Université de Montréal

Réseaux de neurones à relaxation entraînés par critère d'autoencodeur débruitant

par
François Savard

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

Août, 2011

Université de Montréal
Faculté des arts et des sciences

Ce mémoire intitulé:

Réseaux de neurones à relaxation entraînés par critère d'autoencodeur débruitant

présenté par:

François Savard

a été évalué par un jury composé des personnes suivantes:

Pascal Vincent,	président-rapporteur
Yoshua Bengio,	directeur de recherche
Sébastien Roy,	membre du jury

Mémoire accepté le: 22 novembre 2011

RÉSUMÉ

L'apprentissage machine est un vaste domaine où l'on cherche à apprendre les paramètres de modèles à partir de données concrètes. Ce sera pour effectuer des tâches demandant des aptitudes attribuées à l'intelligence humaine, comme la capacité à traiter des données de haute dimensionnalité présentant beaucoup de variations. Les réseaux de neurones artificiels sont un exemple de tels modèles. Dans certains réseaux de neurones dits *profonds*, des concepts "abstraits" sont appris automatiquement.

Les travaux présentés ici prennent leur inspiration de réseaux de neurones profonds, de réseaux récurrents et de neuroscience du système visuel. Nos tâches de test sont la classification et le débruitement d'images quasi binaires. On permettra une rétroaction où des représentations de haut niveau (plus "abstraites") influencent des représentations à bas niveau. Cette influence s'effectuera au cours de ce qu'on nomme **relaxation**, des itérations où les différents niveaux (ou couches) du modèle s'interinfluencent. Nous présentons deux familles d'architectures, l'une, l'architecture complètement connectée, pouvant en principe traiter des données générales et une autre, l'architecture convolutionnelle, plus spécifiquement adaptée aux images. Dans tous les cas, les données utilisées sont des images, principalement des images de chiffres manuscrits.

Dans un type d'expérience, nous cherchons à reconstruire des données qui ont été corrompues. On a pu y observer le phénomène d'influence décrit précédemment en comparant le résultat avec et sans la relaxation. On note aussi certains gains numériques et visuels en terme de performance de reconstruction en ajoutant l'influence des couches supérieures. Dans un autre type de tâche, la classification, peu de gains ont été observés. On a tout de même pu constater que dans certains cas la relaxation aiderait à apprendre des représentations utiles pour classifier des images corrompues. L'architecture convolutionnelle développée, plus incertaine au départ, permet malgré tout d'obtenir des reconstructions numériquement et visuellement semblables à celles obtenues avec l'autre architecture, même si sa connectivité est contrainte.

Mots clés: intelligence artificielle, apprentissage machine, réseaux de neurones, autoencodeur débruitant, réseaux de neurones récurrent

ABSTRACT

Machine learning is a vast field where we seek to learn parameters for models from concrete data. The goal will be to execute various tasks requiring abilities normally associated more with human intelligence than with a computer program, such as the ability to process high dimensional data containing a lot of variations. Artificial neural networks are a large class of such models. In some neural networks said to be *deep*, we can observe that high level (or "abstract") concepts are automatically learned.

The work we present here takes its inspiration from deep neural networks, from recurrent networks and also from neuroscience of the visual system. Our test tasks are classification and denoising for near binary images. We aim to take advantage of a feedback mechanism through which high-level representations, that is to say relatively abstract concepts, can influence lower-level representations. This influence will happen during what we call **relaxation**, which is iterations where the different levels (or layers) of the model can influence each other. We will present two families of architectures based on this mechanism. One, the fully connected architecture, can in principle accept generic data. The other, the convolutional one, is specifically made for images. Both were trained on images, though, and mostly images of written characters.

In one type of experiment, we want to reconstruct data that has been corrupted. In these tasks, we have observed the feedback influence phenomenon previously described by comparing the results we obtained with and without relaxation. We also note some numerical and visual improvement in terms of reconstruction performance when we add upper layers' influence. In another type of task, classification, little gain has been noted. Still, in one setting where we tried to classify noisy data with a representation trained without prior class information, relaxation did seem to improve results significantly. The convolutional architecture, a bit more risky at first, was shown to produce numerical and visual results in reconstruction that are near those obtained with the fully connected version, even though the connectivity is much more constrained.

Keywords: artificial intelligence, machine learning, neural networks, autoencoder, denoising autoencoder, recurrent neural networks.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES SIGLES

CPU	Central Processing Unit
CRBM	Convolutional Deep Belief Network
DAE	Denoising Auto Encoder
DBM	Deep Boltzmann Machine
DBN	Deep Belief Network
DCAE	Deep Convolutional Auto Encoder
GPU	Graphics Processing Unit
MSE	Mean Squared Error
MLP	Multi-Layer Perceptron
NLL	Negative Log Likelihood
p&s	Poivre et sel (pour du bruit)
RBM	Restricted Boltzmann Machine
RSDAE	Relaxing Stacked Denoising Autoencoder

NOTATION

La notation est spécifiée dans le texte, mais voici tout de même quelques symboles fréquents :

a	Activation d'une unité avant nonlinéarité
b	Vecteur de biais des unités d'une couche de neurones
k	Indice d'une itération de relaxation
W	Matrice de poids des connections d'une couche à une autre
x	Vecteur d'entrée d'un modèle
$\tilde{x}$	Vecteur d'entrée *bruité* d'un modèle
y	Vecteur de sortie d'un modèle, ou cible désirée pour supervision
z	Reconstruction (sortie) d'un autoencodeur
σ	Déviation standard lors d'estimation des paramètres d'une distribution normale
$\sigma()$	Nonlinéarité d'une unité d'un modèle (fonction logistique ou *softsign*)
X'	Transposée de la matrice X
v'	Vecteur-colonne pour le vecteur-ligne v, ou *vice-versa*

REMERCIEMENTS

J'aimerais tout d'abord remercier mon directeur de maîtrise, Yoshua Bengio, pour m'avoir fourni maints conseils concernant ces expériences que je lui détaillais longuement et pour m'avoir donné l'opportunité de m'initier au domaine. J'ai énormément appris au cours des deux dernières années, et c'est en grande partie grâce à son aide et à l'environnement au laboratoire qu'il a fondé au sein de l'Université de Montréal.

Le remerciement doit donc aussi aller aux autres membres de ce laboratoire, le LISA, qui m'ont fourni ici et là un soutien précieux dans mes travaux. Certains de ces membres sont aussi auteurs de l'outil principal que j'ai utilisé pour programmer les modèles, Theano, sans lequel certaines expériences auraient été beaucoup plus ardues à réaliser.

J'aimerais aussi remercier mes proches, donc ma compagne, mes parents et mes grands-parents, pour leur crucial support durant cet effort. Ce que j'étudiais pouvait leur paraître un peu distant et abstrait, mais malgré tout ils savaient demeurer encourageants et grâce à eux la persévérance était bien plus facile à trouver.

Je suis aussi reconnaissant envers divers organismes qui ont contribué à ces recherches. Je mentionnerai bien sûr les bourses que j'ai reçues. Ce support financier provenait, directement ou indirectement, du CRSNG et du FQRNT. J'ai aussi utilisé plusieurs grappes de calcul, notamment une du réseau SharcNET (à travers Calcul Canada) et une autre du réseau RQCHP.

CHAPITRE 1

INTRODUCTION ET FONDEMENTS THÉORIQUES

Le présent travail traite d'une approche spécifique d'apprentissage machine, plus spécifiquement d'une forme particulière de réseaux de neurones. Pour bien asseoir la présentation des résultats et l'originalité de la méthode, les concepts théoriques sous-jacents, assez nombreux, seront présentés ici de façon succincte. Une revue de la littérature pertinente, du plus général au plus spécifique, sera faite au cours de la présentation de ces assises conceptuelles.

1.1 Concepts de base d'apprentissage machine

L'apprentissage machine est un domaine vaste regroupant des méthodes permettant de modéliser des distributions de données pour accomplir diverses tâches. Les modèles sont très variés en forme, mais en général l'apprentissage consistera à adapter les **paramètres** θ d'un **modèle** (représenté par une fonction f_θ) pour mieux accomplir la tâche. Un exemple minimaliste est l'adaptation de la pente et ordonnée à l'origine (paramètres) d'une droite à un nuage de points dans le plan cartésien.

Les données auront une forme dépendant du problème, mais viendront généralement comme un ensemble, fini ou infini, d'**exemples** x_i pris en entrée du modèle, chaque exemple étant représenté dans le **jeu de donnée** D par un nombre fixe d (pour **dimension**) de **caractéristiques d'entrée** (étiquettes ou nombres). Le cas considéré le plus souvent dans ce travail est celui où les exemples sont des images de taille fixe et les caractéristiques sont chacun des pixels d'une image, considérée comme un long vecteur plutôt que comme une tableau à deux dimensions.

Parmi les familles de tâches importantes on compte :

- la **classification** : déterminer à quelle classe, parmi un nombre fixe de classes, un exemple appartient ;
- la **régression** : prédire un nombre réel correspondant à l'exemple d'entrée ;

– l'**estimation de densité** : estimer quelle est la densité de probabilité pour la distribution des données au point représenté par l'exemple en entrée.

Dans les deux premiers cas, on devra permettre à l'algorithme, durant sa phase d'apprentissage, de savoir quelle est la bonne **cible** y_i pour chaque entrée x_i. Ce sera une classe dans le premier cas, un nombre réel dans le second. Cette situation se nomme **apprentissage supervisé**. S'il n'y a pas de cible requise, comme dans le troisième cas, il s'agit d'**apprentissage non supervisé**.

1.1.1 Apprentissage par minimisation du risque empirique

Dans les situations considérées ici, l'apprentissage consistera, en pratique, à optimiser une **fonction objectif** définie comme une mesure de l'adéquation du modèle au jeu de données. Le **risque empirique** $R(f_\theta, D)$, terme général pour les fonctions objectifs utilisées ici, est la moyenne d'une **fonction de coût** C sur l'ensemble du jeu de données :

$$R(f_\theta, D) = \frac{1}{N}\sum_{i=1}^{N} C(f_\theta(x_i), y_i)$$

où N est le nombre d'exemples dans D [40].

On cherchera donc à minimiser le risque empirique en ajustant les paramètres θ. Pour une situation de régression, un fonction de coût typique serait la différence, au carré, de la cible de l'exemple et de la valeur prédite par le modèle. En prenant la moyenne sur le jeu de données, ça donne la ***mean squared error*** **(MSE)** :

$$MSE(f_{theta}, D) = \frac{1}{N}\sum_{i=1}^{N} (f_\theta(x_i) - y_i)^2$$

En ajustant les paramètres du modèle pour minimiser ce coût, on adapte le modèle aux données. La fonction de coût sera aussi souvent dérivée en cherchant à rendre les données d'apprentissage probables selon la distribution définie par le modèle, si on est dans un cadre probabiliste. C'est ce qu'on nomme chercher le **maximum de vraisemblance** (*maximum likelihood*).

Ainsi on utilisera souvent la probabilité donnée par le modèle à la sortie observée, pour un exemple donné, pour dériver la fonction de coût ***negative log likelihood*** (**NLL**). Par exemple, dans le cas d'une sortie binaire (0 ou 1), si on voit le modèle comme une distribution de probabilité conditionnelle de la sortie observée :

$$p(y_i = 1|x_i) = f_\theta(x_i)$$

et en considérant la vraisemblance de l'ensemble des données, on obtient :

$$NLL(f_\theta, D) = -log\{\prod_{i=1}^{N} p(y_i|x_i)\} = -\sum_{i=1}^{N} [y_i log\{f_\theta(x_i)\} + (1-y_i) log\{1-f_\theta(x_i)\}]$$

Au passage, mentionnons une formule spécifique pour M sorties binaires pour chaque exemple :

$$NLL(f_\theta, D) = -\sum_{i=1}^{N}\sum_{j=1}^{M} \left[y_{ij} log\{f_{\theta j}(x_i)\} + (1-y_{ij}) log\{1-f_{\theta j}(x_i)\}\right] \tag{1.1}$$

où $f_{\theta j}$ donne la *j-ième* sortie du modèle. Cette formule est aussi applicable si la fonction $f_{\theta j}(x_i)$, ou encore les sorties y_{ij}, ont des valeurs sur *l'intervalle*]0,1[.

1.1.2 Généralisation et séparation du jeu de données

Dans tout ceci il ne faut pas perdre de vue que le jeu de données, s'il est fini, n'est qu'un sous-ensemble du jeu de données théorique infini (ou immense) des échantillons qu'on peut tirer de la distribution réelle des données. Une question fondamentale est donc de savoir si ce qu'on apprendra sur les données vues "généralise" bien aux autres données : est-ce que les prédictions du modèle sont bonnes pour des données qu'il n'aura jamais vues ? C'est le problème de la **généralisation**.

Pour illustrer ce problème, on peut considérer un algorithme très simple de classification : pour chaque exemple dont on ignore la classe, on considère l'exemple le plus

“proche” (selon une métrique quelconque) dans le jeu de données d’entraînement et on retourne la classe associée à ce “plus proche voisin”. Au passage, soulignons qu’il s’agit d’une **approche non paramétrique**, car il n’y a pas d’apprentissage de paramètres à proprement parler : le jeu de données *est* le modèle, en quelque sorte. Le problème survient lorsque la fonction à modéliser est “complexe” et le nombre de dimensions de l’entrée est grand. Quand c’est le cas, choisir simplement le voisin qui minimise une métrique simple peut devenir une piètre stratégie, et cette approche généralise mal à de nouveaux exemples. À titre d’exemple, considérons le problème de déterminer à partir d’une image d’un visage la largeur du nez. L’approche consisterait à comparer tous les pixels de l’image, un à un, à d’autres pixels d’images pour lesquelles on connaît la cible, et choisir la cible de l’image plus proche. C’est bien sûr une très mauvaise stratégie.

On voit donc que le fait de trop se fier aux exemples de l’ensemble d’entraînement peut être trompeur. On nomme **sur-apprentissage (overfitting)** le fait d’apprendre une réponse généralisant mal mais performant bien sur les exemples d’entraînement. On voudra en fait minimiser l’**erreur de généralisation**, l’espérance de la fonction de coût sur la réelle distribution des données (“toutes” les données), et non pas, comme c’est le cas pour l’erreur empirique, sur le jeu de données qu’on utilise pour l’entraînement. C’est pourquoi on réservera une partie du jeu de données, ne servant pas durant l’entraînement, pour estimer l’erreur de généralisation. C’est le **jeu de données de test**, ou ensemble de test.

1.1.3 Hyperparamètres et ensemble de validation

Le sur-apprentissage est un phénomène qui peut se produire si on donne trop de flexibilité au modèle, lui permettant d’ajuster ses paramètres de façon trop précise aux exemples d’entraînement. On dira que le modèle a une trop grande **capacité**. Pour éviter ce problème, une approche simple consiste à réduire le nombre de paramètres du modèle (pour les modèles le permettant). Ce nombre est un **hyperparamètre**, c’est-à-dire qu’il s’agit d’une valeur qui n’est pas apprise au même titre que d’autres paramètres, durant l’apprentissage, mais contrôle de façon plus globale la forme du modèle et le déroulement de l’apprentissage. D’autres exemples d’hyperparamètres sont la durée de

l'apprentissage, le coefficient (poids) de certains termes dans la fonction objectif, etc.

C'est l'existence d'hyperparamètres qui justifie une autre séparation du jeu de données, le **jeu de données de validation**, ou ensemble de validation. Typiquement, on ne connaîtra pas la valeur optimale des hyperparamètres menant à l'apprentissage du "meilleur" modèle. On devra donc essayer plusieurs combinaisons possibles d'hyperparamètres. Chacun des modèles sera entraîné sur l'ensemble d'entraînement en optimisant ses paramètres (et non pas *hyper*paramètres) selon cet ensemble. Par contre, pour comparer les modèles entre eux, on voudra une mesure de l'erreur de généralisation, donc non "biaisée", non basée sur l'ensemble utilisé pour l'entraînement. On réservera donc une partie du jeu de données, l'ensemble de validation, pour évaluer la performance de chaque modèle et choisir le meilleur, selon cette mesure. Cette opération se nomme **sélection de modèle**. On utilisera l'ensemble de test, une autre sous-partie réservée mais distincte, introduite plus haut, comme mesure de la performance finale de l'apprentissage *et de la sélection de modèle*, pour pouvoir comparer une approche (un type de modèle) à d'autres publiées dans la littérature pour le même ensemble de données.

1.2 Les réseaux de neurones artificiels

1.2.1 Régression logistique et séparabilité linéaire

Jusqu'à présent il était question de "modèles" de façon très générale. Dans ce travail, les modèles sont du type **réseaux de neurones artificiels**. Ils portent ce nom car le fonctionnement de l'unité de base de ces modèles, un "neurone", s'apparente (de façon très approximative) au comportement de neurones biologiques : une accumulation d'entrées pondérées, suivie d'une "fonction d'activation", rappelant le comportement "tout-ou-rien" des potentiels d'action des neurones biologiques.

Le comportement d'un neurone individuel s'illustre bien par la **régression logistique**. Si on a n_{in}entrées et qu'on cherche à prédire une étiquette binaire (0/1) de sortie y, une régresion logistique sera donc un produit scalaire entre n_{in} **paramètres** ou **poids** ("***weights***") ajustables w et ces entrées, additionné d'un **biais** b, la somme suivie d'une **nonlinéarité** (fonction non linéaire) de type **fonction logistique** $\sigma(a) = \frac{1}{1+exp(-a)}$:

$$y = \sigma(a) = \sigma(w'x + b)$$

La fonction logistique est une fonction de forme sigmoïdale (“S”) permettant une transition douce entre les valeurs de 0 pour des grandeurs négatives en entrée, et 1 pour des valeurs positives. On l'appellera aussi **fonction sigmoïde**, bien que d'autres fonctions aient une forme semblable.

Supposons qu'on binarise la sortie en considérant tout $y < 0.5$ comme une sortie de 0 et $y > 0.5$ comme une sortie de 1. $\sigma(w'x + b) = 0.5$ implique que $w'x + b = 0$. Cette dernière équation est l'équation d'un hyperplan, et on ajustera les paramètres pour qu'autant que possible on trouve les points de classe 0 d'un côté du plan, et les points de classe 1 de l'autre. S'il existe un tel **hyperplan séparateur** qui sépare parfaitement les deux classes, alors on dit que les données sont **linéairement séparables**, et concrètement ça signifie que le problème de classification est très facile à résoudre.

1.2.2 Apprentissage par descente de gradient

Pour ajuster les paramètres, il existe une multitude de techniques d'optimisation. La technique utilisée dans ce travail est la **descente de gradient stochastique à vitesse d'apprentissage constante**. Étant donnée une valeur courante des paramètres θ, on trouve le vecteur (direction) de changement de ces paramètres résultant en la descente la plus rapide de la fonction de coût. Cela se fait en dérivant la fonction de coût par rapport aux paramètres pour obtenir le gradient. On prendra donc un pas dans cette direction $\frac{\partial C}{\partial \theta}$, **pas de gradient** (ou ***update***) plus ou moins grand selon la **vitesse d'apprentissage** (***learning rate***) ε :

$$\theta' \leftarrow \theta - \varepsilon \frac{\partial C}{\partial \theta}$$

Dans notre cas, la vitesse d'apprentissage est constante. Il faut mentionner qu'il existe toute une gamme de techniques pour améliorer le processus d'optimisation en changeant cette grandeur au cours de l'apprentissage, ou en introduisant d'autres facteurs (comme le cas populaire du ***momentum***). Cependant, comme on pourra le voir

plus tard, il y a déjà suffisamment d'hyperparamètres dans la méthode présentée ; ce genre d'optimisation ne changerait probablement pas les conclusions tirées, vu qu'il n'y a pas beaucoup de comparaison numérique avec d'autres méthodes optimisées de la sorte.

1.2.3 Apprentissage en lot *versus* apprentissage stochastique

Il y a plusieurs façons pour l'apprentissage de parcourir les données durant l'apprentissage. Pour simplifier, supposons qu'on a un jeu de données de taille finie. Le fait de "considérer" (d'apprendre de) tous les exemples du jeu de données une fois est nommé une **époque** d'apprentissage. Il est possible de considérer tous les exemples à chaque pas de gradient, ce qu'on nommera **apprentissage en lot** (batch). À l'autre extrême, on peut considérer un seul exemple à la fois, ce qui se nomme **apprentissage stochastique**. C'est "stochastique" en ceci qu'on supposera qu'un exemple et celui qui le suit dans l'ordre de présentation à l'algorithme d'apprentissage n'ont pas de relation de dépendance et sont tirés de façon uniforme du jeu de données, ce qu'on note par le terme **"indépendant et identiquement distribué" (i.i.d.)**.

La descente de gradient stochastique offre un certain avantage en terme de vitesse d'apprentissage. Comme on peut faire des pas de gradient beaucoup plus petits, la question de la grandeur de chaque pas est un peu moins cruciale, notamment. Le risque en fonction des paramètres et du jeu de données étant long à calculer et non convexe, un problème central est que l'optimisation est longue et mènera vers des optima locaux, plutôt qu'un optimum global. Avec la méthode du gradient stochastique on évitera parfois de rester coincé dans certains minima locaux, la stochasticité nous permettant de nous en échapper [27].

En pratique, dans ce travail, une approche de compromis entre les deux extrêmes est adoptée. L'algorithme considérera des exemples en "***minibatch***" X de quelques dizaines d'exemples à la fois. Cela permet de tirer malgré tout profit de gains en vitesse de calcul vu les opérations matricielles utilisées continuellement. X (majuscule) symbolise le fait qu'il s'agit alors d'une matrice, avec une rangée pour chaque exemple.

1.2.4 Couches cachées et multiples sorties

Dans la plupart des situations d'intérêt, les données ne seront pas linéairement séparables. Il faudra alors pouvoir modéliser des fonctions non linéaires dans l'espace d'entrée, ce qui n'est pas possible avec un seul neurone. L'idée émergeant naturellement est alors d'utiliser plusieurs neurones fonctionnant de cette façon, chacun ayant ses propres paramètres, donc son propre hyperplan. Chaque neurone peut ainsi en venir à représenter une caractéristique différente de l'entrée, qui dépend de façon non linéaire de l'information fournie par les caractéristiques entrées.

En connectant plusieurs neurones de la sorte à l'entrée directement, on constitue ce qu'on nomme une **couche cachée** de neurones. Les paramètres de chaque neurone sont alors placés dans les colonnes d'une matrice W, puisque les paramètres de chaque neurone constituent un vecteur. L'équation des activations pour une couche cachée h devient alors $h = \sigma(XW + b)$ où l'ordre XW (produit matriciel) est utilisé pour montrer ce qui se produit lorsqu'une *minibatch* est traitée : on obtient un vecteur colonne pour chaque unité, représentant l'activation de cette unité pour chaque exemple dans la *minibatch*.

Il y aura un paramètre de biais par neurone, représentant en quelque sorte sa "valeur moyenne". b est donc un vecteur (vecteur-ligne, ici), pas une matrice. L'opérateur d'addition dans cette opération est donc un peu spécial, puisque le vecteur b de biais est additionné à chaque ligne de la matrice résultant de l'opération XW. En terme de programmation, cela se nomme *broadcasting*.

Les neurones d'une couche cachée pourront servir d'entrée à un nouveau neurone de sortie y. On nommera de tels réseaux, ayant une ou plusieurs couches cachées, des **Multi Layer Perceptron** (MLP), en l'honneur d'un des premiers modèles de neurone artificiel, le Perceptron [34]. Cette couche permet à la sortie d'avoir des comportement non linéaires fonction de l'entrée, même si, à la rigueur, l'unité de sortie est linéaire en fonction des valeurs des unités qu'elle prend en entrée directe.

Ici il est utile de noter le sens qu'ont généralement les diagrammes : on parlera de "**couches supérieures**" pour dénoter les couches se rapprochant de la couche de sortie. Donc l'information se "propage vers le haut", comme si elle "montait de niveau", d'une

certaine façon. Le terme "**propagation avant**" fait référence à la propagation vers le haut, aussi.

Si on veut plusieurs classes en sortie pour un problème de classification, on peut utiliser plusieurs sorties ayant chacune leur propre fonction logistique et chacune servant à séparer une des classes de toutes les autres. Typiquement, par contre, on utilisera une fonction ***softmax*** en sortie. Il s'agit d'avoir un ensemble de poids et un biais par unité de sortie, mais de lier toutes les sorties y_i par la fonction :

$$softmax_i(a_i) = \frac{exp(a_i)}{\sum_j exp(a_j)} \tag{1.2}$$

où j varie de 1 au nombre de sorties. Ainsi on normalise les sorties entre elles, et c'est plus facile d'éviter d'avoir de l'ambiguïté sur la sortie choisie par le réseau. De plus on peut considérer les sorties comme paramétrant une distribution de probabilités multinomiale, donnant la probabilité de chaque classe étant donné l'entrée présentée au réseau, car leur somme est 1.

On peut entraîner une réseau à une ou plusieurs couches cachées par descente de gradient, comme expliqué dans la section 1.2.2. On utilisera comme *update* une somme du gradient de chaque sortie par rapport à tous les poids de tous les neurones. Dans le contexte d'un réseau de neurones à plusieurs couches, on nommera ce genre d'apprentissage **rétropropagation** (***backpropagation***), un algorithme fondamental dans le domaine, fonctionnant comme son nom le dit par la propagation de la sortie aux entrées de composantes du gradient [11, 36] .

Nous n'expliquerons pas le détail des formules de rétropropagation, mais mentionnerons simplement qu'il s'agit d'appliquer à répétition la règle de dérivée en chaîne pour obtenir l'impact du changement de chacun des paramètres des neurones sur la fonction de coût. L'outil utilisé pour programmer les modèles fait ce calcul symbolique automatiquement (voir la section réservée aux outils employés), permettant de rapidement changer les détails du modèle.

1.3 Les architectures profondes

Lorsqu'on modélise ou explique un phénomène, il est tout naturel d'avoir recours à la décomposition en sous-parties, d'expliquer des choses complexes en termes de concepts plus simples. Il semble aussi tout naturel de commencer à éduquer un enfant en commençant par des choses simples comme fondement pour des notions plus complexes.

