Université de Montréal

Étude de techniques d'apprentissage non-supervisé pour l'amélioration de l'entraînement supervisé de modèles connexionnistes

par
Hugo Larochelle

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

Thèse présentée à la Faculté des études supérieures et postdoctorales
en vue de l'obtention du grade de Philosophiæ Doctor (Ph.D.)
en informatique

Décembre, 2008

Université de Montréal
Faculté des études supérieures et postdoctorales

Cette thèse intitulée :

Étude de techniques d'apprentissage non-supervisé pour l'amélioration de l'entraînement supervisé de modèles connexionnistes

présentée par :

Hugo Larochelle

a été évaluée par un jury composé des personnes suivantes :

Pascal Vincent,	président-rapporteur
Yoshua Bengio,	directeur de recherche
Douglas Eck,	membre du jury
Geoffrey Hinton,	examinateur externe
Nathalie Loye,	représentant du doyen de la FESP

Thèse acceptée le : 18 mars 2009

RÉSUMÉ

Le domaine de l'intelligence artificielle a pour objectif le développement de systèmes informatiques capables de simuler des comportements normalement associés à l'intelligence humaine. On aimerait entre autres pouvoir construire une machine qui puisse résoudre des tâches liées à la vision (e.g., la reconnaissance d'objet), au traitement de la langue (e.g., l'identification du sujet d'un texte) ou au traitement de signaux sonores (e.g., la reconnaissance de la parole).

Une approche développée afin de résoudre ce genre de tâches est basée sur l'apprentissage automatique de modèles à partir de données étiquetées reflétant le comportement intelligent à émuler. Entre autre, il a été proposé de modéliser le calcul nécessaire à la résolution d'une tâche à l'aide d'un réseau de neurones artificiel, dont il est possible d'adapter le comportement à l'aide de la rétropropagation [99, 131] d'un gradient informatif sur les erreurs commises par le réseau. Populaire durant les années 80, cette approche spécifique a depuis perdu partiellement de son attrait, suite au développement des méthodes à noyau. Celles-ci sont souvent plus stables, plus faciles à utiliser et leur performance est souvent au moins aussi élevée pour une vaste gamme de problèmes.

Les méthodes d'apprentissage automatique ont donc progressé dans leur fonctionnement, mais aussi dans la complexité des problèmes auxquels elles se sont attaqué. Ainsi, plus récemment, des travaux [12, 15] ont commencé à émettre des doutes sur la capacité des machines à noyau à pouvoir efficacement résoudre des problèmes de la complexité requise par l'intelligence artificielle. Parallèlement, Hinton et al. [81] faisaient une percée dans l'apprentissage automatique de réseaux de neurones, en proposant une procédure permettant l'entraînement de réseaux de neurones d'une plus grande complexité (i.e., avec plus de couches de neurones cachées) qu'il n'était possible auparavant.

C'est dans ce contexte qu'ont été conduits les travaux de cette thèse. Cette thèse débute par une exposition des principes de base de l'apprentissage automatique (chapitre 1) et une discussion des obstacles à l'obtention d'un modèle ayant une bonne performance de généralisation (chapitre 2). Puis, sont présentées les contributions apportées dans le cadre de cinq articles, contributions qui sont toutes basées sur l'utilisation d'une certaine

forme d'apprentissage non-supervisé.

Le premier article (chapitre 4) propose une méthode d'entraînement pour un type spécifique de réseau à une seule couche cachée (la machine de Boltzmann restreinte) basée sur une combinaison des apprentissages supervisé et non-supervisé. Cette méthode permet d'obtenir une meilleure performance de généralisation qu'un réseau de neurones standard ou qu'une machine à vecteurs de support à noyau, et met en évidence de façon explicite les bénéfices qu'apporte l'apprentissage non-supervisé à l'entraînement d'un réseau de neurones.

Ensuite, dans le second article (chapitre 6), on étudie et étend la procédure d'entraînement proposée par Hinton et al. [81]. Plus spécifiquement, on y propose une approche différente mais plus flexible pour initialiser un réseau à plusieurs couches cachées, basée sur un réseau autoassociateur. On y explore aussi l'impact du nombre de couches et de neurones par couche sur la performance d'un réseau et on y décrit différentes variantes mieux adaptées à l'apprentissage en ligne ou pour données à valeurs continues.

Dans le troisième article (chapitre 8), on explore plutôt la performance de réseaux profonds sur plusieurs problèmes de classification différents. Les problèmes choisis ont la propriété d'avoir été générés à partir de plusieurs facteurs de variation. Cette propriété, qui caractérise les problèmes liés à l'intelligence artificielle, pose difficulté aux machines à noyau, tel que confirmé par les expériences de cet article.

Le quatrième article (chapitre 10) présente une amélioration de l'approche basée sur les réseaux autoassociateurs. Cette amélioration applique une modification simple à la procédure d'entraînement d'un réseau autoassociateur, en « bruitant » les entrées du réseau afin que celui-ci soit forcé à la débruiter.

Le cinquième et dernier article (chapitre 12) apporte une autre amélioration aux réseaux autoassociateurs, en permettant des interactions d'inhibition ou d'excitation entre les neurones cachés de ces réseaux. On y démontre que de telles interactions peuvent être apprises et sont bénéfiques à la performance d'un réseau profond.

Mots-clés : apprentissage non-supervisé, réseau de neurones artificiel, machine de Boltzmann restreinte, autoassociateur, autoencodeur, architecture profonde

ABSTRACT

The objective of the field of artificial intelligence is the development of computer systems capable of simulating a behavior reminiscent of human intelligence. In particular, we would like to build a machine that would be able to solve tasks related to vision (e.g., object recognition), natural language (e.g., topic classification) or signal processing (e.g., speech recognition).

The general approach developed in the sub-field of machine learning to solve such tasks is based on using labeled data to train a model to emulate the desired behavior. One such model that has been proposed is the artificial neural network, which can adapt its behavior using a backpropagated gradient [99, 131] that is informative of the errors made by the network. Popular during the 80's, this specific approach has since lost some of its appeal, following the development of kernel methods. Indeed, kernel methods are often found to be more stable, easier to use, and their performance usually compares favorably on a vast range of problems.

Since the foundation of the field, machine learning methods have progressed not only in their inner workings, but also in the complexity of problems they can tackle. More recently however, it has been argued [12, 15] that kernel methods might not be able to solve efficiently enough problems of the complexity that is expected from artificial intelligence. At the same time, Hinton et al. [81] put forth a breakthrough in neural network training, by developing a procedure able to train more complex neural networks (i.e., with more layers of hidden neurons) than previously possible.

This is the context in which the work presented in this thesis started. This thesis begins with the introduction of the basic principles of machine learning (Chapter 1) as well as the known obstacles to achieve good generalization performance (Chapter 2). Then, the work from five papers is presented, with each of these papers' contribution relying on a form of unsupervised learning.

The first paper (Chapter 4) presents a training method for a specific form of single hidden layer neural network (the Restricted Boltzmann Machine), based on the combination of supervised and unsupervised learning. This method achieves a better general-

ization performance than a standard neural network and a kernel support vector machine. This observation emphasizes the beneficial effect of unsupervised learning for training neural networks.

Then, the second paper (Chapter 6) studies and extends the training procedure of Hinton et al. [81]. More specifically, we propose a different but more flexible approach for initializing a deep (i.e., with many hidden layers) neural network, based on autoassociator networks. We also empirically analyze the impact of varying the number of layers and number of hidden neurons on the performance of a neural network, and we describe variants of the same training procedure that are more appropriate for continuous-valued inputs and online learning.

The third paper (Chapter 8) describes a more intensive empirical evaluation of training algorithms for deep networks, on several classification problems. These problems have been generated based on several factors of variations, in order to simulate a property that is expected from artificial intelligence problems. The experiments presented in this paper tend to show that deep networks are more appropriate than shallow models, such as kernel methods.

The fourth paper (Chapter 10) develops an improved variation of the autoassociator network. This simple variation, which brings better generalization performance to deep networks, modifies the autoassociator network's training procedure by corrupting its input and forcing the network to denoise it.

The fifth and final paper (Chapter 12) contributes another improved variation on autoassociator networks, by allowing inhibitory/facilitatory interactions between the hidden layer neurons. We show that such interactions can be learned and can be beneficial to the performance of deep networks.

Keywords: unsupervised learning, neural network, Restricted Boltzmann Machine, autoassociator, autoencoder, deep architecture, deep learning

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

REMERCIEMENTS

Je remercie Yoshua Bengio pour m'avoir enseigné l'importance de la curiosité, de l'audace et de la persévérance dans la recherche scientifique. C'est grâce à ces trois valeurs et à ton soutien indéfectible que j'ai pu me rendre aussi loin comme chercheur.

De façon générale, je tiens aussi à remercier tous mes collègues que j'ai côtoyés aux laboratoires LISA et GAMME. Parfois peut-être inconsciemment, vous m'avez aidé à pousser mes propres limites afin que je puisse devenir non seulement un meilleur chercheur, mais aussi une meilleure personne.

De plus, je remercie mes parents pour m'avoir encouragé et soutenu dans ma décision de continuer mes études au doctorat. Vous savoir derrière moi tout ce temps fut essentiel.

Finalement, j'aimerais remercier ma femme, Angèle, qui plus que quiconque a su me soutenir lors des moments plus difficiles où le moral était moins bon.

À ma petite famille, Angèle et Mélie...

CHAPITRE 1

INTRODUCTION À L'APPRENTISSAGE AUTOMATIQUE

L'**apprentissage automatique** correspond au domaine se consacrant au développement d'algorithmes permettant à une machine d'apprendre à partir d'un ensemble de données, i.e., d'y extraire des concepts et patrons caractérisant ces données. Bien que la motivation originale de ce domaine était de permettre la mise sur pied de systèmes manifestant une **intelligence artificielle**, les algorithmes issus de ce domaine sont maintenant répandus dans bien d'autres domaines, tels la bioinformatique, la finance, la recherche d'information et le forage de données.

Plus spécifiquement, on peut définir un algorithme d'apprentissage comme suit :

Définition. *Un algorithme d'apprentissage est un algorithme prenant en entrée un ensemble de données $\mathcal{D}$ et retournant une fonction f.*

On désigne alors $\mathcal{D}$ comme **ensemble d'entraînement** ou ensemble d'apprentissage, et la fonction f comme **modèle**. Suite à l'exécution d'un algorithme d'apprentissage, on dira que le modèle f a été entraîné sur l'ensemble $\mathcal{D}$.

On considère que l'ensemble de données $\mathcal{D}$ contient sous forme de vecteurs l'information nécessaire pour résoudre un problème donné. Le modèle f doit ainsi pouvoir recevoir de tels vecteurs en argument et on désignera par **sortie** la valeur de l'application de f sur un vecteur. L'utilisation de vecteurs n'exclut pas la possibilité de travailler avec des données plus complexes, telles des graphes.

De plus, on considère normalement que tous les éléments de $\mathcal{D}$, que l'on nomme **exemples d'apprentissage**, sont indépendants et identiquement distribués. On désigne cette hypothèse par **hypothèse I.I.D.** Cette hypothèse n'exclut pas complètement le cas de données séquentielles, puisque les vecteurs contenus dans $\mathcal{D}$ peuvent être de tailles variables et ainsi correspondre à des séquences. Les séquences elles-mêmes doivent par contre être indépendantes entre elles.

Il est à noter que le choix de la définition d'apprentissage automatique fait ici n'a pas comme objectif d'être parfaitement général. Effectivement, sa simplicité ne permet pas

de bien couvrir certains types d'apprentissage, tel l'apprentissage par renforcement. Cependant, ce choix spécifique permet de traiter plus directement des types d'apprentissage mis en cause dans les travaux de cette thèse, et ainsi devrait faciliter leur compréhension par le lecteur.

1.1 Les types d'apprentissage automatique

Il existe plusieurs types différents d'apprentissage automatique, qui se distinguent essentiellement par leur objectif, i.e., la nature de ce qui doit être appris. Bien qu'ils puissent trouver application dans des contextes différents, ces types d'apprentissage peuvent aussi être combinés dans un même système. Dans le cadre de cette thèse, on parlera surtout de l'**apprentissage supervisé** et de l'**apprentissage non-supervisé**, et on abordera brièvement l'**apprentissage par renforcement**.

1.1.1 Apprentissage supervisé

L'apprentissage supervisé correspond au cas où l'objectif de l'apprentissage est déterminé explicitement via la définition d'une cible à prédire. Dans ce cas, $\mathcal{D}$ correspond à un ensemble de n paires d'**entrées** $\mathbf{x}_t$ et de **cibles** associées y_t :

$$\mathcal{D} = \{(\mathbf{x}_t, y_t) | \mathbf{x}_t \in \mathcal{X}, y_t \in \mathcal{Y}\}_{t=1}^{n} .$$

Typiquement, un tel ensemble est récolté en fournissant l'ensemble des entrées à un groupe de personnes et en leur demandant d'associer à chacune de ces entrées une cible appropriée dans le contexte du problème à résoudre. La tâche d'un algorithme d'apprentissage est alors d'entraîner un modèle qui puisse imiter ce processus d'étiquetage par un humain, i.e., qui puisse prédire pour une entrée $\mathbf{x}$ quelconque la valeur de la cible y qui aurait normalement été donnée par un humain. Cependant, les algorithmes d'apprentissage ne se limitent pas à la modélisation du comportement de l'humain, et peuvent être utilisés pour modéliser la relation liant des paires d'entrées et de cibles provenant d'un autre phénomène (e.g., la relation entre une action et son prix à la bourse telle que

générée par les marchés financiers).

La nature de l'ensemble $\mathcal{Y}$ d'où proviennent les cibles dépendra du type de problème à résoudre. Deux types de problèmes fréquents sont les problèmes de **classification** et de **régression**.

Classification

Dans le cadre d'un problème de classification, $\mathcal{Y}$ correspond à un ensemble fini de classes auxquelles peuvent appartenir les différentes entrées possibles $\mathbf{x} \in \mathcal{X}$. Dans le cas où $\mathbf{x}$ est une séquence, $\mathcal{Y}$ peut aussi être un ensemble de séquences de classes. Voici d'ailleurs quelques exemples de problèmes de classification :

Reconnaissance de caractères : $\mathbf{x} \in \mathcal{X}$ correspond à la représentation vectorielle de la valeur des pixels d'une image et $y \in \mathcal{Y}$ au caractère associé (par exemple, un chiffre entre 0 et 9).

Classification syntaxique : $\mathbf{x} \in \mathcal{X}$ correspond à une phrase dans une langue donnée et $\mathbf{y} \in \mathcal{Y}$ est la séquence des classes syntaxiques (ou *part of speech tags*) de chacun des mots de $\mathbf{x}$.

La figure 1.1 présente une illustration de problèmes de classification en 2 dimensions. Afin de mesurer la performance d'un modèle de classification, on utilise normalement le taux d'erreur de classification, i.e., la proportion du nombre de cas où la sortie donnée par le modèle ne correspond pas à la cible attendue. Lorsque le modèle f donne une probabilité pour chaque classe possible, la log-vraisemblance négative de la classe cible peut aussi être utilisée. Comme on le verra plus tard, cette mesure de performance a l'avantage d'être dérivable par rapport à la sortie du modèle, et peut ainsi être optimisée à l'aide d'un algorithme de descente de gradient.

Régression

Dans un problème de régression, $\mathcal{Y}$ correspond à un ensemble de valeurs continues ou de vecteurs de valeurs continues. Voici quelques exemples de tels problèmes :

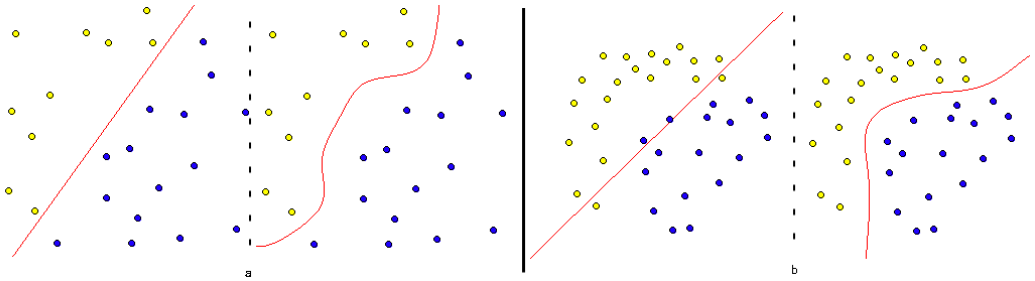


Figure 1.1 – Illustration de deux problèmes de classification en 2 dimensions, un linéairement séparable (à gauche), l'autre pas (à droite). Pour chaque problème, un classifieur linéaire et un classifieur non-linéaire ont été entraînés à partir d'exemples d'entraînement séparés en deux classes (en jaune et en bleu). La **surface de décision**, déterminant la limite entre la région associée à chaque classe par un classifieur, est affichée en rouge.

Prédiction du poids : $\mathbf{x} \in \mathcal{X}$ correspond à des caractéristiques physiques d'une personne (par exemple son âge, son sexe, etc.) et $y \in \mathcal{Y}$ correspond à son poids.

Prédiction de la valeur d'une action : $\mathbf{x} \in \mathcal{X}$ correspond à une séquence d'indicateurs du marché pour différentes journées et $\mathbf{y} \in \mathcal{Y}$ correspond à la progression de la valeur d'une action.

La figure 1.2 présente un exemple de problème de régression en une dimension. La performance d'un modèle de régression est typiquement mesurée en calculant la différence au carré entre la sortie du modèle et la cible attendue, mais bien d'autres mesures sont possibles.

1.1.2 Apprentissage non-supervisé

L'apprentissage non-supervisé correspond au cas où aucune cible n'est prédéterminée. Ainsi, l'ensemble d'entraînement ne contient que des entrées :

$$\mathcal{D} = \{\mathbf{x}_t | \mathbf{x}_t \in \mathcal{X}\}_{t=1}^{n}$$

et ne définit pas explicitement la nature de la fonction f qui doit être retournée par l'algorithme d'apprentissage. Ainsi, c'est plutôt l'utilisateur qui doit spécifier le problème à résoudre. Par contre, dans tous les cas, le modèle capture certains éléments de la vé-

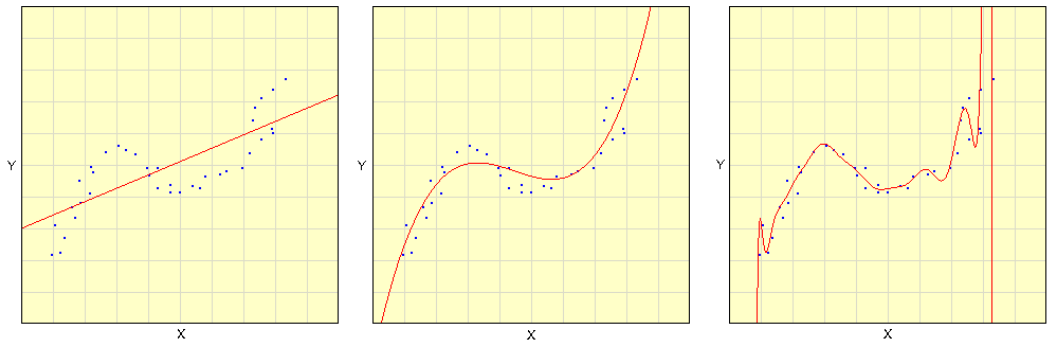


Figure 1.2 – Illustration d'un problème de régression en une dimension, où la cible y prend une valeur réelle. De gauche à droite, la prédiction est produite par un modèle linéaire, quadratique et polynomial de degré 20.

ritable distribution ayant généré $\mathcal{D}$. Voici un bref survol des problèmes d'apprentissage non-supervisé les plus répandus.

Estimation de densité

Pour ce problème, f doit fournir une estimation de la fonction de densité ou de probabilité de la distribution ayant généré les éléments de $\mathcal{D}$. Afin de résoudre ce problème, une approche possible pour un algorithme d'apprentissage consiste à spécifier une forme précise de fonction de densité p valide pour le modèle f, puis d'ajuster les paramètres de la fonction de densité afin d'avoir une valeur élevée de $f(\mathbf{x}_t) = p(\mathbf{x}_t)$ pour tous les exemples de $\mathcal{D}$. Afin d'être une fonction de densité valide, la valeur de $p(\mathbf{x})$ doit toujours être non-négative et doit intégrer à 1 sur tout l'ensemble $\mathcal{X}$.

Ainsi, après entraînement, le modèle contient alors de l'information importante sur la structure statistique des entrées. Par exemple, lorsqu'un des éléments x_i d'une entrée est manquant, il est possible d'utiliser un tel modèle d'estimation de densité afin de trouver une valeur vraisemblable pour cet élément en utilisant le fait que

$$p(x_i|x_1, \ldots, x_{i-1}, \ldots, x_{i+1}, x_d) = \frac{p(\mathbf{x})}{\sum_{x_i^* \in \mathcal{X}_i} p(x_1, \ldots, x_i^*, \ldots, x_d)} ,$$

où $\mathcal{X}_i$ est l'ensemble des valeurs que peut prendre x_i. Il est même possible d'entraî-

ner un modèle d'estimation de densité explicitement selon ce critère, appelé la **pseudo-vraisemblance** [19], mesurant la qualité de la prédiction faite par un modèle de chaque entrée étant données les autres. Ce critère est d'ailleurs plus approprié dans une situation où prédire une entrée étant données les autres est effectivement le type d'opération que l'on souhaite faire avec le modèle. Un modèle de densité peut aussi être utilisé afin de "purifier" une entrée qui a été corrompue par du bruit, puisque le modèle de densité contient de l'information sur les configurations vraisemblables d'entrées non-bruitées. Encore une fois, il existe un critère d'entraînement, appelé *Score Matching* [**? ?**], dont l'utilisation est alors plus appropriée pour cette tâche. Comme on le verra plus tard, ces dernières applications ne forment que deux exemples parmi tant d'autres d'utilisation d'un modèle d'estimation de densité afin de raisonner de façon probabiliste par rapport aux éléments d'une entrée et de sa structure statistique.

Finalement, à partir de la densité apprise, il est normalement possible de générer de nouveaux points selon cette densité, ce qui peut être utile dans certaines applications, par exemple en simulation, ou simplement pour mieux comprendre ce qui a été appris par le modèle.

Extraction de caractéristiques

L'objectif de cette tâche est d'apprendre une nouvelle représentation $f(\mathbf{x})$ de l'entrée qui soit plus utile que la représentation vectorielle originale $\mathbf{x}$. Cette représentation prend généralement la forme d'un vecteur, dont les éléments (appelés « caractéristiques » ou *features*) sont calculés par le modèle f. En général, l'utilité de cette représentation dépend de la tâche que l'on souhaite réellement résoudre. En d'autres termes, l'extraction de caractéristiques n'est normalement pas une fin en soi, mais bien un moyen d'améliorer un autre algorithme d'apprentissage dont le modèle f^* utiliserait alors cette nouvelle représentation et serait entraîné sur le nouvel ensemble de données $\widehat{\mathcal{D}} = \{(f(\mathbf{x}_t), y_t\}_{t=1}^n$.

La qualité de l'extraction de caractéristiques peut être mesurée de différentes façons, en fonction du contexte de l'application. Le critère le plus souvent considéré est l'amélioration de la performance de cet autre algorithme. On souhaite alors que la prédiction $f^*(f(\mathbf{x}))$ d'un modèle entraîné à partir des caractéristiques extraites soit meilleure que

celle d'un modèle entraîné à partir de la représentation des entrées originale. Un autre critère est le gain en efficacité de la prédiction $f^*(f(\mathbf{x}))$. Ce gain peut être dû à la réduction du calcul nécessaire pour faire une prédiction ou à la réduction du coût réel du calcul, dans le cas où un coût (e.g., en argent) est associé à l'obtention des éléments de l'entrée.

Il existe plusieurs approches à l'extraction de caractéristiques. Une technique simple consiste à simplement sélectionner les éléments de $\mathbf{x}$ les plus utiles. Cette approche est appelée la **sélection de caractéristiques** [65]. Formellement, on a alors :

$$f(\mathbf{x}) = (x_{s_1}, x_{s_2}, \ldots, x_{s_r})$$

où $\mathbf{s}$ est le vecteur des indices d'éléments de $\mathbf{x}$ sélectionnés. Les techniques de sélection de caractéristiques peuvent aussi servir à la création de nouvelles caractéristiques. Effectivement, en déterminant une façon automatique de générer un large ensemble de caractéristiques candidates à partir du vecteur d'entrée $\mathbf{x}$ (e.g., en considérant comme caractéristique candidate le résultat de la multiplication d'un sous-ensemble des éléments de $\mathbf{x}$, pour tout sous-ensemble possible), on peut alors appliquer un algorithme de sélection de caractéristiques sur ce nouvel ensemble.

Il est aussi souvent possible de formuler le problème d'extraction de caractéristiques à partir d'un problème d'estimation de densité. En choisissant une fonction de densité utilisant un vecteur de variables latentes $\mathbf{h}$ telle que

$$p(\mathbf{x}) = \sum_{\mathbf{h} \in \mathcal{H}} p(\mathbf{x}, \mathbf{h})$$

où $\mathcal{H}$ est l'ensemble des valeurs que peut prendre $\mathbf{h}$, il est effectivement possible d'utiliser la distribution conditionnelle des variables latentes $p(\mathbf{h}|\mathbf{x})$ afin d'extraire de nouvelles caractéristiques. Par exemple, dans le cadre d'une **machine de Boltzmann restreinte** ou *Restricted Boltzmann Machine* , (voir annexe B du chapitre 6 pour plus de détails) dont les éléments de $\mathbf{h}$ sont binaires, on utilise normalement comme caractéris-

tiques extraites le vecteur des probabilités que chaque variable latente soit égale à 1 :

$$f(\mathbf{x}) = (p(h_1 = 1|\mathbf{x}), \ldots, p(h_H = 1|\mathbf{x})) \ .$$

Dans le cas du **codage par représentation creuse** ou *sparse coding* [116], les éléments de $\mathbf{h}$ prennent une valeur non-négative réelle (dans lequel cas le calcul de $p(\mathbf{x})$ nécessite une intégrale sur $\mathcal{H}$) et on utilise normalement le mode de $p(\mathbf{h}|\mathbf{x})$ comme nouvelle représentation de $\mathbf{x}$ [120] :

$$f(\mathbf{x}) = \underset{\mathbf{h} \in \mathcal{H}}{\operatorname{argmax}}\, p(\mathbf{h}|\mathbf{x}) \ .$$

En utilisant une approche par estimation de densité, le critère dirigeant la création des caractéristiques est donc implicitement défini par le choix du modèle d'estimation de densité.

Réduction de dimensionnalité

Pour cette tâche, f doit associer à un vecteur d'entrée $\mathbf{x}$ une représentation $f(\mathbf{x})$ de plus petite dimensionnalité que $\mathbf{x}$ mais conservant l'essentiel de l'information contenue dans l'entrée. Ainsi, cette tâche peut directement servir à l'extraction de caractéristiques, une telle réduction de la dimensionnalité des entrées pouvant potentiellement améliorer la performance d'un algorithme d'apprentissage supervisé. Cependant, elle peut aussi servir à visualiser des données de $\mathcal{D}$ en 2 ou 3 dimensions.

Plusieurs algorithmes ont été proposés à ce jour, dont les plus populaires sont *Principal Component Analysis* (PCA) [91], *Multidimensional Scaling* (MDS) [41], *Locally Linear Embedding* (LLE) [130], *Isomap* [148], *Semidefinite Embedding* (SDE) [158] et, plus récemment, *t-Distributed Stochastic Neighbor Embedding* (t-SNE) [151]. En général, ces algorithmes cherchent une transformation f préservant le mieux possible une notion de similarité entre les vecteurs d'entrée de $\mathcal{D}$ ou une caractéristique de $\mathcal{D}$, spécifique à l'algorithme (par exemple, la variance des entrées de $\mathcal{D}$ dans le cas de PCA ou les coordonnées locales des voisins de chaque vecteur d'entrée dans le cas de LLE).

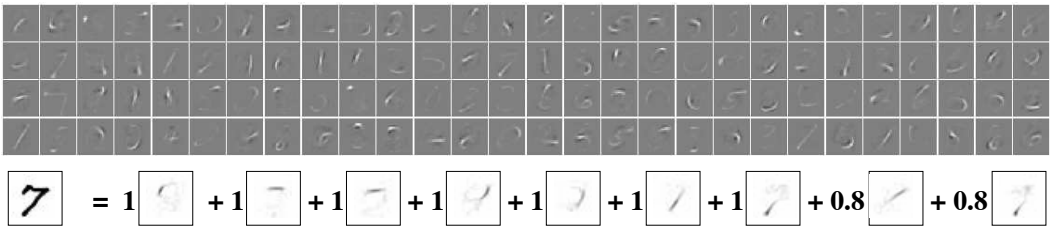


Figure 1.3 – Exemple d'extraction de caractéristiques à l'aide du codage par représentation creuse, générée par [125]. En haut, illustration des caractéristiques extraites pour des images de chiffres, ressemblant à des traits de crayon (en blanc). En bas, illustration de la représentation creuse pouvant être obtenue à partir des caractéristiques extraites (illustrées plutôt en noir). On observe que l'image d'un sept peut alors être représentée par seulement 9 coefficients non-nuls, plutôt que par la représentation par pixels originale.

Ces algorithmes supposent normalement que les entrées sont situées aux alentours d'un sous-espace, de plus petite dimension que l'espace original, appelé **variété**. La dimensionnalité présumée de cette variété est un paramètre qui doit normalement être précisé par l'utilisateur. Il existe d'ailleurs tout un pan de recherche qui se consacre au développement d'algorithmes pouvant déterminer la dimensionnalité intrinsèque des données de $\mathcal{D}$. La figure 1.4 présente un exemple de réduction de dimensionnalité en 2 dimensions.

Groupage

Le problème du groupage (*clustering*) [109] correspond à la recherche d'un partitionnement de l'espace $\mathcal{X}$ en g sous-ensembles (ou groupes) $\mathcal{G}_1, \ldots, \mathcal{G}_g$. Chacun de ces sous-ensembles $\mathcal{G}_i$ peut facultativement être associé à un prototype $\widetilde{\mathbf{x}}_i$ qui résume bien les données de $\mathcal{D}$ contenues dans $\mathcal{G}_i$. Ainsi, f doit donner, pour chaque vecteur d'entrée, l'indice i du sous-ensemble $\mathcal{G}_i$ auquel il appartient. Dans le domaine du forage de données, le groupage est souvent utilisé comme outil de visualisation d'un ensemble de données, dans le but d'y découvrir une possible structure de groupe latente. Le groupage peut aussi être utilisé afin de compresser un ensemble de données en le remplaçant par l'ensemble des prototypes. Il est même possible d'extraire des caractéristiques à l'aide du groupage, par exemple en considérant le vecteur binaire d'association aux différents

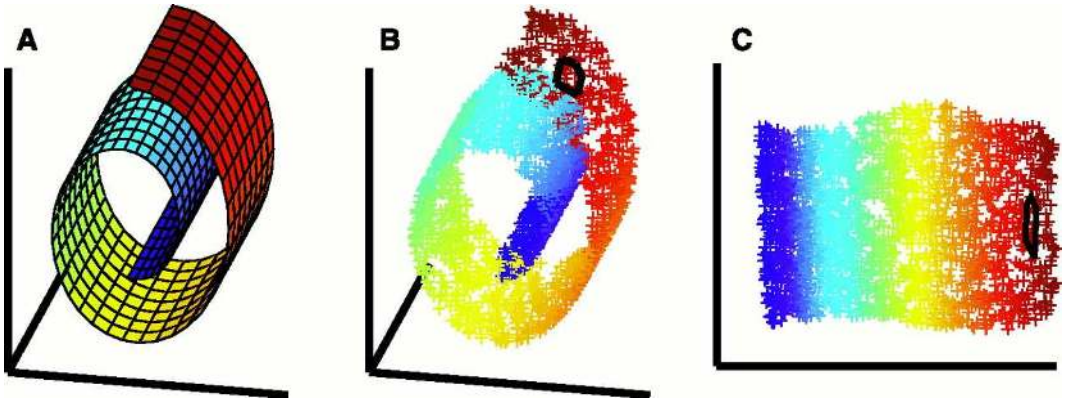


Figure 1.4 – Exemple de réduction de dimensionnalité provenant de [130], obtenu par l'algorithme LLE. Cet algorithme permet d'associer chaque entrée d'un ensemble prédéterminé (B) à une entrée de plus petite dimension (C). Les entrées sont présumées être situées autour d'une variété de plus petite dimension, qui ici prend la forme d'une spirale (A).

sous-ensembles $\left(1_{f(\mathbf{x})=1}, \ldots, 1_{f(\mathbf{x})=g}\right)$ comme vecteur de caractéristiques.

