leurs *features* $\boldsymbol{h}$. Puis, à partir de ces vecteurs $\boldsymbol{h}$ (variables latentes), on peut reconstruire de nouvelles images de digits à l'aide de $\boldsymbol{x} = \sigma(\boldsymbol{W}^*\boldsymbol{h}^T)$ et les comparés aux images de digits initiales. La qualité de la restitution peut être calculée par le biais de la norme L2 entre image reconstruite et image d'origine. Le résultat est donné sur la Figure 16. A part un léger effet de floutage, la reconstruction est assez fidèle.

Considérant le lot d'entraînement des images de digits, on peut également leur associer leurs représentations $\boldsymbol{h}^{(p)}$, et connaissant le label du digit $t^{(p)}$, alors on peut constituer des couples $(\boldsymbol{h}^{(p)}, t^{(p)})$ d'entraînement pour un classificateur linéaire [33] (Equation 9). Par la suite, une fois entraîné ce classificateur peut être évalué

30. `yann.lecun.com/exdb/mnist/`
31. voir d'autres catalogues classiques à l'adresse `https://www.tensorflow.org/datasets/catalog/overview`.
32. Dans cet exercice, on n'utilise aucun biais et les poids sont initialisés en utilisant une distribution gaussienne de moyenne 0 et de sigma 0.1.
33. Par exemple en utilisant une régression logistique de la libraire `scipy` : `https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html`

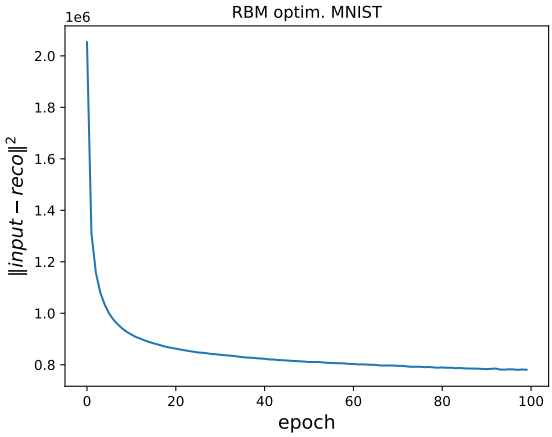

FIGURE 14 – Evolution de $\sum_{i \leq P} \|\boldsymbol{v}^{(i)} - \tilde{\boldsymbol{v}}_i\|^2$ en fonction de l'itération sur le lot d'entraînement (*epoch*).