L'idée de faire émerger des caractéristiques de l'entrée a été mentionnée lors de l'explication de la notion de couche cachée. En effet, les neurones de cette couche intermédiaires en viendront à s'activer selon la présence de certains motifs dans les dimensions de l'entrée. Par contre, avec une seule couche connectée uniquement à l'entrée, ce n'est pas possible pour des unités cachées de se servir d'autres unités cachées pour extraire des caractéristiques qui s'exprimeraient en ces "termes plus simples" que les unités de la première couche représenteraient. Il est alors tout naturel d'essayer d'empiler plusieurs couches avant d'atteindre les neurones de sortie, en vue d'obtenir cette propriété désirable de "compositionnalité". On parlera alors de **réseaux de neurones profonds** [6, 19], la profondeur désignant le fait d'avoir plusieurs nonlinéarités entre l'entrée et la sortie.

On observe d'ailleurs ce phénomène de hiérarchie de modules dans le cerveau humain, notamment dans le cortex visuel. Il y a en effet un sens "avant" à la propagation de l'information, partant des ganglions de la rétine, passant par le noyau géniculé latéral, avant d'atteindre l'aire V1 et d'ensuite prendre divers chemins. Cette division en modules rappelle bien les couches cachées empilées.

Le problème avec cette idée de couches empilées est qu'il est dur d'entraîner de tels réseaux de neurones avec un critère supervisé et descente de gradient seulement, du moins c'est ce que suggèrent certains travaux [6, 15]. Cela ne semble pas fonctionner et les couches inférieures ne semblent pas apprendre de représentations utiles. Ce genre d'entraînement mènerait à du sur-apprentissage, donc une erreur faible sur l'ensemble d'entraînement, mais élevée sur l'ensemble de test. Une explication proposée est que le gradient se propage difficilement à travers plusieurs fonctions d'activation non linéaires, donc les couches inférieures ne sont pas bien entraînées.

L'exception notable à cette règle serait les réseaux convolutionnels, présentés plus

loin. Il faut aussi mentionner ici que des travaux récents suggèrent qu'en entraînant très longtemps et en présentant des variations infinies des exemples d'entraînement afin d'éviter du sur-apprentissage, c'est néanmoins faisable d'entraîner de tels modèles avec les principes de base énoncés jusqu'ici [13].

C'est pour faciliter l'entraînement de réseaux profonds qu'intervient l'idée de **pré-entraînement, non supervisé, vorace (*greedy*), couche-par-couche**. Comme le nom le dit, il s'agit non pas d'entraîner toutes les couches en même temps mais, plutôt, de les entraîner successivement. Une fois la première couche entraînée, on entraînera la seconde en considérant les sorties de la première, pour chaque exemple, comme entrée de la nouvelle couche. C'est "vorace" en ce sens que chaque couche tente d'optimiser son coût sans égard aux coûts des couches qui suivront.

"Non supervisé" indique ici qu'il n'y a pas de cible à prédire durant le pré-entraînement d'une couche mais que plutôt on cherchera des moyens d'extraire des caractéristiques "pertinentes" des exemples venant de la couche précédente, ou de l'entrée. La pertinence ne dépend que des exemples eux-mêmes et non pas d'une cible. On le fera en considérant d'autres critères que le critère supervisé utilisé jusqu'alors. Un critère utilisé de façon importante dans ce travail est celui de l'autoencodeur débruitant, qui sera présenté plus loin.

De manière générale, en cherchant des caractéristiques pertinentes, on cherche, un peu indirectement, à apprendre les **facteurs de variation** des données. Sans entrer dans les détails, un facteur de variation est une dimension abstraite des données assez indépendante d'autres dimensions du même genre. Penser à, par exemple, la hauteur d'une tête dans une photographie d'un visage. C'est une caractéristique de très haut niveau, étant assez difficile à extraire automatiquement des pixels originaux. C'est aussi une caractéristique assez indépendante d'autres caractéristiques telle que la couleur des yeux. Ce seraient de telles caractéristiques qu'on aimerait découvrir durant l'apprentissage. On aimerait que les représentations apprises reflètent des mesures de ces caractéristiques assez explicitement. Ces mesures "expliquent" ou résument d'une certaine façon une image donnée : si on a de bonnes caractéristiques mesurées, on connaît en quelque sorte les "causes" plus profondes de l'image.

Une fois toutes les couches initialisées par préentraînement, on formera un réseau profond qu'il sera possible, maintenant, de raffiner en vue de prédire une sortie. On ajustera donc maintenant tous les paramètres, de toutes les couches, selon le gradient de la fonction de coût supervisée, c'est-à-dire tenant compte de la bonne prédiction par exemple d'entraînement. C'est ce qu'on nomme le ***finetuning* supervisé**. On se trouverait alors à avoir initialisé les poids dans une région de leur espace plus favorable.

Cette façon de pré-entraîner couche par couche et de raffiner de façon supervisée a d'abord été introduite dans le contexte des **Deep Belief Networks** dans Hinton et al. [19], en 2006. C'était là une avancée ayant eu un grand impact sur le domaine. Ils sont expliqués brièvement à la prochaine section.

L'idée des architectures profondes a ensuite rapidement été étendue à d'autres composantes de base, notamment avec des couches entraînées en auto-encodeurs (voir la section 1.5) dans Bengio et al. [4].

La possibilité de composer en montant de niveaux d'abstraction est une caractéristique importante des architectures profondes. Cependant, l'approche s'attaque à plusieurs autres problèmes fondamentaux non couverts ici. Pour un traitement beaucoup plus détaillé, voir Bengio [6].

1.4 Modèles graphiques probabilistes et Restricted Boltzmann Machines

Certains des modèles dont nous parlerons impliquent des notions de modèles graphiques probabilistes, donc nous les présenterons très brièvement (voir le chapitre 8 de Bishop [11] pour une référence). Un **modèle graphique probabiliste** représente des relations de dépendances (et, par le fait même, d'indépendance) statistique entre des variables aléatoires, représentées par les noeuds et arcs d'un graphe. Il est important de remarquer qu'en général ces graphes n'ont *a priori* rien à voir avec les graphes représentant les réseaux de neurones, et ne représentent pas *directement* le déroulement d'un calcul.

Il y a deux grandes familles de modèles, dirigés et non dirigés, selon que les arcs "pointent" dans une direction ou non. Dans les modèles dirigés, la relation de dépendance est directe, et on peut exprimer la probabilité conditionnelle d'une variable en se

basant uniquement sur les variables dont elle dépend, c'est-à-dire celles qui "pointent" vers elle selon un arc dirigé.

Dans les modèles non dirigés, les formules ne sont pas calculées aussi directement. Par contre, on sait que si on connaît la valeur de *toutes* les variables A se connectant à un groupe de variables B, alors la valeur de ces variables B est indépendante du reste graphe C (tout le graphe moins A et B). On dit que les variables B ainsi isolées sont conditionnellement indépendantes du reste du graphe C, conditionnellement aux variables dans A.

C'est important lors du problème de l'**inférence**. Présentons ce concept dans le contexte des ***Restricted Boltzmann Machines*** (**RBM**), un type de modèle probabiliste graphique à connexions non dirigées [39]. Il s'agit de deux couches d'unités (variables aléatoires), les unités d'entrée (unités **visibles**) et de la couche cachée, pouvant chacune prendre les valeurs 0 ou 1 (figure 1.1). Une unité de la couche cachée est connectée à toutes les unités de la couche d'entrée, et *vice versa*.

La distribution de probabilité sur les états de toutes les unités (vecteurs-colonnes x et h) est donnée par une fonction où le négatif du terme dans l'exponentielle est nommé **énergie** :

$$p(x,h) = \frac{e^{-h'Wx-b'x-c'h}}{Z}$$

où W est une matrice des poids entre les deux couches, et b et c sont les vecteurs de biais associées aux unités visibles et cachées respectivement. L'idée est que l'entraînement visera à donner une énergie faible aux exemples d'entraînement, et une énergie élevée ailleurs. À noter que cette fonction ne se calcule pas facilement, ceci dû à la présence d'une constante de normalisation Z dont le calcul prohibitif nécessite une somme sur tous les états possibles des unités.

L'inférence consiste à déterminer la probabilité associée aux valeurs de variables aléatoires dont on ignore la valeur, étant donné les valeurs connues d'autres variables du modèle. Pour les RBM, il s'agira le plus souvent de déterminer la probabilité associée aux valeurs 0/1 des unités de la couche cachée h étant donné un exemple d'entrée x.

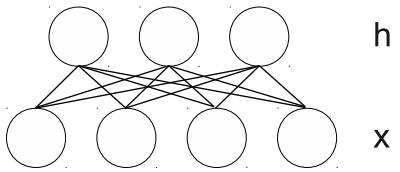


Figure 1.1 – Modèle graphique d'une petite RBM

Grâce à la propriété d'indépendance conditionnelle, les unités de la couche cachée sont indépendantes les unes des autres lorsqu'on connaît les valeurs des variables d'entrée, car elles ne sont pas connectées entre elles directement. On peut donc calculer l'état d'une unité de la couche cachée sans connaître l'état des autres. Ça rend donc l'inférence très rapide.

Dans la percée de 2006 dont il est question ci-haut, les RBMs étaient entraînées avec un algorithme calculant une approximation du gradient de la log vraisemblance des données, la **contrastive divergence (CD)**. Les paramètres appris à chaque couche servaient ensuite à initialiser un modèle de type génératif un peu particulier, à connexions dirigées pour toutes les couches sauf la dernière, nommé ***Deep Belief Network*** (**DBN**). Pour utilisation comme architecture de classification, il est aussi possible de simplement utiliser ces paramètres comme initialisation d'un réseau de neurones profond tel que présenté jusqu'à présent, en ajoutant une dernière couche servant à la sortie, donc la prédiction comme telle.

1.5 Auto-encodeurs

Dans la section 1.3, il était question d'extraction de caractéristiques "pertinentes" à l'entrée de façon non supervisée. L'**autoencodeur** est un principe permettant de trouver de telles caractéristiques.

L'idée est simple mais ingénieuse : on entraînera un réseau de neurones à une couche cachée h à prédire en sa sortie z son entrée x, une tâche qu'on nomme **reconstruction**. Le critère sera donc la minimisation d'une **erreur de reconstruction** $L(x,z)$. En principe, la couche cachée devra donc contenir de l'information pertinente à la reconstruction. Par exemple, si la couche cachée contient moins d'unités qu'il y a d'entrées, alors logique-

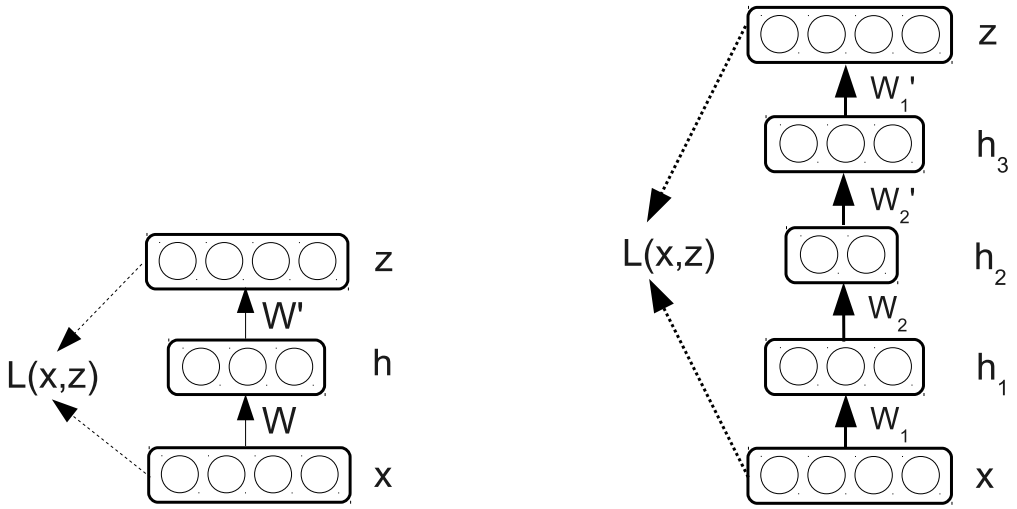
ment pour bien reconstruire elle doit apprendre à “résumer” l’entrée, puisque l’information pour la reconstruction est entièrement contenue dans la couche cachée. Ainsi des caractéristiques “pertinentes” à la reconstruction devront être extraites.

Il faut noter que les paramètres pour passer de h à z, W', ont une forme matricielle qui est la transposée de la forme des paramètres W. Si les poids sont “**liés**” (***tied***) alors il s’agit effectivement de la transposée et durant l’entraînement le contenu de la matrice sera contraint de rester le même pour les deux directions. Autrement, s’ils ne sont pas liés, ils sont libres de prendre des valeurs différentes. En pratique ça signifie maintenir deux matrices distinctes en mémoire.

Il est aussi possible d’effectuer la reconstruction en faisant passer l’information dans plusieurs couches cachées $\{h_1, ..., h_n\}$. On obtient ainsi un **autoencodeur profond**. Comme ce type de réseau contient plusieurs couches, il sera en général préférable de préentraîner les couches comme expliqué dans la section sur les architectures profondes, et de les faire correspondre par paires : la première et la dernière, la seconde et l’avant-dernière, etc., comme on peut voir dans la figure 1.2 par les correspondances entre les noms des paramètres de chacune des couches.

Par contre, dans cette version “de base” de l’autoencodeur, si le nombre d’unités dans la couche cachée est plus grand ou égal au nombre d’entrées, le modèle peut alors apprendre de façon triviale la fonction identité. C’est facile en copiant une à une les valeurs de l’entrée dans la couche cachée, par exemple. On n’obtient donc pas de caractéristiques extraites intéressantes. Pour contourner ce problème, différentes contraintes peuvent être imposées durant l’apprentissage. C’est ce qui est fait avec l’autoencodeur débruitant (voir section 1.6).

Pour des modèles effectuant de la reconstruction, on peut évaluer l’erreur de reconstruction, la différence entre l’entrée originale et l’entrée reconstruite. Cela permet de comparer les modèles entre eux. Pour le faire numériquement, dans ce travail, la MSE ou la NLL seront utilisées. Ces coûts serviront pour l’évaluation des modèles et comme fonctions objectifs pour guider l’entraînement.



(a) Autoencodeur “de base” (b) Autoencodeur profond

Figure 1.2 – **Représentations d’un autoencodeur ordinaire (à gauche) et profond (à droite)**. Dans le cas de l’autoencodeur profond, c’est ici un exemple où deux couches seraient préentraînées avant de former l’autoencodeur profond, donnant des poids liés W_1et W_1' ainsi que W_2et W_2'.

1.6 Auto-encodeurs débruitants et types de bruit

Pour éviter qu’un autoencodeur apprenne la fonction identité, et pour le forcer à apprendre des caractéristiques “robustes” aux variations dans l’entrée, l’**auto-encodeur débruitant** va d’abord corrompre l’entrée en lui ajoutant du bruit mais demander au modèle de prédire en sortie la version originale [41]. La sortie doit donc ressembler le plus possible à l’entrée originale, avant le processus de corruption. Ce faisant, le modèle est contraint d’apprendre à reconnaître des caractéristiques robustes à ce bruit, c’est-à-dire gardant à peu près la même valeur même si le bruit change.

Une des versions utilisées dans ce travail sera celle où le bruit est de type “poivre” dans une image. Ainsi on sélectionnera une certaine proportion de pixels, tirés aléatoirement, dans l’image en entrée et on mettra leur valeur à 0. On peut aussi utiliser d’autres types de bruit, comme un bruit “poivre & sel” : certaines entrées mises à 1, d’autres à 0, plutôt qu’uniquement des 0. Il est même possible d’utiliser des bruits plus complexes, comme des occlusions couvrant plusieurs pixels contigus. Voir la section 4.4 à ce sujet.

Un résultat à retenir, présenté dans la publication mentionnée ci-haut, est qu'un autoencodeur débruitant entraîné sur des images aura tendance à apprendre des "filtres" (paramètres de chaque neurone de la première couche considérés comme des images) correspondant à des sous-parties *locales* de l'image d'entrée. On considérera comme désirable cette propriété, car c'est plus robuste au bruit et ça rappelle les champs réceptifs locaux des neurones de V1. Voir la figure 1.3 à cet effet.

Notons qu'un bruit de 0% durant l'entraînement correspond à un autoencodeur ordinaire. Nous n'avons fait aucune expérience où nous utilisions un tel bruit nul.

1.7 Les architectures convolutionnelles

Beaucoup de jeux de données utilisés en apprentissage machine sont sous forme d'images. Il s'agit de la presque totalité des données utilisées pour le présent travail. Les images représentent une entrée bien particulière avec des propriétés importantes dont il serait dommage de ne pas tirer profit. Pour les analyser, il est bien naturel de se tourner vers l'étude de la vision comme inspiration. Si on fait abstraction des déplacements des yeux, on peut en effet établir une correspondance intéressante entre le champ visuel perçu par la rétine et les images mises en entrées d'un réseau de neurones artificiels.

Le domaine des neurosciences de la vision offre plusieurs observations intéressantes concernant l'architecture supposée du cortex visuel et des aires rattachées. Dans la section sur les architectures profondes, il a été fait allusion au fait que les couches inférieures du cortex visuel semblent représenter des "objets plus simples", les couches supérieures montrant des niveaux d'abstraction plus élevés. Une représentation simpliste du flot d'information provenant des yeux est montrée dans la figure 1.4.

L'information provenant de la rétine atteint le cortex par une aire nommée V1. L'observation la plus importante, alors, est sûrement la localité des champs réceptifs des neurones. En effet, un neurone donné dans cette région réagira à des stimuli présents dans une zone assez restreinte de la rétine. La correspondance entre des neurones de V1 et la rétine forme ainsi une "carte" dite *rétinotopique* dans V1.

Il s'agit là d'une information importante pour la modélisation : inutile qu'un neurone

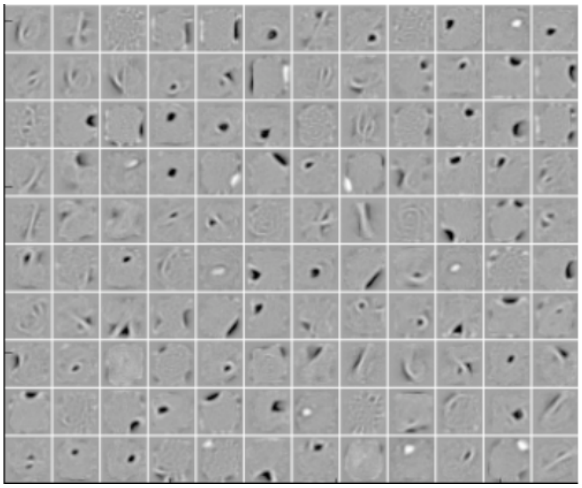


Figure 1.3 – **Exemples de filtres appris lors de l'entraînement d'autoencodeurs débruitants** à niveau de bruit poivre 50%. Image tirée de Vincent et al. [41].

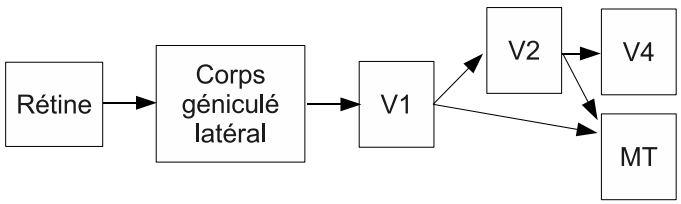


Figure 1.4 – **Représentation schématique du flot de l'information provenant de la rétine**, dans la direction "avant" (feed forward). Inspiré de Behnke [3]. Les "modules" suivant V1 exercent différentes fonctions de plus haut niveau qui seraient reliés notamment à la reconnaissance d'objet (flot dit "ventral" allant vers V4) et au positionnement et traitement du mouvement du sujet et des objets (flot dit "dorsal" allant vers MT).

de la première couche soit connectés à l'ensemble de l'entrée ("rétine") si seulement une petite zone aura une influence non nulle sur son activation.

Des travaux par Hubel & Wiesel en neurosciences [21–23] ont inspiré Fukushima pour la création du modèle **Neocognitron** [18]. Il y propose un modèle hiérarchique avec deux grands types principaux de neurones : les **cellules simples** et **cellules complexes**. Des couches alternées composées de cellules simples, puis complexes, constituent le modèle.

Les cellules simples réagissent à la présence d'un motif dans leur champ réceptif. Un exemple souvent donné sera de détecter la présence petits bouts de contours (edge) dans une orientation particulière dans leur champ réceptif. Si le motif corrèle assez bien avec l'activité dans le champ réceptif, le neurone s'activera. La réponse des cellules simples est apprise durant l'entraînement du modèle Neocognitron.

On remarquera cependant que la détection de contour orientée à 45 degrés n'est pas quelque chose de spécifique à un neurone particulier, mais plutôt partagé par des neurones couvrant l'ensemble du champ visuel. C'est de cette observation que vient l'idée du **partage de poids** des architectures convolutionnelles : on tentera de détecter chaque motif appris à chaque position de l'image d'entrée, sans apprendre un motif séparé pour chaque position.

En pratique chaque couche de cellules simples sera composée d'un certain nombre de **filtres de convolution**. En effet, pour appliquer le même motif à toute l'entrée, une opération de **convolution** du filtre fait le travail. On obtient donc, pour chaque filtre, une **"*feature map*"** de sortie donnant la réponse, à chaque position de l'entrée, de la somme des poids du filtre multipliés un à un aux pixels correspondants dans le voisinage de cette position. Le fait d'avoir plusieurs filtres nous permet d'avoir différents motifs pour une même couche, ce qui est bien sûr désirable si on veut pouvoir détecter, par exemple, plusieurs orientations de bords (dans le cas où des détecteurs de bords sont appris). On obtient donc plusieurs *feature maps*.

La réponse des cellules simples est cependant assez capricieuse sur la position du motif. S'il se déplace un peu, la réponse peut changer beaucoup. C'est là qu'interviennent les cellules complexes, en introduisant de l'invariance sur la position. Ainsi une

cellule complexe prendra en entrée l'état de quelques cellules simples à champs réceptifs proches, le ***pool*** de la cellule complexe. Cette dernière s'activera si l'une ou l'autre des cellules simples est active. On gagne donc une certaine indifférence ou invariance à la position du motif. La réponse des cellules complexes n'a pas à être apprise ; elle est prédéfinie de façon dure.

On connectera donc une première couche de cellules simples à l'entrée, puis des cellules complexes à ces cellules simples. On pourra ensuite recommencer en prenant la sortie de ces premières cellules complexes comme entrées d'une prochaine couche de cellules simples. Ainsi par les couches de cellules simples on pourra détecter des motifs de plus en plus haut niveau et complexité. Par les couches de cellules complexes on gagnera en invariance sur la position du motif et de la position relative de ses sous-parties.

En pratique dans ce travail on utilisera une opération de réponse maximale sur un champ réceptif ("*pool*") de taille fixe. Ainsi typiquement une cellule complexe prendra en entrée la réponse d'une zone de 2 par 2 unités d'une seule feature map et retournera la valeur maximale qui s'y trouve. Cette opération se nomme ***maxpooling***. Il s'agit d'une des façons de faire l'étape de **sous-échantillonnage** (ou **"*pooling*"**) dans un réseau convolutionnel, en pratique. Il n'y aura pas de partage d'aire couverte par deux cellules complexes adjacentes, dans l'implémentation. Le facteur de sous-échantillonnage (*pool_size*), par exemple 2, donne la dimension de la zone servant à calculer une unité d'une couche résultante du pooling.

Les architectures convolutionnelles permettent donc, en résumé, d'apprendre des caractéristiques sous forme de hiérarchie, de simples à complexes, tout en ayant une certaine invariance sur leur position et taille dans l'entrée. L'idée d'entraîner de telles architectures par rétropropagation, pour des fins de classification, est apparue plus tard, avec le modèle LeNet [26]. En effet, le Neocognitron était entraîné par des critères non supervisés n'impliquant pas de gradient.

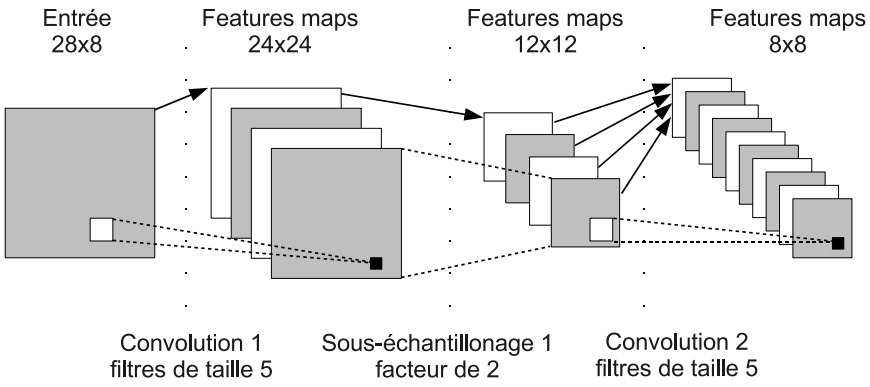


Figure 1.5 – **Diagramme des premières couches d'un réseau convolutionnel**, illustrant différents phénomènes qu'on y retrouve. Les flèches montrent l'entrée qui arrive à la dernière *feature map* de chaque couche, montrant qu'à la dernière convolution la dernière feature map est calculée à partir de toutes les features maps de la couche précédente. On voit aussi que le nombre de feature maps change aux étapes de convolution, ce qui n'est bien sûr pas le cas pour les étapes de sous-échantillonnage. Les convolutions sont illustrées par le calcul d'une unité d'une feature map résultant d'une convolution, l'état de cette unité étant calculé à partir d'une zone de la taille du filtre dans la couche précédente.

CHAPITRE 2

ARCHITECTURES PROFONDES À RELAXATION EXISTANTES

2.1 Définition du terme “relaxation”

Le thème des présents travaux, ce qui les distingue, concerne ce qu’on nomme la “relaxation” de la représentation. Une définition est bien sûr de mise pour mieux cerner le but recherché. Dans le contexte de réseaux de neurones multicouches, on appellera donc **relaxation** le fait de laisser la représentation, donc l’état d’activation des neurones, évoluer en la recalculant d’après l’état à l’étape précédente, sans pour autant changer l’entrée.