L'algorithme de groupage le plus simple et le plus répandu est certainement l'algorithme des K moyennes [106, 145] (*K-means algorithm*). Il s'agit d'un algorithme itératif, qui débute avec une initialisation aléatoire de la position des prototypes dans $\mathcal{X}$, puis qui alterne entre (1) la réassignation de chaque vecteur d'entrée de $\mathcal{D}$ au prototype le plus proche selon la distance euclidienne et (2) la mise à jour des prototypes à la valeur de la moyenne des vecteurs d'entrée associés au même prototype. La figure 1.5 montre quelques étapes de l'exécution de cet algorithme sur un ensemble d'entrées en 2 dimensions.

De façon similaire à l'extraction de caractéristiques, le groupage peut aussi être formulé à partir d'un problème d'estimation de densité. En supposant un modèle de densité utilisant des variables latentes $\mathbf{h}$ pouvant prendre un nombre fini de valeurs, puis en indexant ces valeurs de 1 à $|\mathcal{H}|$, il est possible de dériver un algorithme de groupage simplement en associant à toute entrée $\mathbf{x}$ l'indice de la valeur de $\mathbf{h}$ la plus probable selon $p(\mathbf{h}|\mathbf{x})$. L'algorithme des K moyennes peut d'ailleurs être dérivé de cette manière [21].

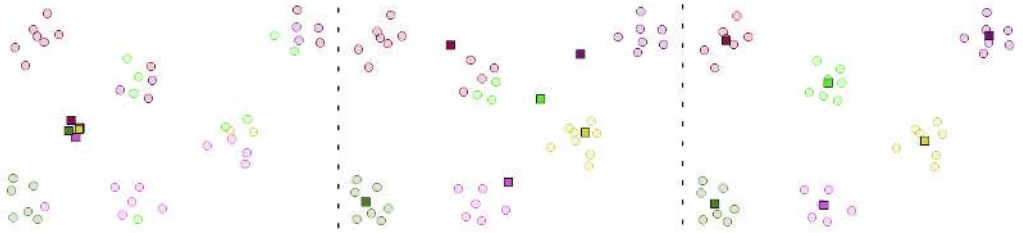


Figure 1.5 – Exemple de groupage obtenu à l'aide de l'algorithme des K moyennes. Les cercles représentent les entrées et les carrés représentent les prototypes. De gauche à droite, l'état de l'algorithme est illustré à l'initialisation, après une étape et après 3 étapes.

1.1.3 Apprentissage par renforcement

L'apprentissage par renforcement a comme objectif d'entraîner un agent à se comporter de façon intelligente dans un environnement donné. Un agent interagit avec l'environnement en choisissant, à chaque temps donné, d'exécuter une action parmi un ensemble d'actions permises. Le comportement intelligent que doit apprendre cet agent est donné implicitement via un signal de renforcement qui, après chaque décision de l'agent, indique s'il a bien ou mal agi. L'agent doit donc se baser sur ce signal afin d'améliorer son comportement, qui est dicté par sa politique d'actions. À chaque temps donné, l'agent a normalement à sa disposition un ensemble de caractéristiques ou indicateurs d'entrée, décrivant l'environnement. Par exemple, si l'agent correspond à un robot, ces indicateurs pourraient être obtenus à partir de capteurs sensoriels brossant un portrait de l'endroit où il se trouve.

Les concepts d'action et de signal de renforcement sont probablement ceux qui distinguent le plus l'apprentissage par renforcement des apprentissages supervisé et non-supervisé. Contrairement à l'apprentissage supervisé, le comportement intelligent à apprendre n'est pas explicitement donné par une cible à prédire mais doit plutôt être défini par l'utilisateur à l'aide d'un signal de renforcement. Cependant, contrairement à l'apprentissage non-supervisé dont l'objectif est de modéliser la structure statistique des entrées, l'apprentissage par renforcement s'intéresse à la notion d'action qui est différente de celle l'entrée, de la même façon que le concept de cible de l'apprentissage supervisé

est différent du concept d'entrée. Ceci étant dit, certains concepts de l'apprentissage par renforcement peuvent être utiles dans un contexte d'apprentissage supervisé, comme le démontre l'algorithme SEARN de **?**], un algorithme de classification pour cibles structurées qui utilise la notion de politique d'actions.

Bien que l'apprentissage par renforcement soit un type d'apprentissage très étudié dans la littérature et important pour le développement d'un système d'intelligence artificielle, il ne sera pas abordé dans la suite de cette thèse, n'étant pas mis en cause par les travaux qui y sont présentés.

1.2 Principes de l'apprentissage automatique

Deux algorithmes d'apprentissage automatique ne se distinguent pas uniquement par le type d'apprentissage qu'ils implémentent. Ils peuvent aussi se distinguer dans la façon qu'ils accomplissent cet apprentissage. Cette section passe en revue ainsi certains des différents principes et concepts généraux qui caractérisent les algorithmes d'apprentissage et qui seront utiles à leur description et compréhension.

1.2.1 Apprentissage par minimisation du risque empirique versus bayésien

Il existe principalement deux philosophies qui régissent la conception des algorithmes d'apprentissage que l'on retrouve dans la littérature, soit l'**apprentissage par la minimisation du risque empirique** (MRE) et l'**apprentissage bayésien**. Par simplicité, je traiterai surtout de l'apprentissage supervisé, bien qu'un raisonnement similaire peut être facilement appliqué dans le cas non-supervisé.

Apprentissage par minimisation du risque empirique

Étant donné un modèle f_Θ, où Θ correspond aux paramètres de ce modèle, et L un coût à minimiser, l'apprentissage par minimisation du risque empirique correspond à

l'opération suivante[1] :

$$\widehat{\Theta} = \underset{\Theta}{\operatorname{argmin}}\, R(\Theta, \mathcal{D}) = \underset{\Theta}{\operatorname{argmin}}\, \frac{1}{n} \sum_{(\mathbf{x}_t, y_t) \in \mathcal{D}} L(f_\Theta(\mathbf{x}_t), y_t) \tag{1.1}$$

où $R(\Theta, \mathcal{D})$ correspond au **risque empirique** et $L(f_\Theta(\mathbf{x}_t), y_t))$ est la fonction de coût associée au risque. On ajoute aussi parfois certaines contraintes sur la valeur que peuvent prendre les paramètres Θ lors de l'optimisation. Pour un modèle f_Θ choisi sans avoir observé l'ensemble Θ, $R(\Theta, \mathcal{D})$ est alors un estimateur non-biaisé du **risque réel**, soit l'espérance $E\left[L(\mathbf{x}, y)\right]$ du coût sous la distribution ayant généré les paires $(\mathbf{x}_t, y_t)$ de $\mathcal{D}$. Il est aussi important de distinguer le risque, qui est le coût optimisé par la procédure d'optimisation, et l'erreur, qui est le coût que l'on souhaite réellement minimiser. Par exemple, dans un problème de classification, l'erreur d'un modèle serait la proportion de ses prédictions erronées de la classe des entrées. Cependant, puisque cette erreur est difficile à optimiser (entre autres parce qu'elle n'est pas dérivable), on se rabat plutôt sur d'autres coûts pour le risque, tels la log-vraisemblance négative.

Ainsi, un algorithme d'apprentissage doit alors préciser la nature de f_Θ et une façon d'accomplir (ou au moins approximer) l'opération de minimisation. Par exemple, f_Θ pourrait correspondre à un **modèle linéaire** (voir section 1.3.1) ou à un **réseau de neurones artificiel** (voir section 1.3.2), et la procédure de minimisation pourrait être une procédure de descente de gradient. C'est la procédure de minimisation qui procède à l'entraînement du modèle f_Θ. Il est d'ailleurs possible que la minimisation ne soit pas complète ou encore que $\widehat{\Theta}$ corresponde à un **minimum local** plutôt qu'à un **minimum global** du risque.

Il existe plusieurs algorithmes différents de descente de gradient. Ceux-ci se distinguent entre autres par la fréquence des mises à jour des paramètres (après un parcours complet de l'ensemble $\mathcal{D}$ ou après le traitement de chaque exemple d'entraînement) ainsi que par l'utilisation (ou pas) de l'information de second ordre sur la relation entre le risque et la valeur des paramètres. Dans cette thèse, l'algorithme de **descente de gra-**

[1] Dans le cas de l'apprentissage non-supervisé, une approche possible consiste à simplement remplacer les cibles y_t par les entrées $\mathbf{x}_t$ dans l'équation 1.1.

dient stochastique sera majoritairement utilisé. Malgré sa simplicité, cet algorithme est très efficace, particulièrement lorsque l'ensemble d'entraînement $\mathcal{D}$ contient une grande quantité d'exemples et que le modèle f_Θ nécessite beaucoup de paramètres [101]. La descente de gradient stochastique procède à la mise à jour des paramètres d'un modèle immédiatement après le traitement de chaque exemple d'apprentissage. Suite au calcul de la sortie $f(\mathbf{x}_t)$ du modèle et du coût associé au risque, l'algorithme tente alors de diminuer ce coût en modifiant chaque paramètre $\boldsymbol{\theta}_i \in \Theta$ dans la direction opposée au gradient $\frac{\partial}{\partial \boldsymbol{\theta}_i} L(f_\Theta(x_t), y_t)$. Le pseudocode de cet algorithme est décrit à la figure 1.2.1.

Apprentissage bayésien

Le principe derrière l'apprentissage bayésien est de considérer les paramètres Θ du modèle choisi comme une variable aléatoire et de l'éliminer à l'aide des règles de base de la théorie des probabilités.

Par exemple, étant donné un modèle probabiliste de vraisemblance $p(y|\mathbf{x}, \Theta)$ ainsi qu'une distribution a priori des paramètres $p(\Theta)$, une façon de faire de l'apprentissage bayésien consiste à faire l'opération suivante :

$$f(\mathbf{x}) = \operatorname*{argmax}_{y \in \mathcal{Y}} p(y|\mathbf{x}, \mathcal{D})$$

où

$$p(y|\mathbf{x}, \mathcal{D}) = \sum_{\Theta} p(y|\mathbf{x}, \Theta) p(\Theta|\mathcal{D})$$

et

$$p(\Theta|\mathcal{D}) = \frac{\prod_{(\mathbf{x}_t, y_t) \in \mathcal{D}} p(y_t|\mathbf{x}_t, \Theta) p(\Theta)}{\sum_{\Theta^*} \prod_{(\mathbf{x}_t, y_t) \in \mathcal{D}} p(y_t|\mathbf{x}_t, \Theta^*) p(\Theta^*)} \ .$$

Dans le cas où Θ appartient à un ensemble continu de valeurs, les sommes ci-haut sont remplacées par des intégrales. Bien que les travaux de cette thèse n'utiliseront pas l'apprentissage bayésien, il est important de savoir que l'apprentissage par MRE n'est pas le seul principe possible pour dériver un algorithme d'apprentissage.

Algorithme de descente de gradient stochastique

(1) Initialiser les éléments de Θ (possiblement aléatoirement)
(2) Pour chaque $(\mathbf{x}_t, y_t) \in \mathcal{D}$
(3) Calculer $L(f_\Theta(\mathbf{x}_t), y_t)$
(4) Mettre à jour les éléments $\boldsymbol{\theta}_i$ de Θ : $\boldsymbol{\theta}_i \leftarrow \boldsymbol{\theta}_i - \Delta \frac{\partial}{\partial \boldsymbol{\theta}_i} L(f_\Theta(x_t), y_t)$
(5) Revenir à (2) si le critère d'arrêt n'est pas satisfait

Figure 1.6 – Pseudocode pour l'algorithme de descente de gradient. Le paramètre Δ, appelé **taux d'apprentissage**, doit être déterminé par l'utilisateur. Un critère d'arrêt doit aussi être précisé. Il peut être souhaitable de diminuer la valeur de Δ après chaque passage à travers $\mathcal{D}$. Une façon répandue est de plutôt utiliser $\frac{\Delta}{1+\beta \cdot it}$ comme taux d'apprentissage, où it est le nombre de fois que les éléments de Θ ont été mis à jour et β est appelé **constante de décroissance** ou *decrease constant*.

1.2.2 Modèles paramétriques et non-paramétriques

On distingue deux types de modèle qui peuvent être utilisés dans le cadre d'un algorithme d'apprentissage : les modèles **paramétriques** et **non-paramétriques**.

Les modèles paramétriques ont une forme précise, qui ne change pas en fonction de la quantité n de données dans $\mathcal{D}$. Ainsi, la taille de Θ ne dépend pas de n, et la **capacité** de représentation des phénomènes contenus dans $\mathcal{D}$ est fixe. Un exemple de modèle paramétrique pouvant être entraîné par apprentissage par MRE est un modèle linéaire. Dans le cas de l'apprentissage bayésien, les modèles gaussiens ou multinomiaux en sont aussi des instances.

Les modèles non-paramétriques, au contraire, deviennent de plus en plus complexes au fur et à mesure que le nombre d'exemples d'apprentissage augmente. La capacité de ces modèles à représenter des phénomènes complexes progresse donc avec n. Des exemples de modèles non-paramétriques entraînés par apprentissage par MRE sont les **machines à noyau** (voir section 1.3.3). Dans le cas de l'apprentissage bayésien, les modèles de mélange utilisant les processus de Dirichlet [126] sont considérés comme étant non-paramétriques.

Pour certains algorithmes, la capacité du modèle est contrôlée explicitement par un

hyper-paramètre, soit un paramètre dont la valeur doit être fixée avant l'entraînement du modèle. C'est le cas du nombre de neurones cachés contenus dans un réseau de neurones artificiel. Dans un tel cas, la distinction entre modèle paramétrique et non-paramétrique n'est pas tout à fait claire. Effectivement, si l'utilisateur décide de toujours utiliser une même valeur prédéterminée pour cet hyper-paramètre, la capacité du modèle est alors fixe, comme dans un modèle paramétrique. Cependant, si l'utilisateur décide d'ajuster la valeur de l'hyper-paramètre en fonction de la performance du modèle sur les données, le terme de modèle non-paramétrique semble alors plus approprié.

1.3 Quelques modèles de classification courants

Le travaux de cette thèse se concentreront surtout sur des tâches de classification. Voici donc une brève description de différents modèles de classification couramment utilisés et qui seront référencés par les travaux de cette thèse.

1.3.1 Modèles linéaires

Parmi les modèles les plus simples se trouvent les modèles linéaires. Pour une tâche de classification, on dit qu'un modèle est linéaire si les 2 régions de l'espace $\mathcal{X}$ associées à n'importe quelle paire de classes par le modèle sont séparables par un hyperplan.

Malgré leur faible capacité, les modèles linéaires sont couramment utilisés, en particulier sur des problèmes où l'entrée est de très haute dimension. Il existe aussi différentes techniques permettant d'augmenter la performance de modèles linéaires, dont l'extraction de caractéristiques (voir section 1.1.2) ainsi que l'astuce du noyau (voir section 1.3.3).

Un exemple d'algorithme d'apprentissage pour modèle linéaire utile à connaître dans le contexte des travaux de cette thèse est le classifieur par **régression logistique linéaire**. Cet algorithme définit

$$f_\Theta(\mathbf{x}) = \mathrm{softmax}(\mathbf{W}\mathbf{x} + \mathbf{b}) \tag{1.2}$$

où $\Theta = \{\mathbf{W}, \mathbf{b}\}$ contient les poids $\mathbf{W}$ et les biais $\mathbf{b}$ du classifieur. Ce classifieur utilise

la **fonction softmax** :

$$\text{softmax}(\mathbf{a})_i = \frac{e^{a_i}}{\sum_{j=1}^{C} e^{a_j}}$$

où C correspond au nombre de classes. En utilisant cette fonction, la sortie vectorielle du modèle $f_\Theta(\mathbf{x})$ peut être interprétée comme une estimation de la distribution conditionnelle de la classe y à assigner à l'entrée $\mathbf{x}$. Dans le cas d'un problème de classification binaire ($\mathcal{Y} = \{0, 1\}$), étant donné la contrainte de sommation à 1 de la sortie du modèle, on suppose plutôt que le modèle estime la probabilité $p(y = 1|\mathbf{x})$ que l'entrée appartienne à la classe 1 en définissant

$$f_\Theta(\mathbf{x}) = \text{sigmoid}(\mathbf{w}^\mathbf{T}\mathbf{x} + b)$$

où on n'a maintenant besoin que d'un seul biais b plutôt que d'un vecteur, ainsi que d'un seul vecteur de poids $\mathbf{w}$ plutôt que d'une matrice $\mathbf{W}$. Dans ce cas, le classifieur utilise la **fonction sigmoïde** :

$$\text{sigmoid}(a) = \frac{1}{1 + e^{-a}} \ .$$

Afin d'entraîner ce modèle, on utilise le coût associé au risque suivant :

$$L(f_\Theta(\mathbf{x}), y) = -\log f_\Theta(\mathbf{x})_y$$

ou dans le cas binaire

$$L(f_\Theta(\mathbf{x}), y) = -y \log f_\Theta(\mathbf{x}) - (1 - y) \log(1 - f_\Theta(\mathbf{x})) \ ,$$

soit la log-vraisemblance conditionnelle négative de l'assignation de l'entrée $\mathbf{x}$ à la classe y selon la distribution conditionnelle $f_\Theta(\mathbf{x})$. Tel que décrit à la section 1.2.1, il est alors possible d'entraîner ce modèle par descente de gradient stochastique[2]. Ainsi, l'entraînement vise à maximiser la vraisemblance conditionnelle de la classification observée dans l'ensemble d'entraînement telle que donnée par le modèle. Il est intéressant

[2]De nombreuses autres approches ont aussi été développées pour l'entraînement d'un classifieur par régression logistique. La descente de gradient stochastique est considérée ici pour sa simplicité et parce qu'elle se comporte généralement bien sur des ensembles d'entraînement de grand taille.

de noter que ce même algorithme peut aussi être dérivé en suivant une approche par maximum d'entropie [18].

Le problème d'optimisation associé à cet algorithme d'apprentissage (voir équation 1.1) est **convexe**. L'avantage que procure une telle propriété est entre autres la garantie (sous certaines conditions peu contraignantes [25]) de la convergence de la descente de gradient stochastique vers un minimum global.

Il est à noter que plusieurs autres algorithmes d'apprentissage pour modèles linéaires ont été développés, se distinguant principalement par le coût utilisé pour l'entraînement. On compte parmi ces algorithmes le perceptron [127, 128], le classifieur aux moindres carrés et la machine à vecteurs de support linéaire [39, 153].

1.3.2 Réseau de neurones artificiel

Le désavantage le plus important des modèles linéaires est leur faible capacité. Effectivement, ils sont incapables de résoudre de manière satisfaisante un problème de classification si la frontière entre les régions de l'espace d'entrée associées aux différentes classes est non-linéaire. Il est d'ailleurs plutôt rare en pratique qu'un problème de classification soit linéaire, ainsi cet inconvénient des modèles linéaires est important. Le réseau de neurones artificiel, aussi appelé perceptron multi-couche, permet de corriger cette situation.

Un réseau de neurones à une couche cachée est définie mathématiquement comme suit :

$$f_\Theta(\mathbf{x}) = \mathrm{softmax}\left(\mathbf{W}h\left(\mathbf{V}\mathbf{x} + \mathbf{c}\right) + \mathbf{b}\right) \tag{1.3}$$

où $\Theta = \{\mathbf{b}, \mathbf{c}, \mathbf{V}, \mathbf{W}\}$ contient les biais $\mathbf{b}$ et connections $\mathbf{W}$ de la **couche de sortie** ainsi que les biais $\mathbf{c}$ et connections $\mathbf{V}$ de la **couche cachée** du réseau. La figure 1.7 présente une illustration graphique d'un réseau de neurones artificiel à une seule couche cachée.

La fonction h est appelée **fonction d'activation**. Pour h, on choisit normalement entre la fonction sigmoïde ou bien la **fonction tangente hyperbolique** :

$$\tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}} \ .$$

On définit aussi le résultat de l'application de ces fonctions d'activation à un vecteur comme étant le vecteur des résultats de l'application de cette fonction à tous les éléments :

$$h(\mathbf{a})_i = h(a_i) \ .$$

Le rôle de la fonction d'activation de la couche cachée est d'introduire un aspect non-linéaire à la prédiction $f_\Theta(\mathbf{x})$ calculée par le réseau de neurones. De plus, le nombre d'éléments (appelés neurones cachés) contenus dans la couche cachée du réseau permet de contrôler sa capacité. Plus il y a de neurones cachés, plus le réseau sera en mesure de calculer une prédiction non-linéaire complexe, capable de bien refléter la relation observée dans l'ensemble d'entraînement entre les entrées $\mathbf{x}_t$ et leur cible y_t. Il a d'ailleurs été démontré qu'en augmentant suffisamment le nombre de neurones cachés d'un tel réseau, il existe nécessairement une valeur des paramètres Θ qui permette d'approcher jusqu'à une précision arbitraire une fonction Borel-mesurable (e.g. une fonction continue) d'un espace de dimension finie vers un espace de dimension finie quelconque [84]. On qualifie ainsi les réseaux de neurones artificiels d'**approximateurs universels**.

Il est possible de lier le réseau de neurones artificiel (équation 1.3) à la classification par régression logistique (équation 1.2) en remarquant que l'équation 1.3 équivaut à l'équation 1.2 où on a remplacé l'entrée $\mathbf{x}$ par la valeur de la couche cachée $h\left(\mathbf{V}\mathbf{x} + \mathbf{c}\right)$. Ainsi, on peut voir la classification à l'aide d'un réseau de neurones comme la classification par régression logistique s'opérant sur une représentation $h\left(\mathbf{V}\mathbf{x} + \mathbf{c}\right)$ apprise de l'entrée $\mathbf{x}$. Selon ce point de vue, la tâche de la couche cachée est de découvrir une représentation des entrées qui permette de les classifier linéairement.

Par exemple, considérons le domaine des fonctions booléennes en deux dimensions (i.e., $\mathcal{X} = \{0, 1\}^2$). Il est bien connu que la fonction binaire $\mathrm{XOR}(x_1, x_2)$

$$\mathrm{XOR}(x_1, x_2) = \begin{cases} 0 & \text{si } x_1 = 0 \text{ et } x_2 = 0 \\ 1 & \text{si } x_1 = 1 \text{ et } x_2 = 0 \\ 1 & \text{si } x_1 = 1 \text{ et } x_2 = 0 \\ 0 & \text{si } x_1 = 1 \text{ et } x_2 = 1 \end{cases}$$

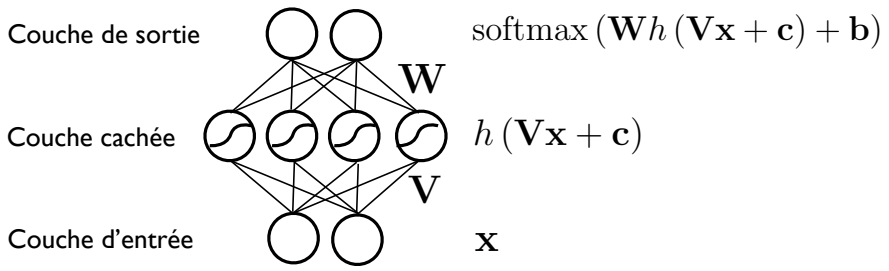


Figure 1.7 – Illustration d'un réseau de neurones artificiel.

ne peut être modélisée par un classifieur linéaire. Cependant, les fonctions OR et AND

$$\mathrm{OR}(x_1, x_2) = \begin{cases} 0 & \text{si } x_1 = 0 \text{ et } x_2 = 0 \\ 1 & \text{si } x_1 = 1 \text{ et } x_2 = 0 \\ 1 & \text{si } x_1 = 1 \text{ et } x_2 = 0 \\ 1 & \text{si } x_1 = 1 \text{ et } x_2 = 1 \end{cases} \quad \mathrm{AND}(x_1, x_2) = \begin{cases} 0 & \text{si } x_1 = 0 \text{ et } x_2 = 0 \\ 0 & \text{si } x_1 = 1 \text{ et } x_2 = 0 \\ 0 & \text{si } x_1 = 1 \text{ et } x_2 = 1 \\ 0 & \text{si } x_1 = 1 \text{ et } x_2 = 1 \end{cases}$$

peuvent l'être, ainsi que toute variation où la négation de certaines des composantes d'entrée est appliquée préalablement (voir figure 1.8). En remarquant que

$$\mathrm{XOR}(x_1, x_2) = \mathrm{OR}\left(\mathrm{AND}(\overline{x_1}, x_2), \mathrm{AND}(x_1, \overline{x_2})\right)$$

il donc possible de modéliser la fonction XOR à l'aide d'un réseau de neurones avec deux neurones cachés modélisant les fonctions $\mathrm{AND}(\overline{x_1}, x_2)$ et $\mathrm{AND}(x_1, \overline{x_2})$ respectivement, et dont les poids de sortie sont ceux d'un classifieur par régression logistique modélisant la fonction $\mathrm{OR}(x_1, x_2)$. Ainsi, en combinant ensemble plusieurs fonctions simples (OR et AND dans cet exemple), un réseau de neurones arrive à modéliser une fonction d'une plus grande complexité (XOR). Cette explication de la complexité du calcul de la sortie d'un réseau de neurones servira d'ailleurs d'inspiration aux travaux de cette thèse.

Cela étant dit, ce dernier exemple est simpliste comparativement aux problèmes que l'on souhaite typiquement résoudre. Entre autres, certains des réseaux de neurones qui

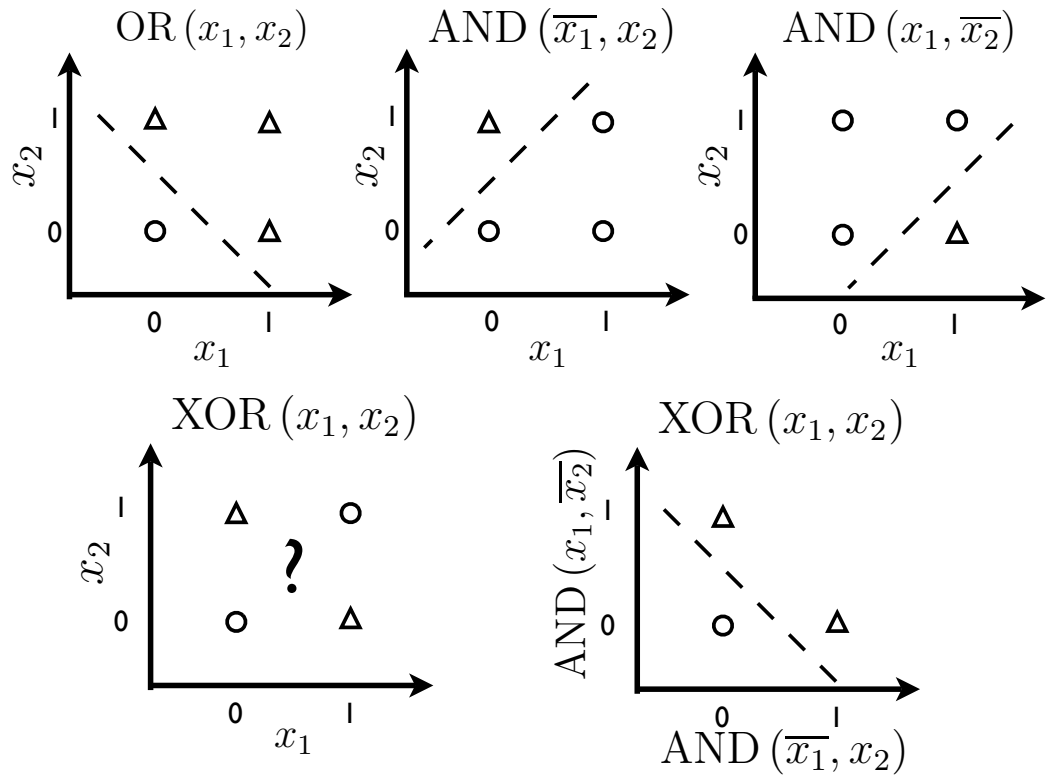


Figure 1.8 – Exemple de modélisation de XOR par un réseau à une couche cachée. En haut, de gauche à droite, illustration des fonctions booléennes $\mathrm{OR}(x_1, x_2)$, $\mathrm{AND}\,(\overline{x_1}, x_2)$ et $\mathrm{AND}\,(x_1, \overline{x_2})$. En bas, on présente l'illustration de la fonction $XOR(x_1, x_2)$ en fonction des valeurs de x_1 et x_2 (à gauche), puis de $\mathrm{AND}\,(\overline{x_1}, x_2)$ et $\mathrm{AND}\,(x_1, \overline{x_2})$ (à droite). Les points représentés par un cercle ou par un triangle appartiennent à la classe 0 ou 1, respectivement. On observe que, bien qu'un classifieur linéaire soit en mesure de résoudre le problème de classification associé aux fonctions OR et AND, il ne l'est pas dans le cas du problème de XOR. Cependant, on utilisant les valeurs de $\mathrm{AND}\,(\overline{x_1}, x_2)$ et $\mathrm{AND}\,(x_1, \overline{x_2})$ comme nouvelle représentation de l'entrée (x_1, x_2), le problème de classification XOR peut alors être résolu linéairement. À noter que dans ce dernier cas, il n'existe que trois valeurs possibles de cette nouvelle représentation, puisque $\mathrm{AND}\,(\overline{x_1}, x_2)$ et $\mathrm{AND}\,(x_1, \overline{x_2})$ ne peuvent être toutes les deux vraies pour une même entrée.

ont été entraînés dans le cadre des travaux de cette thèse contiennent quelques milliers de neurones cachés. Ainsi, la tâche de l'algorithme d'apprentissage est de modifier les paramètres du réseau afin de trouver la nature des caractéristiques de l'entrée que chaque neurone doit extraire pour résoudre le problème de classification. Idéalement, ces carac-

téristiques devront être différentes, puisqu'avoir deux neurones qui calculent la même caractéristique équivaudrait à gaspiller la capacité du modèle. Cependant, ces caractéristiques ne seront pas nécessairement exclusives, c'est-à-dire que pour une entrée donnée, il n'y aura pas seulement un neurone de la couche cachée qui sera actif, mais généralement plusieurs. De plus, il sera possible qu'un même neurone soit actif pour deux entrées totalement différentes (i.e., séparées par une distance euclidienne importante). On dit alors que la couche cachée du réseau de neurones calcule une **représentation distribuée** [75] de l'entrée. L'avantage principal des représentations distribuées est la grande quantité de phénomènes potentiellement présents dans l'entrée qu'elles sont en mesure de représenter. Effectivement, dans le cas simplifié où chaque neurone caché est binaire et ne peut valoir que 0 ou 1[3], un réseau contenant H neurones cachés sera en mesure de coder 2^H phénomènes différents, provenant de la combinaisons des H caractéristiques détectables par la couche cachée.

Ainsi, la capacité des réseaux de neurones est potentiellement très grande. Cependant, la possibilité d'exploiter cette capacité dépend grandement de l'aptitude de l'algorithme d'apprentissage à bien entraîner ces réseaux. Comme pour la régression logistique, on utilise aussi la descente de gradient stochastique appliquée au coût de la log-vraisemblance conditionnelle négative de l'assignation de la classe étant donnée l'entrée. Ceci nécessite le calcul du gradient du coût de chaque exemple par rapport à chacun des paramètres du réseau. Une méthode naïve mais très coûteuse de procéder est de calculer ces gradients indépendamment pour chaque paramètre. Cependant, il est possible de faire bien mieux et de limiter au maximum la quantité de calculs répétés inutilement. Cette méthode, appelée la **rétropropagation de gradient** [99, 131], tire avantage de la règle de dérivation en chaîne afin de factoriser les différents éléments de calcul pour tous les gradients et procéder à leur estimation dans un ordre avantageux. Il en résulte une procédure en deux passes à travers le réseau, qui fonctionne pour un nombre quelconque de couches cachées. Suite à ces deux passes, le calcul du gradient pour tous les paramètres est complété. La première passe est une passe avant, débutant à la couche

[3]Par exemple, on pourrait imposer cette contrainte à un réseau utilisant la fonction sigmoïde dans sa couche cachée en fixant $h(\mathbf{a})_i = 1_{\mathrm{sigmoid}(a_i)>0.5}$

d'entrée et progressant vers la couche de sortie en calculant l'activation de chacune des couches cachées du réseau. La deuxième et dernière passe, la passe arrière, débute à la couche de sortie et propage le gradient du coût par rapport à la sortie du modèle à travers toutes les couches et vers la couche d'entrée, selon la règle de dérivation en chaîne. L'algorithme de rétropropagation de gradient est détaillé à l'annexe A du chapitre 6 (voir la partie *fine-tuning phase* du pseudocode).