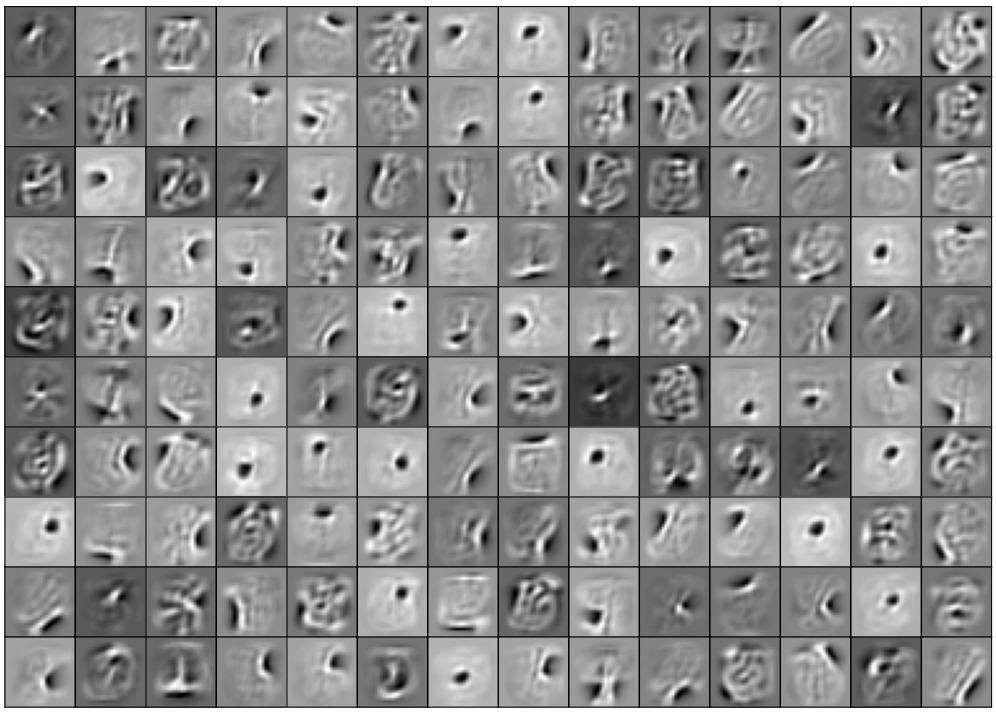

FIGURE 15 – Ensemble des éléments de la matrice $\boldsymbol{W}^*$ optimisée (voir texte).

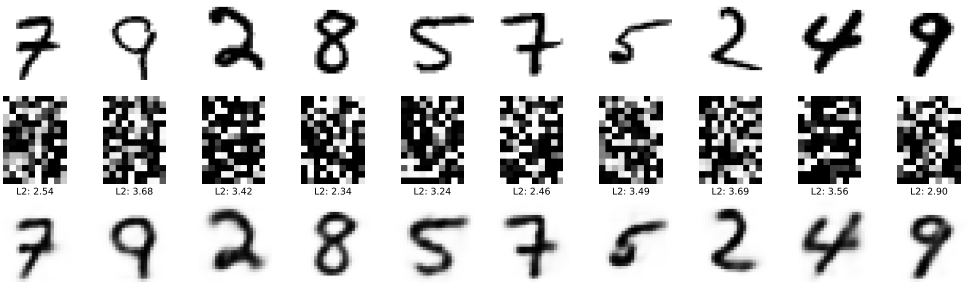

FIGURE 16 – A partir d'images de digit du lot de test (1ère ligne), une fois la matrice de poids optimisée, on peut obtenir les représentations (*features*) de chacune d'elles à savoir les vecteurs $\boldsymbol{h}$ mis en sous forme d'imagettes 14×10 (2nd ligne), puis à partir de ces représentations où l'information est codée et compressée, on peut procéder à la génération de nouvelle image de digits (3eme ligne). On a porté au dessus de ces images reconstruite la norme L2 en prenant à chaque fois l'image de test d'origine (1ère ligne de la même colonne).

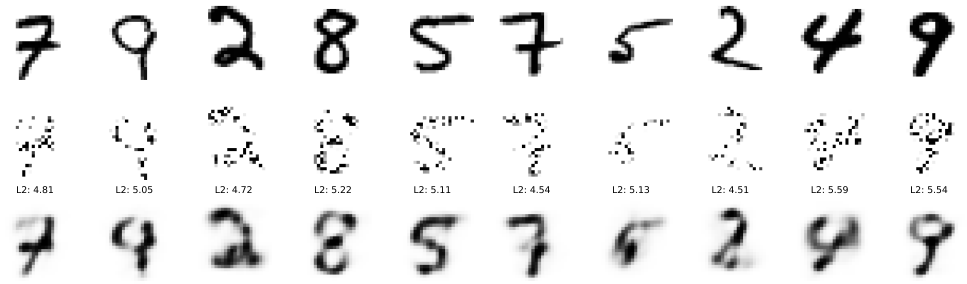

FIGURE 17 – Exemples de reconstitution par RBM d'images de test MNIST détériorées aléatoirement : en haut l'original, au milieu la version dégradée, en bas la version reconstruite. On note la norme L2 entre l'image bruitée ou reconstruite et l'image originale (même colonne).

sur le lot d'images de test de MNIST passé également à travers la RBM pour extraire les *features* comme précédemment. Le score obtenu de la classification est de 94.5% pour le Top-1. Si on avait directement fourni les images au classificateur, son score aurait été d'environ 90% donc la réduction de dimensionalité par l'extraction des *features* par la RBM apporte un gain substantiel. Notons que ce score n'est plus compétitifs en comparaison des performances de méthodes plus raffinées qui atteignent les 100% pour ce lot d'entraînement.

Néanmoins, si la classification linéaire est performante, cela sous-entend que dans l'espace à $H = 140$ dimension, les directions moyennes des 10 digits (des classes) sont assez bien séparées pour que les fluctuations des vecteurs d'une classe donnée ne génèrent pas de "faux" membres d'une autre classe. Ceci dit, cette opération de *"features extraction"* est générale et produit la représentation "$\Phi(x)$" des données. Le tout est de savoir quelle architecture est adaptée pour le problème [34]. Constatons que l'optimisation de la RBM est découplée de son usage final, la classification des digits, c'est un aspect différent des architectures modernes multi-couches qui fusionnent les deux aspects : *feature extraction* et classification finale.