Pour fins de clarté, rappelons la convention adoptée ici qu’en propageant l’information à partir de l’entrée de couche en couche, on dit qu’on va vers le “haut”. Un des objectifs principaux avec la relaxation est donc de permettre aux couches supérieures d’influencer les couches inférieures à travers les **connexions de rétroaction**, connexions allant des couches du “haut” vers celles du “bas”. Ce n’est bien sûr pas possible dans le cas d’un modèle purement à propagation avant. En reprenant l’idée que les couches supérieures représentent des abstractions de “plus haut niveau” ou, dans le cas du système visuel, des objets plus larges et complexes, le but est donc en général de permettre au “global” d’influencer le “local”, au “haut niveau d’abstraction” d’influencer le “bas niveau”.

Les réseaux ainsi formés seront donc une forme particulière de réseaux récurrents, abordés dans une prochaine section. Cependant, il n’y a pas ici d’évolution de l’entrée, comme c’est normalement le cas avec des réseaux récurrents utilisés pour modéliser des séries temporelles.

Pour mieux situer la pertinence des présents travaux, les sections qui suivent expliquent des architectures semblables et les motivations derrière notre approche.

2.2 Les connexions de rétroaction dans le système visuel

Il y a eu dans les dernières dix années, environ, plusieurs travaux portant sur le rôle des connexions de rétroaction dans le système visuel des primates, donc de l'être humain. Nous parlerons de quelques observations soulevées. Avant toute chose, il faut noter que les connexions entre les modules de la figure 1.4 – et en fait bien d'autres modules qui n'y figurent pas – sont pour la grande majorité bidirectionnelles. C'est-à-dire que, par exemple, dans la direction *feedforward* l'information va du corps géniculé latéral à V1, mais il y a aussi des axones propageant l'information en sens inverse.

Ce qui suit est basé principalement sur Bullier [12]. Cet article argumente en faveur d'une réexamination du modèle le plus populaire où les connexions *feedforward* dominent et où la vision est vue comme une série d'étapes de traitement successives. Il est question à la fois des connexions de rétroaction et de **connexions latérales**, c'est-à-dire entre neurones d'un même voisinage dans une même couche.

Les arguments avancés sont multiples, mais il est plus intéressant de s'attarder ici sur l'utilité avancée des connexions de rétroaction que sur l'argumentation elle-même. Mentionnons tout de même que les connexions latérales compteraient pour 60% des liens totaux et que pour V1 les connexions de rétroaction sont en plus grand nombre que les connexions *feedforward* en provenance du corps géniculé latéral.

Des indices reliés à la chronologie de propagation suggèrent que les connexions de rétroaction peuvent jouer un rôle dès les premières étapes de traitement d'un nouveau stimulus. D'abord la propagation d'information vers MT est plus rapide que pour celle atteignant V4. De plus, les connexions latérales dans un voisinage proche sont en fait plus lentes que les connexions de rétroaction (40 ms vs 10 ms).

Des expériences menées par les auteurs suggèreraient que les connexions de rétroaction sont plus utiles lorsqu'il y a ambiguïté dans l'entrée, quand par exemple l'objet à percevoir au premier plan est difficilement discernable sur le fond, l'arrière-plan. Cela peut être dû à une scène complexe, ou encore à un faible contraste. Cela rejoint l'idée plus générale, attribuée à Mumford [31], selon laquelle le traitement purement *feedforward* est approprié quand l'information locale perçue par un neurone permet de prendre

de bonnes “décisions” locales. En présence d’ambiguïté sur l’information locale, il est utile d’aller chercher de l’information de contexte plus global.

Finalement, une autre utilité hypothétique importante des connexions de rétroaction serait au niveau de mécanismes d’attention, c’est-à-dire le fait de se concentrer sur une partie de l’entrée et d’ignorer le reste. Des effets qu’on chercherait à expliquer seraient par exemple la capacité à repérer et se concentrer facilement sur des objets d’une certaine couleur distinctive dans le champ visuel (effet de “*pop-out*”) ou encore se concentrer sur un des côtés du champ visuel. On observe des changements forts entre des réponses de neurones individuels dans V1 lors d’expériences impliquant des tâches de ce genre, plus précisément lorsque les tâches impliquent des contrastes faibles. Cependant, c’est difficile dans ces cas de bien distinguer le rôle des connexions de rétroaction des autres types de connexions entre modules corticaux. Il y a malgré tout des expériences qui suggèrent que les connexions de rétroaction seraient impliquées dans l’attention sur des caractéristiques spécifiques dans une scène encombrée.

2.3 Les réseaux de neurones récurrents

Il existe une large gamme de réseaux de neurones dits **récurrents** où l’état d’activation des unités est récupéré d’une façon ou d’une autre comme partie de l’entrée pour de prochains calculs. Bien souvent les données d’entrées seront des séries temporelles. Ces réseaux forment des systèmes dynamiques, c’est-à-dire qu’il y a un état et une règle d’évolution d’un état à l’autre.

Les réseaux récurrents sont très intéressants en principe, notamment parce qu’ayant une mémoire on s’attendrait à ce qu’ils constituent une base intéressante pour des mécanismes intelligents apprenant à résumer et tenir compte du **contexte** de l’entrée courante. Malheureusement, ils forment une classe de systèmes normalement difficiles à entraîner sur de longues séquences d’entrées.

Une façon de comprendre pourquoi l’entraînement est difficile est de s’intéresser à une méthode d’entraînement typique, la **rétropropagation à travers le temps** (*back-propagation through time*, BPTT) [35]. Supposons qu’on ait un réseau à une seule

couche cachée et des séries de 3 entrées. L'idée est de "dérouler" l'état dans le temps, donc d'empiler les couches aux différents pas de temps comme s'il s'agissait d'un réseau profond. Cependant les *poids entre les couches sont les mêmes*, contrairement à un réseau profond tel que vu jusqu'alors. On entraîne en faisant une descente de gradient sur un coût lié à une cible à prédire en toute fin de la série, donc de ces "couches déroulées".

Bengio et al. [5] montrent qu'un tel système dynamique entraîné par descente de gradient, s'il apprend à représenter et garder en "mémoire" de l'information provenant des premières entrées dans une longue série, fait en sorte que le signal d'entraînement, donc le gradient, devienne très petit en redescendant ces couches déroulées. C'est-à-dire que le signal d'entraînement ne peut pas influencer correctement les premières itérations de la série quand la représentation permet de conserver une information sur le début de la série de façon stable.

Comme ce phénomène de ***vanishing gradients*** est inhérent à cette forme d'entraînement sur de longues séquences, des mécanismes doivent être trouvés pour contourner le problème. Parmi les autres types de réseaux récurrents connus et donnant de bons résultats, on trouve les Echo State Networks [24] et les modèles Long Short Term Memory [20].

Cependant le modèle présenté ici n'est pas entraîné sur une série temporelle. L'entrée à toutes les étapes est en fait constante. Plus de détails sont donnés plus loin, mais notons qu'ainsi on se trouve ici aussi à contourner le problème de *vanishing gradients*, ou enfin c'est un espoir.

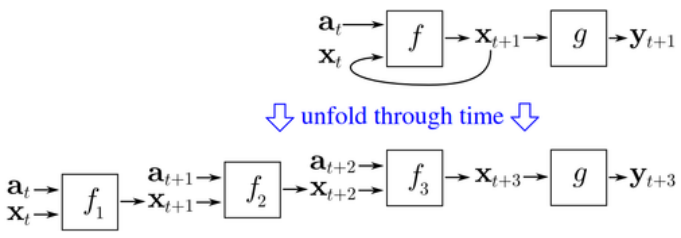


Figure 2.1 – **Illustration du principe de "déroulage" à travers le temps**. Figure prise à http ://en.wikipedia.org/wiki/Backpropagation_through_time

2.4 Les machines de Boltzmann profondes

Un premier modèle se rapprochant beaucoup de ce que nous faisons, et ayant servi d'inspiration à nos travaux, est la **Deep Boltzmann Machine (DBM)** [37, 38]. Il a déjà été question plus tôt des RBM. Les DB*M* sont un empilement de couches de type RBM et préentraînées comme telles, comme c'est le cas des DB*N*. Cependant, contrairement aux DB*N*, les DB*M* conservent les connexions bidirectionnelles entre toutes les couches lors d'un subséquent entraînement non supervisé impliquant toutes les couches. La fonction d'énergie (voir la section sur les RBM) y est effectivement la somme des fonctions d'énergie des RBM qui la composent.

Une DBM est donc un modèle probabiliste graphique à plusieurs couches où les connexions entre unités de couches successives sont non dirigées. Ces connexions ne représentent pas directement un calcul mais plutôt une relation de dépendance statistique entre les unités liées. On dira que certains états d'activations des unités sont plus probables que d'autres, selon une distribution de probabilité paramétrée (ou, de façon équivalente, une fonction d'énergie paramétrée) par les poids entre les unités des différentes couches. Ainsi, plutôt que de trouver l'état des unités par une passe unique *feedforward* vers les couches supérieures, il est nécessaire de faire de l'inférence pour déterminer quels états d'activations des unités cachées sont probables étant donné l'entrée (distribution conditionnelle).

Pour une seule couche cachée (donc une RBM), l'inférence est directe, comme vu précédemment. L'espérance de l'activation de chaque unité cachée est obtenue par un seul calcul rapide équivalent au calcul des activations de neurones avec sigmoïdes. Cependant, avec plusieurs couches, une telle inférence exacte et rapide n'est pas possible.

Une façon de faire cette inférence est de recourir à une méthode approximative nommée ***mean field***. Pour chaque nouvelle entrée x présentée au modèle, on créera une distribution Q approximant $p(h|x)$, où h est la variable aléatoire représentant toutes les couches supérieures. Cette distribution est définie de telle sorte que les unités sont statistiquement indépendantes, ce qui facilite grandement le calcul des valeurs espérées. En

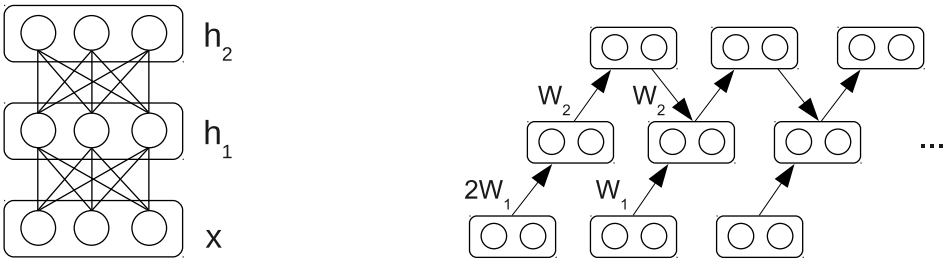


Figure 2.2 – **Illustration du modèle graphique d'une Deep Boltzmann Machine (à gauche) et des mises à jour de l'inférence *mean field* (à droite)**. Ici ce serait avec deux couches cachées, alors que dans les articles originaux il y en a trois. Remarquer que durant la propagation initiale vers le haut, on doit doubler les paramètres des entrées à la première couche cachée, pour compenser vu que par la suite la première couche cachée reçoit en entrée des informations venant de la couche supérieure aussi, ce qui augmente le "poids" total par environ un facteur de 2.

fait on obtiendra la valeur d'une unité en se basant sur les espérances de ses voisines directes dans le graphe, en faisant un calcul identique à celui donnant l'activation de neurones avec fonction logistique ($\sigma(w'x+b)$).

Dans le cas de la DBM, pour combiner les valeurs des couches inférieures et supérieures, on divisera simplement par 2 les entrées venant de ces deux directions. Par contre, comme il s'agit d'un réseau multicouche et que l'on n'a pas, au départ, de valeurs d'activations pour les couches supérieures, il est nécessaire malgré tout de faire un calcul *feedforward* initial avant d'itérer et de recalculer chacune des couches en fonction des autres. On peut ainsi faire une trentaine d'itérations avant d'obtenir les espérances approximatives des activations des unités cachées.

C'est de cette méthode d'inférence dont le modèle utilisé dans notre travail s'inspire, tel que présenté plus loin.

2.5 Architectures convolutionnelles à relaxation

2.5.1 Neural Abstraction Pyramid

Quelques mois après avoir commencé ce travail, et à peu près un mois après avoir attaqué la version convolutionnelle, des recherches nous ont mené aux travaux d'un chercheur nommé Sven Behnke, datant de la période 1998-2003, sur une architecture

qu'il nomme **Neural Abstraction Pyramid** [3]. Ces travaux ne me semblent pas très connus (à en juger par le nombre de citations) mais méritent une attention particulière ici. C'est qu'ils s'apparentent énormément à ce qui est fait pour ce mémoire, en particulier ce qui est fait concernant la version convolutionnelle et pour la méthode employée pour l'entraînement.

Il s'agit d'une archtitecture s'inspirant de plusieurs observations sur le cortex visuel, et partageant beaucoup avec les architectures convolutionnelles introduites plus haut. Les couches ont des liens allant dans les deux directions, donc il y a une rétroaction. De plus, il y a des connexions locales entre neurones d'une même couche, ce qui est une caractéristique importante de la connectivité dans V1. Pour calculer la réaction du réseau à une entrée, les activations sont recalculées de façon itérative à la manière d'un réseau récurrent. Il y a d'autres différences un peu moins importantes, et le vocabulaire utilisé est un peu dépaysant, mais il serait trop long d'entrer dans les détails ici.

Dans un des derniers articles à leur sujet, les paramètres sont entraînés avec un critère non supervisé où la dynamique est déroulée et l'entraînement se fait selon la méthode Backpropagation Through Time (voir la section sur les réseaux récurrents). Il utilise un critère non supervisé de reconstruction où l'entrée n'évolue pas, ce qui est précisément ce qu'on fait dans ce travail.

Il emploie son architecture avec succès sur plusieurs tâches auxquelles il l'adapte, notamment la localisation des yeux dans des images de visages [2] et le débruitement d'images [1].

Malgré ces similarités, des différences importantes sont qu'ici nous traitons prioritairement le cas des couches complètement connectées (par opposition à une connectivité convolutionnelle) et que notre critère non supervisé soit semblable à celui d'un autoencodeur débruitant. De plus, nous utilisons un pré-entraînement couche par couche avec critères non supervisés de reconstruction, ce qui ne me semble pas avoir été le cas dans ces travaux. Les ensembles de données et les tâches étant différentes, ce serait difficile de se comparer numériquement à ces expériences.

2.5.2 Convolutional deep belief network

Lee et al. [28] présentent une autre architecture de type convolutionnelle importante, plus récente et permettant une forme de relaxation aussi. Leur modèle est basé sur une fonction d'énergie semblable aux RBM. Des couches de convolution à l'activation assez "standard" sont alternées avec des couches de *probabilistic max pooling*. Le but de ce dernier type de couche est d'assigner une probabilité aux états des unités précédent le *pooling*, de façon à pouvoir échantillonner un état de la couche inférieure étant donnée l'état de la couche supérieure. Cet échantillonnage se fait par *pool*, c'est-à-dire en pigeant l'état des unités groupées sous une même unité de *maxpooling* de la couche supérieure. Chaque *pool* suivra en fait une distribution multinomiale selon laquelle une seule (ou aucune) des unités peut être activée à la fois.

L'énergie pour chaque couche étant définie, ils forment ensuite un réseau profond qu'ils nomment *Convolutional deep belief network* et peuvent faire de l'inférence par échantillonnage de Gibbs de l'état de chacune des couches tour à tour. Ils peuvent alors faire de la complétion d'image, cachant une moitié de visage et récupérant, après plusieurs itérations d'échantillonnage, une partie de l'autre moitié. Ils obtiennent aussi de bons résultats en classification sur Caltech-101, un jeu de données populaire avec des images couleur de plus grande taille (environ 300x200 pixels) [16]. Notons que grâce à l'architecture convolutionnelle, les images d'entrée peuvent avoir jusqu'à 150 pixels dans l'une ou l'autre des dimensions, ou être réduites à cette taille maximale, si besoin est.

2.6 Autres travaux pertinents

Le sujet de la relaxation et des connexions de rétroaction étant vaste, et touchant des théories intéressantes sur l'attention dans le cortex, il y aurait plusieurs autres travaux qu'il serait pertinent de mentionner. Pour éviter d'allonger grandement le texte, ce sera fait succintement ici.

Du côté de la relaxation comme telle, Bengio et Gingras [8] utilisent un modèle à relaxation du genre présenté dans ce travail mais entraîné par un signal supervisé unique-

ment, donc à une tâche de classification. De plus, c'est dans le cas où des données sont parfois manquantes dans l'entrée. Cependant, les données sont très différentes de celles utilisées ici et les couches sont de plus petite taille. Ils observent des gains très appréciables en vitesse d'entraînement et en performance de classification sur les données utilisées.

Du côté des neurosciences, il y a plusieurs travaux récents qui cherchent à faire le lien, au moins de façon hypothétique, entre l'inférence dans des réseaux du genre des DBM et le cortex. Notamment, Fiser et al. [17] explorent spécifiquement le sujet de l'inférence "optimale", dans le cortex visuel, au sens de travailler avec plusieurs hypothèses en parallèle et de les considérer selon leur probabilités respectives. Ils avancent notamment l'idée que cela pourrait se faire avec un mécanisme similaire aux chaînes de Markov utilisées dans les modèles graphiques comme les DBM. Une idée intéressante qui en ressort est qu'on peut tester cette hypothèse en comparant l'activité dans le cortex lorsqu'un stimulus est présenté, et lorsqu'on demande à un sujet de se remémorer ce stimulus.

Dans la même veine, Lee et Mumford [29] proposent un modèle où les différents modules de la hiérarchie de la vision sont vus comme de grandes variables aléatoires, liées dans un modèle graphique où l'interinfluence des modules leur permet d'atteindre un état où tous doivent tenir compte des autres, évitant ainsi de mauvaises décisions locales hâtives. Ils explorent l'idée que le maintient de multiples hypothèses pourrait se faire avec un mécanisme semblable aux *particle filters*, modèles populaires en vision informatique. Une autre idée importante est que V1 serait un tampon où se dessinerait le résultat des modules d'ordre supérieur au cours du processus d'établissement de consensus entre les différentes parties.

Reichert et al. [33] explorent un modèle d'hallucinations causées par la perte d'une partie du champ visuel grâce à des machines de Boltzmann profondes. Concernant les hallucinations en général, Corlett et al. [14], eux relevant du domaine des neurosciences, proposent qu'elles s'expliqueraient par des problèmes de prédiction entre les modules du cortex, organisés en hiérarchie, se basant sur le cadre probabiliste de Fiser et al. [17] mentionné ci-haut.

CHAPITRE 3

MODÈLES À RELAXATION UTILISÉS DANS CES EXPÉRIENCES

Le terme “relaxation” a été défini à la section 2.1. Dans cette section-ci, il sera question des modèles concrètement utilisés pour ce travail. Il y en a deux, chacun avec ses variations, mais partageant à peu près le même processus de relaxation. Le premier, chronologiquement et en terme de quantité d’expériences effectuées, est la version “**complètement connectée**”, c’est-à-dire où tous les neurones d’une couche se connectent à tous les neurones de la couche supérieure et/ou inférieure, selon le cas. Le second est la **version convolutionnelle**, où la connectivité est locale, où il y a partage de poids et agrégation (*pooling*) pour diminuer la résolution d’une couche à l’autre, s’inspirant du principe des réseaux convolutionnels expliqué dans la section 1.7. Tout d’abord, par contre, le principe fondamental de relaxation utilisée sera expliqué.

Un mot, avant tout, sur les étapes d’entraînement : comme expliqué dans la section sur les architectures profondes, l’entraînement de réseaux profonds sans pré-entraînement donne généralement une mauvaise généralisation. L’approche adoptée ici est d’abord de pré-entraîner couche par couche en autoencodeur débruitant. Les paramètres ainsi obtenus sont ensuite utilisés pour intialiser l’entraînement du réseau relaxant, étape nommée “***finetuning* non supervisé**” ou “***finetuning* en relaxation”** car elle n’implique pas d’information de classe. Une étape intermédiaire d’entraînement en autoencodeur profond a aussi été essayée pour les réseaux convolutionnels, comme expliqué à la section 3.3.

Si les buts d’une expérience incluaient de faire de la classification, une dernière étape de “***finetuning* supervisé**” était effectuée pour apprendre à prédire la classe à partir du réseau.

3.1 Relaxation de type mean field entraînée avec le principe de l'auto-encodeur débruitant

La relaxation effectuée dans ce travail rappelle celle utilisée pour l'inférence approximée *mean field* dans les DBM. Par contre, ici, la notion de déformation (ou addition de bruit) de l'entrée introduit une différence importante, de même que l'interprétation mathématique du calcul.

L'ordre de recalcul des activations des différentes couches est illustré dans la figure 3.1. On voit que, partant d'une entrée x, du bruit (décrit plus loin) est d'abord ajouté pour former $\tilde{x}$ avant de faire une première propagation avant pour initialiser la représentation à toutes les couches. Ensuite ***num_iter*** itérations de relaxation, correspondant à d'autres "diagonales montantes" de propagation avant, sont faites. Cependant, pour ces autres propagation, les calculs de la première couche cachée à l'avant-dernière (***num_hidden_layers*** *- 1)* utilisent en entrée une influence venant de la couche supérieure à l'itération de relaxation précédente. Pour la version complètement connectée, cela signifie additionner les termes *avant la nonlinéarité*. Pour une couche intermédiaire h_n à l'itération de relaxation k, l'activation est donnée par :

$$h_n^k = \sigma(\frac{1}{2}(W_n h_{n-1}^k + b_n + W_{n+1}^{'} h_{n+1}^{k-1} + b_{n+1}^{'})) \tag{3.1}$$

Quelques points à noter, dans cette formule et la figure 3.1 :

- h_0 correspondrait en fait à x et ses reconstructions z_k. La reconstruction n'est influencée que par ce qui vient du haut, donc on oublie les termes concernant d'hypothétiques "h_{-1} et b_0", et il n'y a pas de division par deux.
- Pour la dernière couche (la plus profonde), il n'y a pas de division par deux non plus, et il n'y a une influence que de la couche du dessous, donc on oublie les termes qui correspondraient à une hypothétique couche encore plus élevée.
- L'influence de la couche inférieure vient de l'itération courante, mais l'influence de la couche supérieure vient de l'itération précédente. On peut le voir sur la figure 3.1.
- L'influence de la couche supérieure se fait à travers des matrices de poids qui

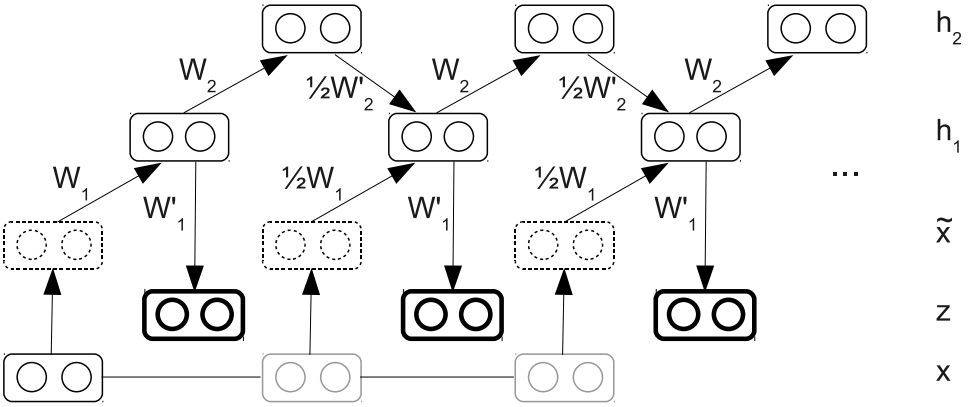


Figure 3.1 – **Diagramme de la relaxation telle qu'utilisée dans ce travail**, ici avec deux couches cachées. En pratique 2 ou 3 couches étaient utilisées. On voit que la même entrée x est réintroduite à chaque itération de relaxation, et est d'abord bruitée en une version $\tilde{x}$ avant d'être propagée à la première couche cachée. Reformulant : la même entrée, avec le même bruit, est utilisée au début de chaque itération de relaxation (sauf exception selon paramètre *relaxed_inputs*). Les reconstructions successives z_k sont influencées par la deuxième couche à partir de la seconde reconstruction.

ont une forme qui est la transposée de l'influence du bas vers le haut. Les poids sont toujours liés dans la version complètement connectée, car des expériences préliminaires montraient que ça ne changeait pas beaucoup les résultats, et c'est deux fois moins large en mémoire ainsi. Dans la version convolutionnelle, vu les rôles un peu différents joués par les deux directions, les poids ne sont pas liés.

– Pour simplifier le passage de l'étape de pré-entraînement à l'étape de relaxation, l'influence dans la nonlinéarité σ est divisée par deux, plutôt que de multiplier par deux après avoir divisé tous les poids au passage à l'étape d'entraînement en relaxation.

La nonlinéarité σ n'est pas nécessairement la fonction logistique. Une autre nonlinéarité utilisée pour la plupart des expériences de la version complètement connectée est la ***softsign :*** $softsign(a) = \frac{a}{1+|a|}$, dont la sortie va de -1 à 1 plutôt que 0 à 1 dans le cas de la fonction logistique. De plus, elle présente comme avantage de "saturer" (tendre vers les valeurs extrêmes -1 et 1) moins rapidement à mesure que $|a|$ augmente. Ça facilite l'apprentissage car si la saturation est trop forte il n'y a plus de gradient permettant de changer les paramètres.

Une option importante concernant la relaxation concerne ce qui est fait avec les reconstructions. Comme on le voit au bas du diagramme, l'entrée à chaque étape est normalement la même. Cependant, on pourrait vouloir réintroduire la reconstruction de l'étape précédente à la place de l'entrée. C'est ce qu'on nommera "**laisser l'entrée relaxer**", et ça correspond à l'option ***relaxed_inputs***.

On nommera ce type de modèle, entraîné de la façon dont nous le faisons, ***"Relaxing Stacked Denoising Auto Encoder (RSDAE)***".

3.2 Architecture pour les tâches de classification

Pour des expériences de classification, deux options ont été utilisées pour prédire la classe à partir des représentations relaxées. La première et plus courante était d'ajouter une couche de sortie, utilisant une fonction *softmax*, qui prend comme entrée les activations de la dernière itération de relaxation de la dernière couche cachée. Une fonction de coût de type NLL était ensuite utilisée pour guider l'apprentissage.