La procédure de rétropropagation permet donc de calculer efficacement les gradients nécessaires pour appliquer l'algorithme de descente de gradient stochastique. Elle rend ainsi cet algorithme plus efficace, mais le problème d'optimisation associé à l'entraînement d'un réseau de neurones n'en demeure pas moins difficile. Effectivement, contrairement au classifieur par régression logistique, l'entraînement d'un réseau de neurones est un problème d'**optimisation non-convexe** pour lequel l'existence d'un seul minimum global n'est pas garantit. Typiquement, ce genre de problème d'optimisation possède ainsi plus d'un minimum local vers lequel une procédure de descente de gradient peut potentiellement converger. Ainsi, il n'existe pas de technique garantissant l'atteinte de l'optimum global lors de l'entraînement. Cependant, étant donnée la très faible capacité d'un classifieur par régression logistique ou de tout autre classifieur linéaire, on observe souvent en pratique que l'inconvénient qu'apporte un problème d'optimisation non-convexe est fortement compensé par l'avantage apporté par l'utilisation d'un modèle non-linéaire.

1.3.3 Machine à noyau

Mais est-il possible d'obtenir le meilleur des deux mondes, soit d'avoir un algorithme d'apprentissage utilisant un modèle non-linéaire mais dont l'entraînement est convexe ? Tel que mentionné dans la section précédente, un réseau de neurones peut être vu comme une régression logistique sur une représentation de l'entrée non-linéaire apprise. Ce qui rend un réseau de neurones non-linéaire, c'est la non-linéarité de la représentation, et ce qui rend l'optimisation non-convexe, c'est l'obligation d'apprendre cette représentation. Ainsi, en déterminant d'avance une représentation non-linéaire de l'entrée, $\phi(\mathbf{x})$, puis en procédant à l'apprentissage d'un classifieur par régression logistique utilisant cette

représentation, on obtient alors ce que l'on veut, soit un algorithme de classification non-linéaire convexe.

Le choix de la représentation $\boldsymbol{\phi}(\mathbf{x})$ a évidemment un impact crucial sur la performance de classification, mais il existe une approche très simple pour augmenter la capacité du classifieur. Sachant qu'il est toujours possible de classifier linéairement toute assignation en deux classes de n points (différents) disposés dans un espace en $n-1$ dimensions [152], il suffit donc d'augmenter la taille de la représentation $\boldsymbol{\phi}(\mathbf{x})$ pour augmenter arbitrairement la capacité du classifieur. Suivie naïvement, cette technique peut cependant s'avérer coûteuse, puisqu'en augmentant la taille de $\boldsymbol{\phi}(\mathbf{x})$ on augmente aussi la quantité de calculs à effectuer pour classifier une entrée.

Heureusement, il existe un truc appelé **astuce du noyau** [2] permettant d'éliminer la dépendance sur la taille de $\boldsymbol{\phi}(\mathbf{x})$ du calcul de la sortie, pour certains choix de $\boldsymbol{\phi}(\mathbf{x})$. Cette astuce s'applique à des algorithmes utilisant les données seulement via le calcul de produits scalaires $\mathrm{k}(\mathbf{x}_t, \mathbf{x}_{t'}) = \boldsymbol{\phi}(\mathbf{x}_t)^{\mathbf{T}}\boldsymbol{\phi}(\mathbf{x}_{t'})$ entre les exemples d'entraînement. Si le résultat de la fonction de noyau $\mathrm{k}(\mathbf{x}_t, \mathbf{x}_{t'})$ peut être calculé plus rapidement que de la façon directe $\sum_{i=1}^{H} \phi(\mathbf{x}_t)_i \phi(\mathbf{x}_{t'})_i$, il est alors possible de réaliser des gains importants en temps de calcul. Par exemple, le **noyau polynomial hétérogène**

$$\mathrm{k}(\mathbf{x}_t, \mathbf{x}_{t'}) = (1 + \mathbf{x}_t^{\mathbf{T}}\mathbf{x}_{t'})^d \tag{1.4}$$

permet de calculer le produit scalaire d'une représentation $\boldsymbol{\phi}(\mathbf{x})$ contenant tous les produits d'au plus d éléments de $\mathbf{x}$. Pour une entrée en D dimensions, bien que cette nouvelle représentation soit de taille $\sum_{i=1}^{d} \binom{D}{i}$, le calcul nécessité par l'équation 1.4 n'est que d'ordre $O(D)$.

La classification par régression logistique est un exemple d'algorithme pouvant profiter du astuce du noyau. Considérons le cas de la classification binaire. La procédure d'entraînement par descente de gradient stochastique nécessite alors le calcul du gradient du coût associé au risque par rapport aux poids $\mathbf{w}$, soit :

$$\frac{\partial}{\partial \mathbf{w}} L(f_\Theta(\boldsymbol{\phi}(\mathbf{x}_t)), y_t) = -\left(y_t - f_\Theta(\boldsymbol{\phi}(\mathbf{x}_t))\right) \boldsymbol{\phi}(\mathbf{x}_t) \ .$$

En initialisant à $\mathbf{w} = 0$ la procédure de descente de gradient, on peut donc dire qu'à tout moment, il existe un vecteur $\boldsymbol{\alpha}$ qui est tel que les poids du classifieur peuvent alors s'écrire comme suit :

$$\mathbf{w} = \sum_{t=1}^{n} \alpha_t \boldsymbol{\phi}(\mathbf{x}_t) \ .$$

Ceci implique aussi que le modèle peut être réécrit comme suit :

$$f_\Theta(\boldsymbol{\phi}(\mathbf{x})) = \text{sigmoid}\left(\sum_{t=1}^{n} \alpha_t \boldsymbol{\phi}(\mathbf{x}_t)^{\mathbf{T}} \boldsymbol{\phi}(\mathbf{x})\right) \tag{1.5}$$

où Θ, qui doit contenir l'information nécessaire pour le calcul de la sortie, contient maintenant la valeur du vecteur $\boldsymbol{\alpha}$ ainsi que toutes les entrées de l'ensemble d'entraînement pour lesquelles $\alpha_i \neq 0$. Sous cette nouvelle paramétrisation du modèle, la règle de mise à jour des paramètres de Θ pour un exemple $(\boldsymbol{\phi}(\mathbf{x}_t), y_t)$ peut maintenant être résumée par la mise à jour suivante :

$$\alpha_i \leftarrow \alpha_i + \Delta \left(y_t - f_\Theta(\boldsymbol{\phi}(\mathbf{x}_t))\right) \ . \tag{1.6}$$

Le calcul de la sortie du modèle ainsi que la mise à jour des paramètres du modèle (équations 1.5 et 1.6) sont donc maintenant écrits sous une forme ne nécessitant que le calcul de produits scalaires entre des entrées. Il est alors possible de remplacer ces produits scalaires par le calcul d'un noyau, obtenant ainsi un **classifieur par régression logistique à noyau**. Il est à noter que plusieurs autres algorithmes de classification linéaires ont une version à noyau, le plus populaire étant probablement la machine à vecteurs de supports à noyau [24, 39].

Bien qu'il soit possible parfois de dériver un noyau à partir d'une représentation donnée, il est souvent plus facile de choisir directement un noyau sans trop se préoccuper de la nature de la représentation calculée. Cependant, le noyau choisi doit avoir une représentation $\boldsymbol{\phi}(\mathbf{x})$ associée, sans quoi l'utilisation d'un noyau peut entraîner des difficultés. Pour cela, le noyau doit être **défini positif**, i.e., pour tout ensemble d'entrée $\{\mathbf{x}_t\}_{t=1}^{n}$, la

matrice de Gram M associée

$$M_{t,t'} = \mathrm{k}(\mathbf{x}_t, \mathbf{x}_{t'})$$

doit être définie positive. Un noyau défini positif couramment utilisé est le **noyau gaussien**

$$\mathrm{k}(\mathbf{x}_t, \mathbf{x}_{t'}) = e^{-\frac{||\mathbf{x}_t - \mathbf{x}_{t'}||^2}{2\sigma^2}}$$

dont la représentation $\phi(\mathbf{x})$ associée est de taille infinie, i.e., correspond à une fonction.

On voit donc que l'astuce du noyau, en augmentant arbitrairement la dimensionnalité de la représentation $\phi(\mathbf{x})$, peut grandement augmenter la capacité d'un classifieur linéaire. Il est d'ailleurs possible de démontrer qu'une machine à noyau est un approximateur universel, que ce soit dans le cas des noyaux gaussien et polynomial, où σ^2 et d contrôlent la qualité de l'approximation, respectivement. De plus, contrairement au réseau de neurones, cette propriété est jumelée à un entraînement correspondant à une optimisation convexe. Ainsi, on observe en pratique que les machines à noyau peuvent facilement atteindre un risque moyen minimal sur les données d'entraînement. On serait alors porté à croire que les machines à noyau forment de meilleurs algorithmes d'apprentissage que les réseaux de neurones. Cependant, dans le prochain chapitre, on verra que la situation est loin d'être aussi simple.

CHAPITRE 2

LE DÉFI DE L'APPRENTISSAGE AUTOMATIQUE : LA GÉNÉRALISATION

L'objectif de l'apprentissage automatique est de fournir à l'utilisateur un modèle f qui sera en mesure de bien résoudre une tâche prédéterminée. Afin de pouvoir prétendre être en mesure d'atteindre cet objectif, le modèle doit ainsi bien se comporter sur de nouvelles entrées, i.e., des entrées qui ne se trouvent pas explicitement dans l'ensemble d'entraînement mais qui ont été générées par le même processus. Par exemple, dans le cas de la classification, on souhaite que f obtienne une erreur de classification qui soit la plus petite possible sur ces nouvelles entrées. On parle alors de la performance de **généralisation** de l'algorithme d'apprentissage ayant produit f. Cette performance est normalement mesurée empiriquement, à l'aide d'un ensemble $\mathcal{D}_{\text{test}}$ de nouveaux exemples qui ne se trouvent pas dans l'ensemble d'entraînement $\mathcal{D}$. On appelle $\mathcal{D}_{\text{test}}$ l'**ensemble de test**. Bien que la plupart des concepts qui sont présentés dans ce chapitre ne sont pas spécifiques à l'apprentissage par MRE (voir section 1.2.1), on se limitera ici à ce type d'apprentissage. On se concentrera aussi sur le problème de la classification, puisque celui-ci est au cœur des travaux présentés dans cette thèse.

Typiquement, on observe une différence importante entre l'erreur obtenue sur l'ensemble d'entraînement et celle obtenue sur l'ensemble de test, la première étant normalement plus basse que la seconde. De plus, ces erreurs progressent de façons bien distinctes au fur et à mesure que progresse l'entraînement du modèle. Au départ, l'entraînement du modèle fait baisser ces deux erreurs. On dit alors que le modèle est dans un état de **sous-apprentissage**, signifiant que le modèle n'a pas terminé d'extraire de l'ensemble d'entraînement l'information qui lui permettrait de mieux généraliser. Ensuite, le modèle arrive éventuellement au point où l'information non-extraite de l'ensemble d'entraînement correspond plus à du bruit ou de l'information spécifique à cet ensemble qui ne permet pas d'améliorer la performance de généralisation. On voit alors l'erreur sur l'ensemble de test augmenter, bien que l'erreur d'entraînement continue de diminuer, puisque cette erreur est optimisée plus directement par l'algorithme d'apprentissage. On

dit alors que le modèle est dans un état de **surapprentissage**. La figure 2.1 illustre graphiquement un exemple typique de progression des erreurs d'entraînement et de test.

Ainsi, on remarque bien l'importance du choix du nombre d'itérations (qui est un hyper-paramètre) pour l'entraînement d'un algorithme, lorsque la procédure d'entraînement est itérative et nécessite plusieurs parcours de $\mathcal{D}$. Pour ce faire, il est possible d'utiliser un autre ensemble $\mathcal{D}_{\text{valid}}$ d'exemples ne se trouvant pas dans $\mathcal{D}$, afin de suivre la progression de la performance de généralisation. L'ensemble $\mathcal{D}_{\text{valid}}$, appelé **ensemble de validation**, doit aussi être différent de l'ensemble de test, qui lui ne doit aucunement être utilisé dans la procédure d'entraînement puisqu'il sert à simuler l'application du modèle sur de nouvelles entrées. À l'aide de cet ensemble de validation, on peut alors déterminer le moment de l'arrêt de l'optimisation en surveillant la progression de l'erreur calculée sur $\mathcal{D}_{\text{valid}}$ et en arrêtant l'optimisation lorsque l'erreur se met à augmenter. Cette procédure est connue sous le nom d'**arrêt prématuré** (*early stopping*). Il est aussi possible de continuer l'optimisation même si l'erreur de validation augmente, mais de retenir l'état de l'optimisation au moment correspondant à la meilleure performance de validation et d'y revenir si celle-ci s'avère être la meilleure globalement.

Dans l'exemple précédent, les états de sous-apprentissage et surapprentissage dépendent du nombre d'itérations d'un algorithme d'apprentissage itératif. Cependant le même genre de phénomène peut être observé lorsque d'autres hyper-paramètres sont variés. Entre autre, la même relation illustrée à la figure 2.1 serait observée si le nombre d'itérations d'entraînement était remplacé par la taille de la couche cachée d'un réseau de neurones artificiel ou le degré d du noyau polynomial hétérogène d'une machine à noyau (en supposant que les autres hyper-paramètres du modèle sont fixés). Intuitivement, en augmentant trop la capacité de ces modèles en faisant un mauvais choix pour leurs hyper-paramètres, la procédure d'entraînement peut alors extraire de l'ensemble d'entraînement plus d'information spécifique à cet ensemble, non liée au problème d'apprentissage que l'on essaie de résoudre. Ainsi, les concepts de sous-apprentissage et surapprentissage ne sont pas spécifiques aux algorithmes d'apprentissage itératifs, mais s'appliquent bien en général. Ici aussi, on peut alors utiliser un ensemble de validation afin de comparer la performance de généralisation pour différents choix de valeurs

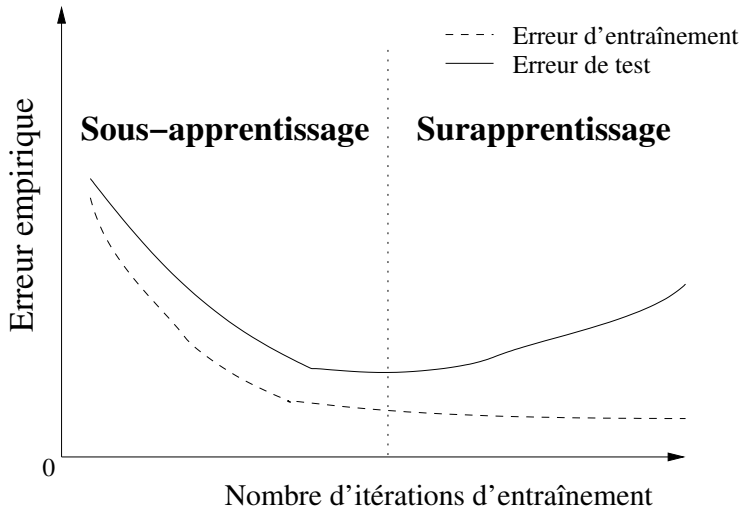


Figure 2.1 – Relation normalement observée entre le nombre d'itérations d'un algorithme d'apprentissage et les erreurs d'entraînement et de test. En général, l'erreur d'entraînement est plus petite que l'erreur de test et l'erreur d'entraînement diminue constamment au fur et à mesure que l'entraînement progresse. Cependant, on observe typiquement une courbe en « U » dans la relation entre l'erreur de test et le nombre d'itérations d'entraînement.

d'hyper-paramètres et choisir celui qui est le plus avantageux.

Il est important de noter que, une fois l'ensemble de validation utilisé afin de sélectionner une valeur pour les hyper-paramètres, l'erreur sur cet ensemble ne peut plus être utilisée afin d'estimer la performance de généralisation de l'algorithme d'apprentissage. Effectivement, puisque cette erreur a été directement minimisée par la procédure de sélection des hyper-paramètres, l'ensemble de validation a été utilisé comme une sorte d'ensemble d'entraînement, pour optimiser les hyper-paramètres. L'ensemble de test $\mathcal{D}_{\text{test}}$ est alors essentiel afin de déterminer de façon non-biaisée la performance de généralisation d'un algorithme et comparer plusieurs algorithmes ensemble.

La nécessité de mettre des données de côté pour former un ensemble de validation peut devenir désavantageuse lorsqu'on a à la base peu de données d'entraînement. Aussi, plus le modèle a d'hyper-paramètres, plus on souhaitera tester de combinaisons de valeurs pour ces hyper-paramètres, ce qui nécessite un ensemble de validation d'autant plus grand afin d'éviter un "surapprentissage" sur l'ensemble de validation. Pour cette raison, des techniques bayesiennes de sélection de modèle, qui ne requièrent pas d'en-

semble de validation, ont été proposées. Effectivement, en définissant des distributions a priori sur les paramètres et hyper-paramètres du modèle, il est alors possible d'obtenir une distribution a posteriori sur l'espace des hyper-paramètres en marginalisation sur les paramètres et en conditionnant sur les données d'entraînement. À partir de cette distribution, on peut alors choisir la combinaison d'hyper-paramètres ayant la plus grande probabilité. On peut même utiliser une moyenne de modèles obtenus à partir de différentes valeurs d'hyper-paramètres échantillonnées selon cette distribution. Ces méthodes nécessitent une intégrale sur l'ensemble des paramètres du modèle qui est souvent difficile à calculer, mais certaines approximations de ces intégrales sont possibles (voir entre autres MacKay [107, 108]). Bien que ces méthodes puissent être utiles, l'utilisation d'un ensemble de validation est souvent préférée pour sa simplicité. Les travaux de cette thèse seront ainsi basés sur cette approche.

On vient donc de voir une procédure simple permettant de favoriser l'entraînement d'un modèle ayant une bonne performance de généralisation, en se basant sur l'utilisation d'un ensemble de validation. On dit qu'une telle procédure est une procédure de **sélection de modèle**[1], puisqu'elle permet de faire un bon choix pour la capacité du modèle ainsi que pour la durée d'entraînement pour ce modèle. Cette procédure n'interagit pas directement avec le mécanisme d'entraînement de l'algorithme d'apprentissage, et ainsi peut être appliquée à n'importe quel algorithme utilisant un ou plusieurs hyper-paramètres.

Mais serait-il possible d'adapter directement le fonctionnement même d'un algorithme d'apprentissage et favoriser davantage une meilleure généralisation ? La réponse est oui, et la suite de ce chapitre se concentre sur différentes adaptations qu'il est possible d'appliquer à un algorithme d'apprentissage afin d'améliorer encore plus sa performance de généralisation. Plus spécifiquement, les prochaines sections traitent de la **régularisation**, de l'**apprentissage non-local** et de la **découverte d'une représentation hiérarchique**. Il est important de noter que ces approches sont complémentaires à la sélection de modèle, qui sera toujours appliquée, telle que décrite précédemment.

[1] Il existe dans la littérature plusieurs autres techniques de sélection de modèle. Cependant, la procédure décrite dans ce chapitre est très populaire et fonctionne très bien en pratique, malgré sa simplicité. C'est pour cette raison que cette procédure a été choisie pour les travaux de cette thèse.

2.1 Régularisation

On a mentionné précédemment qu'en augmentant trop la capacité d'un modèle, il était possible d'atteindre un état de surapprentissage et ainsi d'obtenir une performance de généralisation moins élevée. De plus, différents modèles ont différents hyper-paramètres contrôlant leur capacité. Dans le cas d'un réseau de neurones à une couche cachée, c'est le nombre de neurones cachés qui contrôle la capacité. Imaginons maintenant une situation où l'ensemble d'entraînement contient très peu de données, tellement qu'un réseau de neurones possédant plus d'une dizaine de neurones cachés tombe inévitablement dans un état de surapprentissage. Cependant, supposons aussi que nous connaissions suffisamment le problème à résoudre pour savoir qu'un réseau de neurones à seulement 10 neurones cachés ne peut pas vraisemblablement résoudre ce problème, et que la seule raison pour laquelle on se restreint à 10 neurones est que l'ensemble d'entraînement est trop petit pour bien entraîner un réseau à plus de neurones cachés sans surapprentissage. Il serait donc utile de pouvoir contrôler autrement la capacité d'un tel modèle sans devoir trop se limiter dans le nombre de paramètres que contient le modèle.

Une approche générale fréquemment utilisée pour exercer un tel contrôle sur la capacité d'un modèle est appelée **régularisation**. La régularisation fonctionne en imposant une pénalité $\Omega(\Theta)$ aux paramètres Θ afin de favoriser certaines configurations des paramètres qui sont plus « souhaitables » ou « plausibles » que d'autres. Plus précisément, on ajuste le critère d'entraînement basé sur le risque empirique $R(\Theta, \mathcal{D})$ de l'équation 1.1 comme suit :

$$\widehat{\Theta} = \underset{\Theta}{\operatorname{argmin}}\, R(\Theta, \mathcal{D}) + \lambda\Omega(\Theta) \tag{2.1}$$

où la pénalisation de régularisation est pondérée par le facteur λ. On considère donc que λ est un hyper-paramètre additionnel du modèle. Une autre justification derrière l'utilisation de la régularisation est qu'elle permet de diminuer la variance de la solution (i.e., les paramètres du modèle) trouvée par l'algorithme d'apprentissage. Par variance, on fait référence à l'importance de la variation de la solution lorsque l'ensemble d'entraînement est regénéré et l'algorithme d'apprentissage relancé. Effectivement, puisque le terme de régularisation impose une certaine préférence sur les configurations des paramètres, cer-

taines de ces configurations seront alors implicitement écartées des solutions qui auraient pu être trouvées par un algorithme minimisant le risque non régularisé (équation 1.1). Cependant, un terme de régularisation vise aussi à restreindre la capacité du modèle. Ainsi, il est probable que, pour une valeur de λ trop élevée, il ne soit plus possible de résoudre parfaitement la tâche à apprendre et que l'on ait alors augmenté le biais de la solution trouvée par l'algorithme. L'ajout d'un terme de régularisation permet donc de trouver un bon **compromis biais-variance** pour l'algorithme d'apprentissage.

Un type de régularisation souvent utilisé est basé sur la pénalisation de la norme ℓ^2 au carré des paramètres de Θ, soit :

$$\Omega(\Theta) = \sum_{\boldsymbol{\theta} \in \Theta} ||\boldsymbol{\theta}||_2^2 = \sum_{\boldsymbol{\theta} \in \Theta} \sum_{i=1}^{l(\theta)} \theta_i^2$$

où $l(\theta)$ est le nombre d'éléments de θ. Cette régularisation encourage donc les paramètres du modèle à ne pas trop s'éloigner de la valeur 0, selon la distance euclidienne. D'autres distances peuvent être utilisées. Un autre choix populaire est la norme ℓ^1

$$\Omega(\Theta) = \sum_{\boldsymbol{\theta} \in \Theta} ||\boldsymbol{\theta}||_1 = \sum_{\boldsymbol{\theta} \in \Theta} \sum_{i=1}^{l(\theta)} |\theta_i| \ ,$$

qui a la propriété de fixer à la valeur 0 certains des éléments θ_i qui sont moins utiles à la résolution du problème [149].

Comme dans le cas du nombre d'itérations d'entraînement ou des hyper-paramètres tels le nombre de neurones cachés ou le degré d'un noyau polynomial, on observe normalement une courbe en « U » dans la relation entre l'erreur observée en test et la valeur de λ. Plus précisément, la figure 2.1 où l'on aurait remplacé le nombre d'itérations par $\frac{1}{\lambda}$ illustrerait bien le genre de relation que l'on s'attend à observer. De plus, pour certains modèles, il est possible de démontrer qu'utiliser une technique d'arrêt prématuré est équivalent à l'utilisation d'un terme de régularisation. Effectivement, en sélectionnant l'hyper-paramètre du taux d'apprentissage Δ d'un algorithme de descente de gradient utilisant l'arrêt prématuré, on contrôlerait alors indirectement la capacité de façon simi-

laire à la pénalisation ℓ^2 lorsqu'utilisée dans le cadre d'une optimisation complète du critère d'entraînement de l'équation 2.1 [36]. L'intuition derrière cette relation est que les deux approches empêchent l'optimisation d'assigner aux paramètres une valeur trop éloignée de zéro[2]. Voir aussi Amari et al. [3], Cataltepe et al. [31], Hagiwara et Kuno [67], Siöberg et Ljung [143], Wang et al. [156] pour plus de détails sur la relation entre l'arrêt prématuré et la régularisation.

Cette dernière intuition est utile pour comprendre le principe de la régularisation en général. L'idée fondamentale derrière la régularisation est que pénaliser (encourager) certaines configurations des paramètres qui sont présumées comme étant nuisibles (favorables) à la performance de généralisation du modèle poussera la procédure d'entraînement vers des configurations plus avantageuses. Cependant, développer une bonne pénalisation $\Omega(\Theta)$ n'est pas toujours facile. À part l'intuition générale que les paramètres ne devraient pas être trop grands (qui est implémentée par les pénalisations ℓ^1 et ℓ^2), dériver une pénalisation appropriée pour un problème spécifique nécessite une bonne compréhension de celui-ci ainsi que du modèle utilisé.

Heureusement, il existe des techniques d'apprentissage automatique permettant de découvrir une régularisation utile uniquement à partir de données. Les prochaines sections décrivent certaines de ces approches, classées en 3 catégories : l'**apprentissage hybride génératif/discriminant**, l'**apprentissage semi-supervisé** et l'**apprentissage multi-tâche**.

2.1.1 Apprentissage hybride génératif/discriminant

Considérons l'algorithme de classification par régression logistique présenté à la section 1.3.1. Parce que le modèle $f_\Theta(\mathbf{x})$ de régression logistique calcule une estimation de la probabilité conditionnelle $p(y|\mathbf{x})$ et que l'entraînement consiste à ajuster spécifiquement cette estimation (i.e., le coût utilisé est $L(f_\Theta(\mathbf{x}_t), y_t) = -\log f_\Theta(\mathbf{x}_t)_{y_t}$), on dit que cet algorithme utilise un **apprentissage discriminant**.

À l'opposé, considérons le modèle suivant, où $p(\mathbf{x}|y)$ est définie comme étant une

[2]On suppose ici que les paramètres sont initialisés près de zéro avant le début de l'optimisation, ce qui est le cas normalement en pratique.

distribution gaussienne de moyenne $\boldsymbol{\mu}_y$ et matrice de covariance diagonale $\boldsymbol{\Sigma}$:

$$p(\mathbf{x}|y) = \frac{1}{(2\pi)^{\frac{D}{2}}|\boldsymbol{\Sigma}|^{\frac{1}{2}}} e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_y)^{\mathbf{T}}\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu}_y)}$$

où on a un paramètre de moyenne $\boldsymbol{\mu}_y$ spécifique pour chaque classe y mais on utilise la même matrice de covariance diagonale globale $\boldsymbol{\Sigma}$. On utilise aussi une distribution multinomiale pour $p(y)$, correspondant à la probabilité a priori qu'un exemple appartienne à chaque classe. Ce modèle, appelé **modèle de Bayes naïf**, permet donc d'estimer la probabilité jointe d'une entrée et de sa classe $p(y, \mathbf{x}) = p(\mathbf{x}|y)p(y)$. Ainsi, ce modèle peut être entraîné simplement en modifiant les paramètres $\boldsymbol{\mu}_y$ et $\boldsymbol{\Sigma}$ afin de maximiser la vraisemblance des données d'entraînement, ou de façon équivalente en minimisant la log-vraisemblance négative des données d'entraînement. Pour ce faire, il suffit de définir

$$f_\Theta(\mathbf{x}) = (p(\mathbf{x}|y=1)p(y=1), \ldots, p(\mathbf{x}|y=C)p(y=C))$$

et d'utiliser le même coût que la régression logistique, soit :

$$L(f_\Theta(\mathbf{x}_t), y_t) = -\log f_\Theta(\mathbf{x}_t)_{y_t} \ .$$

Bien que ce modèle ne spécifie pas explicitement une estimation de la probabilité conditionnelle $p(y|\mathbf{x})$, il est possible d'obtenir une telle estimation en dérivant la probabilité $p(y|\mathbf{x})$ comme suit :

$$p(y|\mathbf{x}) = \frac{p(\mathbf{x}|y)p(y)}{\sum_{y'=1}^{C} p(\mathbf{x}|y')p(y')} \ .$$

Ainsi, ce modèle peut être utilisé afin de classifier de nouvelles entrées. Parce que ce modèle définit une probabilité jointe d'une entrée et de sa cible associée et que l'entraînement vise explicitement à optimiser cette probabilité jointe, on dit que ce modèle est entraîné par **apprentissage génératif**.

Bien que le type d'apprentissage utilisé par la régression logistique et le modèle de Bayes naïf ne soit pas le même, ces deux modèles sont linéaires. Effectivement, dans les deux cas, la décision de classifier une entrée $\mathbf{x}$ dans la classe y, qui est prise en

choisissant la classe y ayant la probabilité $p(y|\mathbf{x})$ la plus élevée, peut être décrite comme le choix de la classe y telle que pour tout $y^* \neq y$ on a

$$\boldsymbol{\omega}_y^{\mathbf{T}}\mathbf{x} + \beta_y > \boldsymbol{\omega}_{y^*}^{\mathbf{T}}\mathbf{x} + \beta_{y^*} \ . \tag{2.2}$$

La valeur ou paramétrisation des $\boldsymbol{\omega}_y$ et β_y dépend alors du modèle considéré. Dans le cas de la régression logistique, on a directement $\boldsymbol{\omega}_y = \mathbf{W}_{y,\cdot}^{\mathbf{T}}$ et $\beta_y = b_y$, où $\mathbf{W}_{y,\cdot}$ est la rangée numéro y de $\mathbf{W}$. Dans le cas du modèle de Bayes naïf, le dérivation est moins directe. On sait que l'on doit classifier $\mathbf{x}$ dans la classe y lorsque la probabilité $p(y|\mathbf{x})$ est plus grande que $p(y^*|\mathbf{x})$ pour tout $y^* \neq y$. De façon équivalente, on classifie $\mathbf{x}$ dans y lorsque, pour tout $y^* \neq y$, la condition suivante est satisfaite :

$$\begin{array}{rcl}
\log p(y|\mathbf{x}) & > & \log p(y^*|\mathbf{x}) \\
\Updownarrow & & \\
\log p(\mathbf{x}|y)p(y) - \log \sum_{y'=1}^{C} p(\mathbf{x}|y')p(y') & > & \log p(\mathbf{x}|y^*)p(y^*) - \log \sum_{y'=1}^{C} p(\mathbf{x}|y')p(y') \\
\Updownarrow & & \\
\log p(\mathbf{x}|y) + \log p(y) & > & \log p(\mathbf{x}|y^*) + \log p(y^*) \\
\Updownarrow & & \\
\begin{array}{c} -\frac{D}{2}\log 2\pi - \frac{1}{2}\log|\boldsymbol{\Sigma}| \\ -\frac{1}{2}\left(\mathbf{x}-\boldsymbol{\mu}_y\right)^{\mathbf{T}}\boldsymbol{\Sigma}^{-1}\left(\mathbf{x}-\boldsymbol{\mu}_y\right) \\ +\log p(y) \end{array} & > & \begin{array}{c} -\frac{D}{2}\log 2\pi - \frac{1}{2}\log|\boldsymbol{\Sigma}| \\ -\frac{1}{2}\left(\mathbf{x}-\boldsymbol{\mu}_{y^*}\right)^{\mathbf{T}}\boldsymbol{\Sigma}^{-1}\left(\mathbf{x}-\boldsymbol{\mu}_{y^*}\right) \\ +\log p(y^*) \end{array} \\
\Updownarrow & & \\
\begin{array}{c} -\frac{1}{2}\mathbf{x}^{\mathbf{T}}\boldsymbol{\Sigma}^{-1}\mathbf{x} + \boldsymbol{\mu}_y^{\mathbf{T}}\boldsymbol{\Sigma}^{-1}\mathbf{x} \\ -\frac{1}{2}\boldsymbol{\mu}_y^{\mathbf{T}}\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_y + \log p(y) \end{array} & > & \begin{array}{c} -\frac{1}{2}\mathbf{x}^{\mathbf{T}}\boldsymbol{\Sigma}^{-1}\mathbf{x} + \boldsymbol{\mu}_{y^*}^{\mathbf{T}}\boldsymbol{\Sigma}^{-1}\mathbf{x} \\ -\frac{1}{2}\boldsymbol{\mu}_{y^*}^{\mathbf{T}}\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_{y^*} + \log p(y^*) \end{array} \\
\Updownarrow & & \\
\begin{array}{c} \boldsymbol{\mu}_y^{\mathbf{T}}\boldsymbol{\Sigma}^{-1}\mathbf{x} \\ +\left(\log p(y) - \frac{1}{2}\boldsymbol{\mu}_y^{\mathbf{T}}\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_y\right) \end{array} & > & \begin{array}{c} \boldsymbol{\mu}_{y^*}^{\mathbf{T}}\boldsymbol{\Sigma}^{-1}\mathbf{x} \\ +\left(\log p(y^*) - \frac{1}{2}\boldsymbol{\mu}_{y^*}^{\mathbf{T}}\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_{y^*}\right) \end{array}
\end{array} \tag{2.3}$$

où on remarque que l'équation 2.3 est équivalente à l'équation 2.2 si l'on remplace les

termes $\boldsymbol{\mu}_y^{\mathbf{T}}\boldsymbol{\Sigma}^{-1}$ par $\boldsymbol{\omega}_y^{\mathbf{T}}$ et $\left(\log p(y) - \frac{1}{2}\boldsymbol{\mu}_y^{\mathbf{T}}\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_y\right)$ par β_y.