Reconstruction d'images dégradées. Un autre usage d'une RBM une fois les poids connus est de reconstruire des imagettes détériorées comme nous avons pu le faire avec les réseaux d'Hopfield (Section 3). Un exemple est donné sur la Figure 17. Pour ce faire, une fois une image dégradée, on procède à une ou plusieurs itérations de "génération d'un vecteur $\boldsymbol{h}$" puis "génération d'un vecteur $\boldsymbol{v}$". A la fin, le vecteur $\boldsymbol{v}$ donne un digit qui se rapproche au mieux du digit non dégradé. Dans le même genre d'esprit, on peut utiliser la RBM pour débruiter des images comme illustré sur la Figure 18. Les résultats sont assez satisfaisants même si cela ne constitue pas l'état de l'art.

34. ibid. 5

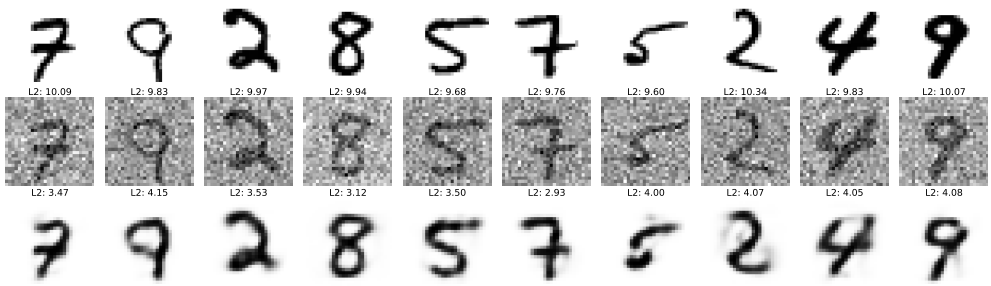

FIGURE 18 – Exemples de reconstruction d'images bruitée que l'on assimile alors à un débruitage. Légende similaire à la Figure 17.

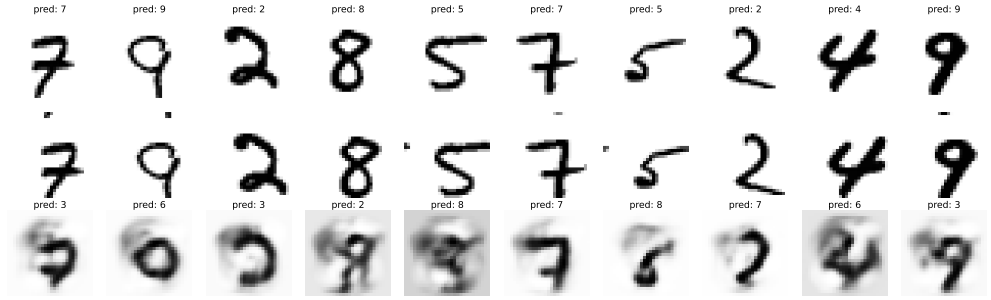

FIGURE 19 – Exemples de faille de classification d'images translatées de 3 pixels dans les 2 directions. Première ligne, l'image d'origine avec le label obtenu par le classificateur linéaire basé sur les représentations issus de la RBM. La seconde ligne montre les images où les digits sont décalées. La troisième ligne donne les images reconstruites par la RBM à partir des images décalées et la classification obtenue à partir des *features* obtenues également. Clairement, le taux de mauvaises classifications augmente sacrément.