La seconde option était de faire une prédiction à chaque itération de relaxation (sauf la propagation avant initiale), donc basée sur chaque itération à la dernière couche cachée. Il s'agissait d'une même couche *softmax* (mêmes paramètres), mais appliquée à chaque étape. Le coût alors utilisé était une moyenne de coût NLL sur chacune de ces itérations. Cette option était nommée ***classft_cost_at_all_steps***.

Typiquement on voudra entraîner en propageant le gradient du coût jusqu'aux paramètres des couches inférieures, c'est-à-dire faire un *finetuning* des couches inférieures. C'est que ça donne de biens meilleurs résultats que de n'entraîner qu'un classifieur à régression logistique à la dernière couche, comme le soulignent les expériences de Lamblin et Bengio [25]. L'option ***finetune_lower_layers*** spécifie si ce comportement est actif. L'autre option, de ne pas ajuster les paramètres des couches inférieures, est intéressante en ceci qu'elle permet de déterminer si la relaxation amène des "caractéristiques non supervisées" permettant de séparer linéairement les classes.

En regard de la classification, il faut aussi noter qu'un réseau à "0 étapes de relaxation" est équivalent à un réseau purement *feedforward*. C'est donc ainsi qu'il était

facilement possible de comparer les résultats avec et sans relaxation.

3.3 Auto-encodeur profond et confirmation de l'utilisation des couches profondes

Un dilemme important dans l'analyse des résultats de la relaxation est de déterminer quelle est l'influence réelle des couches supérieures, ou même s'il elles ont une influence quelconque. En effet, dans bien des cas il se pourrait que les couches inférieures fassent tout le travail et que les couches supérieures ne participent pas vraiment à la reconstruction de façon utile.

Une façon assez crue de sonder ce comportement est donc de forcer l'information à passer par ces couches, ce qui se produit lorsqu'on entraîne en autoencodeur profond (voir la section 1.5). C'est ce qui a été fait dans le cas de la version convolutionnelle, où la question était importante durant le déverminage du code, notamment.

3.4 Version convolutionnelle

La **version complètement connectée** de nos modèles à relaxation est telle que décrite ci-haut, c'est-à-dire que toutes les unités d'une couche sont connectées à toutes les unités des couches voisines en haut et en bas, selon ce qui s'applique. Cependant, comme les entrées considérées dans ce travail sont toutes des images, réutiliser des idées de connectivité convolutionnelle était intéressant. C'est cependant une direction assez large, donc il a fallu faire des choix quant à l'architecture précise.

Notons au passage qu'il y a plusieurs motivations derrière l'utilisation de réseaux convolutionnels plutôt que des réseaux complètement connectés. La première serait que les filtres appris peuvent normalement être appliqués à de plus grandes images qu'à l'entraînement. Une autre raison intéressante est qu'intuitivement, au moins pour la première couche, il semble normal qu'émergent des filtres plus locaux si on veut de la robustesse au bruit, et c'est ce qu'on observe (section 1.6). Comme il y aura donc répétition d'un même filtre à différents endroits, autant en tirer profit en réduisant le nombre de paramètres appris (*weight sharing*). Aussi, en soi, une architecture s'inspirant du cortex visuel exerce un certain attrait.

Les architectures convolutionnelles où de la rétroaction est présente font face à un problème inexistant dans le cas des versions complètement connectées, ou des architectures convolutionnelles purement *feedforward,* c'est-à-dire la façon dont la rétroaction est faite d'une couche ayant une résolution plus faible à une couche inférieure à résolution plus élevée. En effet, il faut se rappeler que dans une telle architecture, il y a des couches de convolution, où la dimension de la couche reste sensiblement la même. Par contre, il y a aussi des couches de sous-échantillonnage, bien souvent du *maxpooling,* qui causent une perte d'information. Comment alors inverser le *pooling* lors de la rétroaction (***unpooling***) ?

Une solution a déjà été présentée plus haut, dans la section 2.5.2 : le *probabilistic max pooling* [28]. Ça convient bien dans leur cas puisqu'ils utilisent de l'échantillonnage. Nous avons essayé plusieurs solutions, dont une qui était elle aussi basée sur une fonction d'énergie, d'autres impliquant des connexions latérales, mais au final une solution simple et rapide à exécuter a été retenue : la valeur prise par l'unité de *pooling* de la couche supérieure est simplement répétée plusieurs fois lors de la rétroaction, après la convolution en sens inverse. Ainsi si on a des *pool* de taille 2x2, on fera une convolution en sens inverse (prochain paragraphe), puis le "pixel" (unité) en haut à gauche sera répété dans une zone de 2x2, et ainsi de suite, donnant une image deux fois plus grande. Comme ça revient à "étirer" les représentations de la couche supérieure (sans interpolation), on nommera ça ***stretch unpooling***.

Ainsi, le préentraînement couche par couche va comme suit. Pour la première couche, plusieurs convolutions, chacune avec un filtre différent, sont appliquées à l'entrée et suivis d'une nonlinéarité (appliquée à chaque unité de façon indépendante). On obtient donc une série de *feature maps*, autant qu'il y a de filtres. On a ensuite d'autres filtres, au même nombre mais de paramètres indépendants, qui effectuent une convolution passant de ces *features maps* à une sortie de la taille de l'entrée (convolution en "sens inverse"), et dont l'influence est sommée puis passée dans une nonlinéarité. On a donc la structure d'un autoencodeur (encodage puis décodage), et on entraîne ces deux ensembles de paramètres par le principe de l'autoencodeur débruitant.

La première couche étant préentraînée, pour entraîner la seconde couche (de convo-

lution), on commence par faire suivre les *feature maps* de la première d'un *maxpooling*. Ensuite une autre couche de convolution, puis une convolution en "sens inverse", sont ajoutées. À la sortie de la convolution inverse, on a donc une représentation dont la taille équivaut à la sortie du *maxpooling*. Pour l'agrandir, on "l'étirera" par un facteur inverse au facteur de pooling, toutes les unités dans un même *pool* héritant de la même valeur. Finalement, pour entraîner cette seconde paire de paramètres de convolution, on entraînera en autoencodeur débruitant, bruitant la sortie non *maxpoolée* et tentant de reconstruire en tenant compte de cet étirement. Le chemin suivi par l'information durant l'entraînement de cette seconde couche est illustré à la figure 3.2.

Il faut ici préciser pourquoi le coût est appliqué après l'étirement. Le but est de tenir compte de l'*unpooling* dans l'entraînement, en rapprochant la reconstruction de l'exemple *non* sous-échantillonné autant que possible. C'est aussi qu'auparavant nous avions expérimenté avec une autre forme d'*unpooling* plus complexe, sur laquelle nous n'élaborerons pas.

On peut ainsi entraîner plusieurs couches de convolution, et en pratique nous en utiliserons 3. On peut aussi construire des graphes de relaxation où on aura de la rétroaction par cette opération d'étirement. Cependant, dans cette version, contrairement à ce qui est fait dans l'équation 3.1, plutôt que de combiner l'information du haut et du bas par une division par 2 *avant* la nonlinéarité, on applique la nonlinéarité avant de combiner les deux influences.

Le but de cette variation était de permettre différentes intégrations d'information dans un voisinnage proche d'un neurone donné, en utilisant potentiellement un autre neurone "intégrateur" intermédiaire. Nous n'élaborerons pas sur cette piste d'expériences non poursuivie. Notons malgré tout que de cette façon, l'opération de combinaison des influences ressemble plus à un "OU" logique qu'en sommant avant la nonlinéarité, car les valeurs en sortie des deux sigmoïdes étant nécessairement positives, si l'une ou l'autre des unités est non nulle, la valeur combinée le sera également.

On obtient donc une moyenne pondérée des deux activations, tout simplement :

$$h^k_{l,p,i,j} = \alpha \times \sigma(bas) + (1-\alpha) \times \sigma(haut) \tag{3.2}$$

où α est le coefficient spécifiant le poids relatif des influences et *haut*/*bas* sont les "influences" des deux directions, qui résultent de la convolution pour la fenêtre correspondant à $h^k_{l,p,i,j}$, l'état activation de l'unité cachée (avec *unpooling* pour le cas *haut*, mais ça reste le résultat d'une convolution). Les indices *l, p, i, j* de l'unité cachée désignent respectivement la couche, la *feature map*, et la position en x et y dans cette feature map. L'exposant *k* désigne l'itération de relaxation.

On peut alors choisir de donner plus ou moins de poids à l'une ou l'autre des directions, haut ou bas, selon la couche et selon l'itération de relaxation. On obtient donc un tableau de **paramètres d'intégration** ayant ces dimensions (nombre de couches $\times$ nombre d'itérations de relaxation). On peut leur donner une valeur fixe de 0.5, ou encore les optimiser comme n'importe quel autre paramètre durant l'apprentissage par descente de gradient. L'intention est de permettre aux "abstractions" associées aux couches supérieures d'exercer une plus grande influence, et ce selon une sorte de "procédure au cours de la relaxation" aux paramètres exacts appris.

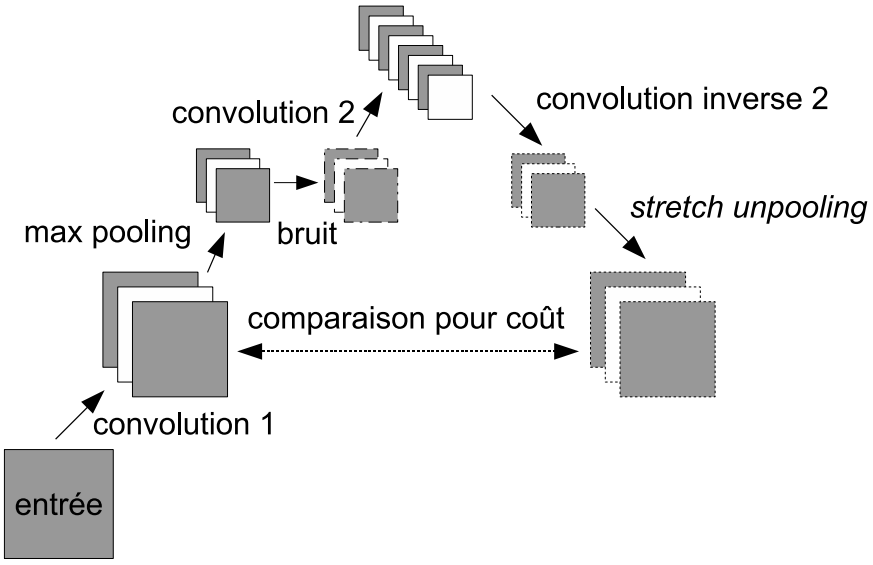


Figure 3.2 – **Version convolutionnelle : illustration de préentraînement de la seconde couche** de convolution avec *stretch unpooling*, dans l'architecture convolutionnelle utilisée.

CHAPITRE 4

EXPÉRIENCES ET RÉSULTATS

4.1 Jeux de données

4.1.1 MNIST et MNIST 7x7

MNIST est un jeu de données très commun dans la communauté entourant les réseaux de neurones profonds. Il s'agit de 70 000 images en tons de gris (principalement les extrêmes, noir et blanc) de chiffres manuscrits, donc avec des classes de 0 à 9. Elles ont une dimension de 28x28 pixels. 10 000 d'entre elles sont réservées pour un jeu de données de test, qui est standard à tous ceux utilisant MNIST. 10 000 autres sont choisies ici pour servir de jeu de donnée de validation. La tâche typique sur MNIST est la classification, où la meilleure performance actuellement publiée est d'une erreur de 0.39% sur le jeu de données de test [32].

MNIST, et différentes variations selon le bruit ajouté, a été utilisé ici comme jeu de données principal. Nous avons aussi utilisé une variation qu'on nommera **MNIST 7x7** où on réduit simplement la taille des images par un facteur de 4 dans chaque direction (donnant des images de 7x7 pixels), avec une interpolation bicubique. Les jeux de données d'entraînement, de validation et de test restent autrement les mêmes.

4.1.2 Caltech Silhouettes

Il s'agit ici d'un jeu de données d'images 28x28 binaires, donc en noir et blanc, aussi. Cependant les classes sont plus nombreuses, au nombre de 101. Il y a au maximum une centaine d'exemples par classe, et les classes sont plus variées : on y trouve des objets courants (voitures, avions, ...) et des animaux. Le jeu de données a été principalement utilisé pour pouvoir se comparer en performance de reconstruction à la publication qui l'introduisait [30].

Figure 4.1 – Exemples tirés de MNIST (gauche) et de Caltech Silhouettes (droite, noir et blanc inversés, tel qu'utilisé ici)

4.2 Déroulement des expériences

4.2.1 Hyperparamètres et choix communs à toutes les expériences

Quelques choix étaient les mêmes dans toutes les expériences, et pour éviter de répéter, les voici :

- La taille de *minibatch* était de 10 ;
- Dans tous les cas où l'entraînement était par un critère d'autoencodeur débruitant, au préentraînement couche par couche ou au *finetuning* en relaxation :
  - le coût (erreur de reconstruction) utilisé était une NLL (équation 1.1) [1] ;
  - le critère d'arrêt était un arrêt prématuré basé sur l'erreur de reconstruction, avec des limites de nombre d'époques minimales et maximales variant selon l'expérience.
- Dans tous les réseaux complètement connectés :
  - les unités cachées avec une nonlinéarité *softsign*, et seule la reconstruction de l'entrée, à la première couche, utilisait une sortie par fonction logistique ;
  - le préentraînement de trois couches était fait, mais à l'étape de relaxation, certaines tâches abandonnaient la dernière couche et devenaient des modèles à deux couches cachées plutôt que trois ;
  - les poids de chaque couche, pour l'encodage et le décodage, étaient liés.
- Dans tous les cas de classification, la couche de sortie était une *softmax* (équation 1.2), entraînée avec un coût NLL (équation 1.1). Par contre, le fait qu'elle soit faite à toutes les itérations ou non (*classft_cost_at_all_steps*), ou qu'il y ait *finetuning* des couches inférieures ou non (*finetune_lower_layers*) pouvaient varier.

1. À noter que nous utilisions la NLL pour l'entraînement, et la MSE pour l'évaluation. Ça peut paraître incongru. L'explication est que les premières expériences où nous nous intéressions quantitativement à l'erreur de reconstruction étaient pour fins de comparaison avec Marlin et al. [30], qui utilisent la MSE. Nous avons donc conservé cette métrique pour les autres expériences.

- Dans tous les cas de *finetuning* supervisé, donc de classification, le critère d'arrêt était un arrêt prématuré basé sur l'erreur de classification sur l'ensemble de validation, avec un nombre minimum et maximum d'époques variant selon l'expérience.
- Pour les cas avec architecture convolutionnelle, l'architecture (taille des filtres, facteur de sous-échantillonage) restait constante. Se référer à la section 4.11 qui les concerne pour les détails.
- Il n'y a jamais de tâche à une seule étape de relaxation, car ça revient à un autoencodeur d'une seule couche. De manière générale, il n'y a pas de tâche où le nombre d'étapes de relaxation est inférieur au nombre de couches, car dans ce cas la dernière couche ne serait pas utilisée.

4.2.2 Résumé d'hyperparamètres et paramètres architecturaux variables

Jusqu'à présent plusieurs paramètres de configuration architecturaux et hyperparamètres variant par expérience ont été mentionnés. Il est bon de les résumer pour en avoir une vue d'ensemble. Se référer aux sections précédentes pour explications de chacun d'entre eux.

Tout d'abord, résumons les hyperparamètres qui variaient d'une expérience à l'autre (et non mentionnés à la section précédente), au sens propre :

- nombre d'unités des couches cachées, ou nombre de filtres dans le cas convolutionnel ;
- la vitesse d'apprentissage (*learning rate*) pour les différentes étapes d'apprentissage ;
- la condition d'arrêt : arrêt prématuré (et sur quel coût) ou nombre d'époques fixe ;
  - le nombre d'époques minimal et maximal si un arrêt prématuré était utilité ;
- le type et le niveau de bruit poivre, poivre et sel, ou des occlusions, utilisés durant l'entraînement (à distinguer du bruit utilisé lors des tests) ;
  - la densité d'occlusions si elles étaient utilisées ;
- le nombre d'itérations de relaxation (*num_iter*).

Maintenant résumons les paramètres de niveau architectural, souvent des paramètres dont l'influence était l'objet direct d'étude pour l'une ou l'autre des expériences :

- y avait-il relaxation des entrées ou non ? (*relaxed_inputs*) ;
- En cas de classification :
  - coût de classification était-il sur la dernière étape de relaxation, ou toutes ? (*classft_cost_at_all_steps*) ;
  - y avait-il *finetuning*, ou non, des couches inférieures durant l'apprentissage en classification ? (*finetune_lower_layers*) ;
  - y avait-il, ou non, relaxation des représentations ? (l'absence est indiquée par un nombre d'itérations de 0) ;
- En cas d'architecture convolutionnelle :
  - utilisait-on, ou non, d'une étape de *finetuning* en autoencodeur profond ? (DCAE) ;
  - apprenait-on les paramètres d'intégration, ou étaient-ils fixes ?

4.2.3 Exploration des hyperparamètres

Contrairement aux paramètres des réseaux, on ne possède généralement pas de gradient nous permettant de nous orienter dans l'espace des hyperparamètres. Ainsi il faut essayer plusieurs valeurs d'hyperparamètres un peu à tâtons afin d'optimiser pour obtenir les meilleurs performances en généralisation, comme expliqué dans la première section.

Il y a plusieurs méthodes pour le faire. Une méthode souvent utilisée est **l'exploration d'une grille** de valeurs, c'est-à-dire de considérer pour chaque hyperparamètre un certain nombre fixe de valeurs possibles et d'essayer *toutes* les combinaisons des valeurs possibles. En termes mathématiques, ça revient à explorer l'entièreté du produit cartésien des ensembles de valeurs pour chaque hyperparamètre. Si par exemple on a trois hyperparamètres avec 5 valeurs chacunes, on a donc $5 \times 5 \times 5$ combinaisons à explorer, chacune étant en fait une tâche d'apprentissage à part entière pouvant demander des heures, voir des jours, sur un noeud de calcul d'une grappe (*cluster*) informatique.

Comme dans ce travail on aura souvent une dizaine d'hyperparamètres avec de 2 à 5 valeurs ou plus pour chacun, le nombre de possibilités à explorer est immense, ce qui rend l'approche en grille impraticable. Une façon de diminuer le nombre de possibilités est d'en sélectionner un sous-ensemble aléatoirement. Une façon encore plus radicale est de donner une plage de valeurs (plutôt que des valeurs fixes) pour chaque hyperparamètre

et échantillonner dans ces intervalles un petit nombre prédéterminé de combinaisons, par exemple 50.

Dans cette **approche par échantillonnage**, on n'aura donc très probablement jamais exactement la même valeur pour un hyperparamètre, ce qui permet de mieux couvrir la plage de valeurs pour chaque hyperparamètre. Ainsi, si un de ceux-ci est très sensible, c'est-à-dire que seule une petite partie de sa plage de valeurs mène à de bons résultats, mais les autres hyperparamètres ne le sont pas vraiment, on a de meilleures chances d'avoir des tâches qui auront une bonne valeur pour le premier [9].

En pratique, ici, c'est l'approche par échantillonnage qui est adoptée. Parfois, par contre, il y aura des hyperparamètres dont les valeurs sont échantillonnées dans un ensemble fixé d'avance, malgré tout. C'est bien sûr le cas pour les paramètres à valeurs discrètes et petites, mais aussi pour les hyperparamètres dont on veut analyser l'influence. Il est en effet plus facile d'avoir des valeurs fixes pour ceux-ci, car ça facilite la comparaison. Par exemple, si on cherche à comparer la performance de reconstruction avec des niveaux de bruits différents, il est nécessaire d'avoir des niveaux de bruit comparables. Si chaque tâche a un niveau de bruit différent, alors les conclusions sont moins claires.

4.2.4 Division des tâches en étapes

Comme mentionné dans diverses sections précédentes, l'entraînement de réseaux profonds se fait souvent en étapes. Il y aura typiquement une étape de préentraînement et une étape de *finetuning* supervisé. Comme les deux étapes sont longues et leurs hyperparamètres sont assez indépendants, on peut sauvegarder les résultats après la première étape et ainsi partager entre plusieurs tâches de la seconde étape les résultats d'une seule des tâches de la première. C'est l'approche adoptée ici. Nous aurons typiquement une cinquantaine de tâches de préentraînement et environ 150 tâches de *finetuning*, pour les deux types de *finetuning* expliqués dans les sections sur la relaxation. Chaque tâche de préentraînement est donc réutilisée trois fois. À noter que dans le cas de la version complètement connecté, on préentraînait toujours 3 couches, mais on pouvait n'en utiliser que 2, au choix, dans les tâches qui suivaient.

4.2.5 Arrêt prématuré avec l'erreur de reconstruction

Un problème à éviter lors de l'entraînement avec un ensemble de données fini est le surentraînement, c'est-à-dire de s'adapter trop fortement aux données de cet ensemble en généralisant mal à d'autres données semblables hors de l'ensemble. Avec une capacité suffisamment grande, ça se produira assez inévitablement si on entraîne trop longtemps. Une façon d'éviter ça est d'arrêter l'entraînement quand on voit que la performance de généralisation devient mauvaise. On calculera donc périodiquement l'erreur de classification sur l'ensemble de validation et ne conservera que les paramètres ayant donné les meilleures performances sur cet ensemble au cours de l'entraînement. Quand on voit que l'erreur remonte depuis un certain temps, on peut cesser l'entraînement.

Ici une approche semblable est aussi utilisée basée sur l'erreur de reconstruction sur l'ensemble d'entraînement, pour éviter de faire durer celui-ci trop longtemps quand l'erreur de reconstruction stagne depuis plusieurs époques, et pour éviter de surapprendre sur l'ensemble d'entraînement (apprendre les exemples d'entrée non bruités "par coeur").

4.3 Outils utilisés pour l'expérimentation

Il convient de faire un tour rapide de l'implémentation plus technique.

Le langage utilisé, à la base, était Python. C'est un langage idéal pour expérimenter et prototyper, entre autres, étant donné que des structures de données communément utilisées sont intégrées à même la syntaxe et que les variables ne sont pas "fortement typées". Autrement dit, on peut rapidement y implémenter des algorithmes faisant usages de constructions communes de programmation sans être trop "verbeux", contrairement à d'autres langages plus orientés sur la vérification statique, comme Java.

Côté apprentissage machine comme tel, l'outil principal utilisé était Theano [10], une librairie de code Python permettant de définir un graphe de calcul d'opérations matricielles. Un avantage important est la capacité de calculer symboliquement un autre graphe du même genre correspondant au calcul du gradient de certains résultats par rapport à certaines variables. Le compilateur de graphe effectue aussi une optimisation selon des motifs communs dans les graphes de calcul utilisés pour les réseaux de neurones. Le

graphe devient donc du code C++ (ou CUDA) compilé.

Ainsi on peut facilement faire des changements dans une fonction de coût, recalculer automatiquement le gradient du coût par rapport aux paramètres du modèle, et lancer un apprentissage efficace sans avoir à manuellement faire l'optimisation du calcul. Le calcul représenté par le graphe compilé peut dans la plupart des cas être fait sur *GPU* (*graphic processing unit* d'une carte graphique), ce qui peut l'accélérer par un facteur de 10 à 100. Theano est une librairie développée au laboratoire d'apprentissage machine de l'Université de Montréal, le LISA.

Côté matériel, les tâches étaient lancées sur des grappes de calcul à divers endroits. La grappe du LISA, Condor, a été utilisée pour certaines expériences à réseau complètement connecté. Il s'agit d'une grappe avec environ 350 noeuds de type séquentiel (CPU). Pour d'autres expériences à réseaux complètement connectés, et pour presque toutes les expériences à réseau convolutionnel, la grappe de *GPUs* du réseau SHARCNet, Angel, a été utilisée. Il s'agit de 44 cartes graphiques NVIDIA permettant d'utiliser du code CUDA tel que produit, notamment, par la librairie Theano.

Pour lancer les expériences, les isoler les unes des autres, choisir les hyperparamètres et récupérer les résultats, nous avons développé une librairie de code qui repose uniquement sur l'utilisation de fichiers contenant les paramètres de chacune des expériences, ainsi que les résultats. Elle utilise certaines fonctionnalités de la librairie Jobman, développée au LISA, qui effectue une tâche semblable mais fondée sur l'utilisation de bases de données.

4.4 Types de bruit et de déformation des images

Il y a 3 types de bruit utilisés dans les expériences (figure 4.2). Précisons d'abord que les images utilisées sont quasi "binaires". Autrement dit, les pixels sont en grande majorité soit noirs, soit blancs. Il faut aussi spécifier que le "noir" des images est le fond et correspond à la valeur 0. Les objets (caractères dans le cas de MNIST) sont "blancs", et ces pixels blancs correspondent à la valeur 1. Les valeurs intermédiaires entre 0 et 1 sont des tons de gris.

Le bruit “**poivre**” correspond à mettre certains pixels à une valeur de 0. Chaque pixel a donc une probabilité ***corruption_level*** d’être mis à 0.

Le bruit **“poivre et sel**” correspond à, pour chaque pixel, le sélectionner avec une probabilité *corruption_level.* Chaque pixel sélectionné se voit ensuite assigné la valeur 0 ou 1 selon une probabilité de 50%.

La déformation de type “**occlusions**” est un peu plus compliquée. Le but est de superposer des parties de caractères de MNIST sur l’image originale, venant faire des “taches” blanches (voir figure 4.2). Plus en détail, on commence par bâtir une grande image tampon de 2000x2000 pixels et y mettre des sous-parties de caractères de MNIST dont la taille est entre 5 et 15 pixels de large et haut. Chaque sous-partie de caractère est ajoutée, par une opération “max” entre cette partie du tampon et le bout de caractère, à une partie aléatoire de la grande image. Selon le paramètre ***occlusion_density*** on remplira ainsi plus ou moins la grande image tampon.

Une fois ce grand tampon créé et prêt pour utilisation, on déformera les caractères un à un en faisant un (autre) *max* entre une zone aléatoire de la grande image et un exemple du jeu de données d’entraînement. Ainsi un exemple se retrouve occludé par des sous-parties de caractères du jeu de données MNIST. Cette façon de générer des occlusions s’inspire d’un filtre de traitement d’image présenté dans Bengio et al. [7].