Ainsi, on pourrait penser que ces deux modèles devraient obtenir des performances de généralisation similaires. Cependant, des travaux théoriques et expérimentaux [115] ont démontré qu'en général, l'erreur[3] de classification asymptotique (lorsque qu'on augmente le nombre d'exemples d'entraînement) de la régression logistique est plus basse ou égale à celle d'un modèle de Bayes naïf. Par contre, le modèle de Bayes naïf atteint normalement plus rapidement son erreur asymptotique. Ainsi, on observe que pour un nombre limité d'exemples d'entraînement, le modèle de Bayes naïf (apprentissage génératif) obtient une meilleure performance de généralisation, et que le portrait est inversé lorsqu'on augmente suffisamment le nombre d'exemples d'entraînement. Une analyse différente mais arrivant à une conclusion similaire a aussi été présentée pour l'entraînement de modèles exponentiels discriminants et génératifs [105]. Les travaux de Liang et Jordan [105], Ng et Jordan [115] indiquent aussi que dans le cas (peu probable) où la famille de distribution $p(y, \mathbf{x})$ choisie pour le modèle entraîné inclut la distribution génératrice des exemples, on s'attend à ce que l'apprentissage génératif soit toujours plus performant.

On peut déjà remarquer une similarité entre l'apprentissage génératif et la régularisation. Effectivement, tout comme la régularisation, lorsque peu de données d'entraînement sont disponibles et que les risques de surapprentissage sont élevés, on observe que l'apprentissage génératif permet d'améliorer la performance de généralisation par rapport à celle de l'apprentissage discriminant. Cette relation avec la régularisation peut être rendue plus explicite en remarquant que le risque optimisé en apprentissage génératif peut être séparé en deux termes comme suit :

$$\frac{1}{n}\sum_{(\mathbf{x}_t,y_t)\in\mathcal{D}} -\log p(y_t,\mathbf{x}_t) = \frac{1}{n}\sum_{(\mathbf{x}_t,y_t)\in\mathcal{D}} -\log p(y_t|\mathbf{x}_t) + \frac{1}{n}\sum_{\mathbf{x}_t\in\mathcal{D}_\mathcal{X}} -\log p(\mathbf{x}_t)$$

où le premier est le risque optimisé par l'apprentissage discriminant et le second est un terme n'utilisant que l'ensemble des entrées de $\mathcal{D}$ (noté $\mathcal{D}_\mathcal{X}$). On peut donc obtenir l'approche par apprentissage génératif en changeant la pénalisation $\Omega(\Theta)$ de l'équation 2.1

[3] On parle ici d'erreur en généralisation, et non pas d'erreur sur l'ensemble d'entraînement.

par la pénalisation dépendante des données d'entrée $\mathcal{D}_{\mathcal{X}}$ suivante :

$$\Omega(\Theta, \mathcal{D}_{\mathcal{X}}) = \frac{1}{n} \sum_{\mathbf{x}_t \in \mathcal{D}_{\mathcal{X}}} - \log p(\mathbf{x}_t) \tag{2.4}$$

et en fixant λ à 1. Pour obtenir l'apprentissage discriminant, cette pénalisation aurait alors un poids nul.

Une extension naturelle des apprentissages génératif et discriminant serait de permettre l'utilisation d'une valeur de λ entre 0 et 1 [27]. Cette approche est dite **apprentissage hybride génératif/discriminant** et on observe fréquemment qu'une valeur intermédiaire de λ est optimale du point de vue de la performance de généralisation [26]. L'apprentissage hybride génératif/discriminant est donc un type d'apprentissage combinant l'apprentissage supervisé et non-supervisé.

Il existe aussi plusieurs variantes du même principe général d'apprentissage hybride génératif/discriminant qui ont été développées, sans nécessairement être basées sur l'optimisation du risque régularisé par une pénalisation $\Omega(\Theta, \mathcal{D}_{\mathcal{X}})$. Par exemple, Lasserre et al. [96] ont démontré qu'il était possible d'obtenir de l'apprentissage hybride en définissant un modèle génératif des entrées et cibles comme suit :

$$p(y, \mathbf{x}|\Theta, \Theta') = p(y|\mathbf{x}, \Theta)p(\mathbf{x}|\Theta')$$

où les paramètres $\Theta \cup \Theta'$ du modèle sont donnés explicitement dans la formule ci-dessus, et où Θ et Θ' sont interchangeables dans la mesure où ils contiennent des paramètres de même dimensionnalité. Lasserre et al. [96] proposent donc de séparer les paramètres du modèle en paramètres discriminants Θ et paramètres génératifs Θ', de définir une probabilité a priori $p(\Theta, \Theta')$ afin d'imposer (ou pas, dans lequel cas $p(\Theta, \Theta') = p(\Theta)p(\Theta')$) une certaine similarité entre les deux types de paramètres, et de simplement maximiser la vraisemblance de données d'entraînement selon ce modèle, sans régularisation. Pour une discussion sur la relation entre cette approche et l'approche via l'utilisation d'une régularisation, voir Bouchard [26]. Plusieurs autres approches décrites comme hybrides ont été développées, étant parfois spécifiques à un modèle en particulier [121] ou suffi-

samment générales pour s'appliquer à une vaste gamme de modèles [49, 110].

Finalement, le principe général de l'apprentissage hybride génératif/discriminant n'est pas applicable à un modèle seulement s'il définit des distributions $p(y|\mathbf{x})$ et $p(\mathbf{x})$. L'idée fondamentale derrière l'apprentissage hybride est d'introduire de l'apprentissage non-supervisé via la pénalisation $\Omega(\Theta, \mathcal{D}_{\mathcal{X}})$. C'est cette idée qui est derrière l'algorithme de régression des moindres carrés partiels (« partial least squares ») [54] par exemple, qui tente d'améliorer la performance d'un modèle de régression linéaire en ajoutant un coût de reconstruction de l'entrée à partir d'une représentation linéaire de plus petite dimension.

2.1.2 Apprentissage semi-supervisé

En permettant de réduire le surapprentissage, la régularisation est en quelque sorte une façon de compenser pour le manque de données. Effectivement, l'approche la plus sure pour améliorer la performance de généralisation d'un modèle serait d'augmenter la quantité de données étiquetées utilisées en entraînement. Malheureusement, le coût associé à cette approche est souvent élevé, puisque qu'elle requiert un travail manuel de classification de la part d'humains. Cependant, en plus des données étiquetées $\mathcal{D}$, il est souvent aussi possible d'obtenir un ensemble de données non-étiquetées $\mathcal{U}$, ne contenant que des entrées $\mathbf{x}_t$ sans cibles y_t associées. De plus, on vient de voir qu'il est possible d'améliorer la performance de généralisation d'un modèle en incorporant de l'apprentissage non-supervisé à un critère d'entraînement supervisé. Ainsi, plutôt que de n'utiliser que les entrées de $\mathcal{D}$ dans le critère non-supervisé, on pourrait facilement choisir d'y ajouter les données de $\mathcal{U}$. On dit de cette approche qu'elle procède par **apprentissage semi-supervisé**.

De façon générale, l'apprentissage semi-supervisé consiste à utiliser un ensemble d'exemples non-étiquetés $\mathcal{U}$, en plus de l'ensemble des exemples étiquetés $\mathcal{D}$, afin d'influencer la procédure d'un algorithme d'apprentissage et améliorer sa performance. Tel que décrit ci-haut, dans le cadre spécifique d'un apprentissage génératif ou hybride génératif/discriminant, il suffit d'ajouter l'ensemble $\mathcal{U}$ à l'ensemble $\mathcal{D}$ dans le terme de régularisation de l'équation 2.4, i.e., d'utiliser comme régularisation $\Omega(\Theta, \mathcal{D}_{\mathcal{X}} \cup \mathcal{U})$ afin

d'obtenir un apprentissage semi-supervisé. De la même façon, si une régularisation non-supervisée a déjà été définie (comme c'est le cas pour la régression des moindres carrés partiels [54]), il est possible d'ajouter les données non-étiquetées dans le calcul de cette régularisation. Les travaux de Ando et Zhang [4], Collobert et Weston [37], qui se basent sur l'ajout de « tâches auxiliaires » consistant chacune en la prédiction d'un élément de l'entrée à partir des autres, ont d'ailleurs démontré qu'à l'aide d'une grande quantité de données non-étiquetées la performance de généralisation d'un classifieur linéaire ou d'un réseau de neurones pouvaient être grandement améliorée.

Dans un cadre d'apprentissage purement discriminant, aucune hypothèse n'est posée sur la forme de $p(\mathbf{x})$ par le modèle, ainsi une approche différente doit être suivie. On doit alors dériver soi-même un terme de régularisation n'utilisant que l'information données par $\mathcal{U}$ (et possiblement $\mathcal{D}_{\mathcal{X}}$ aussi) afin d'influencer l'optimisation du modèle. Par exemple, certains travaux de la littérature ont proposé de pénaliser une certaine propriété $r(f(\mathbf{x}_t))$ de la sortie sur les exemples non-étiquetés, en définissant

$$\Omega(\Theta, \mathcal{U}) = \sum_{x_t \in \mathcal{U}} r(f(\mathbf{x}_t)). \tag{2.5}$$

Dans le cas d'un modèle dont la sortie $f(\mathbf{x}_t)$ peut être interprétée comme une distribution sur l'assignation des classes, on peut alors tenter de minimiser l'entropie de cette distribution pour tous les exemples non-étiquetés [63], ce qui encourage l'entraînement d'un modèle f ayant confiance en sa classification sur les exemples non-étiquetés. Une idée similaire a aussi été proposée dans le cadre des machines à vecteurs de support, où l'on souhaite maximiser la marge entre la surface de décision de f et l'assignation des classes la plus favorable possible pour les exemples non-étiquetées [90, 152]. Ces deux approches sont d'ailleurs liées, la deuxième étant un cas limite de la première [64].

Une autre forme populaire de régularisation est la suivante :

$$\Omega(\Theta, \mathcal{U}) = \sum_{\mathbf{x}_t, \mathbf{x}_{t'} \in \mathcal{U}} w(\mathbf{x}_t, \mathbf{x}_{t'}) D(f(\mathbf{x}_t), f(\mathbf{x}_{t'})) \tag{2.6}$$

où $D(f(\mathbf{x}_t), f(\mathbf{x}_{t'}))$ définit une distance entre la sortie pour deux entrées $\mathbf{x}_t$ et $\mathbf{x}_{t'}$, et

$w(\mathbf{x}_t, \mathbf{x}_{t'})$ pondère cette distance selon chaque paire d'entrées. Un choix courant [45, 169] est

$$D(f(\mathbf{x}_t), f(\mathbf{x}_{t'})) = ||f(\mathbf{x}_t) - f(\mathbf{x}_{t'})||^2 \text{ et } w(\mathbf{x}_t, \mathbf{x}_{t'}) = e^{-||\mathbf{x}_t - \mathbf{x}_{t'}||^2}$$

où l'on peut aussi imposer $w(\mathbf{x}_t, \mathbf{x}_{t'}) = 0$ pour toute paire d'entrées $\mathbf{x}_t$ et $\mathbf{x}_{t'}$ qui ne sont pas proches voisins [8]. Ce choix est donc approprié si on croit que la cible à prédire varie peu dans une région dense de l'espace des entrées. Il est à remarquer que cette pénalisation est nulle lorsque le modèle donne la même prédiction pour toute entrée, et que c'est le terme du risque empirique (ainsi qu'un choix de valeur approprié pour le facteur λ) qui rend cette solution inappropriée, du moins sur les données étiquetées. Afin de s'assurer que la sortie du modèle sur les données non-étiquetées ne soit pas constante, un terme additionnel encourageant la diversité des prédictions sur ces données (e.g. en encourageant la distribution marginale de la sortie du modèle à avoir une entropie élevée) pourrait être ajouté.

La littérature sur l'apprentissage semi-supervisé est très vaste et couvre plusieurs autres approches ne se limitant pas à l'utilisation d'une des deux régularisations des équations 2.5 et 2.6. Par exemple, Chapelle et Zien [34] ont proposé d'utiliser les idées derrières ces deux régularisations en même temps. D'autres approches ont été développées sans avoir en tête l'interprétation de l'apprentissage semi-supervisé comme régularisation, mais peuvent tout de même s'y rattacher. Un tel exemple est le ***self-training***, où un algorithme d'apprentissage utilise ses propres prédictions sur des exemples non-étiquetés afin d'élargir son ensemble d'entraînement. Cette idée n'est pas jeune [1, 53, 139], mais a été appliquée avec succès plus récemment dans le cadre de la désambiguïsation de sens [167] et peut être vue comme une régularisation par la minimisation de l'entropie de la prédiction du modèle sur des entrées non-étiquetées [63]. Il y a aussi le ***co-training***, où l'on suppose que les composantes d'entrée $\mathbf{x}_t$ peuvent être divisées en deux parties $\mathbf{x}_t^{(1)}$ et $\mathbf{x}_t^{(2)}$ contenant chacune suffisamment d'information pour bien prédire la cible y_t. En entraînant deux modèles différents sur ces deux représentations, on peut alors utiliser leurs prédictions sur des entrées non-étiquetées afin qu'ils puissent élar-

gir mutuellement leurs ensembles d'entraînement. La régularisation implicite de cette approche force donc les sorties de chaque modèle à être similaires sur les entrées non-étiquetées. Cette régularisation est aussi explicitement utilisée par d'autres approches qui sont dites du ***multi-view learning*** [23, 43].

Une autre approche très répandue dans la littérature est basée sur le principe de transduction (par opposition à l'induction de fonction, qui est le principe derrière les approches vues jusqu'à maintenant), où l'on suppose connaître d'avance l'ensemble des points non-étiquetés pour lesquels on souhaite faire une prédiction et où la classification de toutes ces entrées se fait de façon jointe. C'est le cas des approches à base de graphes [22, 168], où un graphe est construit en associant un nœud à chaque entrée de $\mathcal{D}_{\mathcal{X}}$ et $\mathcal{U}$. On ajoute alors entre certaines paires de nœuds des arêtes pondérées selon une distance entre les entrées associées, puis on détermine une opération sur les graphes qui permettra de propager les cibles associées aux entrées de $\mathcal{D}_{\mathcal{X}}$ vers les nœuds associés aux entrées de $\mathcal{U}$. Malheureusement, ce genre de méthode ne définit pas directement comment classifier une nouvelle entrée présentée après avoir classifié les entrées de $\mathcal{U}$. Une approche possible, proposée par Delalleau et al. [45], est d'utiliser la formule analytique de la prédiction pour une nouvelle entrée $\mathbf{x}^*$, qui aurait été calculée si elle avait été incluse dans $\mathcal{U}$ sous la contrainte que les prédictions pour tous les points de l'ensemble d'entraînement $\mathcal{D} \cup \mathcal{U}$ soient celles obtenues sans $\mathbf{x}$ (i.e., on suppose que l'ajout de $\mathbf{x}$ ne change pas drastiquement les prédictions faites pour ces exemples). Typiquement, on obtient alors une formule pour f similaire à celle obtenue avec une machine à noyau, mais contenant des termes additionnels pour chacune des entrées de $\mathcal{U}$.

2.1.3 Apprentissage multi-tâche

En plus de données non-étiquetées, il arrive parfois que des ensembles d'exemples étiquetés $\{\mathcal{D}^{(i)}\}_{i=1}^{l}$ **associés à d'autres tâches** soient aussi disponibles. L'**apprentissage multi-tâche** [7, 29, 30] tente ainsi d'exploiter ces données, en plus de l'ensemble des exemples étiquetés $\mathcal{D}$ dédiés à la tâche qui nous intéresse, afin d'influencer la procédure d'un algorithme d'apprentissage et améliorer sa performance. Une façon simple de le

faire consiste à ajouter un terme de régularisation exploitant ces données comme suit :

$$\Omega\left(\Theta, \{\mathcal{D}^{(i)}\}_{i=1}^{l}\right) = \sum_{i=1}^{l} \lambda_i \frac{1}{n_i} \sum_{(\mathbf{x}_t, y_t) \in \mathcal{D}^{(i)}} L_i(f_\Theta(\mathbf{x}_t), y_t)$$

où λ_i permet de contrôler l'importance de l'apprentissage des tâche auxiliaires, les coûts $L_i(f_\Theta(\mathbf{x}_t), y_t)$ sont choisis en fonction des différentes tâches et n_i est le nombre d'exemples contenus dans $\mathcal{D}^{(i)}$. La sortie du modèle $f_\Theta(\mathbf{x}_t)$ doit donc fournir une prédiction pour chacune des tâches à résoudre, et chaque coût $L_i(f_\Theta(\mathbf{x}_t), y_t)$ agit alors sur différents éléments de la sortie.

Bien que popularisé dans le contexte des réseaux de neurones par Caruana [29], ce principe peut être utilisé par plusieurs autres modèles, par exemple par les classifieurs k plus proches voisins, les régresseurs à noyau et les arbres de décisions [30], ainsi que les machines à vecteurs de support à noyau [51].

Il est aussi possible de dériver un algorithme d'apprentissage semi-supervisé à partir d'un algorithme d'apprentissage multi-tâche, en construisant automatiquement des tâches auxiliaires à partir de l'entrée et en utilisant des ensembles de données non-étiquetées pour générer les ensembles $\mathcal{D}^{(i)}$. C'est essentiellement ce qu'ont proposé Ando et Zhang [4], pour qui les tâches auxiliaires consistent en la prédiction d'une entrée étant donnée la valeur des autres.

2.2 Fléau de la dimensionnalité et généralisation non-locale

On a vu jusqu'à maintenant qu'il est possible de réduire les effets néfastes du surapprentissage à l'aide d'un terme de régularisation influencé par des données d'entraînement. Ceci permet d'éviter d'avoir à trouver les raisons pour lesquelles un modèle surapprend. Mais bien que ces raisons ne soient pas toutes connues, il y a tout de même certains aspects spécifiques d'un problème d'apprentissage qui ont été identifiés par les chercheurs en apprentissage automatique comme posant des obstacles à la généralisation. Un de ces obstacles bien connu est le **fléau de la dimensionnalité**.

Le fléau de la dimensionnalité fait référence à la difficulté qu'ont plusieurs algo-

rithmes d'apprentissage à bien généraliser à de nouvelles entrées lorsqu'elles appartiennent à un espace de haute dimension. Pour fins d'explication, imaginons un problème de classification où chaque entrée $\mathbf{x}_t$ appartient à un hypercube $\mathcal{X} = [0,1]^D$ et est associée à une cible $y_t \in \{-1, 1\}$ selon une relation qui n'est pas connue. Imaginons maintenant un algorithme d'apprentissage automatique simple qui retourne un modèle f faisant une prédiction de classification comme suit. Ce modèle divise d'abord chaque axe de $\mathcal{X}$ en m morceaux, formant m^D petits hypercubes. Étant donné une nouvelle entrée $\mathbf{x}$, ce modèle trouve dans quel hypercube l'entrée $\mathbf{x}$ appartient et retourne l'étiquette associée au plus grand nombre d'exemples d'apprentissage appartenant à cet hypercube. Dans le cas d'une égalité, une étiquette dans $\mathcal{Y}$ est tirée aléatoirement et est retournée.

Pour que ce modèle estime bien la relation réelle entre les entrées et leur cible, il est nécessaire qu'au moins une entrée de l'ensemble d'entraînement se trouve dans chacun des hypercubes. Ainsi, le nombre d'exemples d'apprentissage nécessaire pour que cet algorithme généralise bien grandit de façon exponentielle en fonction de la dimension D de l'espace. Ceci est un constat plutôt catastrophique pour les problèmes où D est grand, d'où le nom « fléau de la dimensionnalité ». Il faut cependant remarquer qu'il est possible que certains des hypercubes soient vides sans que la performance de généralisation en soit affectée, si cet hypercube correspond à une région de l'espace où on ne s'attend à observer aucune entrée $\mathbf{x}$. Pour cette raison, le fléau de la dimensionnalité peut ne pas s'appliquer à la dimensionnalité explicite de l'entrée (D) mais plutôt à la dimensionnalité implicite de la variété (voir la section 1.1.2) autour de laquelle proviennent les entrées en général. De plus, même si la dimensionnalité implicite des données est élevée, la tâche de classification peut tout de même être relativement simple à résoudre. Ceci pourrait être le cas si la surface de décision à apprendre était simple ou si le nombre de régions associées aux différentes classes n'était pas trop grand. Dans un tel cas, l'algorithme simpliste ci-haut pourrait regarder dans les cibles des exemples d'entraînement dans les hypercubes voisins aux hypercubes vides afin de faire une prédiction ayant une chance relativement élevée d'être correcte. Pour cette raison, Bengio et al. [12] parlent plutôt du « fléau des fonctions hautement variables ».

Bien que le modèle donné en exemple ci-haut puisse paraître simpliste, on observe

en pratique que le fléau de la dimensionnalité pose problème à bien d'autres algorithmes d'apprentissage répandus dans la littérature. Pour certains algorithmes, il est même possible de faire une analyse théorique de l'impact du fléau de la dimensionnalité. Par exemple, il a été démontré que le modèle des fenêtres de Parzen (*Parzen windows*, une machine à noyau gaussien pour l'estimation de densité) [119] peut nécessiter dans le pire des cas un nombre d'exemples de l'ordre de $m^{(4+D)/5}$ afin d'obtenir une erreur en dimension D équivalente à celle obtenue par le même modèle en dimension 1 entraîné avec m exemples d'apprentissage (voir [69, 142]). Bengio et al. [12] présentent aussi différents arguments théoriques illustrant que le fléau de la dimensionnalité peut être nocif pour les machines à noyau gaussien en général, comme les machines à vecteurs de support.

Une des raisons pour laquelle tous ces modèles sont victimes de ce fléau vient du fait que leur capacité à généraliser dans une région de l'espace dépend du nombre d'exemples d'entraînement situés dans cette région. Lorsque ces modèles tentent un prédiction pour une entrée $\mathbf{x}$, ce sont effectivement les exemples situés dans le voisinage de cet exemple qui seront utilisés pour calculer cette prédiction. On qualifie une telle façon de généraliser de **locale**.

À l'opposé, un algorithme qui ne possède pas ce handicap est dit à généralisation **non-locale**. Tout algorithme correspondant à un modèle linéaire en est un exemple. Effectivement, une transformation linéaire $f(\mathbf{x}) = \mathbf{A}\mathbf{x}$ peut entre autres ignorer une partie d'une entrée $\mathbf{x}$ en fixant certaines colonnes de $\mathbf{A}$ à 0. Imaginons un problème de régression où la cible y d'une entrée $\mathbf{x}$ est indépendante de la première coordonnée de $\mathbf{x}$, un algorithme d'apprentissage pourrait alors apprendre à fixer la première colonne de $\mathbf{A}$ à 0 et ainsi bien généraliser même à des entrées dont la première composante de bruit est différente de toutes celles des exemples d'entraînement (voir la figure 2.2 pour une autre exemple). De plus, puisqu'un réseau de neurones artificiel utilise normalement une représentation distribuée (voir section 1.3.2), où chaque neurone correspond à une transformation linéaire suivie d'une transformation non-linéaire, on considère un tel modèle comme étant non-local aussi. Une machine à noyau peut également généraliser de façon non-locale, à condition qu'elle utilise un noyau $\mathrm{k}(\mathbf{x}_t, \mathbf{x}_{t'})$ lui aussi non-local, i.e., un

noyau dont la valeur ne décroît pas inévitablement vers 0 lorsque la distance entre les deux entrées $\mathbf{x}_t$ et $\mathbf{x}_{t'}$ augmente. Autrement, si le noyau est dit local (comme c'est le cas avec le noyau gaussien), la généralisation sera elle aussi locale.

Il est à noter que le nature de la généralisation non-locale dépend grandement du modèle. Dans le cas d'un réseau de neurones, sa capacité de généralisation non-locale permet (entre autres) d'ignorer la valeur de certaines composantes d'entrée. Si d'autres types de régularité du problème d'apprentissage ne peuvent être modélisés de façon compacte à l'aide du genre de généralisation non-locale utilisée par le modèle, la généralisation non-locale ne sera pas utile. Cependant, la capacité d'ignorer certaines composantes d'entrée risque fortement d'être utile dans des problèmes où des composantes de bruit ou non pertinentes peuvent être observées (e.g., pour des problèmes de vision, de traitement de la langue ou de traitement sonore), ainsi on peut s'attendre à ce que ce type de généralisation non-locale soit utile.

Ainsi, une première façon simple de diminuer les conséquences du fléau de la dimensionnalité est d'utiliser un modèle à généralisation non-locale. Une technique alternative serait aussi de tenter de s'attaquer directement au fléau de la dimensionnalité à l'aide d'un modèle de réduction de dimensionnalité (voir la section 1.1.2) et ainsi compresser l'information contenue par une entrée dans un moins grand nombre de dimensions. Pour qu'une telle approche fonctionne bien cependant, il est nécessaire que le modèle de réduction de dimensionnalité généralise bien lui aussi à de nouvelles entrées. Et puisque ces modèles prennent souvent la forme d'une machine à noyau local [17], on observe qu'ils éprouvent de la difficulté à bien généraliser [16]. Pour cette raison, il est plus souvent proposé de réduire la dimensionnalité de l'entrée de façon linéaire. Cependant, une telle transformation linéaire risque de ne pas avoir suffisamment de capacité pour bien compresser l'information de l'entrée dans un petit nombre de dimensions, et pourrait ainsi éliminer de l'information utile pour le problème à résoudre. Plusieurs approches ont donc été développées afin d'assurer que la transformation linéaire préserve l'information utile à la résolution du problème [61, 62, 159, 160]. Ce faisant, le modèle en arrive à ressembler un peu plus à un réseau de neurones, calculant une représentation intermédiaire qui sera utile à la résolution de la tâche supervisée. Ainsi, considérer l'uti-

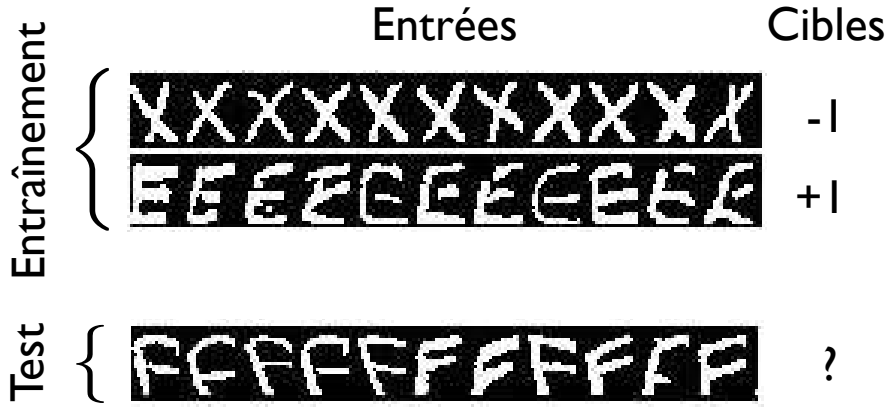


Figure 2.2 – Illustration d'un exemple de problème nécessitant une généralisation non-locale. Le problème consiste à indiquer si l'image d'un caractère contient un trait horizontal dans le bas de l'image ($y = +1$) ou pas ($y = -1$). Dans l'ensemble d'entraînement, on trouve des exemples positifs (caractères «E») et négatifs (caractères «X») qui sont très différents des exemples de l'ensemble de test (caractères «F», pour lesquels $y = -1$). Typiquement, un modèle à généralisation locale fera alors une mauvaise prédiction ($y = +1$) en test, puisque pour l'image d'un «F», le nombre de pixels communs (i.e., ayant la même valeur) avec un «E» est en général plus grand qu'avec un «X». Un modèle à généralisation non-locale cependant pourrait être en mesure d'ignorer les pixels qui ne sont pas mis en cause par la relation entre l'entrée et la cible.

lisation d'un modèle non-local dès le départ peut s'avérer être plus direct et plus simple.

Il est à noter qu'un modèle à généralisation non-locale n'est pas pour autant assuré d'avoir une bonne performance en haute dimension. Effectivement, plus la dimensionnalité des entrées d'un problème est élevée, plus il est vraisemblable que le modèle devra contenir un grand nombre de paramètres pour résoudre ce problème et ainsi être plus vulnérable au surapprentissage. La régularisation jouera donc quand même un rôle primordial dans l'obtention d'une bonne performance de généralisation.

2.3 Découverte d'une représentation hiérarchique

On vient de voir qu'une caractéristique parfois utile pour favoriser une bonne performance de généralisation dans un modèle est de s'assurer qu'il puisse généraliser de façon non-locale. De cette façon, on impose une certaine contrainte sur le type de cal-

cul effectué par le modèle. Ainsi, on peut se demander s'il existe d'autres contraintes qui puissent favoriser la généralisation. Plus particulièrement, pouvons-nous trouver de telles contraintes si l'on se limite aux problèmes normalement associés à un comportement intelligent de la part de l'humain, soit dans les domaines de la vision, du traitement de la langue ou de signaux sonores ?

Effectivement, une caractéristique souvent observée dans les données de vision, langagières ou sonores est leur organisation sous une forme hiérarchique. Par exemple, plusieurs objets prennent (approximativement) la forme de polyèdres formés de plusieurs faces, elles-mêmes formées d'arêtes. De la même façon, une phrase est composée de mots, qui eux sont produits à l'aide de lettres. Le signal sonore d'une pièce musicale lui sera décomposable en différents accords ou notes joués, qui peuvent à leur tour être décomposés en plusieurs signaux simples de fréquence unique. L'illustration de telles décompositions hiérarchiques est présentée à la figure 2.3.

D'ailleurs, des travaux consacrés à l'étude des différentes régions du cerveau et de leur fonction avancent que le système visuel humain serait organisé de façon hiérarchique en différentes régions du cerveau (régions V1 jusqu'à V5). La région recevant le plus directement ses stimuli de l'œil (V1) aurait la tâche de détecter des patrons visuels simples, soit des contours ou « edges » [85]. Puis, la région suivante (V2) dans la hiérarchie détecteraient des patrons plus complexes, entre autres des paires ou triplets de traits joints par différents angles et autres formes plus complexes [72, 87]. Ce processus de complexification des patrons détectés continuerait ensuite en montant dans la hiérarchie. De plus, cette interprétation des observations expérimentales est renforcée par plusieurs travaux en apprentissage automatique ayant démontré qu'il est possible d'entraîner, de façon non-supervisée et sur un ensemble de données d'images naturelles, des modèles détectant le même genre de patrons que ceux détectés dans V1 [116] et V2 [103].

Ainsi, pour ce type de problèmes, il semblerait que l'utilisation d'un modèle ayant une **architecture profonde**, i.e., nécessitant le calcul de plusieurs niveaux de représentations de plus en plus abstraites pourrait être bénéfique. Au delà de la similarité d'une telle architecture avec l'organisation de certaines régions du cerveau, il existe aussi des arguments plus théoriques (présentés à la section 6.3) appuyant l'utilisation d'une archi-

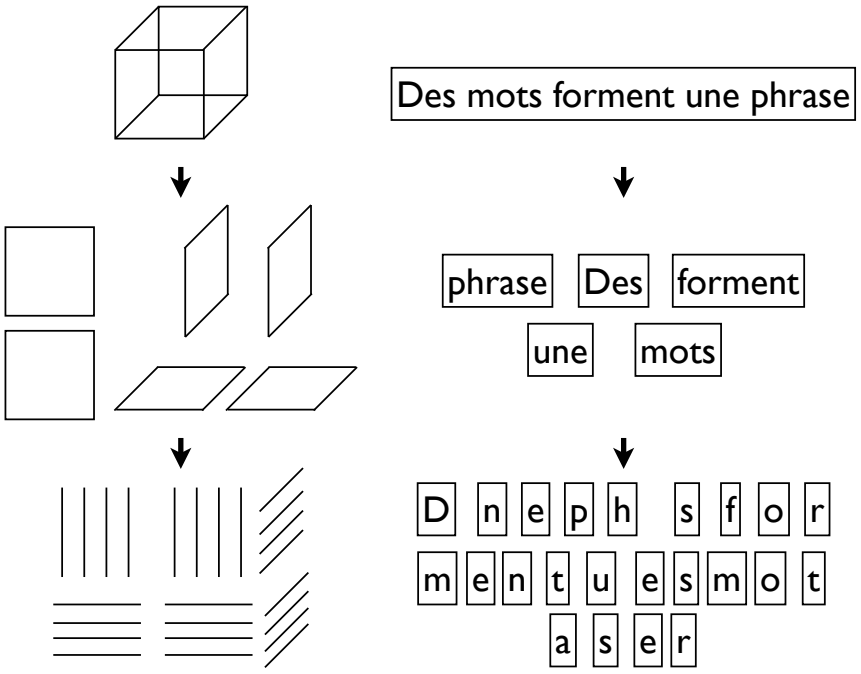


Figure 2.3 – Illustration de la décomposition hiérarchique d'un représentation d'un objet en surfaces et en arêtes (à gauche) et d'une phrase en mots et en lettres (à droite).

tecture profonde.