4.4 Discussion

L'architecture RBM a donné lieu à des variantes comme par exemple celle permettant de s'affranchir de réponses binaires des neurones en choisissant par exemple des neurones à réponse gaussienne. Les RBMs sont souvent intégrées dans des architectures pour des problèmes d'apprentissage non-supervisé. Elles sont utilisées en ingénierie : ex. pour l'extraction de features dans le domaine du "pattern recognition", dans les systèmes de "recommandation" en tout genre, pour la reconnaissance de signaux radars en milieu très bruité, etc. Des améliorations ont vu le jour comme l'usage du "Dropout" pour réduire l'overfitting (Roder et al., 2022). Enfin, des architectures en cascade ont été utilisées comme les DBM et autre Deep Belief Network de G. Hinton (Hinton, 2009). Les personnes intéressées peuvent se référer par exemple à la revue de Zhang et al. (2018). Notons que les RBM ont été utilisés pour réduire la complexité d'entraîner des architectures telles les Auto-Encoder (Hinton and Salakhutdinov, 2006) (Figure 1) lesquelles sont des architectures à réduction de dimensionnalité de l'entrée et dont la sortie est contrainte lors de l'entraînement à minimiser l'erreur quadratique (par exemple) avec l'entrée. Nous ne nous étendons pas plus avant sur ce sujet qui ouvre le vaste domaine des *modèles génératifs*.

Cependant, avant de conclure, il parait intéressant d'aborder une thématique qui dépasse le cadre des Machines de Boltzmann. Il s'agit d'un défaut que l'on peut mettre facilement en évidence avec la RBM optimisée sur le lot MNIST. Il suffit de translater de 3 pixels l'image d'origine pour que la classification par la suite soit complètement flouée comme le montre la Figure 19. La translation a généré des *exemples adversaires* pour l'architecture RBM. Ce type de défaut a plutôt été révélé dans le contexte des architectures profondes (Szegedy et al., 2014).

A posteriori, face à ce type de problème plusieurs attitudes (ni exclusives ni exhaustives) sont à l'œuvre comme par exemple :

1. on peut modifier la façon d'entraîner le modèle. Par exemple, il serait sans doute possible d'améliorer la robustesse du modèle durant la phase d'entraînement en lui présentant des images translatées, voire même légèrement orientées différemment ou bien déformées. Il s'agit de méthode de *"data*

augmentation" qui en l'occurrence essaye de faire apprendre au modèle une certaine variabilité des images concernant des invariances de la réponse vis-à-vis de transformations (translation globale ou difféomorphisme). Cette méthode est largement rentrée dans les us et coutumes des personnes qui utilisent quotidiennement les réseaux de neurones en général.

2. une seconde voie opère une réflexion à partir des *symétries du problème* : quelles sont les transformations qui laisse le problème invariant ? C'est dans ce cadre qu'on été mis au point les *réseaux de neurones convolutionnels* à partir des idées de réduction du nombres de poids par des invariances spatiales (Le Cun et al., 1989) et qui donna le point de départ de l'essor moderne des réseaux de neurones. Voilà en substance se qu'écrivaient Yann LeCun, Léon Bettou, Yoshua Bengio et Patrick Haffner en 1998 (LeCun et al., 1998) *"[...] Mais le principal défaut des réseaux non structurés pour les applications d'image ou de parole réside dans le fait qu'ils ne possèdent aucune invariance intrinsèque par rapport aux translations ou aux distorsions locales des entrées. [...]* ". Ces développements amèneront à l'élaboration de l'architecture "AlexNet" en 2012 (Krizhevsky et al., 2012) qui donna des résultats spectaculaires sur le lot d'images ImageNet [35] et surclassa toutes les meilleures méthodes antérieures.

   Cette voie attire la réflexion sur les *opérateurs* qui doivent être mis en place pour construire la représentation $\Phi(x)$ de l'entrée x que l'on a appelé *feature extraction*. Cela ne dit pas que d'autres types d'exemples adversaires puissent se révéler gênant mais au moins on les repoussent dans un espace de plus en plus restreint. Ici, les mathématiques tentent de reprendre la main pour comprendre les dessous des réseaux de neurones, surtout pour expliquer le succès des CNNs, tout en construisant un corpus théorique qui permet de cerner les problèmes d'instabilités [36].