Finalement, notons à nouveau que pour aucune expérience nous n’avons entraîné avec un bruit de 0%, qui correspond à un autoencodeur ordinaire. Cela aurait été possible, et les résultats auraient pu être instructifs, mais nous avons concentré nos efforts sur un entraînement avec un critère de débruitement.

4.5 Analyse et présentation

4.5.1 Incertitude sur un minimum

Une situation courante est d’avoir une dizaine (ou plus) de tâches avec chacune une performance numérique, et de vouloir obtenir une valeur pour le minimum qui reflète l’incertitude associée à ces valeurs. On veut aussi éviter de désavantager indûment des ensembles de résultats où par malchance l’échantillonnage d’hyperparamètres n’aurait

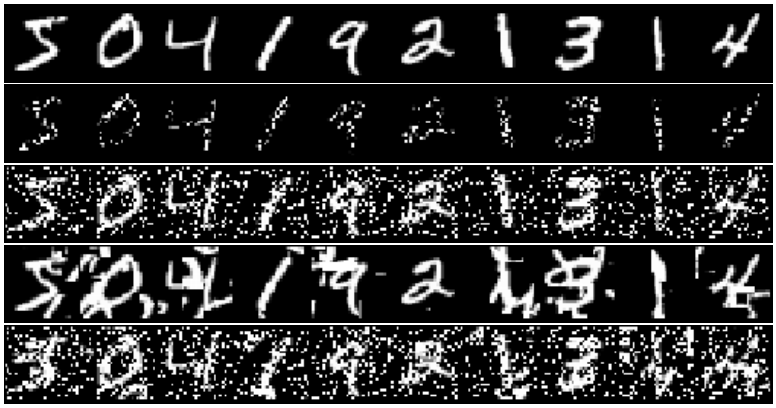


Figure 4.2 – **Exemples de différents types de bruit**. De haut en bas : images originales, bruit poivre 60%, bruit poivre & sel 30%, occlusions avec une *occlusion_density* de 0.005. La dernière série présente une combinaison des deux derniers types de bruit.

sélectionné que de médiocres combinaisons, par exemple en comparant un ensemble des performances avec relaxation, et sans relaxation.

Une méthode pour le faire est d'échantillonner à répétition un sous-ensemble de valeurs de performance et de conserver le minimum de chacun de ces sous-ensembles. En prenant la moyenne ("minimum moyen") et la déviation standard (σ) de ces valeurs, on obtient une valeur et une forme de mesure d'incertitude. On utilisera ici des sous-ensembles avec une cardinalité du 2/3 de celle de l'ensemble de valeurs initiales, et 50 répétitions. Cette méthode est un exemple de *bootstrapping.*

4.5.2 Autres aspects d'analyse

Une observation à garder en tête est que quand l'entrée est fixe (*relaxed_inputs* prend la valeur Faux), toute amélioration à la reconstruction doit nécessairement provenir de l'influence des couches supérieures, et de l'évolution de l'état à ce niveau.

4.5.3 Présentation des hyperparamètres

Les hyperparamètres sont présentés sous forme textuelle, plutôt que de tableaux. C'est que certains détails s'expliquent mieux ainsi, d'autant plus que d'une expérience à l'autre certains sont récupérés et d'autres non.

Les vitesses d'apprentissage (*learning rate*) ne sont pas donnés car leur sens dépend de la méthode exacte de calcul du coût, qui a parfois changé (ce qui n'altère pas les conclusions).

4.6 Expériences avec bruit "poivre" sur MNIST

La première expérience faite pour ce travail implique des réseaux complètement connectés entraînés à débruiter du bruit de type "poivre" sur MNIST. Le but était de déterminer si la relaxation aide sur des tâches de débruitement, ce qui est tout naturel vu la façon d'entraîner. Nous avons aussi essayé le modèle sur diverses tâches de classification : avec ou sans bruit sur l'entrée, avec ou sans *finetuning* supervisé des couches inférieures lors de la phase d'entraînement supervisé.

Dans tous les cas, il y avait 50 tâches de préentraînement, 150 tâches de *finetuning* non supervisé avec relaxation, et 150 tâches de classification lorsque cette étape était faite. Trois tâches de *finetuning* partageaient la même tâche de préentraînement, donc. Le *corruption_level* (niveau de bruit poivre, ici) variait toujours entre 20% et 80%. Il y avait 2 ou 3 couches de 1000 unités cachées chacune, excluant la couche de régression logistique utilisée au haut du réseau lors de la classification. Lors de la relaxation, *num_iter*, le nombre d'itérations de relaxation, pouvait être 0, 2, 4, 6 ou 8. Au préentraînement, chaque couche pouvait être entraînée entre 5 et 50 époques. À la relaxation, c'était de 5 à 50 époques pour l'ensemble du modèle. Les hyperparamètres étaient sélectionnés par échantillonnage.

Les tâches étaient exécutées sur la grappe de calcul du LISA, Condor. Chaque époque de préentraînement, pour chaque couche, prenait de 5 à 10 minutes, selon la machine. Au *finetuning* non supervisé, chaque époque pouvait prendre jusqu'à 70 minutes, dans le pire des cas (8 étapes de relaxation, 3 couches). Les tâches les plus rapides pour cette étape (2 itérations) prenaient environ 10 minutes par époque.

À noter que pour cette série d'expériences le niveau de bruit à chaque étape, incluant le bruit éventuel à la classification, était sélectionné indépendamment. Ainsi une tâche de classification voyant un bruit de 30% pourrait hériter de paramètres préentraînés par

une tâche où le niveau de bruit est 60%.

4.6.1 Reconstruction avec bruit poivre

Il est bien sûr très intéressant de visualiser les reconstructions à chaque étape de relaxation. La première étape n'étant influencée que par la représentation à la première couche, on s'attend à ce qu'elle donne de moins bons résultats que les couches suivantes, ou du moins on l'espère. C'est effectivement ce qu'on observe, tel que montré à la figure 4.3. À noter que toute amélioration dans les itérations suivant la première vient forcément de l'influence des couches cachées 2, à partir de la seconde reconstruction, et 3, à partir de la troisième (s'il y a une troisième couche). En effet, comme la même entrée est répétée à chaque étape, sans l'influence des représentations aux niveaux supérieurs, on obtiendrait toujours le même résultat qu'à la première reconstruction.

Notons que **l'apparence s'améliore fortement dans les premières étapes de relaxation, et de moins en moins à mesure que les itérations progressent**. C'est une tendance qui a été observée dans beaucoup d'expériences subséquentes.

4.6.2 Reconstructions à partir d'entrée avec occlusions

Le bruit poivre est le bruit présent à l'entraînement. C'est cependant intéressant de voir si le modèle peut compenser pour d'autres types de bruits, même sans avoir été entraîné sur ce type de bruit. Deux sont montrés aux figures 4.4 et 4.5, des occlusions noires et des occlusions blanches. On voit que ça fonctionne plutôt bien pour les occlusions noires, et on observe des complétions de parties de caractères qui reposent entièrement sur l'identification implicite de la classe du caractère, et que parfois une mauvaise classe est identifiée (voir le "9" qui devient un "7"). Il faut garder à l'esprit que le réseau voit toujours, en entrée, la même version bruitée (*relaxed_inputs* est à Faux) donc la complétion ne peut venir que de représentations dans les couches cachées supérieures.

Le fait que ça fonctionne bien avec les occlusions noires, mais pas les blanches, semble indiquer que la fonction de reconstruction apprise (représentée par le réseau) est habituée à ce qu'on pourrait appeler un bruit "soustractif". Selon cette hypothèse, on

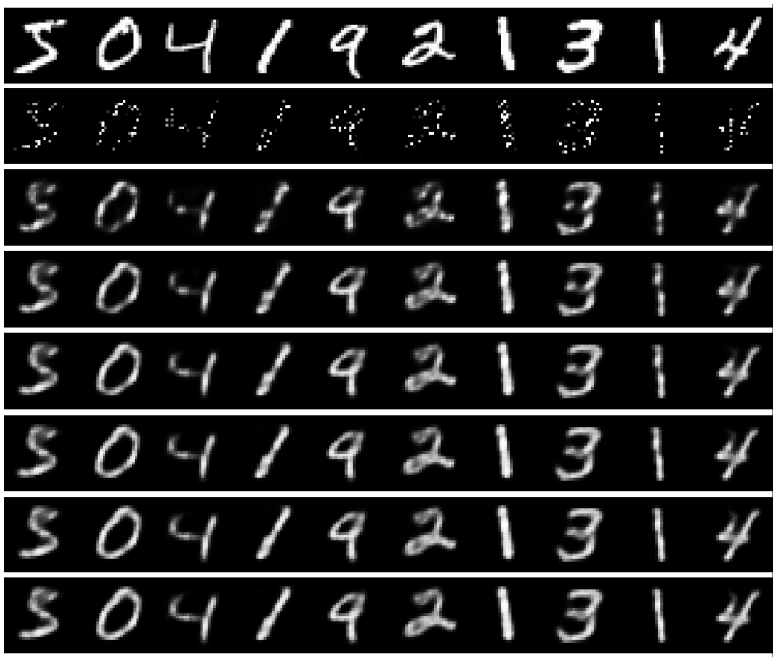


Figure 4.3 – **Expériences avec bruit poivre : exemple de reconstructions à différentes étapes de la relaxation**. La première ligne, celle du haut, montre les images originales. La seconde, la version bruitée qui est présentée au réseau à chaque étape. Ici il s'agit de 10 exemples tirés du jeu de données d'entraînement, bruités à 80%, et d'un modèle à 6 étapes de relaxation. On voit bien la complétion progressive, plus marquée aux premières itérations, mais tout de même visible dans les dernières. Bien que la première reconstruction, influencée uniquement par la première couche, comble déjà une bonne partie des pixels manquants, les itérations suivantes améliorent beaucoup la qualité, à l'oeil.

apprendrait donc à modéliser non seulement les caractères, mais aussi le type de bruit.

4.6.3 Classification avec finetuning, sans bruit sur l'entrée

Il s'agit ici de la première expérience en classification. Partant du réseau préentraîné, puis *finetuné* de façon non supervisée avec relaxation, on entraîne une couche de régression logistique avec *softmax* pour les 10 classes de MNIST, en *finetunant* les couches inférieures apprises durant les phases non supervisées (*finetune_lower_layers* est à Vrai). À noter que pour les réseaux à "0 étape de relaxation", l'étape de *finetuning* non supervisé avait simplement été sautée.

La régression logistique prend en entrée la représentation à la dernière couche, à la dernière itération de relaxation (*classft_cost_at_all_steps* est à Faux). On compare la sortie à la bonne classe avec un coût NLL. Ce coût est dérivé par rapport aux paramètres de toutes les couches pour effectuer les pas d'apprentissage. Le nombre d'époques était fixé à 5, le but n'étant pas d'obtenir les meilleures performances, mais simplement de voir l'impact de la relaxation. Il n'y avait pas de bruit sur l'entrée durant la classification. Chaque tâche prenait entre 5 et 100 minutes par époque pour s'exécuter, selon le nombre d'itérations (de 0 à 8) et le noeud de calcul.

Comme le montre le tableau 4.I, la performance en classification ne semble pas être améliorée par la relaxation avec cette architecture, bien que cela ne nuise pas énormément. Par contre, cela mène à une plus grande incertitude sur la performance et plus de tâches ne convergent pas.

Une hypothèse pour expliquer l'absence d'amélioration est qu'il est possible que l'information de classe doive, dès la première propagation avant, être déjà en grande partie identifiée. C'est d'ailleurs ce qui semble permettre de compléter des caractères en reconstruction avec occlusion. Ainsi, l'information de classe étant déjà extraite à la propagation initiale, bien que peut-être un peu bruitée, les itérations de relaxation n'aideraient pas vraiment à améliorer la classification.

Une autre hypothèse serait que comme le coût de classification est appliqué à la dernière itération de relaxation, le gradient peut difficilement bien tenir compte de l'impact des premières itérations, et les gains faits lors du *finetuning* en relaxation (l'é-

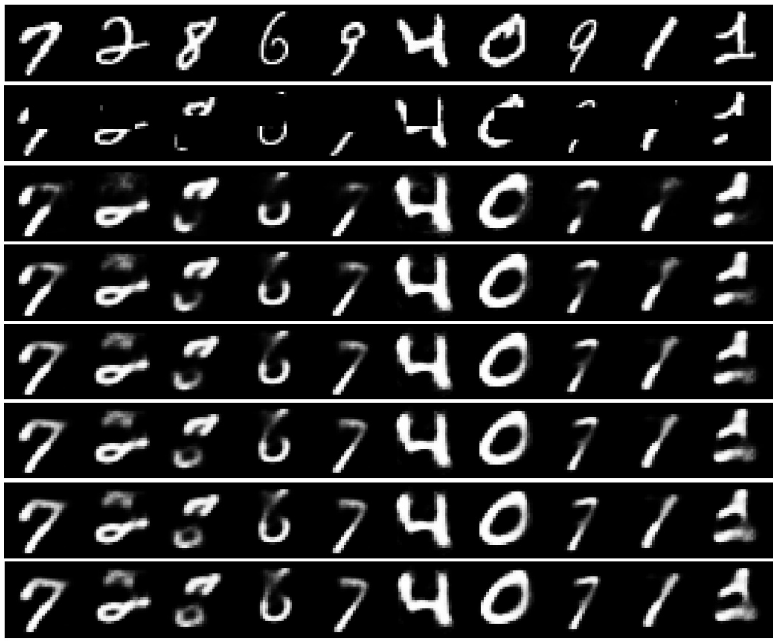


Figure 4.4 – **Expériences avec bruit poivre : reconstructions avec occlusions noires sur l'entrée**. Les occlusions sont des carrés de dimension 12x12 pixels. On voit bien des complétions basées sur une reconnaissance de la classe, voir du style d'écriture. À noter, par contre, que c'est ici sur des images de l'ensemble d'entraînement, pas de test, donc ces caractères ont été vus déjà par le modèle à l'entraînement.

Figure 4.5 – **Expériences avec bruit poivre : reconstructions avec occlusions blanches sur l'entrée**. Toutes les reconstructions ont cette apparence, dès la première itération. On voit donc que le modèle n'est pas du tout capable de gérer ce type d'occlusion, probablement car ce qu'il a vu à l'entraînement n'est qu'un bruit "soustractif".

tape intermédiaire non supervisée) seraient un peu effacés lors du *finetuning* supervisé. C'est cette hypothèse qui a motivé le fait de mettre le coût à toutes les étapes (*classft_cost_at_all_steps*) dans des expériences sur d'autres types de bruit.

4.6.4 Classification avec finetuning, avec bruit sur l'entrée

Cette expérience est très semblable en architecture à la précédente, sauf en ceci qu'on laisse du bruit sur l'entrée. C'est du bruit poivre, tout comme pour les autres étapes. Cependant, le nombre d'époques variait entre 25 et 50, car nous avions constaté un phénomène où l'erreur remontait puis redescendait, ce qui est peut-être dû au niveau de bruit qui changeait. Les temps d'exécution étaient semblables à ceux de l'expérience précédente.

Le niveau de bruit en entrée était un paramètre échantillonné entre 20% et 80%, ce qui n'était peut-être pas le choix le plus judicieux pour l'analyse. Pour comparer quantitativement, on prendra donc les tâches dans des intervalles allant de 20% à 40% de bruit, de 30% à 50%, etc. jusqu'à 80%. On comparera les minimum atteints dans chacun des ces intervalles sans relaxation, et avec relaxation (tout nombre d'itérations confondus).

Les résultats sont regroupés dans le tableau 4.II et les performances brutes de chaque tâche sont illustrées à la figure 4.6. Comme on peut constater, les tâches avec relaxation semblent toujours inférieures (ou égales) en performances aux tâches sans relaxation.

Nombre d'itérations	Tâches	Minimum moyen	σ	Tâches n'ayant pas convergé
0	33	1.93%	0.09%	0
2	30	2.03%	0.08%	2
4	32	2.13%	0.04%	0
6	22	2.10%	0.18%	6
8	33	2.19%	0.14%	6

Tableau 4.I – **Expériences avec bruit poivre : meilleurs taux d'erreur de classification avec *finetuning*, sans bruit sur l'entrée**. Pour le calcul d'incertitude sur le minimum, voir la section 4.5.1. Les tâches n'ayant pas convergé avaient des erreurs au-delà de 80%.

Ainsi, avec ou sans bruit sur l'entrée, dans la mesure où les couches inférieures sont *finetunées*, les performances semblent être meilleures sans relaxation.

Comme on peut le constater dans le graphique, il y a plus d'incertitude associée aux tâches avec relaxation. Dans des expériences réalisées beaucoup plus tard, nous avons réalisé que ce n'était pas inhérent à la relaxation, mais plutôt au choix d'hyperparamètres qui est plus sensible. Avec un meilleur choix, les points se rapprochaient du vrai minimum atteint. En choisissant mieux les hyperparamètres ici, on peut donc penser qu'on aurait pu réduire l'incertitude, et par la même occasion le minimum moyen. C'est donc là quelque chose dont il faut tenir compte quand on regarde ces chiffres et graphiques.

4.6.5 Classification sans finetuning (caractéristiques non supervisées), sans bruit sur l'entrée

Dans la section sur les architectures profondes, le concept de découverte automatique de facteurs de variation a été introduit. Il est intéressant de savoir si des caractéristiques apprises de façon entièrement non supervisée permettent de bien classifier les données, sans *finetuner* les couches préentraînées. On a alors une idée d'à quel point l'information de classe est bien séparée automatiquement même si on ne l'a jamais introduite explicitement au cours des étapes préliminaires.

Ici on préentraîne donc couche par couche et effectue un *finetuning* non supervisé avec relaxation. Par contre, à l'étape de classification, on ne modifie que les paramètres de la couche de régression logistique (*finetune_lower_layers* est à False). Ainsi, si on voulait parfaitement classifier, la représentation à la dernière couche apprise de manière non supervisée devrait être linéairement séparable.

Dans l'expérience présente, il n'y a pas de bruit sur l'entrée pour la classification, à la différence de l'expérience qui suivra. On entraînait pour un nombre fixe d'époques (5), simplement pour avoir une idée de ce qui fonctionne le mieux : avec ou sans relaxation. Comme il n'y avait pas de *finetuning,* les tâches s'exécutaient beaucoup plus rapidement, prenant au maximum environ 10 minutes par époque.

Comme le montre le tableau 4.III de meilleures performances, dans ce cas-ci non plus la relaxation ne semble pas aider, et le risque qu'une tâche ne converge pas est un

	Sans relaxation		Avec relaxation	
Niveau de bruit	Minimum moyen	Incertitude (σ)	Minimum moyen	Incertitude (σ)
20% à 40%	1,54%	0,05%	1,84%	0,1%
30% à 50%	1,90%	0,05%	2,12%	0,03%
40% à 60%	2,27%	0,02%	2,44%	0,16%
50% à 70%	2,64%	0,16%	3,04%	0,41%
60% à 80%	3,62%	0,42%	4,88%	0,06%

Tableau 4.II – **Expériences avec bruit poivre : taux d'erreur en classification avec *finetuning* des couches inférieures, et en présence de bruit sur l'entrée**. Pour le calcul d'incertitude sur le minimum, voir la section 4.5.1. Comme le bruit avait un niveau échantillonné de manière uniforme entre 20% et 80%, les minima ont été calculés pour des intervalles de valeurs du niveau de bruit.

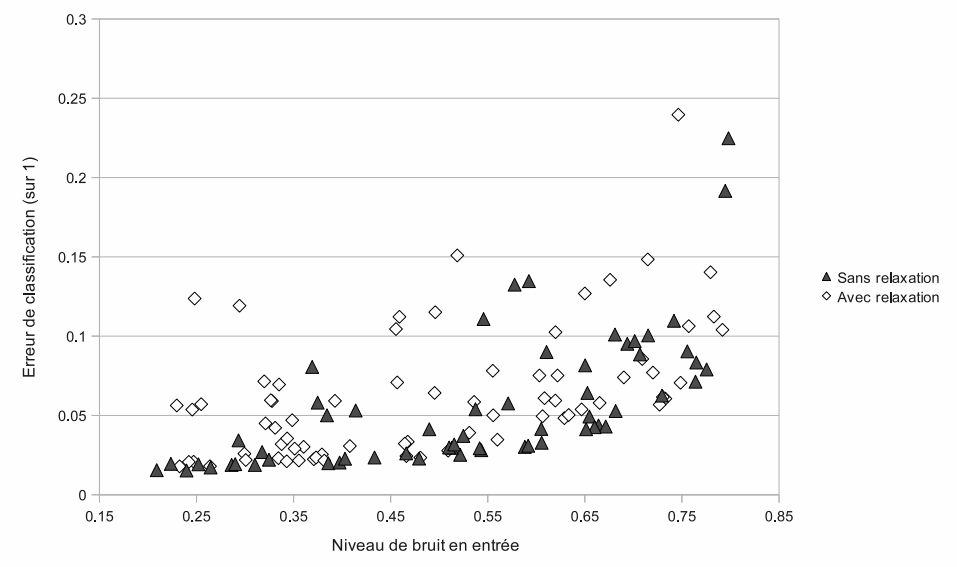


Figure 4.6 – **Expériences avec bruit poivre : performances de classification avec *finetuning*, avec bruit sur l'entrée**. Erreur de classification sur l'ensemble de validation de MNIST en fonction du bruit sur l'entrée, et selon que la tâche utilisait la relaxation ou non. Cette figure ne montre pas des tâches avec relaxation n'ayant pas convergé, une dizaine au total.

peu plus élevé avec relaxation.

4.6.6 Classification sans finetuning (caractéristiques non supervisées), avec bruit sur l'entrée

Pour terminer cette série d'expériences, le cas restant était d'essayer d'extraire des caractéristiques de façon non supervisée afin de les utiliser pour classifier une entrée bruitée. Il s'agit donc du même principe que dans l'expérience précédente (pas de *finetuning* des couches inférieures), à cette différence près que l'entrée à classifier est bruitée.

Comme le montrent la figure 4.7 et le tableau 4.IV, cette fois-ci, contrairement au reste des expériences en classification, et donc de façon un peu surprenante, on observe un gain appréciable du fait de laisser la représentation cachée relaxer. Une explication possible serait que l'information de classe est beaucoup moins mise en évidence et raffinée à la dernière couche, de par l'absence de *finetuning* supervisé. Les caractéristiques de haut niveau seraient donc moins directement liées aux classes, bien que l'information s'y trouve de manière un peu moins directe. L'impact d'une représentation bruitée serait alors plus grand.

Plus illustrer cette idée, on peut penser à une première unité qui représente directement la présence de la classe "1". Un peu de bruit sur cette unité n'a pas un grand impact sur la capacité à déterminer la valeur qu'elle devrait prendre en absence de bruit. Par contre, une caractéristique comme la "largeur en pixels du caractère" est moins directement liée à la classe, bien qu'on puisse quand même penser qu'un caractère étroit a de plus fortes chances d'être un "1". Par contre, la largeur exacte est plus utile à connaître, car d'autres caractères étroits existent. Donc en introduisant du bruit sur cette mesure, la prédiction de la classe peut être plus grandement affectée. En laissant la représentation relaxer, l'hypothèse serait que le bruit affecte moins la représentation à la dernière couche. On peut observer un tel phénomène de débruitement dans les couches supérieures dans la section 4.11.3 sur les représentations dans la version convolutionnelle.

Nombre d'itérations	Tâches	Minimum	σ	Tâches n'ayant pas convergé
0	33	3,38%	0,14%	0
2	30	3,71%	0,13%	0
4	32	3,97%	0,02%	3
6	22	3,57%	0,18%	2
8	33	3,53%	0,19%	3

Tableau 4.III – **Expériences avec bruit poivre : meilleurs taux d'erreur de classification sans *finetuning* des couches inférieures, sans bruit sur l'entrée**. Le minimum donné correspond au "minimum moyen" de toutes les tâches ayant ce nombre d'itérations de relaxation. Pour le calcul d'incertitude sur le minimum, voir la section 4.5.1. Les tâches n'ayant pas convergé avaient des erreurs au-delà de 60%.

	Sans relaxation		Avec relaxation	
Niveau de bruit	Minimum moyen	σ	Minimum moyen	σ
20% à 40%	3,36%	0,06%	2,68%	0,19%
30% à 50%	3,61%	0,08%	2,78%	0,25%
40% à 60%	4,25%	0,15%	3,55%	0,17%
50% à 70%	5,91%	0,13%	4,33%	0,18%
60% à 80%	7,16%	0,06%	4,90%	0,10%

Tableau 4.IV – **Expériences avec bruit poivre : performances en classification sans *finetuning* des couches inférieures, et en présence de bruit sur l'entrée**. Pour le calcul d'incertitude sur le minimum, voir la section 4.5.1. Comme le bruit avait un niveau échantillonné de manière uniforme entre 20% et 80%, les minima ont été calculés pour des intervalles de valeurs du niveau de bruit.

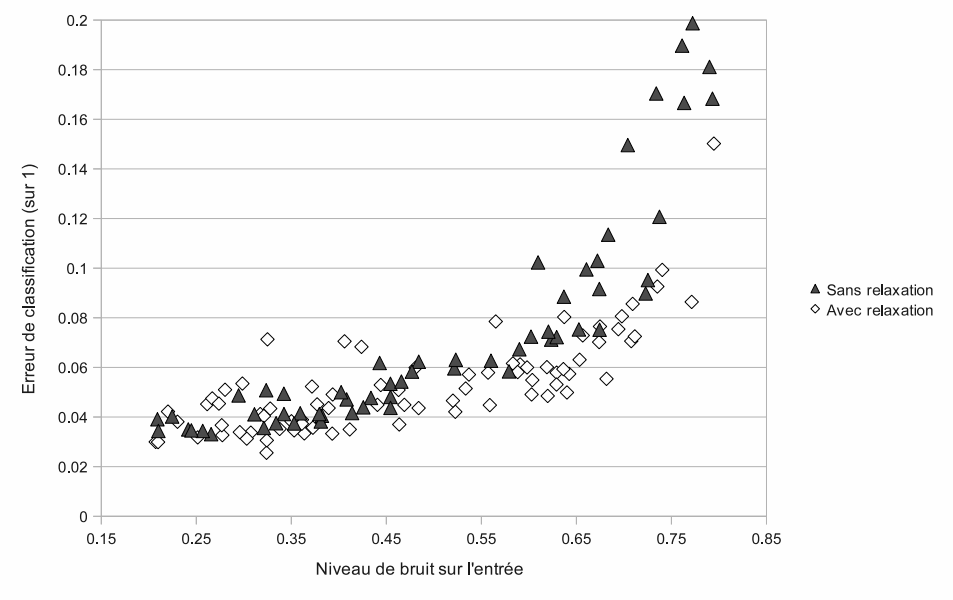


Figure 4.7 – **Expériences avec bruit poivre : classification sans *finetuning*, avec bruit sur l'entrée**. Il s'agit des erreurs de classification (sur l'ensemble de validation de MNIST) finale pour chacune des 150 tâches, identifiés selon que la tâche utilisait ou non la relaxation. On voit bien que les tâches avec relaxation semblent bien au-dessous des tâches sans celle-ci, effet d'autant plus marqué pour les bruits forts.