Dans le cadre de cette thèse, c'est donc cette approche qui sera privilégiée. Plus particulièrement, on cherchera à utiliser ce principe dans le cadre des réseaux de neurones artificiels. Leur efficacité de calcul, leur puissance de représentation ainsi que leur utilisation de représentations distribuées font des réseaux de neurones une classe de modèles de choix pour le développement d'un modèle à architecture profonde. Malheureusement, entraîner un réseau de neurones à plusieurs couches cachées est un problème difficile. En plus de la complexité de l'optimisation non-convexe liée à leur entraînement, l'espace des représentations hiérarchiques paramétrisées par ce genre de réseau est tellement vaste que des problèmes de surapprentissage peuvent facilement survenir lorsque l'ensemble des données d'entraînement n'est pas suffisamment grand pour bien identifier la meilleure ou même simplement une bonne représentation hiérarchique. Dans un tel cas, l'utilisation de la régularisation devrait s'avérer essentielle à l'apprentissage d'une

telle représentation. Le prochain chapitre se concentre donc sur l'expérimentation d'une technique de régularisation pour un réseau de neurones simple, basée sur l'apprentissage hybride génératif/discriminant.

CHAPITRE 3

PRÉSENTATION DU PREMIER ARTICLE

3.1 Détails de l'article

Classification using Discriminative Restricted Boltzmann Machines
Hugo Larochelle et Yoshua Bengio
Publié dans *Proceedings of the 25th Annual International Conference on Machine Learning, Omnipress* en 2008.

3.2 Contexte

Généralement, les machines à noyau sont considérées comme formant l'état de l'art pour la résolution d'un problème de classification. En particulier, la machine à vecteurs de support à noyau gaussien est souvent le premier modèle dont on considère l'utilisation pour un problème de classification quelconque. En plus d'avoir démontré une capacité de généralisation impressionnante pour une vaste gamme de problèmes, ce modèle offre une certaine stabilité d'utilisation, grâce entre autres à sa qualité d'approximateur universel et à sa capacité à être entraîné à l'aide d'une optimisation convexe. Par contre, on a vu à la section 2.2 que par sa généralisation locale, ce modèle pouvait facilement être soumis au fléau de la dimensionnalité pour des problèmes où la dimensionnalité D des entrées est grande.

Une alternative logique à l'utilisation d'une machine à vecteurs de support serait alors de considérer un réseau de neurones à une couche cachée. Bien que son entraînement soit non-convexe, ce modèle est aussi un approximateur universel mais surtout peut généraliser de façon non-locale. De plus, un réseau de neurones peut apprendre directement une représentation utile des données plutôt que de se faire imposer une représentation précise comme dans toute machine à noyau. Malheureusement, on observe souvent en pratique une moins bonne performance de généralisation chez les réseaux de neurones. Une explication possible est que l'entraînement d'un réseau de neurones soit

trop vulnérable au surapprentissage pour des tailles de couche cachée suffisamment importantes, puisque l'espace des représentations utiles ainsi paramétrisé est alors lui aussi trop grand. On doit alors se restreindre à un nombre limité de neurones cachés, ce qui réduit la complexité des problèmes que peut résoudre le réseau de neurones. Une autre approche fréquemment utilisée consiste à utiliser une régularisation générique (e.g., une pénalisation de la norme ℓ^2 des poids ou une procédure d'arrêt prématuré), mais le succès d'une telle approche est limité par la nature non-informative (i.e., indépendante du problème d'apprentissage) de la régularisation.

Parallèlement, une classe spécifique de réseaux de neurones, les machines de Boltzmann restreintes, ont émergé de l'apprentissage automatique comme une approche performante pour l'apprentissage de représentations utiles de données. Ce type de modèle est typiquement basé sur l'apprentissage non-supervisé, et est normalement utilisé dans une approche d'extraction de caractéristiques (voir section 1.1.2) en fournissant un prétraitement bénéfique des données à un algorithme d'apprentissage supervisé quelconque.

3.3 Contributions

Dans cet article, on présente une façon simple et efficace d'entraîner une machine de Boltzmann restreinte de façon supervisée, en tirant profit de la régularisation apportée par l'apprentissage hybride génératif/discriminant, soit une régularisation adaptée aux données du problème. Les expériences conduites dans le cadre de cet article démontrent que l'algorithme d'apprentissage proposé obtient une meilleure performance qu'une machine à vecteurs de support à noyau gaussien ou qu'un réseau de neurone standard. Cette amélioration de la performance semble être obtenue spécifiquement grâce à la régularisation hybride, qui permet d'entraîner un modèle utilisant une représentation plus grande qu'il aurait été possible sans surapprentissage dans le cas non régularisé. De plus, cet algorithme contient moins d'hyper-paramètres que l'approche originale consistant à utiliser une machine de Boltzmann restreinte seulement comme prétraitement pour un autre algorithme.

3.4 Commentaires

Tout d'abord, il est à remarquer que la notation utilisée dans chacun des articles de cette thèse n'est pas toujours exactement la même que celle utilisée dans le reste du texte de la thèse. Ainsi, il est important de porter attention à la définition interne de la notation de chaque article. De plus, le texte des articles insérés dans cette thèse diffère de leur version originalement publiée, puisque des ajustements ont dû être faits afin de répondre aux commentaires formulés par les membres du jury de cette thèse.

Ensuite, il y a certains anachronismes volontaires dans l'ordre de présentation des articles de cette thèse. Dans le cas présent par exemple, cet article a été publié après celui du chapitre 8 (il y fait d'ailleurs référence). Effectivement, l'ordre de présentation des articles est plutôt motivé par l'obtention d'un exposé des travaux de cette thèse présentant progressivement et de façon cohérente les différentes notions étudiées par cette recherche.

Cet article souligne particulièrement bien le rôle primordial joué par l'apprentissage non-supervisé dans l'obtention d'une représentation utile des données, un concept essentiel à la compréhension des autres articles de cette thèse. Les résultats comparatifs en apprentissage semi-supervisé de cet article permettent aussi de mettre en évidence les difficultés des machines à noyau à bien généraliser en haute dimension, tel que prédit par les travaux de Bengio et al. [12].

De plus, bien que la suite de cette thèse explorera l'utilisation de plus d'une couche cachée, la facilité d'utilisation de l'algorithme d'apprentissage proposé (liée entre autres au nombre limité d'hyper-paramètres utilisés) font de cette approche une alternative appropriée à des problèmes pour lesquels une seule couche cachée est suffisante.

Finalement, suite à la publication de cet article, les travaux de Williams et Hinton [164] ont été portés à mon attention, De façon similaire, ceux-ci proposent d'entraîner une machine de Boltzmann de façon discriminante. Cependant, leur approche utilise uniquement l'apprentissage discriminant, et ne tire pas profit de la régularisation apportée par l'apprentissage génératif.

CHAPTER 4

CLASSIFICATION USING DISCRIMINATIVE RESTRICTED BOLTZMANN MACHINES

4.1 Abstract

Recently, many applications for Restricted Boltzmann Machines (RBMs) have been developed for a large variety of learning problems. However, RBMs are usually used as feature extractors for another learning algorithm or to provide a good initialization for deep feed-forward neural network classifiers, and are not considered as a stand-alone solution to classification problems. In this paper, we argue that RBMs provide a self-contained framework for deriving competitive non-linear classifiers. We present an evaluation of different learning algorithms for RBMs which aim at introducing a discriminative component to RBM training and improve their performance as classifiers. This approach is simple in that RBMs are used directly to build a classifier, rather than as a stepping stone. Finally, we demonstrate how discriminative RBMs can also be successfully employed in a semi-supervised setting.

4.2 Introduction

Restricted Boltzmann Machines (RBMs) [144] are generative models based on latent (usually binary) variables to model an input distribution, and have seen their applicability grow to a large variety of problems and settings in the past few years. From binary inputs, they have been extended to model various types of input distributions [81, 162]. Conditional versions of RBMs have also been developed for collaborative filtering [135] and to model motion capture data [147] and video sequences [146].

RBMs have been particularly successful in classification problems either as feature extractors for text and image data [60] or as a good initial training phase for deep neural network classifiers [74]. However, in both cases, the RBMs are merely the first step of another learning algorithm, either providing a preprocessing of the data or an initializa-

tion for the parameters of a neural network. When trained in an unsupervised fashion, RBMs provide no guarantees that the features implemented by their hidden layer will ultimately be useful for the supervised task that needs to be solved. More practically, model selection can also become problematic, as we need to explore jointly the space of hyper-parameters of both the RBM (size of the hidden layer, learning rate, number of training iterations) and the supervised learning algorithm that is fed the learned features. In particular, having two separate learning phases (feature extraction, followed by classifier training) can be problematic in an online learning setting.

In this paper, we argue that RBMs can be used successfully as stand-alone non-linear classifiers alongside other standard classifiers like neural networks and Support Vector Machines, and not only as feature extractors. We investigate training objectives for RBMs that are more appropriate for training classifiers than the common generative objective. We describe Discriminative Restricted Boltzmann Machines (DRBMs), i.e. RBMs that are trained more specifically to be good classification models, and Hybrid Discriminative Restricted Boltzmann Machines (HDRBMs) which explore the space between discriminative and generative learning and can combine their advantages. We also demonstrate that RBMs can be successfully adapted to the common semi-supervised learning setting [33] for classification problems. Finally, the algorithms investigated in this paper are well suited for online learning on large datasets.

4.3 Restricted Boltzmann Machines

Restricted Boltzmann Machines are undirected generative models that use a layer of hidden variables to model a distribution over visible variables. Though they are most often trained to only model the inputs of a classification task, they can also model the joint distribution of the inputs and associated target classes (e.g. in the last layer of a Deep Belief Network in Hinton et al. [81]). In this section, we will focus on such joint models.

We assume given a training set $\mathcal{D}_{train} = \{(\mathbf{x}_i, y_i)\}$, comprising for the i-th example an input vector $\mathbf{x}_i$ and a target class $y_i \in \{1, \ldots, C\}$. To train a generative model on

such data we consider minimization of the negative log-likelihood

$$\mathcal{L}_{gen}(\mathcal{D}_{train}) = -\sum_{i=1}^{|\mathcal{D}_{train}|} \log p(y_i, \mathbf{x}_i). \tag{4.1}$$

An RBM with n hidden units is a parametric model of the joint distribution between a layer of hidden variables (referred to as neurons or features) $\mathbf{h} = (h_1, \ldots, h_n)$ and the observed variables made of $\mathbf{x} = (x_1, \ldots, x_d)$ and y, that takes the form

$$p(y, \mathbf{x}, \mathbf{h}) \quad \propto \quad e^{-E(y,\mathbf{x},\mathbf{h})}$$

where

$$E(y, \mathbf{x}, \mathbf{h}) \quad = \quad -\mathbf{h^T W x} - \mathbf{b^T x} - \mathbf{c^T h} - \mathbf{d^T}\vec{y} - \mathbf{h^T U}\vec{y}$$

with parameters $\Theta = (\mathbf{W}, \mathbf{b}, \mathbf{c}, \mathbf{d}, \mathbf{U})$ and $\vec{y} = (1_{y=i})_{i=1}^{C}$ for C classes. This model is illustrated in Figure 4.3. For now, we consider for simplicity binary input variables, but the model can be easily generalized to non-binary categories, integer-valued, and continuous-valued inputs [81, 162]. It is straightforward to show that

$$\begin{aligned} p(\mathbf{x}|\mathbf{h}) &= \prod_i p(x_i|\mathbf{h}) \\ p(x_i = 1|\mathbf{h}) &= \text{sigmoid}(b_i + \sum_j W_{ji} h_j) \qquad (4.2) \\ p(y|\mathbf{h}) &= \frac{e^{d_y + \sum_j U_{jy} h_j}}{\sum_{y^*} e^{d_{y^*} + \sum_j U_{jy^*} h_j}} \qquad (4.3) \end{aligned}$$

where sigmoid is the logistic sigmoid. Equations 4.2 and 4.3 illustrate that the hidden units are meant to capture predictive information about the input vector as well as the

target class. $p(\mathbf{h}|y, \mathbf{x})$ also has a similar form:

$$\begin{aligned} p(\mathbf{h}|y, \mathbf{x}) &= \prod_j p(h_j|y, \mathbf{x}) \\ p(h_j = 1|y, \mathbf{x}) &= \text{sigmoid}(c_j + U_{jy} + \sum_i W_{ji} x_i). \end{aligned}$$

When the number of hidden variables is fixed, an RBM can be considered a parametric model, but when it is allowed to vary with the data, it becomes a non-parametric model. In particular, Freund et Haussler [55], Le Roux et Bengio [97] showed that an RBM with enough hidden units can represent any distribution over binary vectors, and that adding hidden units guarantees that a better likelihood can be achieved, unless the generated distribution already equals the training distribution.

In order to minimize the negative log-likelihood (eq. 4.1), we would like an estimator of its gradient with respect to the model parameters. The exact gradient, for any parameter $\theta \in \Theta$ can be written as follows:

$$\begin{aligned} \frac{\partial \log p(y_i, \mathbf{x}_i)}{\partial \theta} &= -\mathbb{E}_{\mathbf{h}|y_i, \mathbf{x}_i} \left[\frac{\partial}{\partial \theta} E(y_i, \mathbf{x}_i, \mathbf{h}) \right] \\ &\quad + \mathbb{E}_{y, \mathbf{x}, \mathbf{h}} \left[\frac{\partial}{\partial \theta} E(y, \mathbf{x}, \mathbf{h}) \right]. \end{aligned}$$

Though the first expectation is tractable, the second one is not. Fortunately, there exists a good stochastic approximation of this gradient, called the contrastive divergence gradient [77]. This approximation replaces the expectation by a sample generated after a limited number of Gibbs sampling iterations, with the sampler's initial state for the visible variables initialized at the training sample $(y_i, \mathbf{x}_i)$. Even when using only one Gibbs sampling iteration, contrastive divergence has been shown to produce only a small bias for a large speed-up in training time [28].

Online training of an RBM thus consists in cycling through the training examples and updating the RBM's parameters according to Algorithm 1, where the learning rate is controlled by λ.

Computing $p(y, \mathbf{x})$ is intractable, but it is possible to compute $p(y|\mathbf{x})$, sample from

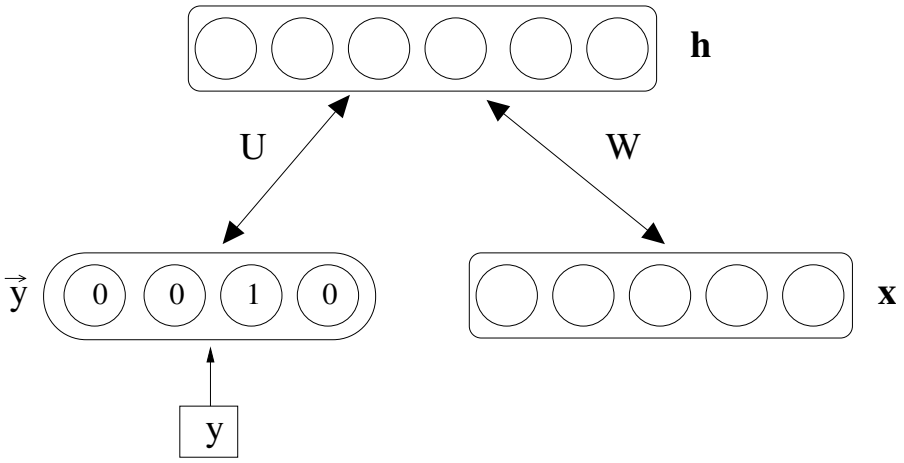


Figure 4.1: Restricted Boltzmann Machine modeling the joint distribution of inputs and target classes

it, or choose the most probable class under this model. As shown in Salakhutdinov et al. [135], for reasonable numbers of classes C (over which we must sum), this conditional distribution can be computed exactly and efficiently, by writing it as follows:

$$p(y|\mathbf{x}) = \frac{e^{d_y} \prod_{j=1}^{n} \left(1 + e^{c_j + U_{jy} + \sum_i W_{ji} x_i}\right)}{\sum_{y^*} e^{d_{y^*}} \prod_{j=1}^{n} \left(1 + e^{c_j + U_{jy^*} + \sum_i W_{ji} x_i}\right)}. \tag{4.4}$$

Precomputing the terms $c_j + \sum_i W_{ji} x_i$ and reusing them when computing

$$\prod_{j=1}^{n} \left(1 + e^{c_j + U_{jy^*} + \sum_i W_{ji} x_i}\right)$$

for all classes y^* permits to compute this conditional distribution in time $O(nd + nC)$. Also, if $p(y|\mathbf{x})$ must be computed for a set of inputs given a fixed RBM, more computational savings can be made by leveraging the fact that the parameters of the RBM are not changing between evaluations of $p(y|\mathbf{x})$. Precomputing the terms $e^{U_{jy^*}}$ for all $j \in \{1, \ldots, n\}$ and for all classes, they can be reused in the computation of $p(y|\mathbf{x})$ for the different inputs. Only the terms $e^{c_j + \sum_i W_{ji} x_i}$ must be computed individually for each input, and only for $j \in \{1, \ldots, n\}$ (no dependence on the number of classes). Then, by multiplying the appropriate exponentiated terms, the terms $e^{c_j + U_{jy^*} + \sum_i W_{ji} x_i}$ can be

Algorithm 1 Training update for RBM over $(y, \mathbf{x})$ using Contrastive Divergence

Input: training pair $(y_i, \mathbf{x}_i)$ and learning rate λ
% Notation: $a \leftarrow b$ means a is set to value b
% $a \sim p$ means a is sampled from p

% Positive phase
$y^0 \leftarrow y_i$, $\mathbf{x}^0 \leftarrow \mathbf{x}_i$, $\widehat{\mathbf{h}}^0 \leftarrow \text{sigmoid}(\mathbf{c} + W\mathbf{x}^0 + \mathbf{U}\vec{y^0})$

% Negative phase
$\mathbf{h}^0 \sim p(\mathbf{h}|y^0, \mathbf{x}^0)$, $y^1 \sim p(y|\mathbf{h}^0)$, $\mathbf{x}^1 \sim p(\mathbf{x}|\mathbf{h}^0)$
$\widehat{\mathbf{h}}^1 \leftarrow \text{sigmoid}(\mathbf{c} + W\mathbf{x}^1 + \mathbf{U}\vec{y^1})$

% Update
for $\theta \in \Theta$ **do**
  $\theta \leftarrow \theta - \lambda \left(\frac{\partial}{\partial \theta} E(y^0, \mathbf{x}^0, \widehat{\mathbf{h}}^0) - \frac{\partial}{\partial \theta} E(y^1, \mathbf{x}^1, \widehat{\mathbf{h}}^1) \right)$
end for

obtained and the total number of costly e^a operations can be reduced.

4.4 Discriminative Restricted Boltzmann Machines

In a classification setting, one is ultimately only interested in correct classification, not necessarily to have a good $p(\mathbf{x})$. Because our model's $p(\mathbf{x})$ can be inappropriate, it can then be advantageous to optimize directly $p(y|\mathbf{x})$ instead of $p(y, \mathbf{x})$:

$$\mathcal{L}_{disc}(\mathcal{D}_{train}) = -\sum_{i=1}^{|\mathcal{D}_{train}|} \log p(y_i|\mathbf{x}_i). \tag{4.5}$$

We refer to RBMs trained according to $\mathcal{L}_{disc}$ as Discriminative RBMs (DRBMs). Since RBMs (with enough hidden units) are universal approximators for binary inputs, it follows also that DRBMs are universal approximators of conditional distributions with binary inputs.

A DRBM can be trained by contrastive divergence, as has been done in conditional RBMs [147], but since $p(y|\mathbf{x})$ can be computed exactly, we can compute the exact gra-

dient:

$$\begin{aligned}\frac{\partial \log p(y_i|\mathbf{x}_i)}{\partial \theta} &= \sum_j \text{sigmoid}(o_{yj}(\mathbf{x}_i))\frac{\partial o_{yj}(\mathbf{x}_i)}{\partial \theta} \\ &- \sum_{j,y^*} \text{sigmoid}(o_{y^*j}(\mathbf{x}_i))p(y^*|\mathbf{x}_i)\frac{\partial o_{y^*j}(\mathbf{x}_i)}{\partial \theta}\end{aligned}$$

where $o_{yj}(\mathbf{x}) = c_j + \sum_k W_{jk}x_k + U_{jy}$. This gradient can be computed efficiently and then used in a stochastic gradient descent optimization. This discriminative approach has been used previously for fine-tuning the top RBM of a Deep Belief Network [74].

4.5 Hybrid Discriminative Restricted Boltzmann Machines

The advantage brought by discriminative training usually depends on the amount of available training data. Smaller training sets tend to favor generative learning and bigger ones favor discriminative learning [115]. However, instead of solely relying on one or the other perspective, one can adopt a hybrid discriminative/generative approach simply by combining the respective training criteria. Though this method cannot be interpreted as a maximum likelihood approach for a particular generative model as in Lasserre et al. [96], it proved useful here and elsewhere [27]. In this paper, we used the following criterion:

$$\mathcal{L}_{hybrid}(\mathcal{D}_{train}) = \mathcal{L}_{disc}(\mathcal{D}_{train}) + \alpha\mathcal{L}_{gen}(\mathcal{D}_{train}) \tag{4.6}$$

where the weight α of the generative criterion can be optimized (e.g., based on the validation set classification error). Here, the generative criterion can also be seen as a data-dependent regularizer for a DRBM. We will refer to RBMs trained using the criterion of equation 4.6 as Hybrid DRBMs (HDRBMs).

To train an HDRBM, we can use stochastic gradient descent and add for each example the gradient contribution due to $\mathcal{L}_{disc}$ with α times the stochastic gradient estimator associated with $\mathcal{L}_{gen}$ for that example. This hybrid approach based on a weighted combination of discriminative and generative gradients has also previously been used to train

a Deep Belief Network (see last section of Hinton [74]). In that particular case, the top-level RBM can be seen as being trained like an HDRBM while other lower-layer parameters are also being trained according to a hybrid discriminative/generative criterion.

4.6 Semi-supervised Learning

A frequent classification setting is where there are few labeled training data but many unlabeled examples of inputs. Semi-supervised learning algorithms [33] address this situation by using the unlabeled data to introduce constraints on the trained model. For example, for purely discriminative models, these constraints are often imposed on the decision surface of the model. In the RBM framework, a natural constraint is to ask that the model be a good generative model of the unlabeled data, which corresponds to the following objective:

$$\mathcal{L}_{unsup}(\mathcal{D}_{unlab}) = -\sum_{i=1}^{|\mathcal{D}_{unlab}|} \log p(\mathbf{x}_i) \tag{4.7}$$

where $\mathcal{D}_{unlab} = \{(\mathbf{x}_i)\}_{i=1}^{|\mathcal{D}_{unlab}|}$ contains unlabeled examples of inputs. To train on this objective, we can once again use a contrastive divergence approximation of the log-likelihood gradient:

$$\begin{aligned} \frac{\partial \log p(\mathbf{x}_i)}{\partial \theta} &= -\mathbb{E}_{y,\mathbf{h}|\mathbf{x}_i}\left[\frac{\partial}{\partial \theta} E(y_i, \mathbf{x}_i, \mathbf{h})\right] \\ &\quad +\mathbb{E}_{y,\mathbf{x},\mathbf{h}}\left[\frac{\partial}{\partial \theta} E(y, \mathbf{x}, \mathbf{h})\right] \end{aligned}$$

The contrastive divergence approximation is slightly different here. The first term can be computed in time $O(Cn + nd)$, by noticing that it is equal to

$$\mathbb{E}_{y|\mathbf{x}_i}\left[\mathbb{E}_{\mathbf{h}|y,\mathbf{x}_i}\left[\frac{\partial}{\partial \theta} E(y_i, \mathbf{x}_i, \mathbf{h})\right]\right].$$

One could either average the usual RBM gradient $\frac{\partial}{\partial \theta} E(y_i, \mathbf{x}_i, \mathbf{h})$ for each class y (weighted by $p(y|\mathbf{x}_i)$), or sample a y from $p(y|\mathbf{x}_i)$ and only collect the gradient for that value of y.

In the sampling version, the online training update for this objective can be described by replacing the statement $y^0 \leftarrow y_i$ with $y^0 \sim p(y|\mathbf{x}_i)$ in Algorithm 1. We used this version in our experiments.

In order to perform semi-supervised learning, we can weight and combine the objective of equation 4.7 with those of equations 4.1, 4.5 or 4.6

$$\begin{aligned} \mathcal{L}_{semi-sup}(\mathcal{D}_{train}, \mathcal{D}_{unlab}) &= \mathcal{L}_{\text{TYPE}}(\mathcal{D}_{train}) \\ &\quad +\beta \mathcal{L}_{unsup}(\mathcal{D}_{unlab}) \end{aligned} \tag{4.8}$$

where TYPE $\in \{gen, disc, hybrid\}$. Online training according to this objective simply consists in applying the appropriate update for each training example, based on whether it is labeled or not.

4.7 Related Work

As mentioned earlier, RBMs (sometimes also referred to as harmoniums [162]) have already been used successfully in the past to extract useful features for another supervised learning algorithm. One of the main contributions of this paper lies in the demonstration that RBMs can be used on their own without relying on another learning algorithm, and provide a self-contained framework for deriving competitive classifiers. In addition to ensuring that the features learned by the RBM's hidden layer are discriminative, this approach facilitates model selection since the discriminative power of the hidden layer units (or features) can be tracked during learning by observing the progression of classification error on a validation set. It also makes it easier to tackle online learning problems relatively to approaches where learning features (hidden representation) and learning to classify are done in two separate phases [14, 81].

Gehler et al. [60], Xing et al. [165] have shown that the features learned by an RBM trained by ignoring the labeled targets can be useful for retrieving documents or classifying images of objects. However, in both these cases, the extracted features were linear in the input, were not trained discriminatively and had to be fed to another learning

algorithm which ultimately performed classification. McCallum et al. [110] presented Multi-Conditional Learning (MCL)[1] for harmoniums in order to introduce a discriminative component to harmoniums' training, but the learned features still had to be fed to another learning algorithm.

RBMs can also provide a good initialization for the parameters of neural network classifiers [74], however model selection issues arise, for instance when considering the appropriate number of learning updates and the magnitude of learning rates of each training phase. It has also been argued that the generative learning aspect of RBM training was a key element to their success as good starting points for neural network training [14], but the extent to which the final solution for the parameters of the neural network is influenced by generative learning is not well controlled. HDRBMs can be seen as a way of addressing this issue.

Finally, though semi-supervised learning was never reported for RBMs before, Druck et al. [49] introduced semi-supervised learning in hybrid generative/discriminative models using a similar approach to the one presented in section 4.6. However, they worked with log-linear models, whereas the RBMs used here can perform non-linear classification. Log-linear models depend much more on the discriminative quality of the features that are fed as input, whereas an RBM can learn useful features using their hidden variables, at the price of non-convex optimization.

4.8 Experiments

We present experiments on two classification problems: character recognition and text classification. In all experiments, we performed model selection on a validation set before testing. For the different RBM models, model selection[2] consisted in finding good

[1] We experimented with a version of MCL for the RBMs considered in this paper, however the results did not improve on those of HDRBMs.

[2] Model selection was done with a grid-like search over λ (between 0.0005 and 0.1, on a log scale), n (50 to 6000), α for HDRBMs (0 to 0.5, on a log scale) and β for semi-supervised learning (0, 0.01 or 0.1). In general, bigger values of n were found to be more appropriate with more generative learning. If no local minima was apparent, the grid was extended. The biases $\mathbf{b}$, $\mathbf{c}$ and $\mathbf{d}$ were initialized to 0 and the initial values for the elements of the weight matrices $\mathbf{U}$ and $\mathbf{W}$ were each taken from uniform samples in $\left[-m^{-0.5}, m^{-0.5}\right]$, where m is the maximum between the number of rows and columns of the matrix.

values for the learning rate λ, the size of the hidden layer n and good weights for the different types of learning (generative and semi-supervised weights). Also, the number of iterations over the training set was determined using early stopping according to the validation set classification error, with a look ahead of 15 iterations. Paired t-tests with a 5% significance level were used to statistically assess the significance of performance differences.

4.8.1 Character Recognition

We evaluated the different RBM models on the problem of classifying images of digits. The images were taken from the MNIST dataset, where we separated the original training set into training and validation sets of 50000 and 10000 examples and used the standard test set of 10000 examples. The results are given in Table 4.1. The ordinary RBM model is trained generatively (to model (x, y)), whereas RBM+NNet is an unsupervised RBM used to initialize a one-hidden layer supervised neural net (as in [14]). We give as a comparison the results of a Gaussian kernel SVM and of a regular neural network (random initialization, one hidden layer, hyperbolic tangent hidden activation functions).

First, we observe that a DRBM outperforms a generative RBM. However, an HDRBM appears able to make the best out of discriminative and generative learning and outperforms the other models.

We also experimented with a sparse version of the HDRBM model, since sparsity is known to be a good characteristic for features of images. Sparse RBMs were developed by Lee et al. [103] in the context of deep neural networks. To introduce sparsity in the hidden layer of an RBM in Lee et al. [103], after each iteration through the whole training set, the biases $\mathbf{c}$ in the hidden layer are set to a value that maintains the average of the conditional expected value of these neurons to an arbitrarily small value. This procedure tends to make the biases negative and large. We follow a different approach by simply subtracting a small constant δ value, considered as an hyper-parameter[3], from

[3]To chose δ, given the selected values for λ and α for the "non sparse" HDRBM, we performed a second grid-search over δ (10^{-5} and 0.1, on a log scale) and the hidden layer size, testing bigger hidden

the biases after each update, which is more appropriate in an online setting or for large datasets.

This sparse version of HDRBMs outperforms all the other RBM models, and yields significantly lower classification error then the SVM and the standard neural network classifiers. The performance achieved by the sparse HDRBM is particularly impressive when compared to reported performances for Deep Belief Networks (1.25% in Hinton et al. [81]) or of a deep neural network initialized using RBMs (around 1.2% in Bengio et al. [14] and Hinton [74]) for the MNIST dataset with 50000 training examples.

The discriminative power of the HDRBM can be better understood by looking a the rows of the weight matrix $\mathbf{W}$, which act as filter features. Figure 4.2 displays some of these learned filters. Some of them are spatially localized stroke detectors which can possibly be active for a wide variety of digit images, and others are much more specific to a particular shape of digit.

4.8.2 Document Classification

We also evaluated the RBM models on the problem of classifying documents into their corresponding newsgroup topic. We used a version of the 20-newsgroup dataset[4] for which the training and test sets contain documents collected at different times, a setting that is more reflective of a practical application. The original training set was

layer sizes then previously selected.

[4]This dataset is available in Matlab format here: `http://people.csail.mit.edu/jrennie/20Newsgroups/20news-bydate-matlab.tgz`

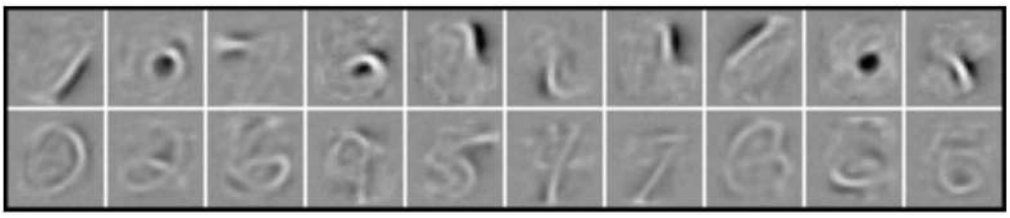


Figure 4.2: Filters learned by the HDRBM on the MNIST dataset. The top row shows filters that act as spatially localized stroke detectors, and the bottom shows filters more specific to a particular shape of digit.