5 Conclusion

Les réseaux de J. J. Hopfield et des Machines de Boltzmann de G. Hinton ont été des pièces maîtresses pour élaborer des architectures qui ont mené par la suite aux succès d'abord des réseaux convolutionnels profonds puis des grands modèles génératifs. Ces réseaux couvrent quasiment tous les domaines du traitement du signal, de l'image, la vidéo, du son, de la parole et tous les tâches liées aux traitement de textes, les problèmes inverses, etc. Notons que le prix Nobel de Chimie 2024 récompense Demis Hassabis et John Jumper qui ont réussi à utiliser le modèle `Alphafold` (Jumper et al., 2021) pour prédire la structure de presque toutes les protéines connue. La question est plutôt de savoir quel champ disciplinaire n'est pas impacté par ces développements tout comme les applications dans la vie quotidienne. Cependant, comme nous l'avons mentionné, tout n'est pas totalement clair : pourquoi cela fonctionne si bien. On manque encore de théorèmes de base. La recherche mathématique est active sur ce sujet dans différents champs comme les statistiques, les probabilités, l'algorithmique mais aussi les processus stochastiques, les équations aux dérivées partielles, voire la géométrie algébrique, etc. Il reste donc beaucoup à découvrir sur les pas de ces pionniers récompensés cette année.

Références

Ackley, D. H., Hinton, G. E., and Sejnowski, T. J. (1985). A learning algorithm for boltzmann machines. *Cognitive science*, 9(1) :147–169.

Bartlett, P. and Maass, W. (2003). *Vapnik-Chervonenkis dimension of neural nets*, pages 1188–1192. MIT Press, 2nd ed. edition.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*, volume 4 of *Information science and statistics*. Springer.

Cortes, C. and Vapnik, V. (1995). Support vector networks. *Machine Learning*, 20 :273–297.

Cover, T. (1965). Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition. *IEEE Transactions on Electronic Computers*, EC–14 :326–334.

Crisanti, A., Amit, D. J., and Gutfreund, H. (1986). Saturation level of the hopfield model for neural network. *Europhysics Letters*, 2(4) :337.

De Marzo, G. and Iannelli, G. (2023). Effect of spatial correlations on hopfield neural network and dense associative memories. *Physica A : Statistical Mechanics and its Applications*, 612 :128487.

35. `https://www.image-net.org/`
36. Ibid. 5

Demircigil, M., Heusel, J., Löwe, M., Upgang, S., and Vermet, F. (2017). On a model of associative memory with huge storage capacity. *Journal of Statistical Physics*, 168 :288–299.

Devlin, J. (2018). Bert : Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv :1810.04805*.

Hebb, D. O. (1949). Organization of behavior. new york : Wiley. *J. Clin. Psychol*, 6(3) :335–307.

Hertz, J. A. (2018). *Introduction to the theory of neural computation*. Crc Press.

Hinton, G. E. (2002). Training products of experts by minimizing contrastive divergence. *Neural Comput.*, 14(8) :1771–1800.

Hinton, G. E. (2009). Deep belief networks. *Scholarpedia*, 4(5) :5947. revision #63393.

Hinton, G. E. (2012). A practical guide to training restricted boltzmann machines. In *Proceedings of Neural Networks : Tricks of the Trade (2nd ed.)*, pages 599–619.

Hinton, G. E., Osindero, S., and Teh, Y.-W. (2006). A Fast Learning Algorithm for Deep Belief Nets. *Neural Computation*, 18(7) :1527–1554.

Hinton, G. E. and Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *Science*, 313(5786) :504–507.

Hinton, G. E. and Sejnowski, T. J. (1986). Learning and relearning in Boltzmann machines. In Rumelhart, D. E. and Mcclelland, J. L., editors, *Parallel Distributed Processing : Explorations in the Microstructure of Cognition, Volume 1 : Foundations*, pages 282–317. MIT Press, Cambridge, MA.

Hopfield, J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences of the United States of America*, 79 :2554–8.

Hopfield, J. J. (1984). Neurons with graded response have collective computational properties like those of two-state neurons. *Proceedings of the National Academy of Sciences*, 81(10) :3088–3092.

Jain, V., Koehler, F., and Mossel, E. (2018). The mean-field approximation : Information inequalities, algorithms, and complexity. In *Conference On Learning Theory*, pages 1326–1347. PMLR.

Jumper, J., Evans, R., Pritzel, A., Green, T., Figurnov, M., Ronneberger, O., Tunyasuvunakool, K., Bates, R., Žídek, A., Potapenko, A., Bridgland, A., Meyer, C., Kohl, S., Ballard, A., Cowie, A., Romera-Paredes, B., Nikolov, S., Jain, R., Adler, J., and Hassabis, D. (2021). Highly accurate protein structure prediction with alphafold. *Nature*, 596 :1–11.