4.7 Expérience avec bruit “poivre” et MNIST 7x7

La relaxation ne semblait pas aider, dans la plupart des cas, lors des premières expériences en classification avec *finetuning*, même avec bruit sur l'entrée, il était naturel de se demander pourquoi et de chercher à confirmer des hypothèses pour l'expliquer. C'était notamment étrange car Bengio et Gingras [8], avec un arrangement similaire, obtenaient de biens meilleurs résultats grâce à la relaxation, en entraînant avec un critère purement supervisé.

Une hypothèse qui nous est alors apparue plausible était que le jeu de données utilisé dans le cas de Bengio et Gingras [8] présentait moins de redondance dans chaque exemple. “Redondance” désigne ici le fait que des pixels voisins peuvent compenser pour un pixel manquant ou bruité, les voisinages étant fortement corrélés. Avec une telle redondance, l'hypothèse serait que l'impact du bruit lors de la classification est moindre, étant donné qu'il y a de l'information supplémentaire qui compense dans l'entrée. Notre hypothèse de base étant que la relaxation aide à débruiter et donnant graduellement une représentation plus “claire” à la couche précédant la couche de classification, l'idée serait alors qu'avec moins de redondance dans l'entrée, elle présenterait ici un bénéfice.

Nous nous sommes donc attaqués à trouver un jeu de données avec moins de redondance, mais avec tout de même des abstractions et motifs de haut niveau utiles. Une réponse naturelle était de simplement réduire la redondance dans les images de MNIST en réduisant leur résolution, ici à 7x7 pixels (par interpolation bicubique).

Nous avons donc lancé 50 tâches de préentraînement. Il y avait 2 ou 3 couches de 300 unités. Le préentraînement de chaque couche variait entre 20 et 50 époques. Le niveau de bruit de type poivre était échantillonné dans l'intervalle de 20% à 60%, . L'étape suivant directement, le *finetuning* non supervisé en relaxation, permettait des tâches de 0 à 8 étapes de relaxation, avec une forte proportion de tâches de 0, pour comparer les résultats avec et sans relaxation. Le bruit était alors sélectionné de façon indépendante. Ici aussi de 20 à 50 époques étaient possible pour cette étape, pour tout le modèle. L'étape finale de classification avec *finetuning* supervisé, avec bruit sur l'entrée du même niveau qu'à l'étape de *finetuning* en relaxation, impliquait la moyenne d'un coût de classification

softmax appliqué à toutes les étapes de relaxation. De 10 à 30 époques étaient possibles à cette étape.

L'erreur de classification en fonction du niveau de bruit, selon qu'il y avait ou non relaxation (tous nombres d'itérations confondus), est donné à la figure 4.8. On y voit que malheureusement ici encore la relaxation ne semble pas aider, bien qu'elle ne semble pas nuire vraiment non plus. Il y a plus de tâches n'ayant pas convergé du côté des tâches avec relaxation, mais nous croyons que c'est dû à un mauvais choix de vitesse d'apprentissage pour ces tâches et non pas à un problème fondamental avec la relaxation comme telle (étant donné qu'il y a amplement de tâches ayant convergé).

Une chose curieuse, par contre, est la linéarité apparente de la relation, très différente des courbes obtenues lors des autres expériences avec MNIST original (28x28). Une hypothèse pour expliquer ce phénomène est justement liée à la redondance : dans le cas où l'entrée contient beaucoup de redondance, l'impact d'un bruit faible est largement moindre. Par contre, à mesure que le bruit grandit, il y a de moins en moins possibilité de compenser l'information manquante, et la dégradation pour une même augmentation du bruit est plus grande. Ici, comme il y a dès le départ moins de redondance, le bruit ajouté résulte en une perte d'information qui serait directement proportionnelle à ce bruit. En spéculant un peu plus, la ligne qu'on voit au bas des points correspondrait alors à une sorte de limite de performance atteignable. La comparaison "relaxation *vs* sans relaxation" proche d'une telle limite serait alors peu concluante.

4.8 Expériences en reconstruction, avec bruit poivre et sel, sur MNIST et Caltech Silhouettes

Comme nous observions des performances intéressantes en reconstruction dans les premières expériences avec bruit poivre, il devenait intéressant d'essayer de se comparer en performance de reconstruction avec d'autres travaux. Un problème rapidement constaté est qu'il n'y a malheureusement pas de tâche de reconstruction "standard" dans la littérature d'apprentissage machine, comparable à la tâche de classification de l'ensemble de test de MNIST.

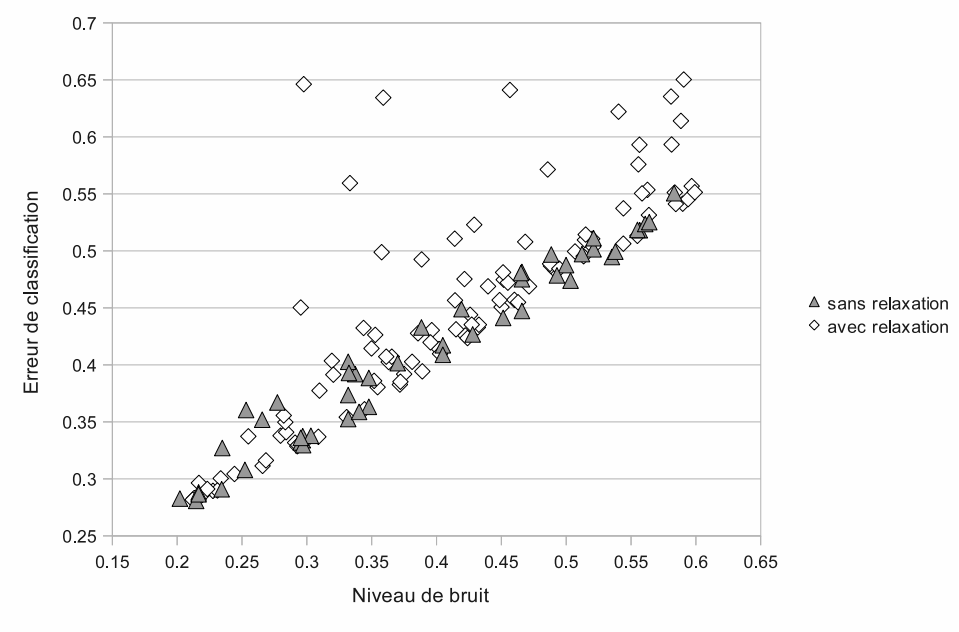


Figure 4.8 – **Expérience avec bruit “poivre” et MNIST 7x7 : erreur de classification** sur l’ensemble de validation de MNIST 7x7 en fonction du bruit sur l’entrée. On voit que les expériences sans relaxation semblent toujours supérieures ou égales en performance aux tâches avec relaxation, bien que la différence semble mince et toutes deux semblent s’approcher d’une droite implicite, probablement une limite de performance.

Après maintes recherches, l'article Marlin et al. [30] a surgi, où sont présentés des résultats, en reconstruction avec bruit et pour d'autres tâches, d'une méthode assez récente et performante, au moins en regard des autres types de tâches. La méthode présentée ne sera pas expliquée en détail ici, mais elle repose sur différents critères d'entraînement pour RBM. La reconstruction se fait à partir, entre autres, de données de MNIST et Caltech Silhouettes, ce dernier étant un ensemble de données qu'ils introduisent avec l'article. Ils y mettent du bruit poivre et sel (description à la section 4.4) de différents niveaux, puis mesurent la performance (MSE) de reconstruction.

C'est donc ce que nous avons fait ici. À noter qu'il y a de petites différences dans le jeu de données. La première est que nous n'avons pas "binarisé" les images (mis un seuil sur les niveaux de gris), bien qu'elles soient en grande partie binaires, déjà. La seconde est que pour Caltech Silhouettes le blanc et le noir ont été inversés. Hormis ces différences, les résultats en MSE devraient être *plutôt* comparables, mais en rétrospective il aurait été préférable de binariser les images pour obtenir des résultats *véritablement* comparables (surtout pour la binarisation de MNIST).

On entraînera donc, au préentraînement comme au *finetuning* non supervisé, sur du bruit poivre et sel. Il s'agit de réseaux ayant 300 unités cachées de type *softsign* par couche, et à 2 ou 3 couches, de façon à rester assez comparable en terme de capacité (nombre de paramètres) aux modèles de l'article de comparaison. Le niveau de bruit poivre et sel va de 20% à 80%, et le bruit choisi au préentraînement était indépendant du bruit choisi lors de l'étape de *finetuning* non supervisé à relaxation. Lors de cette étape, le nombre d'itérations de relaxation allait de 2 à 8. Pour MNIST, le nombre d'époques variait entre 20 et 50 pour le préentraînement, de même pour l'étape de *finetuning* non supervisé. Le nombre exact était déterminé par le critère d'arrêt prématuré basé sur l'erreur de reconstruction. Pour Caltech Silhouettes, compte tenu de la petite taille du jeu de données, les limites étaient multipliées par un facteur 10.

Les tâches étaient exécutées sur la grappe de GPU Angel du réseau SharcNET et finissaient très rapidement. Pour MNIST, le préentraînement prenait environ 30 secondes par époque, pour toutes les couches. À la relaxation, les tâches les plus lentes, donc avec 8 étapes de relaxation et 3 couches, demandaient environ 2.5 minutes par époque. Pour

Caltech Silhouettes, vu la taille très petite du jeu de données, chaque époque était environ 10 fois plus rapide.

4.8.1 Calcul de la MSE prise comme mesure de performance d'une tâche

Il y aura ici deux types de performances rapportées. On calcule la MSE sur l'ensemble de test [2] à toutes les époques. Les performances dites "première couche seulement" sont la meilleure MSE sur l'ensemble de test lors du préentraînement de la première couche. Ça donne une idée de ce qu'on peut atteindre sans relaxation, avec un autoencodeur débruitant standard. Ça servira de *baseline*. Rappelons qu'avec une seule couche, un autoencodeur débruitant a tendance à apprendre des filtres assez locaux dans l'image, donc l'intuition est qu'une seule couche ne peut pas assembler ces caractéristiques simples par composition (voir la section 1.6).

Le deuxième type de performance est "avec relaxation", et implique toutes les couches. Le calcul se fait à chaque époque. On prendra la MSE sur l'ensemble de test, par itération de relaxation. On obtiendra donc la MSE de la première itération, de la seconde, etc. de façon séparée. Selon la tâche et son hyperparamètre *num_iter*, on peut donc avoir de 2 à 8 MSE par époque. Au final on prendra comme performance de la tâche la meilleure MSE de test atteinte, toutes itérations de relaxation et époques d'entraînement confondues.

Mentionnons cependant que les modèles à une seule couche, sans relaxation, sont en désavantage en terme de capacité, car ils n'ont que 300 unités, *versus* 600 et 900 au total pour les modèles avec relaxation. Une procédure de comparaison peut-être plus juste aurait été de faire varier la taille de cette couche unique dans une série d'expériences et de sélectionner la taille optimale (capacité assez grande sans surapprentissage).

2. La procédure standard aurait été de calculer la MSE sur l'ensemble de *validation* pour toutes les tâches, de sélectionner la meilleure (sélection de modèle) pour un niveau de bruit donné, puis de rapporter sa MSE pour l'ensemble de *test* comme performance pour ce niveau de bruit. Nous avons suivi une approche différente en calculant des marges d'erreur pour les minima trouvés sur le jeu de données de test. En suivant la procédure standard, et si nous avions binarisé les images de MNIST, la comparaison avec les résultats de Marlin et al. [30] serait un peu plus solide.

Nous ne croyons pas que cela aurait changé qualitativement les conclusions, par contre, les résultats sur l'ensemble de validation et de test devant être assez semblables, et la différence de performance trouvée (vs Marlin et al. [30]) à niveau de bruit plus élevé étant très grande. En fait on peut voir que dans les cas de bruit élevé, les tâches sont presque toutes largement meilleures, donc peu importe l'approche de sélection de minimum choisie, il est très probable que le minimum resterait largement inférieur en terme d'erreur.

4.8.2 Résultats pour les deux jeux de données, MNIST et Caltech Silhouettes

Les phénomènes observés étant sensiblement les mêmes pour les deux jeux de données, nous présenterons ici les résultats ensemble. Le tableau 4.V et la figure 4.9 présentent les résultats obtenus pour les deux approches implémentées par nos propres moyens, ainsi que les résultats de Marlin et al. [30] pour leur méthode la plus performante.

Deux observations évidentes sautent aux yeux. La première est que les autoencodeurs débruitants, d'une seule couche, sont plus performants pour cette tâche que la méthode présentée dans l'article de comparaison. C'est d'autant plus clair lorsque les niveaux de bruit augmentent. Nous croyons que c'est dû au fait que nous entraînons spécifiquement à la tâche de débruitement, ce qui met nos méthodes en avantage clair car la tâche en question n'est introduite *qu'après* l'entraînement dans le cas de cet article.

La seconde observation frappante est que les modèles avec relaxation sont légèrement, mais significativement, meilleurs que les modèles n'impliquant qu'une seule couche. Les moyennes des colonnes d'amélioration sont 5.46% pour MNIST et 4.77% pour Caltech Silhouettes. Il faut rappeler que ça pourrait être dû à une meilleure optimisation des hyperparamètres pour le cas des modèles avec relaxation. C'est ce que suggèrent les points montrant que certaines tâches d'une seule couche n'auraient pas eu le temps d'atteindre de bonne performances (c.-à-d. qu'ils semblent être des données aberrantes), alors que ce n'est pas le cas pour les tâches avec relaxation. Tout de même, de par le groupement des points de chaque type de tâche, on voit que les modèles à une couche semblent frapper une limite de performance moins bonne que celle des tâches avec relaxation, et c'est ce que semblent montrer les tableaux présentant les minima atteints et leur incertitude.

Comme la capacité des réseaux à une seule couche est inférieure, on ne peut pas conclure que la relaxation est meilleure à capacité égale, bien sûr. On peut par contre dire que pour une même première couche, il y a un effet bénéfique à introduire la rétroaction des couches supérieures et la relaxation, alors que ça aurait pu nuire.

Une chose importante à mentionner est qu'une amélioration bien visible à l'oeil peut ne pas être aussi frappante numériquement. C'est que souvent l'amélioration se fera

dans une portion assez faible de l'image, ou encore consistera à rendre certains "pixels" (unités de sortie) un peu plus certains, c.-à-d. plus blancs. Ainsi, lorsqu'on approche de la limite de performance de reconstruction (image de sortie égale à l'image d'entrée), des gains pourtant visibles sont plus difficiles à percevoir dans les chiffres.

4.9 Expériences avec bruit poivre et sel et occlusions

Comme on pouvait le constater dans les expériences en reconstruction avec bruit poivre et sel, la relaxation semble permettre un petit gain en performance, mais une seule couche cachée offre une performance semblable. L'intuition qui vient rapidement à l'esprit pour expliquer ça serait qu'il y a beaucoup de redondance locale permettant de compenser pour un pixel bruité par son voisinage immédiat, faisant en sorte que les motifs assez locaux appris en général dans une première couche débrouillent plutôt bien ce type de bruit.

Le but ici est de tester cette hypothèse et de créer une tâche plus ardue, avec des déformations de l'image obligeant à aller chercher de l'information plus loin que le voisinage immédiat. Avec une capacité plutôt faible à la première couche cachée, ça devrait donner la chance aux couches supérieures de composer avec les caractéristiques apprises à bas niveau, et si l'hypothèse est bonne on devrait observer un meilleur gain en performance en utilisant la relaxation. On utilisera donc des occlusions telles que décrites à la section 4.4.

Nous laisserons malgré tout du bruit poivre et sel car dans des expériences passées nous avions pu constater qu'avec uniquement des occlusions l'apprentissage semblait se concentrer à reconstruire parfaitement les parties non bruitées, ce qui menait les paramètres appris à une mauvaise solution.

Ici, par contre, il n'y a pas d'article avec lequel se comparer, donc on n'aura que des résultats avec relaxation, et sans relaxation, c'est-à-dire résultantes d'une seule couche d'autoencodeur débruitant.

Ces expériences étaient faites sur les machines avec GPU de la grappe Angel de SharcNET.

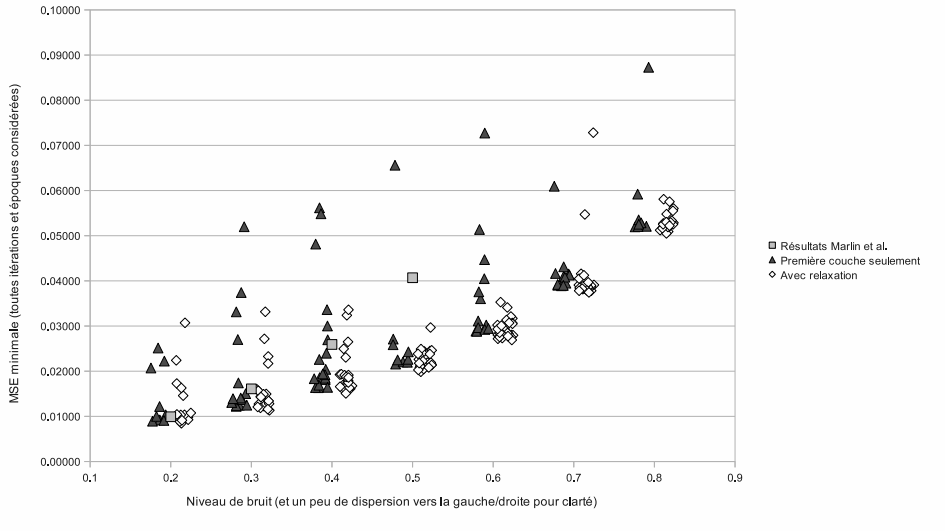
	Une couche seulement			Avec relaxation			Amélioration
Niveau de bruit	#	Min $\times 10^2$	σ $\times 10^2$	#	Min $\times 10^2$	σ $\times 10^2$	
20%	9	0.90	0.01	17	0.86	0.02	4.4%
30%	15	1.23	0.01	20	1.14	0.02	7.3%
40%	20	1.64	0.03	23	1.54	0.05	6.1%
50%	9	2.17	0.02	26	2.00	0.02	7.8%
60%	13	2.89	0.01	30	2.70	0.02	6.6%
70%	12	3.894	0.004	25	3.76	0.02	3.4%
80%	9	5.199	0.004	22	5.07	0.03	2.5%

(a) Résultats sur MNIST

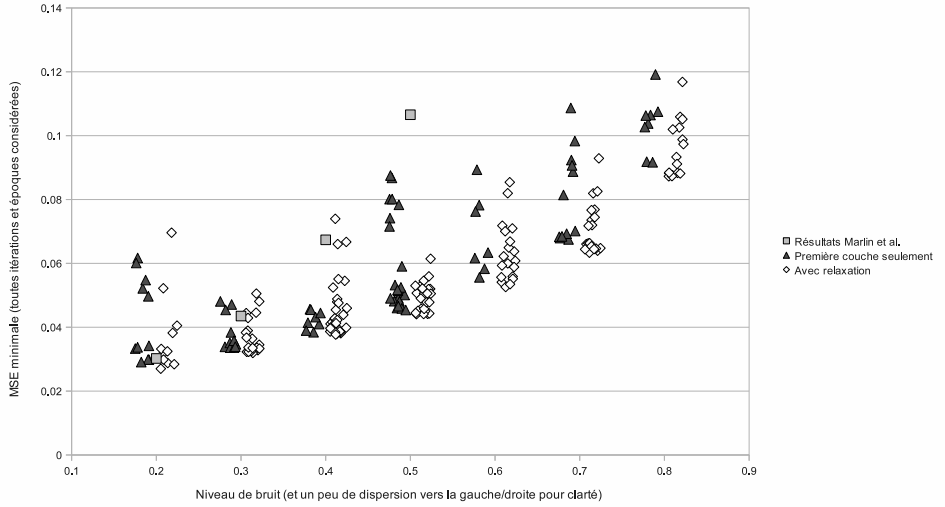
	Une couche seulement			Avec relaxation			Amélioration
Niveau de bruit	#	Min $\times 10^2$	σ $\times 10^2$	#	Min $\times 10^2$	σ $\times 10^2$	
20%	11	2.92	0.03	11	2.76	0.07	5.5%
30%	11	3.36	0.01	21	3.21	0.02	4.5%
40%	8	3.86	0.04	30	3.79	0.02	1.8%
50%	22	4.56	0.03	25	4.410	0.002	3.3%
60%	7	5.7	0.2	19	5.30	0.06	7.0%
70%	12	6.77	0.03	19	6.35	0.03	6.2%
80%	8	9.2	0.2	16	8.73	0.01	5.1%

(b) Résultats sur Caltech Silhouettes

Tableau 4.V – **Expérience en reconstruction avec bruit poivre & sel sur MNIST (haut) et Caltech Silhouettes (bas) : minima de MSE selon niveau de bruit** (colonne "Min"), comparant la performance d'une seule couche d'autoencodeur débruitant avec un réseau à 2 ou 3 couches utilisant relaxation (tous nombres d'itérations confondus). Les valeurs montrées ici ont été multipliées par 10^2. Pour le calcul d'incertitude sur le minimum, voir la section 4.5.1. L'amélioration est calculée avec la formule $\frac{min_{une\ couche} - min_{avec\ relaxation}}{min_{une\ ccouche}} \times 100$ en utilisant les minima moyens donnés ici (c.-à-d. colonne "Min").



(a) Résultats sur MNIST



(b) Résultats sur Caltech Silhouettes

Figure 4.9 – **Expériences avec bruit poivre et sel : erreur de reconstruction MSE sur l'ensemble de test de MNIST (haut) et Caltech Silhouettes (bas)**, selon le niveau de bruit en entrée, avec bruit sur la position pour distinguer les points (c.-à-d. qu'autrement les groupes de points seraient normalement horizontalement alignés, rendant la densité difficile à distinguer). Les triangles gris foncé, déplacés vers la gauche, sont les MSE minimales obtenues durant l'entraînement pour chaque tâche de préentraînement, donc la performance de la première couche seulement. Les losanges blancs, déplacés vers la droite, sont les MSE minimales obtenues (durant l'entraînement, toutes itérations de relaxation confondues), pour chaque tâche, durant le *finetuning* non supervisé avec relaxation. Les couleurs représentent le nombre d'itérations de relaxation. Les carrés gris pâle sont les performances rapportées dans Marlin et al. [30] pour leur méthode la plus performante, mais le niveau de bruit maximal essayé était 50%.

4.9.1 Expérience de reconstruction sur MNIST avec occlusions et bruit poivre et sel

La première expérience avec occlusions impliquait de reconstruire des chiffres de MNIST. Pour 4 différents niveaux d'occlusions (0.002, 0.005, 0.01, 0.015), et pour 7 niveaux de bruit poivre et sel, de 20% à 80%, donc pour 28 combinaisons des deux paramètres, quelques tâches étaient lancées. Durant le préentraînement des premières couches, la meilleure performance de reconstruction MSE sur l'ensemble de test était conservée [3]. L'entraînement était très rapide, prenant environ 30 secondes par époque peu importe la couche au préentraînement, et au maximum environ 4 minutes par époque pour le *finetuning* non supervisé en relaxation.

Il y a donc 115 valeurs de MSE pour ce type de tâche, au total, toutes combinaisons confondues. Ces dernières servaient ensuite à comparer avec les meilleures performances MSE à la fin du *finetuning* non supervisé avec relaxation. Il y a 140 valeurs pour cet autre type de tâche. Si les nombres semblent sans lien (115 et 140), c'est qu'il y avait des problèmes techniques arrêtant aléatoirement certaines tâches. En fait 150 tâches à relaxation avaient été lancées à partir d'une cinquantaine de tâches de préentraînement (avec 3 copies en passant de l'un à l'autre), mais d'autres tâches de préentraînement ont été lancées pour avoir plus de données de comparaison.

Les tâches impliquaient des réseaux à 1000 unités de type *softsign* par couche, avec 2 ou 3 couches. De 2 à 8 itérations étapes de relaxation étaient possibles. Le même niveau de bruit (poivre et sel et occlusions) était utilisé au préentraînement et au *finetuning* non supervisé.

Compte tenu de petit nombre de tâches (de 0 à une dizaine) par combinaison de niveau d'occlusion et niveau de bruit, montrer les performances brutes induirait en erreur. Il est en fait difficile d'obtenir une idée claire de ce qui se passe pour une combinaison donnée. Plutôt nous prendrons ici l'amélioration moyenne considérant le minimum

3. C'est ici un méthode qui diffère un peu de l'approche standard, de façon semblable à ce qui est dit à la note de bas de page 2 à la page 67. Selon cette approche, nous aurions fait la sélection de modèle (choisir la meilleure tâche pour chaque combinaison de bruits) en se basant sur l'erreur sur l'ensemble de *validation* puis rapporter la performance sur l'ensemble de *test* de cette tâche. Encore une fois, à l'intuition, cela n'aurait probablement pas changé l'amélioration observée d'une méthode à l'autre.

atteint pour chacune des deux méthodes, pour chacune des combinaisons. L'amélioration pour une combinaison donnée est donnée par $\frac{min_{une\ couche} - min_{avec\ relaxation}}{min_{une\ couche}} \times 100$, où les "min" sont les réels points minimum pour une combinaison donnée (donc pas de calcul de l'incertitude sur le résultat pour une combinaison donnée).