Model	Error
RBM ($\lambda = 0.005, n = 6000$)	3.39%
DRBM ($\lambda = 0.05, n = 500$)	1.81%
RBM+NNet	1.41%
HDRBM ($\alpha = 0.01, \lambda = 0.05, n = 1500$)	1.28%
Sparse HDRBM (idem + $n = 3000, \delta = 10^{-4}$)	**1.16%**
SVM	1.40%
NNet	1.93%

Table 4.1: Comparison of the classification performances on the MNIST dataset. SVM results for MNIST were taken from http://yann.lecun.com/exdb/mnist/. On this dataset, differences of 0.2% in classification error is statistically significant.

divided into a smaller training set and a validation set, with 9578 and 1691 examples respectively. The test set contains 7505 examples. We used the 5000 most frequent words for the binary input features. The results are given in Figure 4.3. We also provide the results of a Gaussian kernel SVM[5] and of a regular neural network for comparison.

Once again, HDRBM outperforms the other RBM models. However, here the generatively trained RBM performs better then the DRBMs. The HDRBM also outperforms the SVM and neural network classifiers.

In order to get a better understanding of how the HDRBM solves this classification problem, we first looked at the weights connecting each of the classes to the hidden

[5]We used `libSVM` v2.85 to train the SVM model

Model	Error
RBM ($\lambda = 0.0005, n = 1000$)	24.9%
DRBM ($\lambda = 0.0005, n = 50$)	27.6%
RBM+NNet	26.8%
HDRBM ($\alpha = 0.005, \lambda = 0.1, n = 1000$)	**23.8%**
SVM	32.8%
NNet	28.2%

Figure 4.3: Classification performances on 20-newsgroup dataset. Classification performance for the different models. The error differences between HDRBM and other models is statistically significant.

neurons. This corresponds to the columns $\mathbf{U}_{\cdot y}$ of the weight matrix $\mathbf{U}$. Figure 4.4 shows a similarity matrix $\mathbf{M}(\mathbf{U})$ for the weights of the different newsgroups, where $\mathbf{M}(\mathbf{U})_{y_1 y_2} = \text{sigmoid}(\mathbf{U}_{\cdot y_1}^{\mathbf{T}} \mathbf{U}_{\cdot y_2})$. We see that the HDRBM does not use different neurons for different newsgroups, but shares some of those neurons for newsgroups that are semantically related. Another interesting visualization of this characteristic is given in Figure 4.5, where the columns of $\mathbf{U}$ were projected on their two principal components. In both cases, we see that the HDRBM tends to share neurons for similar topics, such as computer (`comp.*`), science (`sci.*`) and politics (`talk.politics.*`), or secondary topics such as sports (`rec.sports.*`) and other recreational activities (`rec.autos` and `rec.motorcycles`).

Table 4.2 also gives the set of words used by the HDRBM to recognize some of the newsgroups. To obtain this table we proceeded as follows: for each newsgroup y, we looked at the 20 neurons with the largest weight among $\mathbf{U}_{\cdot y}$, aggregated (by summing) the associated input-to-hidden weight vectors, sorted the words in decreasing order of their associated aggregated weights and picked the first words according to that order. This procedure attempts to approximate the positive contribution of the words to the conditional probability of each newsgroup.

4.8.3 Semi-supervised Learning

We evaluated our semi-supervised learning algorithm for the HDRBM on both the digit recognition and document classification problems. We also experimented with a version (noted MNIST-BI) of the MNIST dataset proposed by Larochelle et al. [95] where background images have been added to MNIST digit images. This version corresponds to a much harder problem, but it will help to illustrate the advantage brought by semi-supervised learning in HDRBMs. The HDRBM trained on this data used truncated exponential input units (see [14]).

In this semi-supervised setting, we reduced the size of the labeled training set to 800 examples, and used some of the remaining data to form an unlabeled dataset $\mathcal{D}_{unlab}$. The validation set was also reduced to 200 labeled examples. Model selection[6] cov-

[6] $\beta = 0.1$ for MNIST and 20-newsgroup and $\beta = 0.01$ for MNIST-BI was found to perform best.

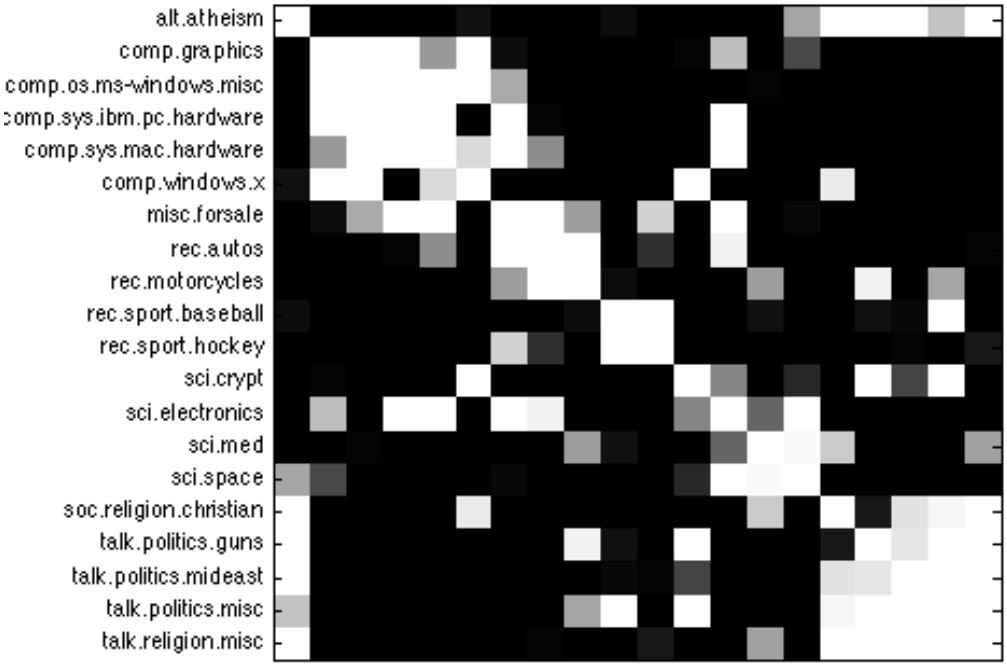


Figure 4.4: Similarity matrix of the newsgroup weight vectors $U_{\cdot y}$.

ered all the parameters of the HDRBM as well as the unsupervised objective weight β of equation 4.8. For comparison purposes, we also provide the performance of a standard non-parametric semi-supervised learning algorithm based on function induction [13], which includes as a particular case or is very similar to other non-parametric semi-supervised learning algorithms such as Zhu et al. [169]. We provide results for the use of a Gaussian kernel (NP-Gauss) and a data-dependent truncated Gaussian kernel (NP-Trunc-Gauss) used in Bengio et al. [13], which essentially outputs zero for pairs of inputs that are not near neighbors. The experiments on the MNIST and MNIST-BI (with background images) datasets used 5000 unlabeled examples and the experiment on 20-newsgroup used 8778. The results are given in Table 4.3, where we observe that semi-supervised learning consistently improves the performance of the HDRBM.

In the past, the usefulness of non-parametric semi-supervised learning algorithms has been demonstrated many times, but usually so on problems where the dimensionality of the inputs is low or the data lies on a much lower dimensional manifold. This is reflected in the result on MNIST for the non-parametric methods. However, for high dimensional

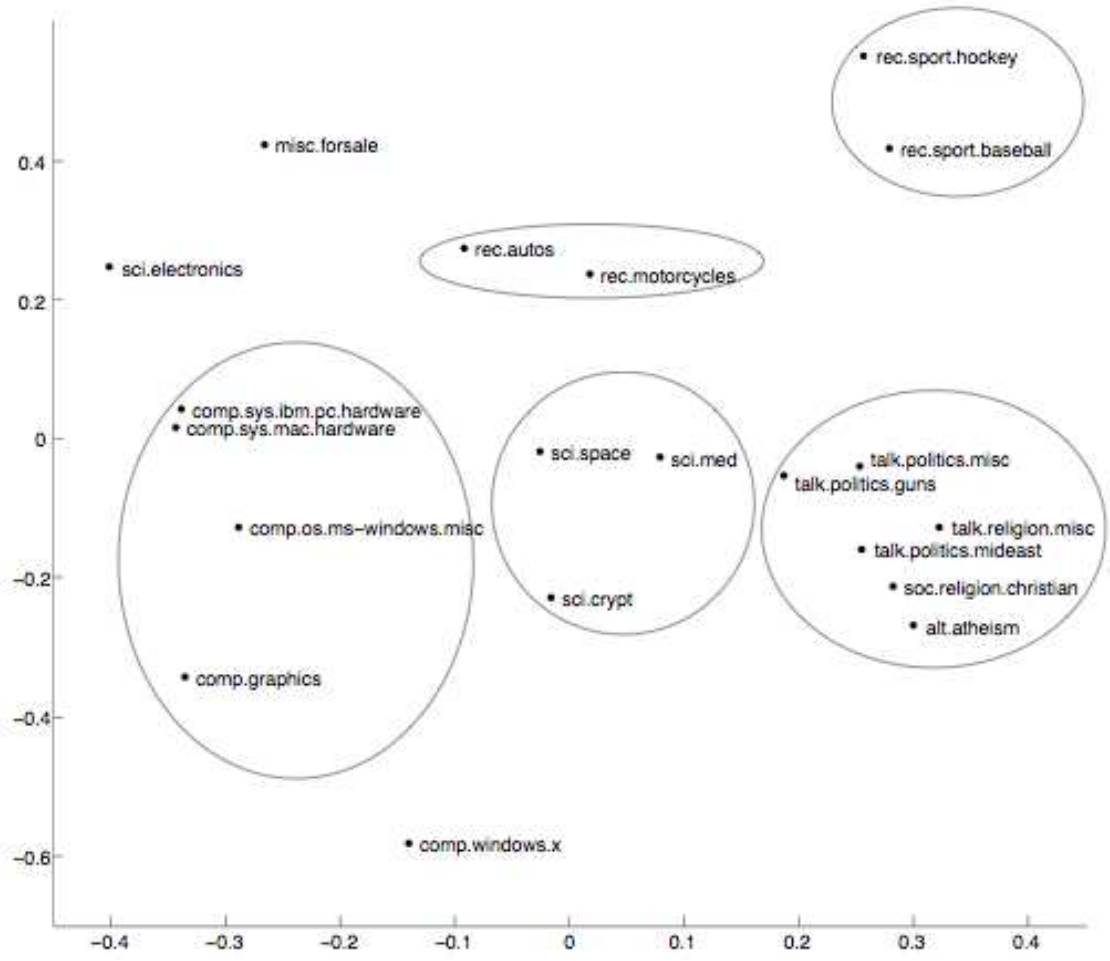


Figure 4.5: Two dimensional PCA embedding of the newsgroup-specific weight vectors $U_{\cdot y}$.

Table 4.2: Most influential words in the HDRBM for predicting some of the document classes

Class	Words
alt.atheism	bible, atheists, benedikt, atheism, religion
comp.graphics	tiff, ftp, window, gif, images, pixel
misc.forsale	sell, condition, floppy, week, am, obo
rec.autos	cars, ford, autos, sho, toyota, roads
sci.crypt	sternlight, bontchev, nsa, escrow, hamburg
talk.politics.guns	firearms, handgun, firearm, gun, rkba

data with many factors of variation, these methods can quickly suffer from the curse of dimensionality, as argued by Bengio et al. [12]. This is also reflected in the results for the MNIST-BI dataset which contains many factors of variation, and for the 20-newsgroup dataset where the input is very high dimensional.

Finally, it is important to notice that semi-supervised learning in HDRBMs proceeds in an online fashion and hence could scale to very large datasets, unlike more standard non-parametric methods.

4.8.4 Relationship with Feed-forward Neural Networks

There are several similarities between discriminative RBMs and neural networks. In particular, the computation of $p(y|\mathbf{x})$ could be implemented by a single layer neural network with softplus and softmax activation functions in its hidden and output layers respectively, with a special structure in the output and hidden weights where the value of the output weights is fixed and many of the hidden layer weights are shared.

The advantage of working in the framework of RBMs is that it provides a natural way to introduce generative learning, which we used here to derive a semi-supervised learning algorithm. As mentioned earlier, a form of generative learning can be introduced in standard neural networks simply by using RBMs to initialize the hidden layer weights. However the extent to which the final solution for the parameters of the neural network is influenced by generative learning is not well controlled. This might explain the superior performance obtained by a HDRBM compared to a single hidden layer neural network

Table 4.3: Comparison of the classification errors in semi-supervised learning setting. The errors in bold are statistically significantly better.

Model	MNIST	MNIST-BI	20-news
HDRBM	9.73%	42.4%	40.5%
Semi-sup HDRBM	8.04%	**37.5%**	**31.8%**
NP-Gauss	10.60%	66.5%	85.0%
NP-Trunc-Gauss	**7.49%**	61.3%	82.6%

initialized with an RBM (RBM+NNet in the tables).

4.9 Conclusion and Future Work

We argued that RBMs can and should be used as stand-alone non-linear classifiers alongside other standard and more popular classifiers, instead of merely being considered as simple feature extractors. We evaluated different training objectives that are more appropriate to train an RBM in a classification setting. These discriminative versions of RBMs integrate the process of discovering features of inputs with their use in classification, without relying on a separate classifier. This insures that the learned features are discriminative and facilitates model selection. We also presented a novel but straightforward semi-supervised learning algorithm for RBMs and demonstrated its usefulness for complex or high dimensional data.

For future work, we would like to investigate the use of discriminative versions of RBMs in more challenging settings such as in multi-task or structured output problems. The analysis of the target weights for the 20-newsgroup dataset seem to indicate that RBMs would be good at capturing the conditional statistical relationship between multiple tasks or in the components in a complex target space. Exact computation of the conditional distribution for the target is not tractable anymore, but there exists promising techniques such as mean-field approximations that could estimate that distribution. Moreover, in the 20-newsgroup experiment, we only used 5000 words in input because generative training using Algorithm 1 does not exploit the sparsity of the input, unlike an SVM or a DRBM (since in that case the sparsity of the input makes the discriminative

gradient sparse too). Motivated by this observation, we intend to explore ways to introduce generative learning in RBMs and HDRBMs which would be less computationally expensive when the input vectors are large but sparse.

Acknowledgments

We thank Dumitru Erhan for discussions about sparse RBMs and anonymous reviewers for helpful comments.

CHAPITRE 5

PRÉSENTATION DU DEUXIÈME ARTICLE

5.1 Détails de l'article

Exploring Strategies for Training Deep Neural Networks
Hugo Larochelle, Yoshua Bengio, Jérôme Louradour et Pascal Lamblin
Publié dans *Journal of Machine Learning Research, MIT Press* en 2009.

5.2 Contexte

On a vu au chapitre précédent qu'à l'aide d'une régularisation basée sur l'apprentissage hybride génératif/discriminant (lui-même fondé sur l'apprentissage non-supervisé), il est possible d'entraîner un réseau de neurones contenant beaucoup plus de neurones cachés sans trop de surapprentissage, et ainsi obtenir des performances du même niveau que l'état de l'art sur des données de vision ou textuelles. Nous avons aussi discuté à la section 2.3 que des données de ce type sont normalement associées à des problèmes liés au comportement intelligent de l'humain et sont bien décrites par une représentation hiérarchique.

5.3 Contributions

Dans cet article, on étudie empiriquement l'approche proposée par Hinton et al. [81][1] pour entraîner un réseau de neurones profond (i.e., à plusieurs couches cachées), basée sur l'entraînement non-supervisé de machines de Boltzmann restreintes. Cette contribution de Hinton et al. [81] est extrêmement importante, étant données les difficultés qu'éprouvait auparavant l'entraînement purement supervisé d'un réseau de neurones à plus de deux couches cachées. Hinton et al. [81] proposent simplement d'entraîner de

[1] Bien que l'on réfère normalement à Hinton et al. [81] pour désigner cette approche, la référence exacte pour cette approche telle qu'appliquée aux réseaux de neurones artificiels standards est Hinton [74], aussi publiée sous forme de rapport technique un an avant [78].

façon vorace chacune des couches cachées d'un réseau de neurones en les considérant chacune comme la couche cachée d'une machine de Boltzmann restreinte. L'entraînement débute à la première couche cachée (i.e., la couche la plus rapprochée de la couche d'entrée) et, une fois l'entraînement non-supervisé de cette couche terminé, progresse vers la couche cachée suivante. Ce processus est répété jusqu'à ce que toutes les couches cachées aient été entraînées. Finalement, une dernière phase d'entraînement supervisé, correspondant à l'algorithme de descente de gradient stochastique standard, est enclenchée afin de compléter l'entraînement de tout le réseau. Ainsi, la procédure de Hinton et al. [81] peut être comprise comme une procédure d'initialisation d'un réseau de neurones.

L'article présenté ici expérimente avec différentes variations de cette procédure vorace. Une des contributions importantes de cet article est la démonstration qu'il est possible d'obtenir des performances similaires de généralisation en initialisant les couches cachées du réseau à l'aide de l'algorithme d'apprentissage d'un réseau autoassociateur (ou autoencodeur). Un réseau autoassociateur est simplement un réseau de neurones entraîné à reconstruire sa propre entrée à partir de la représentation calculée par sa couche cachée. L'avantage le plus important de l'autoassociateur est probablement sa flexibilité. Il existe effectivement moins de contraintes à la modification de la topologie ou du type de calculs effectués par un autoassociateur. Il est aussi facile de dériver une procédure d'entraînement adéquate à un nouvel autoassociateur, via la rétropropagation et la descente de gradient stochastique. D'ailleurs, les articles des chapitres 10 et 12 illustrent bien le genre de variations qu'il est possible de dériver. De plus, comme le démontre cet article, il est possible de combiner les procédures d'entraînement de la machine de Boltzmann restreinte et de l'autoassociateur afin d'améliorer encore plus la performance de généralisation d'un réseau profond.

Une autre contribution intéressante de cet article provient d'expériences sur deux problèmes de classification de caractères écrits. La particularité de ces deux problèmes est que l'un est une version plus difficile de l'autre, obtenu en appliquant une rotation d'un angle aléatoire sur les images d'entrées. En variant le nombre de couches cachées pour ces deux problèmes, on observe que le problème le plus complexe (basé sur les

rotations d'images) nécessite un plus grand nombre de couches cachées que le problème original. Cette expérience met en évidence l'importance que peut avoir l'utilisation d'une architecture hiérarchique profonde pour la résolution de problèmes plus complexes.

L'article apporte aussi plusieurs autres contributions. On y avance, expériences à l'appui, que l'apprentissage non-supervisé n'aide pas seulement à régulariser le réseau, mais peut aussi contribuer à l'obtention d'une meilleure optimisation des paramètres du réseau. On y présente aussi d'autres variantes de la procédure de Hinton et al. [81], par exemple en permettant l'entraînement simultané des machines de Boltzmann restreintes de chaque couche en plus de l'entraînement supervisé global du réseau. Cette dernière variante est plus appropriée dans le cadre d'un apprentissage dit « en ligne », où un flot d'exemples d'entraînement est donné plutôt qu'un ensemble de taille fixe. On y dérive aussi deux variantes des machines de Boltzmann restreintes compatibles pour des composantes d'entrée x_i prenant une valeur continue dans l'intervalle $[0, 1]$ ou dans tous les réels ($]-\infty, \infty[$), les machines de Boltzmann ayant été originalement développées pour des entrées aux composantes binaires.

5.4 Commentaires

Encore une fois, il y a un certain anachronisme dû à la présentation de cet article à ce moment-ci, avant celui du chapitre 8 auquel cet article fait aussi référence. Puisque cet article décrit de façon plus détaillée la procédure générale d'entraînement d'un réseau profond, il était approprié de l'inclure à cette étape-ci de l'exposé des travaux de cette thèse. Il faut savoir aussi que cet article est une version étendue de l'article «Greedy Layer-Wise Training of Deep Networks» [14], qui lui a été publié avant.

CHAPTER 6

EXPLORING STRATEGIES FOR TRAINING DEEP NEURAL NETWORKS

6.1 Abstract

Deep multi-layer neural networks have many levels of non-linearities allowing them to compactly represent highly non-linear and highly-varying functions. However, until recently it was not clear how to train such deep networks, since gradient-based optimization starting from random initialization often appears to get stuck in poor solutions. Hinton et al. recently proposed a greedy layer-wise unsupervised learning procedure relying on the training algorithm of restricted Boltzmann machines (RBM) to initialize the parameters of a deep belief network (DBN), a generative model with many layers of hidden causal variables. This was followed by the proposal of another greedy layer-wise procedure, relying on the usage of autoassociator networks. In the context of the above optimization problem, we study these algorithms empirically to better understand their success. Our experiments confirm the hypothesis that the greedy layer-wise unsupervised training strategy helps the optimization by initializing weights in a region near a good local minimum, but also implicitly acts as a sort of regularization that brings better generalization and encourages internal distributed representations that are high-level abstractions of the input. We also present a series of experiments aimed at evaluating the link between the performance of deep neural networks and practical aspects of their topology, e.g., demonstrating cases where the addition of more depth helps. Finally, we empirically explore simple variants of these training algorithms, such as the use of different RBM input unit distributions, a simple way of combining gradient estimators to improve performance, as well as on-line versions of those algorithms.

6.2 Introduction

Training deep multi-layered neural networks is known to be hard. The standard learning strategy—consisting of randomly initializing the weights of the network and

applying gradient descent using backpropagation—is known empirically to find poor solutions for networks with 3 or more hidden layers. As this is a negative result, it has not been much reported in the machine learning literature. For that reason, artificial neural networks have been limited to one or two hidden layers.

However, complexity theory of circuits strongly suggests that deep architectures can be much more efficient (sometimes exponentially) than shallow architectures, in terms of computational elements and parameters required to represent some functions [10, 15]. Whereas it cannot be claimed that deep architectures are better than shallow ones on every problem [94, 136], there has been evidence of a benefit when the task is complex enough, and there is enough data to capture that complexity [95]. Hence finding better learning algorithms for such deep networks could be beneficial.

An approach that has been explored with some success in the past is based on constructively adding layers. Each layer in a multi-layer neural network can be seen as a representation of the input obtained through a learned transformation. What makes a good internal representation of the data? We believe that it should disentangle the factors of variation that inherently explain the structure of the distribution. When such a representation is going to be used for unsupervised learning, we would like it to preserve information about the input while being easier to model than the input itself. When a representation is going to be used in a supervised prediction or classification task, we would like it to be such that there exists a "simple" (i.e., somehow easy to learn) mapping from the representation to a good prediction. To constructively build such a representation, it has been proposed to use a *supervised* criterion at each stage [52, 104]. However, as we discuss here, the use of a supervised criterion at each stage may be too greedy and does not yield as good generalization as using an unsupervised criterion. Aspects of the input may be ignored in a representation tuned to be immediately useful (with a linear classifier) but these aspects might turn out to be important when more layers are available. Combining unsupervised (e.g., learning about $p(x)$) and supervised components (e.g., learning about $p(y|x)$) can be be helpful when both functions $p(x)$ and $p(y|x)$ share some structure.

The idea of using unsupervised learning at each stage of a deep network was recently

put forward by Hinton et al. [81], as part of a training procedure for the deep belief network (DBN), a generative model with many layers of hidden stochastic variables. Upper layers of a DBN are supposed to represent more "abstract" concepts that explain the input observation $\mathbf{x}$, whereas lower layers extract "low-level features" from $\mathbf{x}$. In other words, this model first learns simple concepts, on which it builds more abstract concepts.

This training strategy has inspired a more general approach to help address the problem of training deep networks. Hinton [78] showed that stacking restricted Boltzmann machines (RBMs)—i.e., training upper RBMs on the distribution of activities computed by lower RBMs—provides a good initialization strategy for the weights of a deep artificial neural network. This approach can be extended to non-linear autoencoders or autoassociators [138], as shown by Bengio et al. [14], and is found in stacked autoassociators network [95], and in the deep convolutional neural network [125] derived from the convolutional neural network [100]. Since then, deep networks have been applied with success not only in classification tasks [14, 95, 122, 125], but also in regression [134], dimensionality reduction [80, 132], modeling textures [118], information retrieval [133], robotics [66], natural language processing [37, 163], and collaborative filtering [135].

In this paper, we discuss in detail three principles for training deep neural networks and present experimental evidence that highlight the role of each in successfully training deep networks:

1. Pre-training one layer at a time in a greedy way;

2. using unsupervised learning at each layer in a way that preserves information from the input and disentangles factors of variation;

3. fine-tuning the whole network with respect to the ultimate criterion of interest.

The experiments reported here suggest that this strategy improves on the traditional random initialization of supervised multi-layer networks by providing "hints" to each intermediate layer about the kinds of representations that it should learn, and thus initializing the supervised fine-tuning optimization in a region of parameter space from

which a better local minimum (or plateau) can be reached. We also present a series of experiments aimed at evaluating the link between the performance of deep neural networks and aspects of their topology such as depth and the size of the layers. In particular, we demonstrate cases where the addition of depth helps classification error, but too much depth hurts. Finally, we explore simple variants of the aforementioned training algorithms, such as a simple way of combining them to improve their performance, RBM variants for continuous-valued inputs, as well as on-line versions of those algorithms.

Notations and Conventions

Before describing the learning algorithms that we will study and experiment with in this paper, we first present the mathematical notation we will use for deep networks.

A deep neural network contains an input layer and an output layer, separated by l layers of hidden units. Given an input sample clamped to the input layer, the other units of the network compute their values according to the activity of the units that they are connected to in the layers below. We will consider a particular sort of topology here, where the input layer is fully connected to the first hidden layer, which is fully connected to the second layer and so on up to the output layer.

Given an input $\mathbf{x}$, the value of the j-th unit in the i-th layer is denoted $\widehat{h}^i_j(\mathbf{x})$, with $i = 0$ referring to the input layer, $i = l + 1$ referring to the output layer (the use of "$\widehat{\ }$" will become clearer in Section 6.4). We refer to the size of a layer as $|\widehat{\mathbf{h}}^i(\mathbf{x})|$. The default activation level is determined by the internal bias b^i_j of that unit. The set of weights W^i_{jk} between $\widehat{h}^{i-1}_k(\mathbf{x})$ in layer $i-1$ and unit $\widehat{h}^i_j(\mathbf{x})$ in layer i determines the activation of unit $\widehat{h}^i_j(\mathbf{x})$ as follows:

$$\widehat{h}^i_j(\mathbf{x}) = \text{sigmoid}\left(a^i_j\right) \text{ where } a^i_j(\mathbf{x}) = b^i_j + \sum_k W^i_{jk}\widehat{h}^{i-1}_k(\mathbf{x}) \; \forall i \in \{1, \ldots, l\} \qquad (6.1)$$

with $\widehat{h}^0(\mathbf{x}) = \mathbf{x}$ and where $\text{sigmoid}(\cdot)$ is the sigmoid squashing function: $\text{sigmoid}(a) = \frac{1}{1+e^{-a}}$ (alternatively, the sigmoid could be replaced by the hyperbolic tangent). Given

the last hidden layer, the output layer is computed similarly by

$$\mathbf{o}(\mathbf{x}) = \widehat{\mathbf{h}}^{l+1}(\mathbf{x}) = f\left(\mathbf{a}^{l+1}(\mathbf{x})\right) \text{ where } \mathbf{a}^{l+1}(\mathbf{x}) = \mathbf{b}^{l+1} + \mathbf{W}^{l+1}\widehat{\mathbf{h}}^{l}(\mathbf{x})$$

where the activation function $f(\cdot)$ depends on the (supervised) task the network must achieve. Typically, it will be the identity function for a regression problem and the softmax function

$$f_j(\mathbf{a}) = \text{softmax}_j(\mathbf{a}) = \frac{e^{a_j}}{\sum_{k=1}^{K} e^{a_k}} \tag{6.2}$$

for a classification problem, in order to obtain a distribution over the K classes.

When an input sample $\mathbf{x}$ is presented to the network, the application of Equation 6.1 at each layer will generate a pattern of activity in the different layers of the neural network. Intuitively, we would like the activity of the first layer neurons to correspond to low-level features of the input (e.g., edge orientations for natural images) and to higher level abstractions (e.g., detection of geometrical shapes) in the last hidden layers.

6.3 Deep Neural Networks

It has been shown that a "shallow" neural network with only one arbitrarily large hidden layer could approximate a function to any level of precision [84]. Similarly, any Boolean function can be represented by a two-layer circuit of logic gates. However, most Boolean functions require an exponential number of logic gates (with respect to the input size) to be represented by a two-layer circuit [157]. For example, the parity function, which can be efficiently represented by a circuit of depth $O(\log n)$ (for n input bits) needs $O(2^n)$ gates to be represented by a depth two circuit[1] [166]. What about deeper circuits? Some families of functions which can be represented with a depth k circuit are such that they require an exponential number of logic gates to be represented by a depth $k-1$ circuit [70]. Interestingly, an equivalent result has been proved for ar-

[1] It should be noted that, though the parity function is inefficiently encoded by a shallow network of logic gates, it can be encoded efficiently with a one hidden layer neural network. This is made possible by the fact that the value of a unit in a neural network is based on a weighted sum of all its inputs, instead of being based on simple boolean operations. This fact illustrates how the capacity of a network (deep or shallow) is highly dependent on the capacity of the units that makes it.

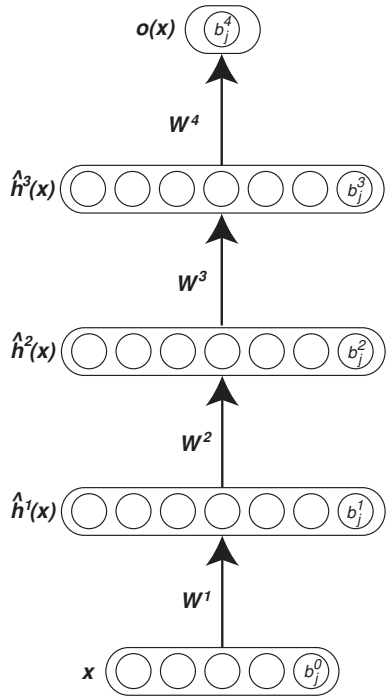


Figure 6.1: Illustration of a deep network and its parameters.

chitectures whose computational elements are not logic gates but linear threshold units (i.e., formal neurons) [71]. The machine learning literature also suggests that shallow architectures can be very inefficient in terms of the number of computational units (e.g., bases, hidden units), and thus in terms of required examples [12, 15]. On the other hand, a highly-varying function can sometimes be represented compactly (with fewer parameters) through the composition of many non-linearities, i.e., with a deep architecture. When the representation of a concept requires an exponential number of elements (more generally exponential capacity), e.g., with a shallow circuit, the number of training examples required to learn the concept may also be impractical. Smoothing the learned function by regularization would not solve the problem here because in these cases the target function itself is complicated and requires exponential capacity just to be represented.

6.3.1 Difficulty of Training Deep Architectures

Given a particular task, a natural way to train a deep network is to frame it as an optimization problem by specifying a supervised cost function on the output layer with respect to the desired target and use a gradient-based optimization algorithm in order to adjust the weights and biases of the network so that its output has low cost on samples in the training set. Unfortunately, deep networks trained in that manner have generally been found to perform worse than neural networks with one or two hidden layers.

We discuss two hypotheses that may explain this difficulty. The first one is that gradient descent can easily get stuck in poor local minima [5] or plateaus of the non-convex training criterion. The number and quality of these local minima and plateaus [57] clearly also influence the chances for random initialization to be in the basin of attraction (via gradient descent) of a poor solution. It may be that with more layers, the number or the width of such poor basins increases. To reduce the difficulty, it has been suggested to train a neural network in a constructive manner in order to divide the hard optimization problem into several greedy but simpler ones, either by adding one neuron [e.g., see 52] or one layer [e.g., see 104] at a time. These two approaches have demonstrated to be very effective for learning particularly complex functions, such as a very non-linear classification problem in 2 dimensions. However, these are exceptionally hard problems, and for learning tasks usually found in practice, this approach commonly overfits.

This observation leads to a second hypothesis. For high capacity and highly flexible deep networks, there actually exists many basins of attraction in its parameter space (i.e., yielding different solutions with gradient descent) that can give low training error but that can have very different generalization errors. So even when gradient descent is able to find a (possibly local) good minimum in terms of training error, there are no guarantees that the associated parameter configuration will provide good generalization. Of course, model selection (e.g., by cross-validation) will partly correct this issue, but if the number of good generalization configurations is very small in comparison to good training configurations, as seems to be the case in practice, then it is likely that the training procedure will not find any of them. But, as we will see in this paper, it appears

that the type of unsupervised initialization discussed here can help to select basins of attraction (for the supervised fine-tuning optimization phase) from which learning good solutions is easier both from the point of view of the training set and of a test set.