Kenton, J. D. M.-W. C. and Toutanova, L. K. (2019). Bert : Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of naacL-HLT*, volume 1, page 2. Minneapolis, Minnesota.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In Pereira, F., Burges, C. J. C., Bottou, L., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc.

Krotov, D. and Hopfield, J. J. (2016). Dense associative memory for pattern recognition. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, page 1180–1188, Red Hook, NY, USA. Curran Associates Inc.

Le Cun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1989). Handwritten digit recognition with a back-propagation network. In *Proceedings of the 2nd International Conference on Neural Information Processing Systems*, NIPS'89, page 396–404, Cambridge, MA, USA. MIT Press.

Lecun, Y. (1988). A theoretical framework for back-propagation. In Touretzky, D., Hinton, G., and Sejnowski, T., editors, *Proceedings of the 1988 Connectionist Models Summer School, CMU, Pittsburg, PA*, pages 21–28. Morgan Kaufmann.

Lecun, Y. and Bengio, Y. (1995). *Convolutional Networks for Images, Speech and Time Series*, pages 255–258. The MIT Press.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11) :2278–2324.

Little, W. A. (1974). The existence of persistent states in the brain. *Mathematical biosciences*, 19(1-2) :101–120.

Liu, Z., Luo, P., Wang, X., and Tang, X. (2015). Deep learning face attributes in the wild. In *Proceedings of the IEEE international conference on computer vision*, pages 3730–3738.

Löwe, M. (1998). On the storage capacity of hopfield models with correlated patterns. *The Annals of Applied Probability*, 8(4) :1216–1250.

MacKay, D. J. C. (2003). *Information Theory, Inference, and Learning Algorithms*. Copyright Cambridge University Press.

Minsky, M. and Papert, S. (1969). *Perceptrons : An Introduction to Computational Geometry*. MIT Press, Cambridge, MA, USA.

Minsky, M. and Papert, S. (1988). *Perceptrons : An Introduction to Computational Geometry*. MIT Press, Cambridge, MA, expanded edition.

Petersen, P. C. and Sepliarskaia, A. (2024). Vc dimensions of group convolutional neural networks. *Neural Networks*, 169 :462–474.

Ramsauer, H., Schäfl, B., Lehner, J., Seidl, P., Widrich, M., Gruber, L., Holzleitner, M., Pavlovic, M., Sandve, G. K., Greiff, V., Kreil, D. P., Kopp, M., Klambauer, G., Brandstetter, J., and Hochreiter, S. (2020). Hopfield networks is all you need. *CoRR*, abs/2008.02217.

Roder, M., de Rosa, G. H., de Albuquerque, V. H. C., Rossi, A. L. D., and Papa, J. P. (2022). Energy-based dropout in restricted boltzmann machines : Why not go random. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 6(2) :276–286.

Rosenblatt, F. (1957). The perceptron - a perceiving and recognizing automaton. Technical Report 85-460-1, Cornell Aeronautical Laboratory, Ithaca, New York.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *nature*, 323(6088) :533–536.

Šíma, J. and Orponen, P. (2003). Continuous-time symmetric hopfield nets are computationally universal. *Neural Computation*, 15(3) :693–733.

Sutskever, I. and Tieleman, T. (2010). On the convergence properties of contrastive divergence. In Teh, Y. W. and Titterington, M., editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 789–795, Chia Laguna Resort, Sardinia, Italy. PMLR.

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I. J., and Fergus, R. (2014). Intriguing properties of neural networks. In Bengio, Y. and LeCun, Y., editors, *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*.

Tolmachev, P. and Manton, J. H. (2020). New insights on learning rules for hopfield networks : Memory and objective function minimisation. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE.

Vapnik, V. N. and Chervoneva, A. (1964). On class of perceptrons. *Automation and remote control*, 25(1) :103.

Varadhan, S. R. S. (2008). Large deviations. *The Annals of Probability*, 36(2) :397 – 419.

Wen, U.-P., Lan, K.-M., and Shih, H.-S. (2009). A review of Hopfield neural networks for solving mathematical programming problems. *European Journal of Operational Research*, 198(3) :675–687.

Zhang, N., Ding, S., Zhang, J., and Xue, Y. (2018). An overview on restricted boltzmann machines. *Neurocomput.*, 275(C) :1186–1199.