Les améliorations par combinaison sont données au tableau 4.VI. On voit donc une amélioration claire dans presque toutes les combinaisons des deux types de bruit, sauf pour le cas le plus simple (le moins bruité selon les deux paramètres). La moyenne globale des améliorations est donc de 6.4%.

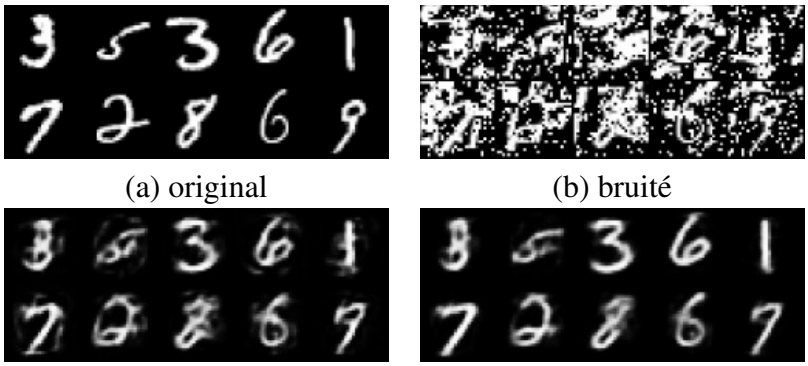
On voit aussi d'autres tendances intéressantes. Premièrement, l'amélioration est de moins en moins importante à mesure qu'on ajoute du bruit poivre et sel. C'est cohérent avec l'idée selon laquelle le bruit poivre et sel est plus facilement enlevé par un modèle à une seule couche (plus facilement que des occlusions). Suivant la même idée, l'amélioration est de plus en plus grande quand on passe de 0.002, à 0.005, puis à 0.01 d'*occlusion_density*.

L'incertitude sur la moyenne de 6.4% étant difficile à évaluer, et la différence avec les améliorations obtenues pour le cas du bruit poivre et sel (expérience précédente sur MNIST et Caltech Silhouettes) étant assez faible, on s'abstiendra de comparer l'amélioration obtenue dans les deux expériences. On peut cependant apprécier un bon gain visuel entre les différentes étapes de relaxation, comme illustré à la figure 4.10. De tels gains, principalement une atténuation progressive des occlusions, ne produisent peut-être pas une énorme différence numérique sur la MSE, mais visuellement sont importants et montrent que des motifs un peu plus larges peuvent être pris en compte grâce à la relaxation.

Encore une fois, la capacité de la première couche étant 2 ou 3 fois inférieure à la capacité des modèles à relaxation, on ne peut pas conclure que la relaxation aide à capacité égale. On peut quand même conclure que pour une même première couche, la relaxation était bénéfique.

occlusion_density	**0.002**	**0.005**	**0.01**	**0.015**	
corruption_level					**Moyenne**
0.2	-17.4%	11.6%	16.4%	12.3%	5.7% (13%*)
0.3	10.2%	12.4%	14.7%	8.1%	11.4%
0.4	11.7%	3.8%	7.3%	8.8%	7.9%
0.5	7.4%	7.9%	7.9%	3.6%	6.7%
0.6	9.4%	9.4%	5.6%	2.8%	6.8%
0.7	**	4.9%	3.6%	2.7%	3.7%
0.8	2.7%	1.3%	3.7%	2.2%	2.5%
Moyenne	4.0%	7.3%	8.5%	5.8%	**6.4%**

Tableau 4.VI – **Expérience de débruitement avec occlusions et bruit p&s sur MNIST : amélioration pour la valeur minimale atteinte** par les tâches pour les combinaisons de *occlusion_density* et *corruption_level* données par la rangée et colonne d'une valeur. C'est donc l'amélioration obtenue en passant de la meilleure tâche avec une couche seulement à la meilleure tâche avec plusieurs couches et relaxation. La valeur est obtenue par $\frac{min_{\text{une couche}} - min_{\text{avec relaxation}}}{min_{\text{une couche}}} \times 100$. (* : 13% si on ne tient pas compte de la valeur négative pour *occlusion_density*=0.002. ** : il n'y avait pas de tâches à une couche seulement pour cette combinaison, donc les moyennes n'en tiennent pas compte.)



(a) original (b) bruité

(c) première reconstruction (d) 5e reconstruction

Figure 4.10 – **Expérience de débruitement avec occlusions et bruit p&s sur MNIST : exemple de reconstructions avec relaxation**, ici de l'ensemble d'entraînement. Il s'agit ici d'une des meilleures tâches pour un niveau de bruit 20% et un niveau d'occlusions 0.015 (le plus élevé). On voit qu'avec une seule itération, donc en n'impliquant que la première couche, une bonne partie du débruitement est déjà effectué. Notamment, tout ce qui se trouve dans une région où n'apparaît jamais de pixel blanc est éliminé dès cette première étape. Cependant, une amélioration visuelle importante, atténuant les occlusions, est obtenue durant la relaxation.

4.9.2 Expérience de classification, avec finetuning, d'entrée bruitée de MNIST avec occlusions et bruit poivre et sel

Ayant obtenu de bonnes améliorations en reconstruction en utilisant la relaxation, il était naturel de se demander si on aurait des gains en performance de classification sur une entrée avec un bruit semblable. Nous avons donc lancé environ 200 tâches de classification avec et sans relaxation. Dans les cas avec relaxation, une couche de classification était branchée à *chacune des étapes de relaxation* (*classft_cost_at_all_steps* était à Vrai) l'objectif étant de rendre le signal correctif plus "proche" des premières itérations. Le coût pour l'entraînement était donc un moyenne des coûts associés à chacune de ces couches. Les tâches pouvaient faire entre 10 et 30 époques, et prenaient entre 30 secondes et 6 minutes par époque pour s'exécuter.

Contrairement à l'intuition, par contre, il n'y a pas de tendance claire indiquant une amélioration grâce à la relaxation, ici. Sans inclure le tableau des améliorations, nous mentionnerons simplement que plusieurs des "améliorations" étaient négatives, et il ne se dégageait pas de tendance claire. Pour obtenir des résultats significatifs, un plus grand nombre de tâches aurait été nécessaire. Comme expliqué plus en détail pour les expériences de classification pour bruit poivre seulement, une hypothèse plausible pour expliquer ce résultat est que de l'information pertinente pour la classification aurait déjà à être extraite dès la première propagation avant.

4.10 Effet du nombre d'itérations de relaxation

Comme le thème central des présents travaux concerne les modèles à relaxation, il est intéressant, au-delà de l'impact de "relaxation" *versus* "absence de relaxation", de se demander combien d'étapes de relaxation est optimal. C'est notamment intéressant à cause du coût supplémentaire de calcul pour chaque itération de relaxation. Bien que nous n'ayons pas mis beaucoup d'effort en ce sens ou fait d'expériences spécifiquement pour répondre à ce genre de questions, nous avons tenté, au courant des expériences, d'analyser pour obtenir des éléments de réponse. Malheureusement, le portrait n'est pas très clair.

4.10.1 La meilleure itération n'est pas la dernière

Un point pertinent, tout d'abord : la meilleure réponse donnée par le modèle n'est pas nécessairement obtenue à la dernière itération, que ce soit en reconstruction ou en classification. Par exemple, dans les expériences à bruit poivre et sel sur MNIST et Caltech Silhouettes, si on s'intéresse à l'amélioration de la performance de reconstruction d'une étape à l'autre, on s'aperçoit qu'en moyenne il y a amélioration dans les premières étapes, puis une légère détérioration vers la fin. Par exemple, pour les tâches ayant 7 étapes de relaxation, les moyenne des améliorations d'une étape à l'autre est [4.55459185, 0.87572045, 0.03914356, -0.1943492 , -0.24806476, -0.24298227]. Ces valeurs sont données par $\frac{min_k - min_{k+1}}{min_{k+1}} * 100$, où k est l'itération et min_k est la meilleure performance à cette étape.

À noter qu'on parle ici de minimum atteint par une tâche pour une itération donnée, donc le minimum pour l'itération 1 de relaxation, pour une tâche, pourrait avoir été pris à une époque différente du minimum pour l'étape 4. En pratique c'était très souvent la même époque (vers la fin), sauf pour la première reconstruction qui avait tendance à devenir un peu moins bonne avec l'entraînement.

Ce phénomène d'amélioration puis détérioration pourrait s'expliquer par le fait que le modèle s'adapte au nombre d'itérations fixe pour lequel il est configuré, et doit tenter de minimiser son coût en tenant compte de ce nombre. Comme les mêmes poids sont utilisés à toutes les itérations de relaxation et les nombres d'itérations sont assez petits, il est probable que les représentations apprises ne sont pas très stables. C'est-à-dire que si on prenait les modèles appris et laissait la relaxation continuer beaucoup plus longtemps en laissant l'entrée relaxer, il y a peu dans la procédure d'apprentissage pour garantir que la reconstruction continuerait à ressembler à l'exemple original. En fait, il y a de fortes chances, dans certains cas, pour que très rapidement la reconstruction devienne très mauvaise. Par exemple, pour le bruit poivre, la stratégie de débruitement semble être *grosso modo* d'ajouter des pixels, comme on pouvait le voir quand des occlusions blanches étaient utilisées ensuite pour tester le modèle.

En résumé, la représentation n'étant pas conçue pour durer de nombreuses itérations

mais plutôt adaptée à un petit nombre fixe, il peut être une bonne “stratégie” d’adapter pour que les itérations centrales soient meilleures. Ce serait plus “payant” selon le coût, si un choix doit être fait.

4.10.2 Nombre d’itérations optimal

Nous avons tenté de déterminer tout bonnement quel est le nombre d’itérations qui mène aux meilleures performances. C’est parfois un peu compliqué car il y a plusieurs niveaux de bruits, donc chaque niveau est en quelque sorte un problème différent ; on ne peut pas comparer directement numériquement deux tâches avec des niveaux de bruit différents.

La stratégie adoptée a donc été de comparer les tâches d’un niveau de bruit donné, puis de cumuler des “performances relatives” à ces groupes de tâches. Ainsi, une tâche à fort niveau de bruit est comparée à d’autres tâches à fort niveau de bruit, et on peut mesurer sa performance relativement à la meilleure tâche de ce groupe avec une formule simple : $(p - p*)/(p*)$, où p est une performance donnée (MSE de reconstruction, par exemple) et $p*$ est la meilleure performance pour le groupe de tâches. Ça donne donc la “détérioration” relative à la meilleure tâche.

Malheureusement, ayant cumulé ces performances relatives pour les tâches de reconstruction pour les expériences sur MNIST avec bruit poivre & sel ou avec occlusions, il n’y avait aucun nombre d’itérations qui ressortait clairement comme meilleur. Pour chaque nombre d’itérations, il était donc possible de trouver des niveaux de bruit où ce nombre d’itération était le meilleur, ou très proche. Il y avait bien des nombres d’itérations où les tâches n’ayant pas convergé étaient plus nombreuses, mais c’est normalement un problème avec d’autres hyperparamètres qui auraient pu être ajustés.

On ne peut donc pas tirer de conclusion ici, autrement que de dire qu’avec les observations faites il ne semble pas y avoir de clair avantage à aller au-delà de 2 ou 3 itérations de relaxation, c’est-à-dire le minimum pour aller chercher de la rétroaction de toutes les couches cachées (respectivement 2 ou 3) dans les configurations utilisées ici. En réalité plus d’expériences ciblées sur ces questions seraient nécessaires. Il est probable que le nombre d’itérations optimal dépend de la tâche précise considérée, de l’utilité

de l'abstraction et de la composition pour le type d'entrée considérée, etc.

4.11 Expériences avec les réseaux convolutionnels

Les réseaux convolutionnels développés pour ce travail ont été décrits à la section 3.4. Nous avons mené des expériences en reconstruction avec ceux-ci, pour voir comment ils se comparent aux réseaux complètement connectés.

Nous avons donc lancé 40 tâches de préentraînements couche par couche tels que décrits dans cette autre section, toujours sur le jeu de données MNIST. Ensuite, selon le cas, on passait soit directement au *finetuning* avec relaxation, ou par une étape intermédiaire en autoencodeur profond. On s'intéressera à l'impact de cette étape intermédiaire, de même qu'à l'impact numérique et visuel d'autres paramètres, soit la relaxation (ou non) des entrées (*relaxed_inputs*) et l'apprentissage (ou la fixation) de poids d'intégration entre les couches.

Il y avait 3 couches, avec 60, 200 et 500 filtres, de tailles 5x5, 5x5 et 4x4, respectivement. Du *maxpooling* était utilisé pour réduire la dimension des images par un facteur 2 entre chaque paire de couches. Ce sont des choix assez communs pour les architectures convolutionnelles sur MNIST. Du *stretch unpooling* était utilisé pour l'*unpooling*. Les images originellement de dimension 28x28 étaient agrandies à 56x56 avant de servir d'entrée, en remplissant les bords avec du noir. C'est pour qu'à la dernière couche il puisse y avoir des images malgré tout assez larges, chaque étape de convolution rognant un peu les bords. Lors de la relaxation, les tâches avaient de 3 à 7 itérations. Le bruit, poivre et sel, variait entre 20% et 60%. Le bruit était le même à toutes les étapes pour une série de tâches héritant de la même tâche de préentraînement. Les taux d'apprentissage étaient divisés par les dimensions des *feature map* à une couche donnée pour compenser pour le partage de poids, lequel provoque une multiplication du gradient pour chaque paramètre. Le nombre d'époques au préentraînement variait entre 8 et 20 pour chaque couche, selon un algorithme d'arrêt prématuré basé sur l'erreur de reconstruction.

Ces tâches étaient lancées sur les machines ayant un GPU du LISA. Les temps de préentraînement étaient d'environ 1 minute par époque pour la première couche, 10 pour

la seconde et 15 pour la dernière.

À l'entrée des couches de convolution, l'entrée normalement sur l'intervalle]0,1[était étirée sur l'intervalle]-1,1[(en multipliant par 2 et soustrayant 1). L'intuition derrière ce changement est que sans celui-ci, "l'absence" d'une partie de l'image est difficile à clairement encoder dans les paramètres. Pour illustrer, penser qu'on cherche à ce qu'un neurone soit à 1 quand l'une de ses multiples entrées est à 0 (absente), tout en permettant de réagir arbitrairement aux autres entrées. C'est difficile à faire car multiplier 0 par n'importe quel poids donne toujours 0. Si par contre l'entrée est absente quand elle prend la valeur -1, alors on peut accorder un poids à cette absence. On aurait aussi bien pu utiliser d'autres non linéarités sur]-1,1[, comme la fonction *tanh* ou *softsign*, mais il faut penser que le *maxpooling* est normalement conçu pour que 0 représente l'absence, bien que ce ne soit pas strictement nécessaire ici.

4.11.1 Entraînement en autoencodeur profond convolutionnel

Un problème avec la relaxation telle que présentée ici est qu'il est difficile de savoir quel est l'impact réel des couches profondes. En effet, comme la reconstruction se produit à la première couche, l'influence de la deuxième couche sur la première, et de la troisième, encore plus indirecte, est difficile à quantifier.

Une façon de s'assurer que l'information "passe à travers" la troisième couche est de configurer les N couches apprises au préentraînement en autoencodeur profond (section 1.5) où les couches 1 et N-1, 2 et N-2, etc. héritent des paires de paramètres W et W^T des couches 1, 2, etc. respectivement. Ainsi, la couche la plus élevée doit avoir une représentation utile, autrement la reconstruction serait mauvaise. Notons que ça demeure convolutionnel, donc dans les couches "descendantes" il doit y avoir une forme d'*unpooling*. On utilise ici aussi du *stretch unpooling*.

On peut aussi continuer l'entraînement avec le principe de l'autoencodeur débruitant dans cette configuration. On montre donc un exemple bruité et, activant tour à tour chacune des couches montante et "redescendante" (encodage et décodage), on tente de reconstruire l'exemple non bruité. En entraînant ainsi, après le préentraînement couche-par-couche, on se trouve à introduire une nouvelle étape de préentraînement précédant la

relaxation. Nous appelons ça **"*deep convolutional autoencoder* (DCAE) *finetuning*"**.

C'est donc une étape possible suivant le préentraînement, et nous avons lancé 40 tâches de ce type à partir des 40 tâches de préentraînement. Le nombre d'époques variait entre 2 et 5, selon un algorithme d'arrêt prématuré basé sur l'erreur de reconstruction. Sur les machines avec GPU du LISA, une époque prenait entre 40 minutes et une heure.

Même s'il s'agit d'une étape intermédiaire, on peut en mesurer la performance de reconstruction. La figure 4.11 montre les résultats de reconstruction (MSE) obtenus pour divers niveaux de bruit, en comparant avec ce qui était obtenu pour des réseaux complètement connectés lancés sur une tâche comparable. Des exemples de reconstructions sur le jeu de données de test, pour une tâche ayant un très bon MSE sur cet ensemble, sont présentés à la figure 4.12, pour donner une idée visuelle de ce qui peut être atteint.

Les résultats sont quand même assez bons, si on se fie aux minima réels atteints (plutôt qu'au nuage de points entier, qui est plutôt dispersé en terme de performances, la plage d'hyperparamètres explorée étant assez grande). Visuellement ça peut être assez satisfaisant aussi. On voit donc que même sans relaxation une architecture convolutionnelle de ce type, malgré la contrainte de localité de champs réceptifs, débruite plutôt bien.

4.11.2 Entraînement de la version convolutionnelle en relaxation

Partant de paramètres de couches préentraînés et potentiellement *finetunés* en DCAE, il est possible de construire un graphe de relaxation, comme expliqué précédemment. Trois options de configuration ont été retenues pour les tests : le passage ou non par l'étape de DCAE, l'apprentissage ou non des paramètres d'intégration, et la relaxation ou non des entrées. Nous avons donc lancé 80 tâches avec relaxation : 40 tâches partant des tâches de préentraînement directement, et 40 tâches partant de tâches entraînées en DCAE. Pour les deux autres options, architecturales, nous avons partagé les tâches à environ moitié-moitié pour chacune des deux, les trois options étant choisies indépendamment. Les tâches pouvaient effectuer entre 2 et 6 époques, et sur les machines à GPU du LISA chaque époque prenait entre 1 et 4 heures, selon le nombre d'itérations de relaxation.

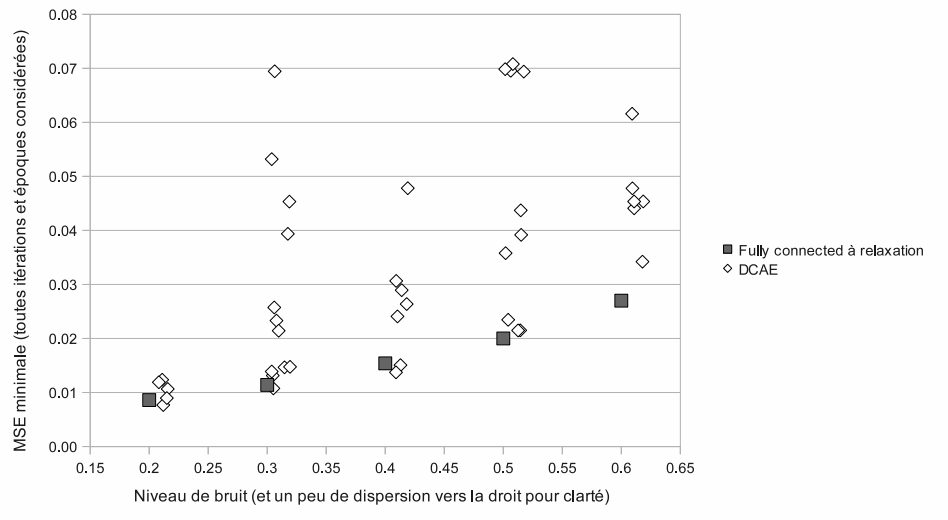
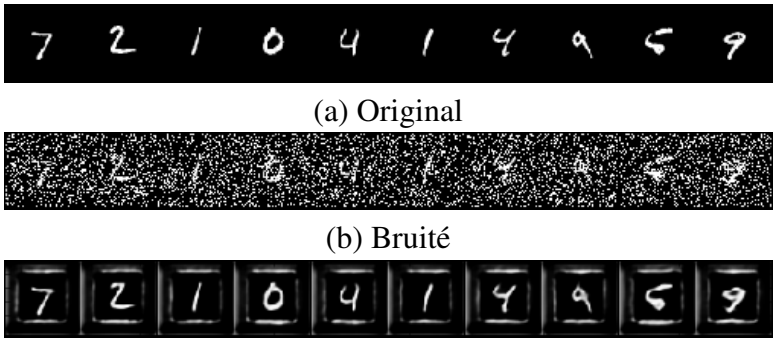


Figure 4.11 – **Tâches d'autoencodeurs profonds convolutionnels : erreur de reconstruction (MSE)**. Meilleure performance de chacune de ces tâches. Le graphe montre aussi le "minimum moyen" rapporté à la section 4.8.2 pour une tâche comparable à réseau complètement connecté avec relaxation.



(a) Original

(b) Bruité

(c) Reconstruction de l'autoencodeur profond

Figure 4.12 – **Tâches d'autoencodeurs profonds convolutionnels : exemple de reconstruction**, pour la meilleure tâche à 40% de bruit en terme de MSE sur le jeu de données de test, pour lequel il est impossible d'apprendre par coeur les formes exactes des caractères au cours de l'entraînement. À noter que le coût de reconstruction ne pénalisant pas la bordure noire autour de l'image d'origine de 28x28, on peut obtenir toutes sortes d'artéfacts dans cette région, comme une petite bordure blanche vue ici.

Dans un premier temps, on peut chercher à savoir numériquement, en regard à la MSE, quels choix étaient les meilleurs. La figure 4.13 montre les résultats pour l'option "avec ou sans étape de *finetuning* en DCAE". On voit donc bien l'avantage de certaines tâches ayant passé par cette étape. L'intuition qui nous semble l'expliquer est bien sûr que cet autre pré-entraînement, impliquant l'ensemble du réseau, favorise l'utilisation des couches profondes dans la "stratégie" qui sera adoptée pour débruiter. Il est en effet assez facile, dans le contexte d'une descente de gradient, de tomber dans une stratégie de débruitement très locale, basée principalement sur la première couche cachée, chemin de l'apprentissage compréhensible mais sous-optimal.

Les deux autres options ne menaient pas à une différence numérique claire, et les nuages de points des cas possibles s'intersectaient trop pour y distinguer une tendance. On peut cependant chercher à savoir visuellement quel est l'impact des différents choix. Par contre, c'est difficile d'évaluer quantitativement des effets purement visuels. C'est pourquoi nous avons procédé à une analyse qualitative de diverses caractéristiques des dernières itérations de reconstruction pour les tâches ayant subi le *finetuning* DCAE, de façon à tenter de rester objectif. Trois caractéristiques étudiées montrent des tendance intéressantes :

- les bords des caractères sont-ils nets (par opposition à flous ou "en vagues") ou non ? ("netteté")
- y a-t-il des taches, du bruit restant, des portions floues n'appartenant pas au caractère ? ("taches")
- y a-t-il des parties de caractère manquants ? ("parties manquantes")

Comme l'évaluation est malgré tout subjective et très approximative, il était inutile d'avoir trop de granularité, donc les échelles vont de 1 à 7. Les résultats sont rapportés à la figure 4.14.

On observe donc assez clairement qu'afin d'obtenir des images nettes et sans taches, des entrées relaxées sont préférables. En vérité, à l'oeil, c'était assez clair que les tâches à entrées relaxées donnait de plus "beaux" résultats. Plus objectivement, il n'y avait pas de tâche sans relaxation des entrées bien nettes ne présentant pas des parties de caractères manquants.

La raison est assez simple, en fait : avec des entrées fixées, la représentation à la première couche de convolution doit nécessairement être une moyenne tenant compte de l'entrée d'origine directement. L'influence peut être diminuée si les poids d'intégration peuvent être appris et réduits en faveur de l'influence du haut. Cependant, en pratique ça n'a que peu de chance de se produire car l'entrée donne beaucoup d'information pour effectuer la reconstruction, donc on ne voudra pas diminuer son influence à ce point.

Par contre, si l'entrée est relaxée, le modèle peut graduellement "retravailler l'image" et en éliminer le bruit. Il est quand même surprenant d'obtenir des bords de caractère nets. On aurait en effet pu s'attendre à ce que l'absence de répétition de l'entrée force le réseau à produire une sortie floue, sa "mémoire" ne lui permettant pas de se "souvenir" de détails précis après quelques itérations. Cependant les résultats numériques en MSE semblent dire que la relaxation des entrées ne provoque pas de différence, et le résultat visuel est très souvent bien net.

La figure 4.15 illustre de bonnes reconstructions pour deux tâches avec et sans relaxation des entrées. On observe que rapidement la version avec entrées relaxées élimine le bruit, phénomène qui n'est pas spécifique à ces deux tâches et probablement expliqué de la même façon que ci-haut. On peut voir le débruitement successif, mais une stratégie purement locale (ne regardant que 5 pixels voisins à la fois) ne pourrait pas compléter des bouts de caractères manquants, car il faut décider des endroits où ajouter des pixels, et ou ne pas le faire, et cela doit être fait selon le contexte environnant. On voit donc bel et bien une influence des couches supérieures sur les couches inférieures, ce qui avait d'abord été confirmé par la version en autoencodeur profond.

Pour ce qui est de l'effet de l'apprentissage (ou non) des paramètres d'intégration, il n'y avait pas de gagnant clair du côté numérique, c'est-à-dire des MSE. Cependant, on semble distinguer que du côté qualitatif ça aiderait de fixer les paramètres à 0.5 plutôt que de les apprendre. Il aurait en réalité été nécessaire de lancer plus de tâches pour savoir réellement ce qui se passe, car l'effet dans le cas où les entrées sont relaxées semble être différent de celui obtenu si les entrées sont fixes ; dans le cas des entrées relaxées, il ne semble pas y avoir d'avantage dans un sens ou dans l'autre, mais il n'y a pas assez de tâches pour confirmer cette conclusion.