6.3.2 Unsupervised Learning as a Promising Paradigm for Greedy Layer-Wise Training

A common approach to improve the generalization performance of a learning algorithm which is motivated by the Occam's razor principle is the use of regularization (such as weight decay) that will favor "simpler" models over more complicated ones. However, using generic priors such as the ℓ^2 norm of the parameters conveys limited information about what the solution to a particular learning task should be. This has motivated researchers to discover more meaningful, data-dependent regularization procedures, which are usually based on unsupervised learning and normally adapted to specific models.

For example, Ando et Zhang [4] use "auxiliary tasks" designed from unlabelled data and that are appropriate for a particular learning problem, to learn a better regularization term for linear classifiers. Partial least squares [54] can also be seen as combining unsupervised and supervised learning in order to learn a better linear regression model when few training data are available or when the input space is very high dimensional.

Many semi-supervised learning algorithms also involve a combination of unsupervised and supervised learning, where the unsupervised component can be applied to additional unlabelled data. This is the case for Fisher-kernels [88] which are based on a generative model trained on unlabelled input data and that can be used to solve a supervised problem defined for that input space. In all these cases, unsupervised learning can be seen as adding more constraints on acceptable configurations for the parameters of a model, by asking that it not only describes well the relationship between the input and the target but also contains relevant statistical information about the structure of the input or how it was generated.

Moreover, there is a growing literature on the distinct advantages of generative and discriminative learning. Ng et Jordan [115] argue that generative versions of discriminative models can be expected to reach their usually higher asymptotic out-of-sample

classification error faster (i.e., with less training data), making them preferable in certain situations. Moreover, successful attempts at exploring the space between discriminative and generative learning have been studied [27, 82, 89, 96].

The deep network learning algorithms that have been proposed recently and that we study in this paper can be seen as combining the ideas of *greedily learning the network* to break down the learning problem into easier steps, *using unsupervised learning* to provide an *effective hint* about what hidden units should learn, bringing along the way a form of regularization that prevents overfitting even in deep networks with many degrees of freedom (which could otherwise overfit). In addition, one should consider the supervised task the network has to solve. The greedy layer-wise unsupervised strategy provides an initialization procedure, after which the neural network is *fine-tuned to the global supervised objective*. The general paradigm followed by these algorithms (illustrated in Figure 6.2 and detailed in Appendix A) can be decomposed in two phases:

1. In the first phase, greedily train subsets of the parameters of the network using a layer-wise and unsupervised learning criterion, by repeating the following steps for each layer ($i \in \{1, \ldots, l\}$)

   Until a stopping criteria is met, iterate through training database by

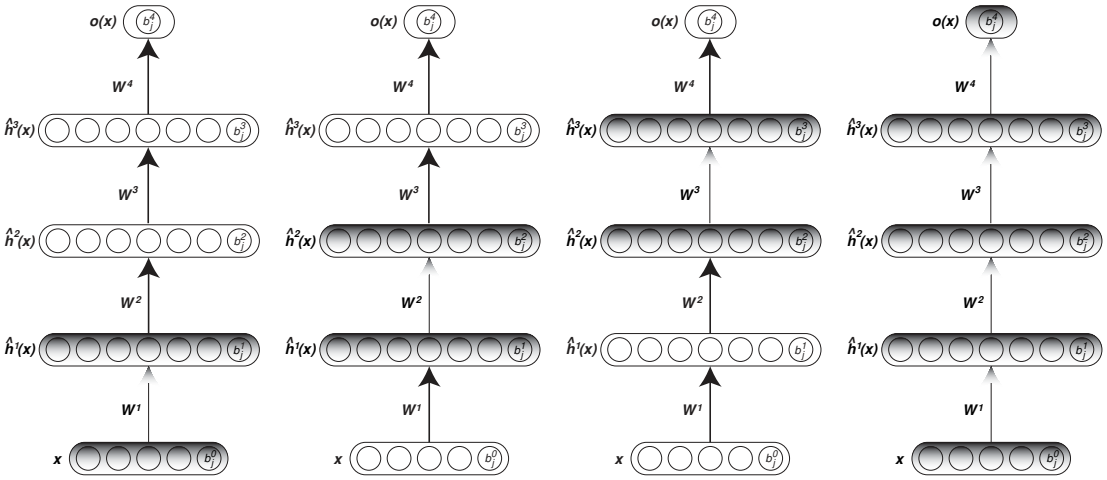
   (a) mapping input training sample $\mathbf{x}_t$ to representation $\widehat{\mathbf{h}}^{i-1}(\mathbf{x}_t)$ (if $i > 1$) and hidden representation $\widehat{\mathbf{h}}^{i}(\mathbf{x}_t)$,

   (b) updating parameters $\mathbf{b}^{i-1}$, $\mathbf{b}^{i}$ and $\mathbf{W}^{i}$ of layer i using some unsupervised learning algorithm.

   Also, initialize (e.g., randomly) the output layer parameters $\mathbf{b}^{l+1}$, $\mathbf{W}^{l+1}$.

2. In the second and final phase, fine-tune all the parameters θ of the network using backpropagation and gradient descent on a global supervised cost function $C(\mathbf{x}_t, y_t, \theta)$, with input $\mathbf{x}_t$ and label y_t, i.e., trying to make steps in the direction $E\left[\frac{\partial C(\mathbf{x}_t, y_t, \theta)}{\partial \theta}\right]$.

Regularization is not explicit in this procedure, as it does not come from a weighted term that depends on the complexity of the network and that is added to the global su-



(a) First hidden layer pre-training (b) Second hidden layer pre-training (c) Third hidden layer pre-training (d) Fine-tuning of whole network

Figure 6.2: Unsupervised greedy layer-wise training procedure.

pervised objective. Instead, it is implicit, as the first phase that initializes the parameters of the whole network will ultimately have an impact on the solution found in the second phase (the fine-tuning phase). Indeed, by using an iterative gradual optimization algorithm such as stochastic gradient descent with early-stopping (i.e., training until the error on a validation set reaches a clear minimum), the extent to which the configuration of the network's parameters can be different from the initial configuration given by the first phase is limited. Hence, similarly to using a regularization term on the parameters of the model that constrains them to be close to a particular value (e.g., 0 for weight decay), the first phase here will ensure that the parameter solution for each layer found by fine-tuning will not be far from the solution found by the unsupervised learning algorithm. In addition, the non-convexity of the supervised training criterion means that the choice of initial parameter values can greatly influence the quality of the solution obtained by gradient descent.

In the next two sections, we present a review of the two training algorithms that fall in paradigm presented above and which are empirically studied in this paper, in Section 6.6.

6.4 Stacked Restricted Boltzmann Machine Network

Intuitively, a successful learning algorithm for deep networks should be one that discovers a meaningful and possibly complex hidden representation of the data at its top hidden layer. However, learning such non-linear representations is a hard problem. A solution, proposed by Hinton [78], is based on the learning algorithm of the restricted Boltzmann machine (RBM) [144], a generative model that uses a layer of binary variables to explain its input data. In an RBM (see Figure 6.3 for an illustration), given an input $\mathbf{x}$, it is easy to obtain a hidden representation for that input by computing the posterior $\widehat{\mathbf{h}}(\mathbf{x})$ over the layer of binary hidden variables $\mathbf{h}$ (we use the "$\widehat{\ }$" symbol to emphasize that $\widehat{\mathbf{h}}(\mathbf{x})$ is not a random variable but a deterministic representation of $\mathbf{x}$).

Hinton [78] argues that this representation can be improved by giving it as input to another RBM, whose posterior over its hidden layer will then provide a more complex representation of the input. This process can be repeated an arbitrary number of times in order to obtain ever more non-linear representations of the input. Finally, the parameters of the RBMs that compute these representations can be used to initialize the parameters of a deep network, which can then be fine-tuned to a particular supervised task. This learning algorithm clearly falls in the paradigm of Section 6.3.2, where the unsupervised part of the learning algorithm is that of an RBM. We will refer to deep networks trained using this algorithm as stacked restricted Boltzmann machine (SRBM) networks. For more technical details about the SRBM network, and how to train an RBM using the contrastive divergence algorithm (CD-k), see Appendix B.

6.5 Stacked Autoassociators Network

There are theoretical results about the advantage of stacking many RBMs into a DBN: Hinton et al. [81] show that this procedure optimizes a bound on the likelihood of the input data when even-numbered and odd-numbered layers have the same size. An additional hypothesis to explain why this process provides a good initialization for the network is that it makes each hidden layer compute a different, possibly more abstract representation of the input. This is done implicitly, by asking that each layer captures

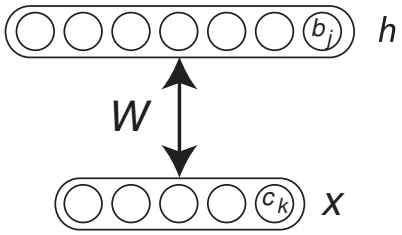


Figure 6.3: Illustration of a restricted Boltzmann machine and its parameters. $\mathbf{W}$ is a weight matrix, $\mathbf{b}$ is a vector of hidden unit biases, and $\mathbf{c}$ a vector of visible unit biases.

features of the input that help characterize the distribution of values at the layer below. By transitivity, each layer contains some information about the input. However, stacking any unsupervised learning model does not guarantee that the representations learned get increasingly complex or appropriate as we stack more layers. For instance, many layers of linear PCA models could be summarized by only one layer. However, there may be other non-linear, unsupervised learning models that, when stacked, are able to improve the learned representation at the last layer added.

An example of such a non-linear unsupervised learning model is the autoassociator or autoencoder network [6, 40, 46, 76, 138]. Autoassociators are neural networks that are trained to compute a representation of the input from which it can be reconstructed with as much accuracy as possible. In this paper, we will consider autoassociator networks of only one hidden layer, meaning that the hidden representation of $\mathbf{x}$ is a code $\widehat{\mathbf{h}}(x)$ obtained from the encoding function

$$\widehat{h}_j(\mathbf{x}) = f\left(a_j\right) \text{ where } a_j(\mathbf{x}) = b_j + \sum_k W_{jk} x_k \ . \tag{6.3}$$

The input's reconstruction is obtained from a decoding function, here a linear transformation of the hidden representation with weight matrix $\mathbf{W}^*$, possibly followed by a non-linear activation function:

$$\widehat{x}_k = g\left(\widehat{a}_k\right) \text{ where } \widehat{a}_k = c_k + \sum_j W^*_{jk} \widehat{h}_j(\mathbf{x}) \ .$$

In this work, we used the sigmoid activation function for both $f(\cdot)$ and $g(\cdot)$. Figure 6.4

shows an illustration of this model.

By noticing the similarity between Equations 6.3 and 6.1, we are then able to use the training algorithm for autoassociators as the unsupervised learning algorithm for the greedy layer-wise initialization phase of deep networks. In this paper, stacked autoassociators (SAA) networks will refer to deep networks trained using the procedure of Section 6.3.2 and the learning algorithm of an autoassociator for each layer, as described in Section 6.5.1.

Though these neural networks were designed with the goal of dimensionality reduction in mind, the new representation's dimensionality does not necessarily need to be lower than the input's in practice. However, in that particular case, some care must be taken so that the network does not learn a trivial identity function, i.e., finds weights that simply "copy" the whole input vector in the hidden layer and then copy it again at the output. For example, a network with small weights W_{jk} between the input and hidden layers (maintaining activations in the linear regime of the activation function f) and large weights W^*_{jk} between the hidden and output layers could encode such an uninteresting identity function. An easy way to avoid such a pathological behavior in the case of continuous inputs is to set the weight matrices $\mathbf{W^T}$ and $\mathbf{W}^*$ to be the same. This adjustment is motivated by its similarity with the parametrization of the RBM model and by an empirical observation that $\mathbf{W^T}$ and $\mathbf{W}^*$ tend to be similar up to a multiplicative factor after training. In the case of binary inputs, if the weights are large, the input vector can still be copied (up to a permutation of the elements) to the hidden units, and in turn these used to perfectly reconstruct the input. Weight decay can be useful to prevent such a trivial and uninteresting mapping to be learned, when the inputs are binary. We set $\mathbf{W^T} = \mathbf{W}^*$ in all of our experiments. Vincent et al. [154] have an improved way of training autoassociators in order to produce interesting, non-trivial features in the hidden layer, by partially corrupting the network's inputs.

The reconstruction error of an autoassociator can be connected to the log-likelihood of an RBM in several ways. Ranzato et al. [122] connect the log of the numerator of the input likelihood with a form of reconstruction error (where one replaces the sum over hidden unit configurations by a maximization). The denominator is the normalization

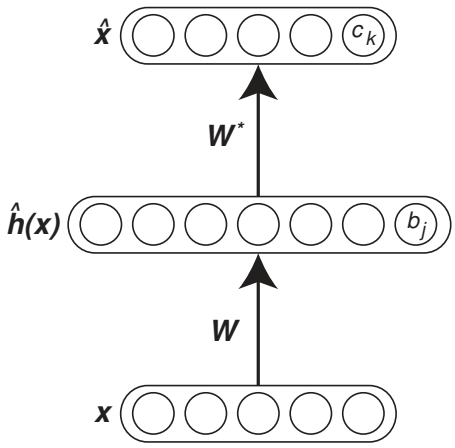


Figure 6.4: Illustration of an autoassociator and its parameters. $\mathbf{W}$ is the matrix of encoder weights and $\mathbf{W}^*$ the matrix of decoder weights. $\widehat{\mathbf{h}}(\mathbf{x})$ is the code or representation of $\mathbf{x}$.

constant summing over all input configurations the same expression as in the numerator. So whereas maximizing the numerator is similar to minimizing reconstruction error for the training examples, minimizing the denominator means that most input configurations should not be reconstructed well. This can be achieved if the autoassociator is constrained in such a way that it cannot compute the identity function, but only minimizes the reconstruction for training examples.

Another connection between reconstruction error and log-likelihood of the RBM was made in Bengio et Delalleau [11]. They consider a converging series expansion of the log-likelihood gradient and show that whereas CD-k truncates the series by keeping the first $2k$ terms and then approximates expectations by a single sample, reconstruction error is a mean-field approximation of the first term in that series.

6.5.1 Learning in an Autoassociator Network

Training an autoassociator network is almost identical to training a standard artificial neural network. Given a cost function, backpropagation is used to compute gradients and perform gradient descent. However, autoassociators are "self-supervised", meaning that the target to which the output of the autoassociator is compared is the input that it was fed.

Previous work on autoassociators minimized the squared reconstruction error:

$$C(\widehat{\mathbf{x}}, \mathbf{x}) = \sum_k (\widehat{x}_k - x_k)^2 \ .$$

However, with squared reconstruction error and linear decoder, the "optimal codes" (the implicit target for the encoder, irrespective of the encoder) are in the span of the principal eigenvectors of the input covariance matrix. When we introduce a saturating nonlinearity such as the sigmoid, and we want to reconstruct values $[0, 1]$, the binomial KL divergence (also known as cross-entropy) seems more appropriate:

$$C(\widehat{\mathbf{x}}, \mathbf{x}) = -\sum_k \left(x_k \log(\widehat{x}_k) + (1 - x_k) \log(1 - \widehat{x}_k)\right) \ . \tag{6.4}$$

It corresponds to the assumption that $\widehat{\mathbf{x}}$ and $\mathbf{x}$ can be interpreted as factorized distributions over binary units. It is well known that the cross-entropy

$$-p \log(q) - (1 - p) \log(1 - q)$$

between two binary distributions parametrized by p and q is minimized when $q = p$ (for a fixed p), making it an appropriate cost function to evaluate the quality of a reconstruction. We used this cost function in all the experiments with SAA networks. Appendix C details the corresponding autoassociator training update.

6.6 Experiments

In this section, we present several experiments set up to evaluate the deep network learning algorithms that fall in the paradigm presented in the Section 6.3.2 and highlight some of their properties. Unless otherwise stated, stochastic gradient descent was used for layer-wise unsupervised learning (first phase of the algorithm) and global supervised fine-tuning (second phase of the algorithm). The datasets were separated in disjoint training, validation and testing subsets. Model selection consisted of finding the best values for the learning rates of layer-wise unsupervised and global supervised learning as

well as the number of unsupervised updates preceding the fine-tuning phase. The number of epochs of fine-tuning was chosen using early-stopping based on the progression of classification error on the validation set. All experiments correspond to classification problems. Hence, to fine-tune the deep networks, we optimized the negative conditional log-likelihood of the training samples' target class (as given by the softmax output of the neural network).

The experiments are based on the MNIST dataset[2] (see Figure 6.5), a benchmark for handwritten digit recognition, as well as variants of this problem where the input distribution has been made more complex by inserting additional factors of variations, such as rotations and background images. The input images are made of 28×28 pixels giving an input dimensionality of 784, the number of classes is 10 (corresponding to the digits from 0 to 9) and the inputs were scaled between 0 and 1.

Successful applications of deep networks have already been presented on a large variety of data, such as images of faces [134], real-world objects [123] as well as text data [37, 80, 133, 163], and on different types of problems such as regression [134], information retrieval [133], robotics [66], and collaborative filtering [135].

In Bengio et al. [14], we performed experiments on two regression datasets, with non-image continuous inputs (UCI Abalone, and a financial dataset), demonstrating the use of unsupervised (or partially supervised) pre-training of deep networks on these tasks. In Larochelle et al. [95], we studied the performance of several architectures on various datasets, including variations of MNIST (with rotations, random background, and image background), and discrimination tasks between wide and tall rectangles, and between convex and non-convex images. On these tasks, deep networks compared favorably to shallow architectures.

Our focus is hence not on demonstrating their usefulness on a wide range of tasks, but on exploring their properties empirically. Such experimental work required several weeks of cumulative CPU time, which restricted the number of datasets we could explore. However, by concentrating on the original MNIST dataset and harder versions of it, we were able not only to confirm the good performance of deep networks, but also to

[2]This dataset can be downloaded from `http://yann.lecun.com/expdb/mnist/`

study practical variations, to help understand the algorithms, and to discuss the impact on a deep network's performance of stepping to a more complicated problem.

6.6.1 Validating the Unsupervised Layer-Wise Strategy for Deep Networks

In this section, we evaluate the advantages brought by the unsupervised layer-wise strategy of Section 6.3.2. We want to separate the different algorithmic concepts behind it, in order to understand their contribution to the whole strategy. In particular, we pursue the following two questions:

1. To what extent does initializing greedily the parameters of the different layers help?

2. How important is unsupervised learning for this procedure?

To address these two questions, we will compare the learning algorithms for deep networks of Sections 6.4 and 6.5 with the following algorithms.

Deep Network Without Pre-Training

To address the first question above, we compare the greedy layer-wise algorithm with a more standard way to train neural networks: using standard backpropagation and stochastic gradient descent, and starting at a randomly initialized configuration of the parameters. In other words, this variant simply puts away the pre-training phase of the other deep network learning algorithms.

Deep Network With Supervised Pre-Training

To address the second question, we run an experiment with the following algorithm. We greedily pre-train the layers using a *supervised criterion* (instead of the unsupervised one), before performing as before a final supervised fine-tuning phase. Specifically, when greedily pre-training the parameters $\mathbf{W}^i$ and $\mathbf{b}^i$, we also train another set of

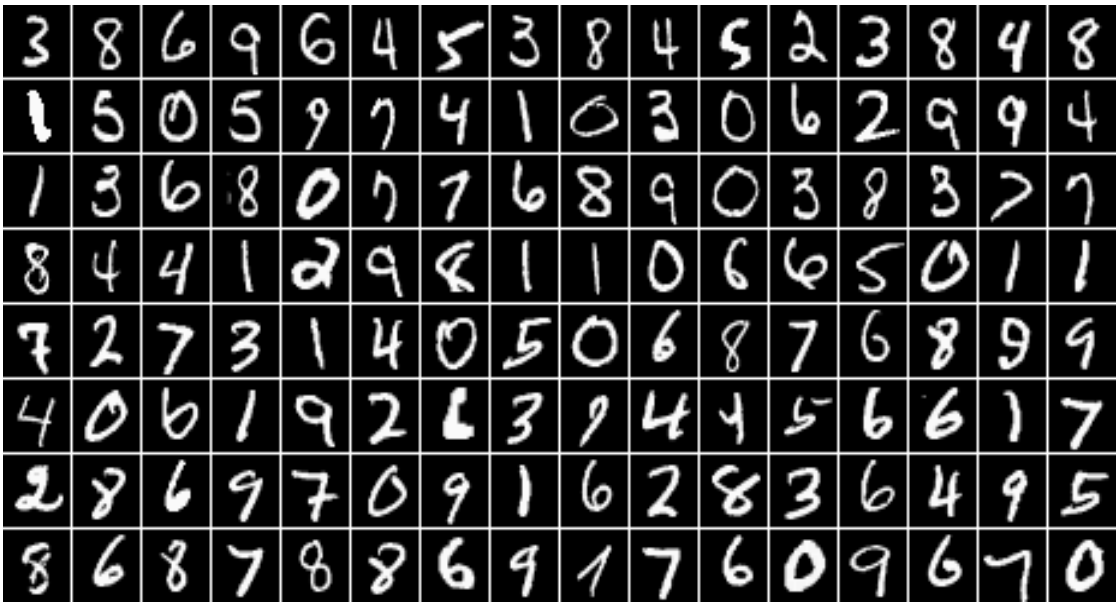


Figure 6.5: Samples from the MNIST digit recognition dataset. Here, a black pixel corresponds to an input value of 0 and a white pixel corresponds to 1 (the inputs are scaled between 0 and 1).

weights $\mathbf{V}^i$ and biases $\mathbf{c}^i$ which connect hidden layer $\widehat{\mathbf{h}}^i(\mathbf{x})$ to a temporary output layer as follows:

$$o^i(\mathbf{x}) = f\left(\mathbf{c}^i + \mathbf{V}^i\widehat{\mathbf{h}}^i(\mathbf{x})\right)$$

where $f(\cdot)$ is the softmax function of Equation 6.2. This output layer can be trained using the same cost as the global supervised cost. However, as this is a greedy procedure, only the parameters $\mathbf{W}^i$, $\mathbf{b}^i$, $\mathbf{V}^i$ and $\mathbf{c}^i$ are updated, i.e., the gradient is not propagated to the layers below. When the training of a layer is finished, we can simply discard the parameters $\mathbf{V}^i$ and $\mathbf{c}^i$ and move to pre-training the next hidden layer, having initialized $\mathbf{W}^i$ and $\mathbf{b}^i$.

Stacked Logistic Autoregression Network

The second question aims at evaluating to what extent any unsupervised learning can help. We already know that stacking linear PCA models is not expected to help improve generalization. A slightly more complex yet very simple unsupervised model for data in

$[0, 1]$ is the logistic autoregression model [see also 56]

$$\widehat{x}_k = \text{sigmoid}\left(b_k + \sum_{j \neq k} W_{kj} x_j\right) \tag{6.5}$$

where the reconstruction $\widehat{\mathbf{x}}$ is log-linear in the input $\mathbf{x}$. The parameters $\mathbf{W}$ and $\mathbf{b}$ can be trained using the same cost used for the autoassociators in Equation 6.4. This model can be used to initialize the weights $\mathbf{W}^i$ and biases $\mathbf{b}^i$ of the i-th hidden layer of a deep network. However, because $\mathbf{W}$ in Equation 6.5 is square, the deep network will need to have hidden layers with the same size as the input layer. Also, the weights on the diagonal of $\mathbf{W}$ are not trained in this model, so we initialize them to zero. The stacked logistic autoregression network will refer to deep networks using this unsupervised layer-wise learning algorithm.

Results

The results for all these deep networks are given in Table 6.1. We also give results for a "shallow", one hidden layer neural network, to validate the utility of deep architectures. Instead of the sigmoid, this network uses hyperbolic tangent squashing functions, which are usually found to work better for one hidden layer neural networks. The MNIST training set was separated into training (50,000) and validation (10,000) sets. The test set has size 10,000. In addition to the hyperparameters mentioned at the beginning of this section, the validation set was used also to select appropriate decrease constants[3] for the learning rates of the greedy and fine-tuning phases. The SRBM and SAA networks had 500, 500 and 2000 hidden units in the first, second and third layers respectively, as in Hinton et al. [81] and Hinton [78]. In the pre-training phase of the SRBM and SAA networks, when training the parameters of the i-th layer, the down-biases c_k where set to be equal to b_k^{i-1} (although similar results were obtained by using a separate set of biases c_k^{i-1} when the $i-1$-th layer is the down-layer). For the deep networks with supervised or no pre-training, different sizes of hidden layers were compared, including sizes similar

[3]When using a decrease constant β, the learning rate for the t^{th} update becomes $\frac{\epsilon_0}{1+t\beta}$, where ϵ_0 is the initial learning rate.

to the stacked logistic autoregression network, and to the SRBM and SAA networks. All deep networks had 3 hidden layers.

Overall, the models that use the unsupervised layer-wise procedure of Section 6.3.2 outperform those that do not. We also observe a slight advantage in the performance of the SRBM network over that of the SAA network (on the MNIST test set, differences of more than 0.1% are statistically significant). The performance difference between the stacked logistic autoregressions network and the deep network with supervised layer-wise pre-training particularly highlights the importance of unsupervised learning. Indeed, even though supervised layer-wise pre-training explicitly trains the hidden layers to capture non-linear information about the input, the overall procedure seems to be too greedy with respect to the supervised task to be learned. On the other hand, even though logistic autoregressions are simple log-linear models and their optimization is blind with respect to the future usage of the weights $\mathbf{W}$ as connections into non-linear hidden layers, the unsupervised nature of training makes them still useful for improving generalization. As a point of comparison, besides the deep networks, the best result on this dataset reported for a learning algorithm that does not use any prior knowledge about the task (e.g., image pre-processing like deskewing or subsampling) is that of a support vector machine with a Gaussian kernel[4], with 1.4% classification error on the test set.

At this point, it is clear that unsupervised layer-wise pre-training improves generalization. However, we could wonder whether it also facilitates the optimization problem of the global fine-tuning. The results of Table 6.1 do not shed any light on this aspect. Indeed, all the networks, even those without greedy layer-wise pre-training, perform almost perfectly on the training set. The explanatory hypothesis we evaluate here is that, without pre-training, the lower layers are initialized poorly, but still allow the top two layers to learn the training set almost perfectly because the output layer and the last hidden layer form a standard shallow but fat neural network. Consider the top two layers of the deep network *with pre-training*: it presumably takes as input a *better representation*, one that allows for better generalization. Instead, the network *without pre-training* sees a "random" transformation of the input, one that preserves enough information about the

[4]See `http://yann.lecun.com/exdb/mnist/` for more details

Models	Train.	Valid.	Test
SRBM (stacked restricted Boltzmann machines) network	0%	1.20%	1.20%
SAA (stacked autoassociators) network	0%	1.31%	1.41%
Stacked logistic autoregressions network	0%	1.65%	1.85%
Deep network with supervised pre-training	0%	1.74%	2.04%
Deep network, no pre-training	0.004%	2.07%	2.40%
Shallow network, no pre-training	0%	1.91%	1.93%

Table 6.1: Classification error on MNIST training, validation, and test sets, with the best hyperparameters according to validation error.

input to fit the training set, but that does not help to generalize. To test this hypothesis, we performed a second series of experiments in which we constrain the top hidden layer to be small (20 hidden units).

The results (Table 6.2) clearly suggest that optimization of the global supervised objective is made easier by greedy layer-wise pre-training. This result for supervised greedy pre-training is also coherent with past experiments on similar greedy strategies [52, 104]. Here, we have thus confirmed that it also applies to unsupervised greedy pre-training. With no pre-training, training error degrades significantly when there are only 20 hidden units in the top hidden layer. In addition, the results obtained without pre-training were found to have much larger variance than those with pre-training, indicating high sensitivity to initial conditions: the unsupervised pre-training more consistently puts the parameters in a "good" basin of attraction for the supervised gradient descent procedure.

Figures 6.6 and 6.7 show the sorts of first hidden layer features (weights going into different hidden neurons) that are learned by the first (bottom) RBM and autoassociator respectively, before fine-tuning. Both models were trained on the MNIST training set of Section 6.6.1 for 40 epochs, with 250 hidden units and a learning rate of 0.005. We see that they both learn visual features characterized by local receptive fields, which ought to be useful to recognize more global shapes (though the autoassociator also learns high frequency receptive fields that are spread over the whole image). This is another account of how unsupervised greedy pre-training is able to help the optimization of the network. Even if the supervised fine-tuning gradient at the first hidden layer is weak, we can see

Models	Train.	Valid.	Test
SRBM network	0%	1.5%	1.5%
SAA network	0%	1.38%	1.65%
Deep network with supervised pre-training	0%	1.77%	1.89%
Deep network, no pre-training	0.59%	2.10%	2.20%
Shallow network, no pre-training	3.6%	4.77%	5.00%

Table 6.2: Classification error on MNIST training, validation, and test sets, with the best hyperparameters according to validation error, when the last hidden layer only contains 20 hidden units

that the first hidden layer appears to learn a relevant representation.

6.6.2 Exploring the Space of Network Architectures

An important practical aspect in using deep network is the choice the architecture or topology of the network. Once we allow ourselves to consider an arbitrary number of hidden layers of arbitrary sizes, some questions naturally arise. First, we would like to know how deep a neural network can be made while still obtaining generalization gains, given a strategy for initializing its parameters (randomly or with unsupervised greedy pre-training). We would also like to know, for a determined depth, what type of architecture is more appropriate. Should the hidden layer's size increase, decrease or stay the same from the first to the last? In this section, we explore those two questions with experiments on the MNIST dataset as well as a variant, taken from Larochelle et al. [95], where the digit images have been randomly rotated. This last dataset, noted MNIST-rotation[5] (see Figure 6.8), contains much more intraclass variability, is much less well described by relatively well separated class-specific clusters and corresponds to a much harder classification problem. The training, validation and test sets contain 10 000, 2 000 and 50 000 examples each. We also generated sets of the same size for the MNIST dataset. We refer to this version with a smaller training set by MNIST-small.

[5]This dataset has been regenerated since Larochelle et al. [95] and is available here: `http://www.iro.umontreal.ca/~lisa/icml2007`.

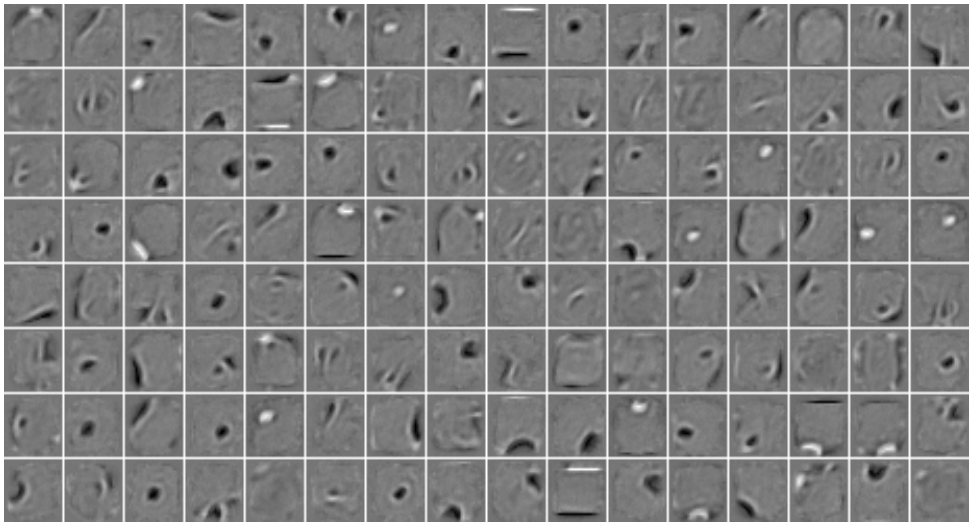


Figure 6.6: Display of the input weights of a random subset of the hidden units, learned by an RBM when trained on samples from the MNIST dataset. The activation of units of the first hidden layer is obtained by a dot product of such a weight "image" with the input image. In these images, a black pixel corresponds to a weight smaller than -3 and a white pixel to a weight larger than 3, with the different shades of gray corresponding to different weight values uniformly between -3 and 3.