Un phénomène intéressant impliquant les paramètres d'intégration, cependant, est que dans les tâches avec une très bonne netteté subjective où les entrées sont relaxées (seulement deux tâches), les paramètres semblent apprendre à favoriser l'information provenant des couches supérieures. Autrement dit, les valeurs des paramètres d'intégration sont plus petites que 0.5, dans un cas beaucoup plus petites, l'effet étant plus marqué pour la première couche. En fait, les tâches à moins bonne netteté semblent apprendre à donner plus de poids à l'information provenant de l'entrée. C'est compatible avec l'intuition selon laquelle la meilleure stratégie pour débruiter est de se fier aux motifs reconnus à haut niveau, mais qu'il est très facile (en terme de descente de gradient) durant l'apprentissage d'aller vers la solution "passable" de simplement rendre l'entrée un peu plus floue.

4.11.3 Phénomène de débruitement dans les couches supérieures

Un aspect intéressant des modèles convolutionnels est que les activations aux couches supérieures gardent une sémantique d'image 2D. Dans toutes les convolutions, on obtient donc des résultats visualisables. En développant le modèle convolutionnel, un jeu de données restreint de deux classes ("5" et "7") avait été utilisé, et nous avions "sondé" ces activations intermédiaires. Un résultat intéressant est que durant la relaxation les couches supérieures semblaient "débruiter" leur représentation. Ça semble naturel, étant donné que le préentraînement force ce comportement, mais c'est tout de même intéressant de remarquer que c'est ce qui se produit aussi après l'entraînement en relaxation. Un exemple de cette visualisation est présenté à la figure 4.16.

4.11.4 Essais avec images de visages en tons de gris

Durant les premières exploration avec la version convolutionnelle, nous avions essayé d'apprendre à débruiter des images en tons de gris. Le jeu de données "Frey faces" était utilisé [4]. Il s'agit de 2000 images du visage de Brendan Frey, images de 20x28 pixels. À peu près la même configuration de réseau était utilisée que pour les tests sur

4. Jeu de données pris à http ://cs.nyu.edu/~roweis/data.html

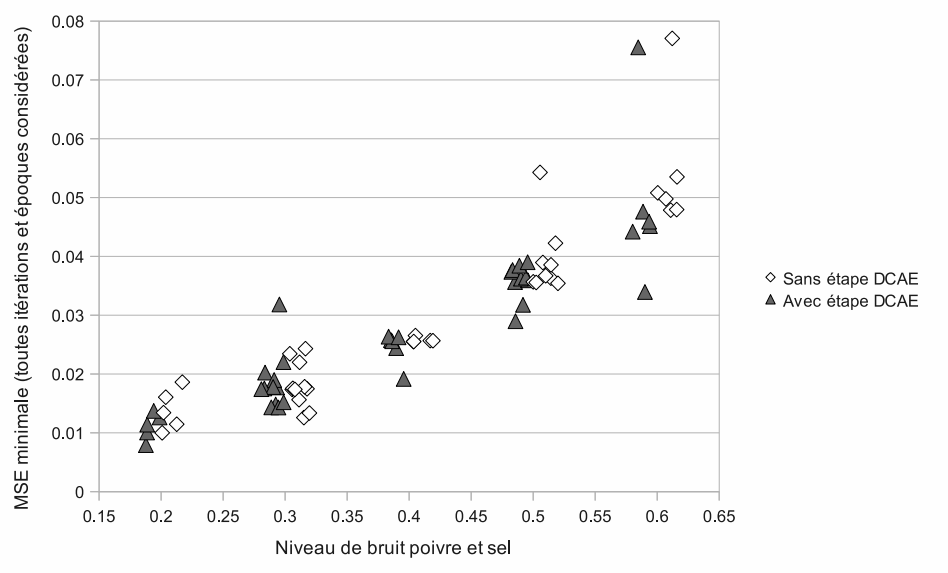


Figure 4.13 – **Tâches avec relaxation pour la version convolutionnelle : MSE de reconstruction minimale**, selon qu'elles soient lancées partant de paramètres ayant passé par une étape de DCAE (triangles gris) ou non (losanges blancs). On voit un avantage assez clair des meilleures tâches ayant passé par l'étape de DCAE.

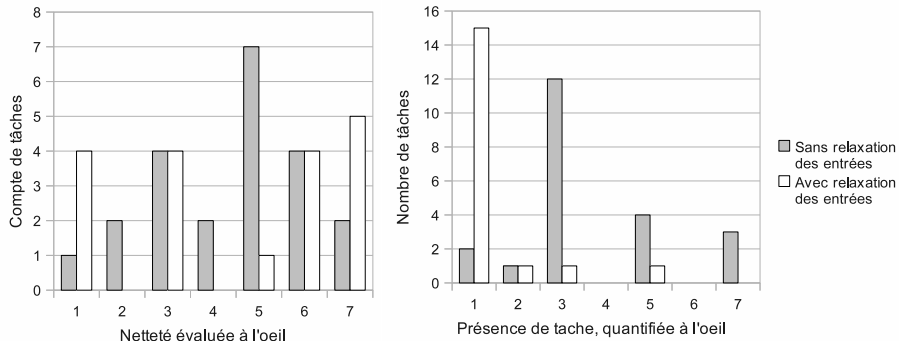
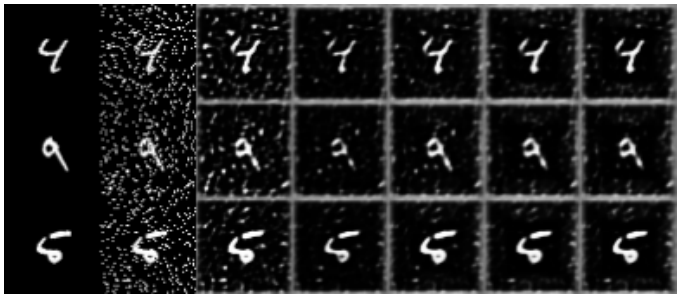
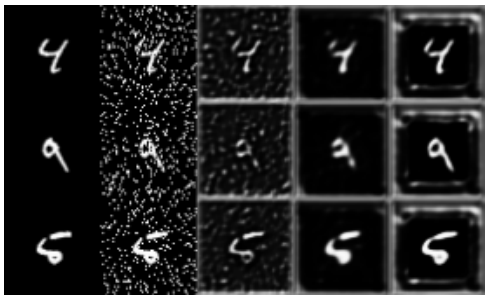


Figure 4.14 – **Tâches avec relaxation pour la version convolutionnelle : histogrammes de la "qualité visuelle" subjective** avec et sans relaxation des entrées. La netteté (à gauche) est évaluée de 1 à 7, où 7 est "très net". La présence de taches (à droite) est évaluée de 1 à 7, où 7 est "très bruité (taché)". Donc dans un cas un chiffre plus grand est meilleur, dans l'autre c'est l'inverse. Noter qu'il n'y a presque pas de taches dans les tâches à entrées relaxées.

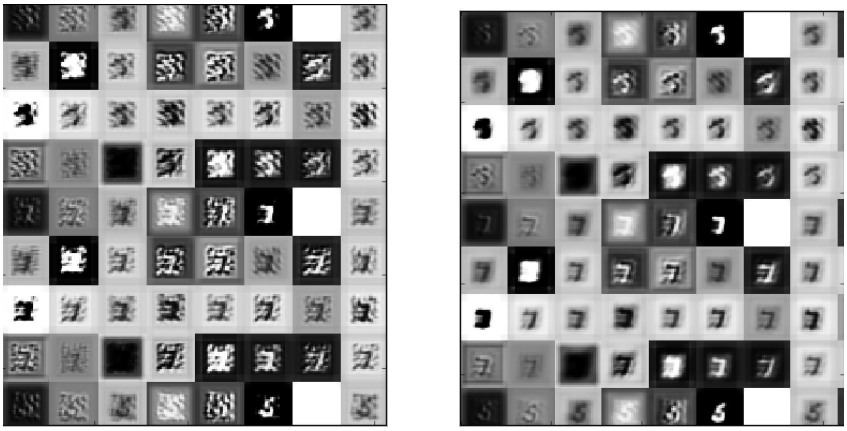


(a) Sans relaxation des entrées



(b) Avec relaxation des entrées

Figure 4.15 – **Tâches avec relaxation pour la version convolutionnelle : reconstructions sur exemples du jeu de données de test**, pour la relaxation version convolutionnelle. Deux tâches, avec entrées non relaxées (haut) et relaxées (bas). De gauche à droite on a d'abord l'entrée originale, l'entrée bruitée, puis les reconstructions successives. En (a) on voit un peu plus de bouts de caractères manquants (sur le "9"), bien que ce soit une des meilleures tâches sans relaxation des entrées.



(a) Propagation avant (b) Deuxième itération de relaxation

Figure 4.16 – **Version convolutionnelle : phénomène de débruitement dans les couches supérieures** durant la relaxation. Les rangées n'ont pas de sémantique particulière, ici. Il s'agit plutôt de plusieurs exemples des classes "5" et "7" d'une même *minibatch*, et de quelques *features maps* consécutives. À gauche : premières activations à la seconde couche. À droite : troisièmes activations à la même couche, après 2 étapes de relaxation.

MNIST décrits ci-haut, mais nous n'avions pas fait d'exploration d'hyperparamètres systématique. Les résultats préliminaires étant mauvais, avec des reconstructions très floues, nous n'avons pas poursuivi dans cette direction. Ça nous semblait indiquer que l'architecture dans sa configuration actuelle était mieux adaptée à des entrées binaires.

4.12 Retour sur la méthodologie

Comme il a déjà été mentionné dans des notes de bas de page précédentes (2 et 3, aux pages 67 et 72), nous avons pour certaines expériences adopté une approche basée sur des intervalles d'incertitude sur l'ensemble de test qui diffère de l'approche standard pour comparaison entre algorithmes. Les conclusions restent à notre avis qualitativement les mêmes, mais si ces travaux avaient à être refaits, l'approche standard serait plus appropriée pour faciliter la comparaison à d'autres recherches là où les différences de performance sont plus petites.

Une autre amélioration importante à la méthodologie aurait été de comparer les modèles sans fixer la capacité, mais plutôt en la laissant libre autant que possible. L'approche

aurait donc été de lancer plusieurs tâches où la capacité varie, de fixer une limite de temps de calcul, et de choisir, parmi les performances obtenues sur l'ensemble de validation, celle qui est la meilleure.

Ici nous fixions une capacité commune à toutes les couches. Entre les tâches d'un même type (avec relaxation, disons) c'est une approche assez correcte. Il aurait par contre été intéresant de comparer les résultats avec relaxation, et ceux à une seule couche sans relaxation, dans ce cadre où la capacité est optimisée de façon indépendante pour chaque type d'algorithme.

Finalement, compte tenu du temps d'entraînement, c'était difficile de laisser les tâches à relaxation s'entraîner pour plus que quelques époques pour leur étape de fine-tuning non supervisé. Il aurait cependant été bon de voir ce qui se produisait lors d'un entraînement beaucoup plus long, de quelques jours voir quelques semaines. On peut penser que cela aurait pu avantager d'autant plus la relaxation dans les tâches de reconstruction, vu que typiquement on comparait avec une seule couche uniquement préentraînée, et qu'au préentraînement on pouvait se permettre d'entraîner plus longtemps (en terme d'époques) tout en ayant un temps de calcul assez court.

CHAPITRE 5

CONCLUSION

Ce mémoire expliquait le contexte, l'approche et les résultats de travaux visant à étudier les performances de modèles à relaxation et rétroaction sur des tâches de débruitement et de classification. De par l'intuition générale que les modèles en couches apprennent des représentations de plus en plus abstraites, que la rétroaction permet aux niveaux supérieurs d'influencer les représentations aux niveaux inférieurs, et par des observations dans cette veine tirées de la neuroscience, cela semblait une voie intéressante à explorer. Des gains de performance ont bel et bien été constatés, principalement sur des tâches de débruitement, et dans une tâche de classification en présence de bruit. Des phénomènes importants, notamment cette influence de représentations aux couches supérieures, ont pu être observés dans des modèles de type complètement connectés ou convolutionnels.

5.1 Concepts sous-jacents et revue de littérature

Nos travaux s'appuient sur un nombre assez grand de concepts tirés du monde de l'apprentissage machine et, dans une moindre mesure, des neurosciences. Tout d'abord, les idées générales de l'apprentissage machine et de l'apprentissage des paramètres de réseaux de neurones ont été présentés. La descente de gradient en vue d'optimiser les paramètres pour satisfaire un coût d'adéquation aux données était la méthode utilisée dans l'entièreté des expériences. Plus spécifiquement, on tentait d'apprendre des représentations robustes permettant de débruiter une entrée, selon des variations sur le principe de l'autoencodeur débruitant, d'abord pour le pré-entraînement couche par couche, mais aussi pour la relaxation ensuite. L'approche s'inscrit globalement dans le cadre des architectures profondes, et le concept de composition et de représentations de plus en plus abstraites était un thème récurrent dans notre analyse.

Différents modèles à relaxation ont été présentés pour expliquer l'inspiration derrière notre approche. L'inspiration la plus directe pour l'approche de relaxation comme

telle provenait de la méthode d'inférence des Deep Boltzmann Machines, bien que nous entraînions avec une approche complètement différente. Notre version convolutionnelle ressemblait en plusieurs points aux travaux sur la Neural Abstraction Pyramid, et dans une moindre mesure à d'autres modèles convolutionnels à relaxation plus récents (CRBM).

Nous n'avons pas tenté de reproduire ces travaux, ni de s'y comparer directement, par contre. Pour comparer la performance de nos modèles, nous utilisions des modèles déjà bien établis comparables, notamment les autoencodeurs débruitants à une couche dans le cas de la reconstruction, ou leur version profonde pour la classification. De plus, MNIST étant un jeu de données très populaire, les performances pouvaient être situées par rapport aux meilleures performances rapportées, mais nos résultats en étant loin nous n'avons pas poussé dans cette direction. Nous avons cependant pu comparer favorablement les autoencodeurs débruitants et nos modèles relaxants à certains résultats de débruitement sur MNIST et Caltech Silhouettes.

5.2 Approche d'entraînement et modèles explorés

L'idée générale de la relaxation telle qu'implémentée ici est de d'abord faire une propagation avant de l'entrée vers les couches supérieures, puis de recalculer les activations de toutes les couches en ayant une influence provenant des couches inférieures, mais aussi supérieures. Il y a donc rétroaction. On dit donc que la représentation peut "relaxer", ce qui a des connotations de "descendre l'énergie" (sans nécessairement que ça soit au sens propre), ou à tout le moins que la représentation aux différentes couches évolue durant les itérations pour être "meilleure" pour certaines tâches.

Nous avions deux versions utilisant cette procédure. La première, et la plus utilisée, avait des couches complètement connectées à ses voisines. Ainsi, on pouvait potentiellement exploiter des motifs impliquant l'ensemble des unités d'entrées, et le modèle était totalement libre d'établir n'importe quelle connectivité. Dans l'autre version, convolutionnelle, on imposait une connectivité locale d'une couche à la suivante, et une répétition des poids, entre autres choses. On pouvait ainsi exploiter la répétition de motifs

utiles à différents endroits des images, et grâce aux réductions de résolution on tentait d'obtenir une certaine invariance aux transformations de l'image. Grâce à un étirement des représentations à plus haut niveau, il était en principe possible, malgré la connectivité locale, d'obtenir une influence du global sur le local dans la relaxation.

5.3 Résultats et tendances observées

Les premières expériences à bruit poivre sur les modèles complètement connectés ont montré qualitativement qu'il était possible d'obtenir un débruitement, voir une déocclusion, visuellement bonne. Cela nous a poussé à faire plus d'expériences en reconstruction. La tâche la plus commune sur MNIST étant la classification, par contre, nous avons tenté plusieurs expériences dans cette direction. Aucune des expériences avec *finetuning*, même plus tard avec d'autres types de bruits et jeux de données, n'a montré de gain grâce à la relaxation, malheureusement. Par contre, dans le cas où la représentation est fixe et apprise de façon non supervisée, donc sans *finetuning*, on a observé un gain appréciable en classification avec des exemples à classifier bruités.

Nous avons poursuivi avec des expériences avec bruit poivre et sel, et avec des occlusions tirées du jeu de données MNIST, pour quantifier les gains en reconstruction. Dans les deux types de bruits, des améliorations de l'ordre d'environ 4% à 6% sur la MSE de reconstruction, en moyenne, étaient observables lorsqu'on passait d'un modèle à une seule couche à un modèle à plusieurs couches. Malgré que les reconstructions à une seule couche étaient souvent déjà plutôt bonnes, l'impact visuel de ces quelques pourcents était qualitativement palpable.

Les expériences en version convolutionnelle visaient tout d'abord à montrer qu'une telle architecture "fonctionne", donc permet de bonnes reconstructions. On a pu le constater, et les reconstructions obtenues, bien que légèrement moins bonnes que celles obtenues avec le modèle complètement connecté, sont quand même très acceptables. On tentait aussi de voir une influence des couches supérieures sur les couches inférieures, du global sur le local, malgré la connectivité locale de chacune des couches. Grâce aux résultats à l'étape d'entraînement en autoencodeur profond, à une analyse visuelle

des reconstructions, et aux paramètres d'intégration appris, on a pu confirmer que ce phénomène se produisait. On a aussi pu observer, à l'oeil, un effet de débruitement dans les couches supérieures.

L'impact du nombre d'itérations de relaxation a été étudié, mais sans conclusion ferme. On sait par contre que la meilleure itération n'est pas la dernière, et que la majeure partie du gain par rétroaction semble être obtenue lors des quelques premières itérations de relaxation. Plus d'expérimentation spécifique à une tâche serait nécessaire pour déterminer le nombre d'itérations idéal pour cette tâche.

5.4 Pistes futures

Les conclusions obtenues ne sont bien sûr que des tendances, et la forme du jeu de données a probablement beaucoup à voir avec les bons résultats. Notamment, le fait que les entrées soient binaires nous semble jouer pour beaucoup. Ainsi, pour d'autres tâches avec entrées d'un autre type, il serait nécessaire d'adapter à tout le moins la première couche. En général, plus d'expérimentation et d'analyse seraient nécessaires pour mesurer l'impact d'un nombre d'itérations plus grand. Une autre tâche intéressante, mais plus distante, est l'application de ce type de relaxation à des données temporelles ; la forme s'y prête assez bien, mais le débruitement deviendrait au passage une prédiction. Si les éléments des séries sont des images, ce serait alors du traitement d'animations ou vidéo.

La forme d'entraînement étudiée, c'est-à-dire avec un critère d'autoencodeur débruitant, a des limites désagréables dans le contexte de la relaxation. Notamment, cette dernière n'y est pas formellement une descente de fonction d'énergie de l'état de toutes les unités, mais est plutôt inspirée d'une forme d'inférence *mean field* d'une distribution conditionnée par l'entrée. Nous croyons qu'en tentant d'autres formes d'entraînements où l'interprétation probabiliste est plus formelle, il y aurait de bonnes chances d'obtenir de meilleurs résultats sur les tâches étudiées.

BIBLIOGRAPHIE

[1] Sven Behnke. Learning iterative image reconstruction in the neural abstraction pyramid. *International Journal of Computational Intelligence and Applications*, 1: 427–438, 2001.

[2] Sven Behnke. Learning face localization using hierarchical recurrent networks. Dans *ICANN*, pages 1319–1324, 2002.

[3] Sven Behnke. *Hierarchical Neural Networks for Image Interpretation*, volume Volume 2766 de *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, novembre 2003.

[4] Y. Bengio, P. Lamblin, P. Popovici et H. Larochelle. Greedy layer-wise training of deep networks. Dans *Advances in Neural Information Processing Systems*, volume 19. MIT Press, Cambridge, MA., 2007.

[5] Y. Bengio, P. Simard et P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994.

[6] Yoshua Bengio. Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, 2(1):1–127, 2009. Also published as a book. Now Publishers, 2009.

[7] Yoshua Bengio, Frédéric Bastien, Arnaud Bergeron, Nicolas Boulanger-Lewandowski, Thomas Breuel, Youssouf Chherawala, Moustapha Cisse, Myriam Côté, Dumitru Erhan, Jeremy Eustache, Xavier Glorot, Xavier Muller, Sylvain Pannetier Lebeuf, Razvan Pascanu, Salah Rifai, François Savard et Guillaume Sicard. Deep learners benefit more from out-of-distribution examples. Dans *JMLR W&CP : Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS 2011)*, avril 2011.

[8] Yoshua Bengio et François Gingras. Recurrent neural networks for missing or

asynchronous data. Dans *Advances in Neural Information Processing Systems 8 (NIPS'95)*, pages 395–401. MIT Press, 1996.

[9] James Bergstra. *Incorporating Complex Cells into Neural Networks for Pattern Classification*. Thèse de doctorat, Université de Montréal, juin 2011.

[10] James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley et Yoshua Bengio. Theano : a CPU and GPU math expression compiler. Dans *Proceedings of the Python for Scientific Computing Conference (SciPy)*, juin 2010. Oral.

[11] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer, 1st ed. 2006. corr. 2nd printing édition, octobre 2007. ISBN 0387310738.

[12] Jean Bullier. 6. what is fed back ? *23 Problems in Systems Neuroscience*, 1:103–133(31), 5 January 2006.

[13] Dan Claudiu Ciresan, Ueli Meier, Luca Maria Gambardella et Jürgen Schmidhuber. Deep big simple neural nets excel on handwritten digit recognition. *CoRR*, abs/1003.0358, 2010.

[14] P. R. Corlett, J. R. Taylor, X. J. Wang, P. C. Fletcher et J. H. Krystal. Toward a neurobiology of delusions. *Progress in Neurobiology*, 92(3):345–369, novembre 2010. ISSN 03010082.

[15] Dumitru Erhan, Pierre-Antoine Manzagol, Yoshua Bengio, Samy Bengio et Pascal Vincent. The difficulty of training deep architectures and the effect of unsupervised pre-training. Dans *Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics (AISTATS 2009)*, pages 153–160, avril 2009.

[16] L. Fei-Fei, R. Fergus et P. Perona. Learning generative visual models from few training examples : an incremental Bayesian approach tested on 101 object categories. Dans *Workshop on Generative-Model Based Vision*, 2004.

[17] J Fiser, B Berkes, G Orbán et M Lengyel. Statistically optimal perception and learning : from behavior to neural representations. *Trends Cogn Sci*, 14:119–130, 2010.

[18] Kunihiko Fukushima. Neocognitron : A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36:193–202, 1980. ISSN 0340-1200. 10.1007/BF00344251.

[19] Geoffrey E. Hinton, Simon Osindero et Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, 2006.

[20] Sepp Hochreiter et Juergen Schmidhuber. Long Short-Term Memory. 9(8):1735–1780, 1997.

[21] D. H. Hubel et T. N. Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *J Physiol*, 160:106–154, 1962.

[22] D. H. Hubel et T. N. Wiesel. Receptive fields and functional architecture of monkey striate cortex. *J Physiol*, 195(1):215–243, 1968.

[23] D. H. Hubel et T. N. Wiesel. Ferrier lecture. Functional architecture of macaque monkey visual cortex. *Proceedings of the Royal Society of London. Series B, Containing papers of a Biological character. Royal Society (Great Britain)*, 198(1130): 1–59, juillet 1977. ISSN 0080-4649.

[24] Herbert Jaeger. Short term memory in echo state networks. 2001.

[25] Pascal Lamblin et Yoshua Bengio. Important gains from supervised fine-tuning of deep architectures on large labeled sets. NIPS*2010 Deep Learning and Unsupervised Feature Learning Workshop, 2010.

[26] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard et L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, 1989.

[27] Y. LeCun, L. Bottou, G. Orr et K. Muller. Efficient backprop. Dans G. Orr et Muller K., éditeurs, *Neural Networks : Tricks of the trade*. Springer, 1998.

[28] Honglak Lee, Roger Grosse, Rajesh Ranganath et Andrew Y. Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. Dans *Proceedings of the 26th International Conference on Machine Learning*, pages 609–616, 2009.

[29] Tai Sing Lee et David Mumford. Hierarchical bayesian inference in the visual cortex. *Journal of The Optical Society of America A-optics Image Science and Vision*, 20, 2003.

[30] Benjamin M. Marlin, Kevin Swersky, Bo Chen et Nando de Freitas. Inductive principles for restricted boltzmann machine learning. *Journal of Machine Learning Research*, 9:509–516, 2010.

[31] David Mumford. Neuronal architectures for pattern-theoretic problems. Dans *Large-Scale Theories of the Cortex*, pages 125–152. MIT Press, 1994.

[32] Marc'Aurelio Ranzato, Christopher Poultney, Sumit Chopra et Yann LeCun. Efficient learning of sparse representations with an energy-based model. Dans J. Platt et al., éditeur, *Advances in Neural Information Processing Systems (NIPS 2006)*. MIT Press, 2006.

[33] David P. Reichert, Peggy Series et Amos J. Storkey. Hallucinations in Charles Bonnet syndrome induced by homeostasis : a deep Boltzmann machine model. 2010.

[34] F. Rosenblatt. The Perceptron : A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65:386–408, 1958.

[35] D. E. Rumelhart, G. E. Hinton et R. J. Williams. *Learning internal representations by error propagation*, pages 318–362. MIT Press, Cambridge, MA, USA, 1986. ISBN 0-262-68053-X.

[36] David E. Rumelhart, Geoff E. Hinton et R. J. Wilson. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

[37] Ruslan Salakhutdinov et Geoffrey Hinton. Deep Boltzmann machines. Dans *Proceedings of the International Conference on Artificial Intelligence and Statistics*, volume 5, pages 448–455, 2009.

[38] Ruslan Salakhutdinov et Hugo Larochelle. Efficient learning of deep boltzmann machines. *Journal of Machine Learning Research - Proceedings Track*, 9:693–700, 2010.

[39] P. Smolensky. *Information processing in dynamical systems : foundations of harmony theory*, pages 194–281. MIT Press, Cambridge, MA, USA, 1986. ISBN 0-262-68053-X.

[40] Vladimir N. Vapnik. *The nature of statistical learning theory*. Springer-Verlag New York, Inc., New York, NY, USA, 1995. ISBN 0-387-94559-8.

[41] Pascal Vincent, Hugo Larochelle, Yoshua Bengio et Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. Dans *Proceedings of the Twenty-fifth International Conference on Machine Learning (ICML'08)*, pages 1096–1103. ACM, 2008.