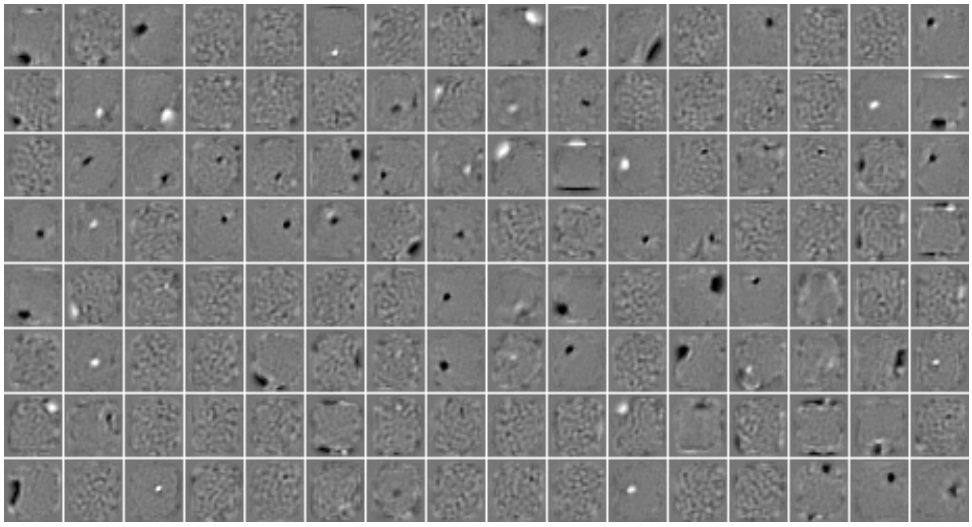


Figure 6.7: Input weights of a random subset of the hidden units, learned by an autoassociator when trained on samples from the MNIST dataset. The display setting is the same as for Figure 6.6.

Figure 6.8: Samples from the MNIST-rotation dataset. Here, a black pixel corresponds to an input value of 0 and a white pixel corresponds to 1 (the inputs are scaled between 0 and 1).

Network Depth

One can wonder whether a neural network can be made too deep, i.e., whether having too many hidden layers can worsen the generalization performance. Of course there are many reasons why this may happen. When a neuron is added, more parameters are inserted in the mathematical formulation of the model, giving it more degrees of freedom to fit the model and hence possibly making it able to overfit. On the other hand, it is less clear to what extent the performance can worsen, since a neuron added at the top layer of a neural network does not increase the capacity the same way a neuron added "in parallel" in a given hidden layer. Also, in the case of an SRBM network, we can imagine that as we stack RBMs, the representation at a hidden layer contains units that correspond to more and more disentangled concepts of the input. Now, consider a hypothetical deep network where the top-level stacked RBM has learned a representation made of units that are mostly independent. An additional RBM stacked on this representation would have no statistical structure to learn. This would initialize the weights of that

new RBM to zero, which is particularly troublesome as the representation at this level would then contain no information about the input. It is not clear if this scenario is plausible and one might even argue that it is not, because unlike the case of directed models like independent component analysis, no independence assumption is made in the prior distribution of the hidden layer of an RBM. But if it were approached the result would be detrimental to supervised classification performance. This particular situation is not expected with stacked autoassociators, as it will always learn a representation from which the previous layer can be reconstructed. Another reason why a deeper architecture could produce worse results is simply that our algorithms for training a deep architecture can probably be improved. In particular, note that the only joint training of all layers that we have done in our experiments, if any, is at the supervised fine-tuning stage.

Table 6.3 presents the classification performance obtained by the different deep networks with up to 4 hidden layers on MNIST-small and MNIST-rotation. The hyperparameters of each layer were *separately* selected with the validation set for all hidden layers, using the following greedy strategy: for a network with l hidden layers, only the hyperparameters for the top layer were optimized, the hyperparameters for the layers below being set to those of the best $l-1$ layers deep network according to the validation performance. We settled for this strategy because of the exponential number of possible configurations of hyperparameters. For standard neural networks, we also tested several random initializations of the weights. For SRBM as well as SAA networks, we tuned the unsupervised learning rates and the number of updates. For MNIST-small, we used hidden layers of 500 neurons, since the experiments by Hinton [78] suggest that it is an appropriate choice. As for MNIST-rotation, the size of each hidden layer had to be validated separately for each layer, and we tested values among 500, 1000, 2000 and 4000.

Table 6.3 show that there is indeed an optimal number of hidden layers for the deep networks, and that this optimum tends to be larger when unsupervised greedy layer-wise learning is used. For the MNIST-small dataset (Table 6.3), the gain in performance between 2 and 3 hidden layers for SRBM and SAA networks is not statistically significant. However, for the MNIST-rotation dataset, the improvement from 2 to 3 hidden layers is clear. This observation is consistent with the increased complexity of the input dis-

Network		MNIST-small	MNIST-rotation
Type	Depth	classif. test error	classif. test error
Neural network (random initialization, + fine-tuning)	1	**4.14** % ± 0.17	15.22 % ± 0.31
	2	**4.03** % ± 0.17	**10.63** % ± 0.27
	3	**4.24** % ± 0.18	11.98 % ± 0.28
	4	4.47 % ± 0.18	11.73 % ± 0.29
SAA network (autoassociator learning + fine-tuning)	1	3.87 % ± 0.17	11.43% ± 0.28
	2	**3.38** % ± 0.16	9.88 % ± 0.26
	3	**3.37** % ± 0.16	**9.22** % ± 0.25
	4	**3.39** % ± 0.16	**9.20** % ± 0.25
SRBM network (CD-1 learning + fine-tuning)	1	3.17 % ± 0.15	10.47 % ± 0.27
	2	**2.74** % ± 0.14	9.54 % ± 0.26
	3	**2.71** % ± 0.14	**8.80** % ± 0.25
	4	**2.72** % ± 0.14	**8.83** % ± 0.24

Table 6.3: Classification performance on MNIST-small and MNIST-rotation of different networks for different strategies to initialize parameters, and different depths (number of layers).

tribution and classification problem of MNIST-rotation, which should require a more complex model. The improvement remains significant when fixing the network's hidden layers to the same size as in the experiments on MNIST-small, as showed in the results of Table 6.4 where the number of units per hidden layer was set to 1000. We also compared the performance of shallow and deep SRBM networks with roughly the same number of parameters. With a shallow SRBM network, the best classification error achieved was 10.47%, with 4000 hidden units (around 3.2×10^6 free parameters). With a 3-layers deep SRBM network, we reached 9.38% classification error with 1000 units in each layer (around 2.8×10^6 parameters): better generalization was achieved with deeper nets having less parameters.

Type of Network Architecture

The model selection procedure of Section 6.6.2 works well, but is rather expensive. Every time one wants to train a 4 hidden layer network, networks with 1, 2 and 3 hidden layers effectively have to be trained as well, in order to determine appropriate hyperpa-

Network			MNIST-rotation
Type	Depth	Layers width	classif. test error
SRBM network	1	1k	12.44 % $\pm$ 0.29
(CD-1 learning	2	1k, 1k	9.98 % $\pm$ 0.26
+ fine-tuning)	3	1k, 1k, 1k	**9.38** % $\pm$ 0.25

Table 6.4: Classification performance on MNIST-rotation of different networks for different strategies to initialize parameters, and different depths (number of layers). All hidden layers have 1000 units.

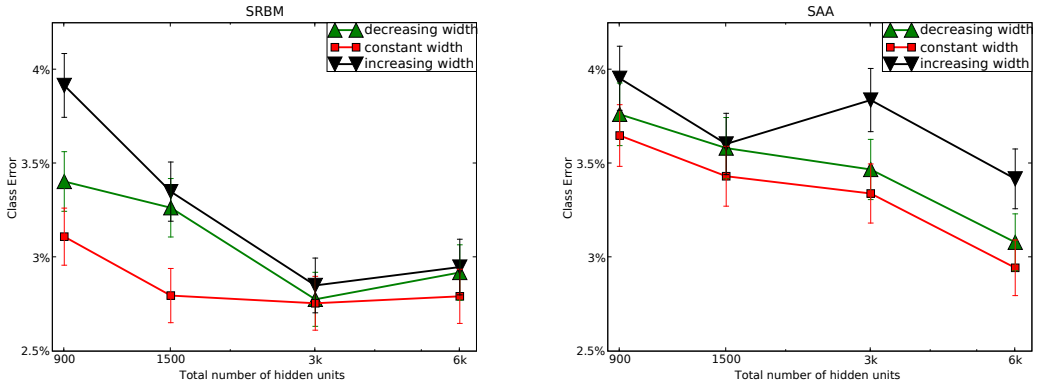
rameters for the lower hidden layers. These networks can't even be trained in parallel, adding to the computational burden of this model selection procedure. Moreover, the optimal hidden layer size for a 1-hidden layer network could be much bigger than necessary for a 4 hidden layer network, since a shallow network cannot rely on other upper layers to increase its capacity.

Let us consider the situation where the number of hidden layers of a deep network has already been chosen and good sizes of the different layers must be found. Because the space of such possible choices is exponential in the number of layers, we consider here only three general cases where, as the layer index increases, their sizes either increases (doubles), decreases (halves) or does not change. We conducted experiments for all three cases and varied the total number of hidden neurons in the network. The same hyperparameters as in the experiment of Table 6.3 had to be selected for each network topologies, however a single unsupervised learning rate and number of updates were chosen for all layers[6].

We observe in Figures 6.9 and 6.10 that the architecture that most often is among the best performing ones across the different sizes of network is the one with equal sizes of hidden layers. It should be noted that this might be a consequence of using the same unsupervised learning hyperparameters for each layer. It might be that the size of a hidden layer has a significant influence on the optimum value for these hyperparameters, and that tying them for all hidden layers induces a bias towards networks with equally-sized hidden layers. However, having untied hyperparameters would make model selection

[6]We imposed this restriction because of the large number of experiments that would otherwise had been required.

too computationally demanding. Actually, even with tied unsupervised learning hyperparameters, the model selection problem is already complex enough (and prone to overfitting with small datasets), as is indicated by the differences in the validation and test classification errors of Table 6.3.



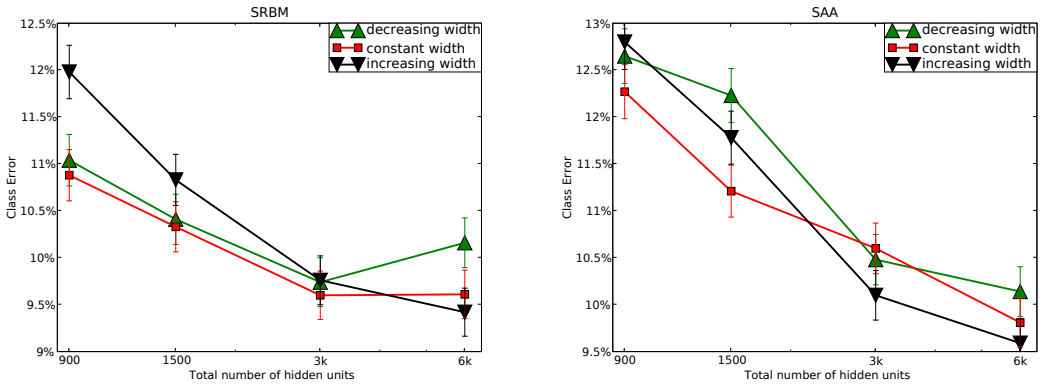
(a) SRBM network.

(b) SAA network.

Figure 6.9: Classification performance on MNIST-small of 3-layer deep networks for three kinds of architectures, as a function of the total number of hidden units. The three architectures have increasing / constant / decreasing layer sizes from the bottom to the top layers. Error-bars represent 95% confidence intervals.

6.7 Continuous-Valued Inputs

In this section, we wish to emphasize the importance of adapting the unsupervised learning algorithms to the nature of the inputs. We will focus on the SRBM network because they rely on RBMs, which are less simple to work with and adapt to the sorts of visible data we want to model. With the binary units introduced for RBMs and DBNs in Hinton et al. [81] one can "cheat" and handle continuous-valued inputs by scaling them to the $[0, 1]$ interval and considering each input continuous value as the probability for a binary random variable to take the value 1. This has worked well for gray-level pixels of handwriting images, which are almost binary, but it may be inappropriate for other kinds of input variables. Previous work on continuous-valued input in RBMs include Chen et Murray [35], in which noise is added to sigmoidal units, and the RBM



(a) SRBM network.

(b) SAA network.

Figure 6.10: Classification performance on MNIST-rotation of 3-layer deep networks for three kinds of architectures. Same conventions as in Figure 6.9.

forms a special form of diffusion network [113]. Welling et al. [162] also show how to derive RBMs from arbitrary choices of exponential distributions for the visible and hidden layers of an RBM. We show here simple extensions of the RBM framework in which only the energy function and the allowed range of values are changed. As can be seen in Figures 6.11 and 6.12 and in the experiment of Section 6.7.3, such extensions have a very significant impact on nature of the solution learned for the RBM's weights and hence on the initialization of a deep network and its performance.

6.7.1 Linear Energy: Exponential or Truncated Exponential

Consider a unit with value x_k in an RBM, connected to units $\mathbf{h}$ of the layer above. $p(x_k|\mathbf{h})$ can be obtained by considering the terms in the energy function that contain x_k. These terms can be grouped in $x_k(\mathbf{W}_{.k}^{\mathbf{T}}\mathbf{h} + c_k)$ when the energy function is linear in x_k (as in Equation 6.7, appendix B), where $\mathbf{W}_{.k}$ is the k-th column of $\mathbf{W}$. If we allow x_k to take any value in interval I, the conditional density of x_k becomes

$$p(x_k|\mathbf{h}) = \frac{e^{x_k(\mathbf{W}_{.j}^{\mathbf{T}}\mathbf{h}+c_k)}\mathbf{1}_{x_k \in I}}{\int_v e^{v(\mathbf{W}_{.j}^{\mathbf{T}}\mathbf{h}+c_k)}\mathbf{1}_{v \in I} dv} \ .$$

When $I = [0, \infty)$, this is an exponential density with parameter $a(\mathbf{h}) = \mathbf{W}_{\cdot j}^{\mathbf{T}}\mathbf{h} + c_k$, and the normalizing integral, equal to $\frac{-1}{a(\mathbf{h})}$, only exists if $a(\mathbf{h}) < 0 \; \forall \mathbf{h}$. Computing the density, the expected value ($\frac{-1}{a(\mathbf{h})}$) and sampling would all be easy, but since the density does not always exist it seems more appropriate to let I be a closed interval, yielding a *truncated exponential* density. For simplicity we consider the case $I = [0, 1]$ here, for which the normalizing integral, which always exists, is

$$\frac{e^{-a(\mathbf{h})} - 1}{a(\mathbf{h})} \; .$$

The conditional expectation of x_k given $\mathbf{h}$ is interesting because it has a sigmoidal-like saturating and monotone non-linearity:

$$E\left[x_k | \mathbf{h}\right] = \frac{1}{1 - e^{-a(\mathbf{h})}} - \frac{1}{a(\mathbf{h})} \; .$$

Note that $E\left[x_k | \mathbf{h}\right]$ does not explode for $a(\mathbf{h})$ near 0, but is instead smooth and in the interval $[0, 1]$. A sample from the truncated exponential is easily obtained from a uniform sample U, using the inverse cumulative F^{-1} of the conditional density $p(x_k|\mathbf{h})$:

$$F^{-1}(U) = \frac{\log(1 - U \times (1 - e^{a(\mathbf{h})}))}{a(\mathbf{h})} \; .$$

The contrastive divergence updates have the same form as for binary units of Equation 6.11, since the updates only depend on the derivative of the energy with respect to the parameters. Only sampling is changed, according to the unit's conditional density. Figure 6.11 shows the filters learned by an RBM with truncated exponential visible units, when trained on MNIST samples. Note how these are strikingly different from those obtained with binomial units.

6.7.2 Quadratic Energy: Gaussian Units

To obtain Gaussian-distributed units, one only needs to add quadratic terms to the energy. Adding $\sum_k d_k^2 x_k^2$ gives rise to a diagonal covariance matrix between units of

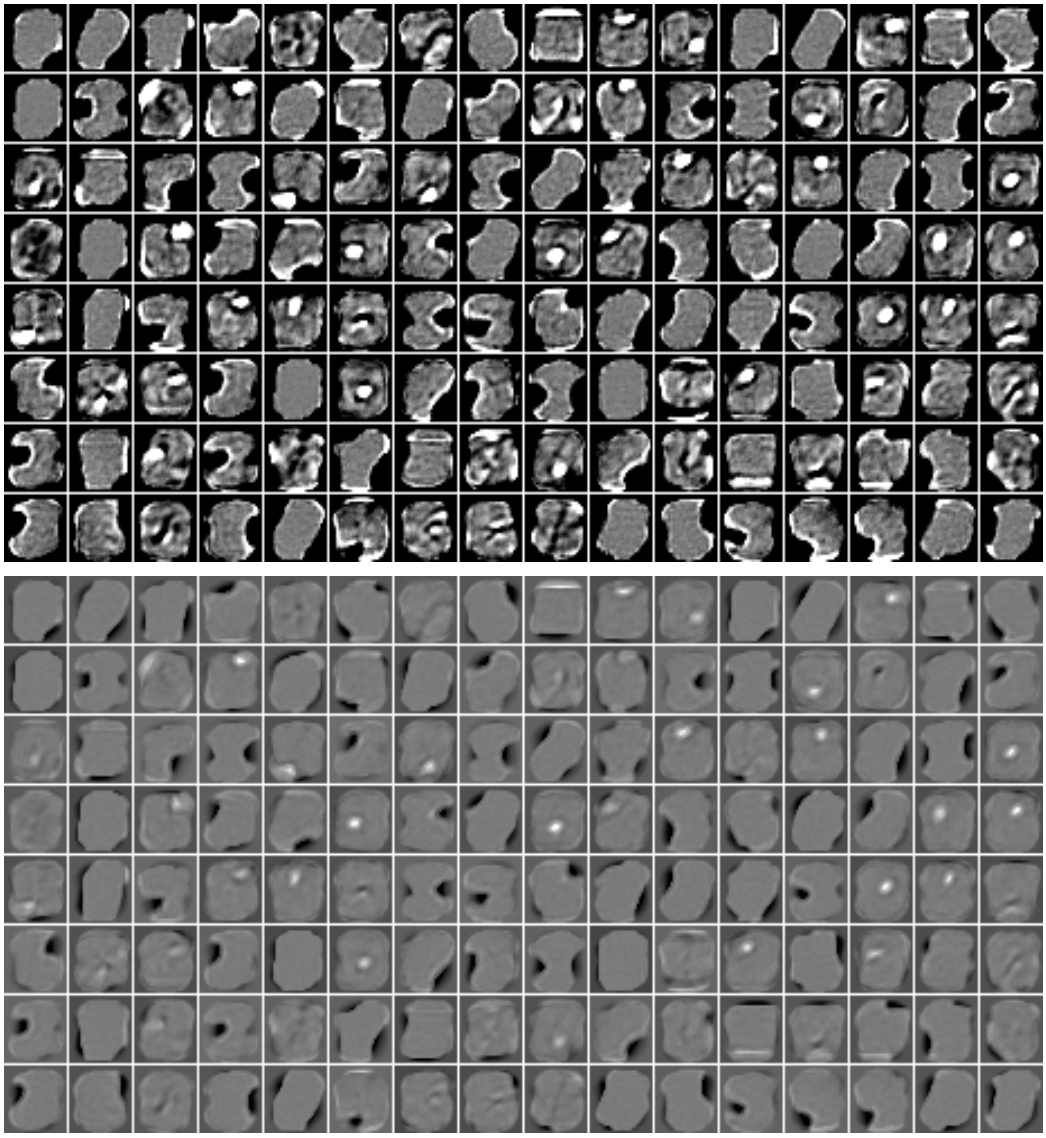


Figure 6.11: Input weights of a random subset of the hidden units, learned by an RBM with truncated exponential visible units, when trained on samples from the MNIST dataset. The top and bottom images correspond to the same filters but with different color scale. On the top, the display setup is the same as for Figures 6.6 and 6.7 and, on the bottom, a black and white pixel correspond to weights smaller than -30 and larger than 30 respectively.

the same layer, where x_k is the continuous value of a Gaussian unit and d_k^2 is a positive parameter that is equal to the inverse of the variance of x_k. In this case the variance is unconditional, whereas the mean depends on the inputs of the unit: for a visible unit x_k with hidden layer $\mathbf{h}$ and inverse variance d_k^2,

$$E\left[x_k|\mathbf{h}\right] = \frac{a(\mathbf{h})}{2d_k^2} \ .$$

The contrastive divergence updates are easily obtained by computing the derivative of the energy with respect to the parameters. For the parameters in the linear terms of the energy function $\mathbf{b}$, $\mathbf{c}$ and $\mathbf{W}$, the derivatives have the same form as for the case of binary units. For quadratic parameter $d_k > 0$, the derivative is simply $2d_k x_k^2$. Figure 6.12 shows the filters learned by an RBM with Gaussian visible units, when trained on MNIST samples.

Gaussian units were previously used as hidden units of an RBM (with multinomial inputs) applied to an information retrieval task [162]. That same paper also shows how to generalize RBMs to units whose marginal distribution is from any member of the exponential family.

6.7.3 Impact on Classification Performance

In order to assess the impact of the choice for the visible layer distribution on the ultimate performance of an SRBM network, we trained and compared different deep networks whose first level RBM had binary, truncated exponential or Gaussian input units. These networks all had 3 hidden layers, with 2000 hidden units for each of these layers. The hyperparameters that were optimized are the unsupervised learning rates and number of updates as well as the fine-tuning learning rate. Because the assumption of binary inputs is not unreasonable for the MNIST images, we conducted this experiment on a modified and more challenging version of the dataset where the background contains patches of images downloaded from the Internet. Samples from this dataset are shown

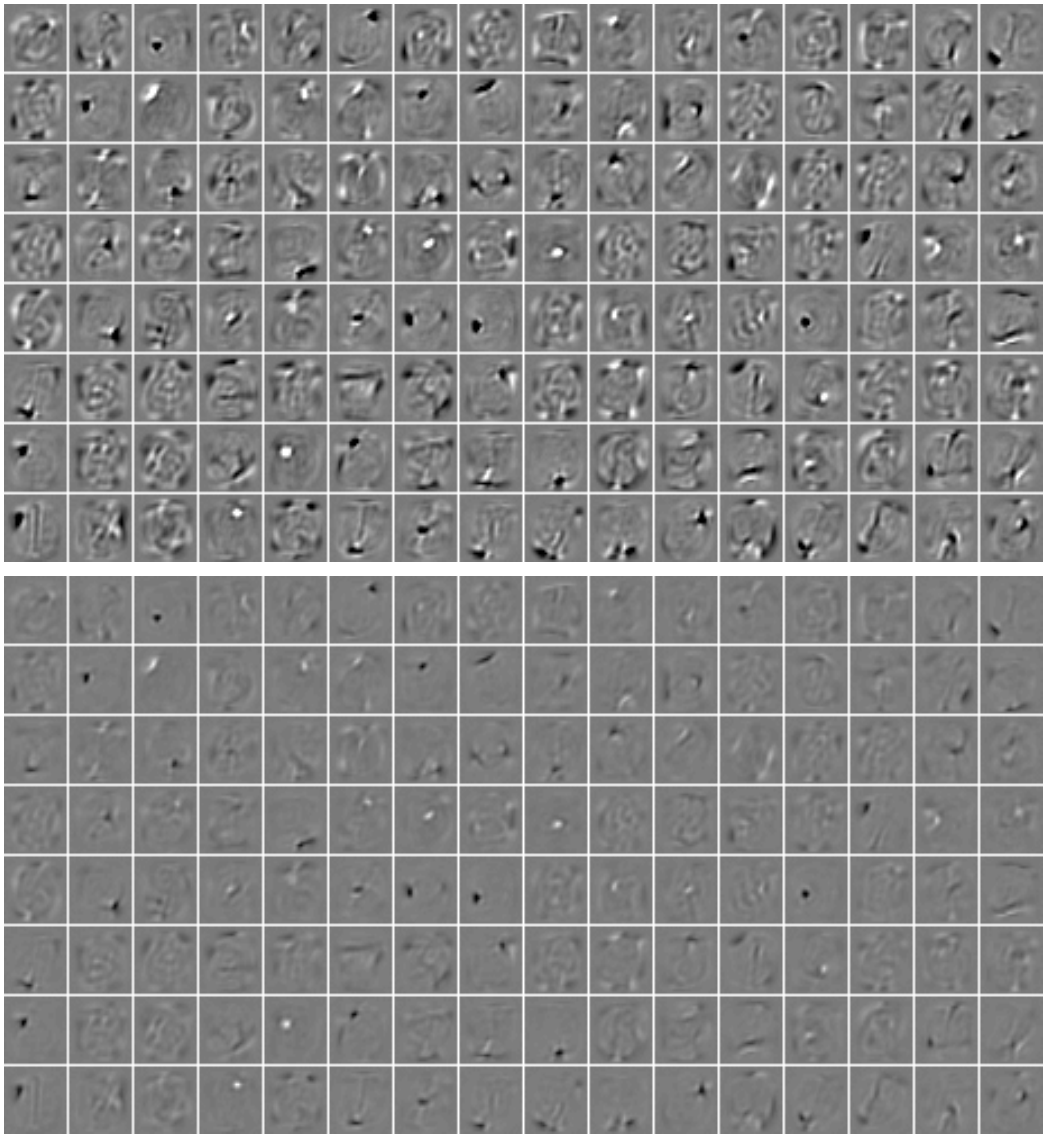


Figure 6.12: Input weights of a random subset of the hidden units, learned by an RBM with Gaussian visible units, when trained on samples from the MNIST dataset. The top and bottom images correspond to the same filters but with different color scale. On top, the display setup is the same as for Figures 6.6 and 6.7 and, on the bottom, a black and white pixel correspond to weights smaller than -10 and larger than 10 respectively.

in Figure 6.13. This dataset is part of a benchmark[7] designed by Larochelle et al. [95]. The results are given in Table 6.5, where we can see that the choice of the input distribution has a significant impact on the classification performance of the deep network. As a comparison, a support vector machine with Gaussian kernel achieves 22.61% error on this dataset [95]. Other experimental results with truncated exponential and Gaussian input units are found in Bengio et al. [14].

6.8 Generating vs Encoding

Though the SRBM and SAA networks are similar in their motivation, there is a fundamental difference in the type of unsupervised learning used during training. Indeed, the RBM is based on the learning algorithm of a *generative model*, which is trained to be able to generate data similar to those found in the training set. On the other hand, the autoassociator is based on the learning algorithm of an *encoding model* which tries to learn a new representation or code from which the input can be reconstructed without too much loss of information.

It is not clear which of the two approaches (generating or encoding) is the most appropriate. The advantage of a generative model is that the assumptions that are made are usually clear. However, it is possible that the problem it is trying to solve is harder than it needs to be, since ultimately we are only interested in coming up with good representations or features of the input. For instance, if one is interested in finding appropriate clusters in a very high dimensional space, using a mixture of Gaussians with full covariance matrix can quickly become too computationally intensive, whereas using the simple k-means algorithm might do a good enough job. As for encoding models, they do not require to be interpretable as a generative model and they can be more flexible because any parametric or non-parametric form can be chosen for the encoder and decoder, as long as they are differentiable.

Another interesting connection between reconstruction error in autoassociators and CD in RBMs was mentioned earlier: the reconstruction error can be seen as an estimator

[7]The benchmark's datasets can be downloaded from `http://www.iro.umontreal.ca/~lisa/icml2007`

Figure 6.13: Samples from the modified MNIST digit recognition dataset with a background containing image patches. Here, a black pixel corresponds to an input value of 0 and a white pixel corresponds to 1 (the inputs are scaled between 0 and 1).

of the log-likelihood gradient of the RBM which has more bias but less variance than the CD update rule [11]. In that paper it is shown how to write the RBM log-likelihood gradient as a series expansion where each term is associated with a sample of the contrastive divergence Gibbs chain. Because the terms become smaller and converge to zero, this justifies taking a truncation of the series as an estimator of the gradient. The reconstruction error gradient is a mean-field (i.e., biased) approximation of the first term, whereas CD-1 is a sampling (i.e., high-variance) approximation of the first two terms, and similarly CD-k involves the first $2k$ terms.

This suggests combining the reconstruction error and contrastive divergence for training RBMs. During unsupervised pre-training, we can use the updates given by both algorithms and combine them by associating a coefficient to each of them. This is actually equivalent to applying the updates one after the other but using different learning rates for both. We tested this idea in the MNIST dataset split of Section 6.6.1, where we had to validate separately the learning rates for the RBM and the autoassociator updates. This combination improved on the results of the SRBM and the SAA networks, obtain-

SRBM input type	Train.	Valid.	Test
Bernoulli	10.50%	18.10%	20.29%
Gaussian	0%	20.50%	21.36%
Truncated exponential	0%	14.30%	14.34%

Table 6.5: Classification error on MNIST with background containing patches of images (see Figure 6.13) on the training, validation, and test sets, for different distributions of the input layer for the bottom RBM. The best hyperparameters were selected according to the validation error.

ing 1.02% and 1.09% on the validation and test set respectively. This improvement was confirmed in a more complete experiment on 6 other folds with mutually exclusive test sets of 10 000 examples, where the mixed gradient variant gave on average a statistically significant improvement of 0.1% on a SRBM network. One possible explanation for the improvement brought by this combination is that it uses a better trade-off between bias and variance in estimating the log-likelihood gradient.

Another deterministic alternative to CD is mean-field CD (MF-CD) of Welling et Hinton [161], and is equivalent to the pseudocode code in Appendix B, with the statements $\mathbf{h}^0 \sim p(\mathbf{h}|\mathbf{x}^0)$ and $\mathbf{v}^1 \sim p(\mathbf{x}|\mathbf{h}^0)$ changed to $\mathbf{h}^0 \leftarrow \text{sigmoid}(\mathbf{b} + \mathbf{W}\mathbf{v}^0)$ and $\mathbf{v}^1 \leftarrow \text{sigmoid}(\mathbf{c} + \mathbf{W}^\mathbf{T}\mathbf{h}^0)$ respectively. MF-CD can be used to test another way to change the bias/variance trade-off, either as a gradient estimator alone, or by combining it to the CD-1 gradient estimate (in the same way we combined the autoassociator gradient with CD-1, previous paragraph). On the MNIST split of Section 6.6.1, SRBM networks with MF-CD and combined CD-1/MF-CD[8] achieved 1.26% and 1.17% on the test set respectively. The improvement brought by combining MF-CD with CD-1 was not found to be statistically significant, based on similar experiments on the 6 other folds.

This suggests that something else than the bias/variance trade-off is at play in the improvements observed when combining CD-1 with the autoassociator gradient. A hypothesis that should be explored is that whereas there is no guarantee that an RBM will encode in its hidden representation all the information in the input vector, an autoassociator is trying to achieve this. In fact an RBM trained by maximum likelihood would be glad to completely ignore the inputs if these were independent of each other. Minimizing

[8]The weight of the CD-1 and MF-CD gradient estimates was considered as a hyperparameter.

the reconstruction error would prevent this, and may be useful in the context where the representations are later used for supervised classification (which is the case here).

6.9 Continuous Training of all Layers of a Deep Network

The layer-wise training algorithm for networks of depth l actually has $l + 1$ separate training phases: first the l phases for the unsupervised training of each layer, and then the final supervised fine-tuning phase to adjust all the parameters simultaneously. One element that we would like to dispense with is having to decide the number of unsupervised training iterations for each layer before starting the fine-tuning. One possibility is then to execute all phases simultaneously, i.e., train all layers based on both their greedy unsupervised and global supervised gradients. The advantage is that we can now have a single stopping criterion (for the whole network). However, computation time is slightly greater, since we do more computations initially on the upper layers, which might be wasted before the lower layers converge to a decent representation, but time is saved on optimizing hyperparameters. When this continuous training variant is used on the MNIST dataset with the same experimental setup as in Section 6.6.1, we reach 1.6% and 1.5% error on the test set respectively for the SRBM network and the SAA network, so unsupervised learning still brings better generalization in this setting. This variant may be more appealing for on-line training on very large datasets, where one would never cycle back on the training data.

However, there seems to be a price to pay in terms of classification error, with this online variant. In order to investigate what could be the cause, we experimented with a 2-phase algorithm designed to shed some light on the contribution of different factors to this decrease. In the first phase, all layers of networks were simultaneously trained according to their unsupervised criterion *without* fine-tuning. The output layer is still trained according to the supervised criterion, however, unlike in Section 6.6.1, the gradient is not backpropagated into the rest of the network. This allows us to monitor the discriminative capacity of the top hidden layer. This first phase also enables us to verify whether the use of the supervised gradient too early during training explains the de-

crease in performance (recall the poor results obtained with purely supervised greedy layer-wise training). Then, in the second phase, 2 options were considered:

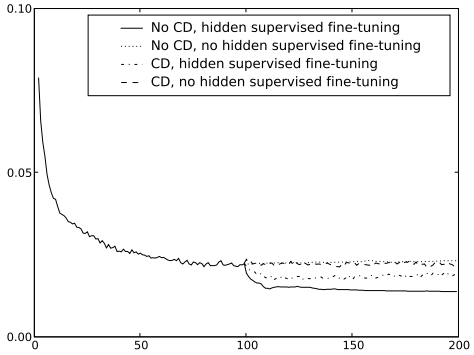
1. fine-tune the whole network according to the supervised criterion and stop layer-wise unsupervised learning;

2. fine-tune the whole network and maintain layer-wise unsupervised learning (as in the previous experiment).

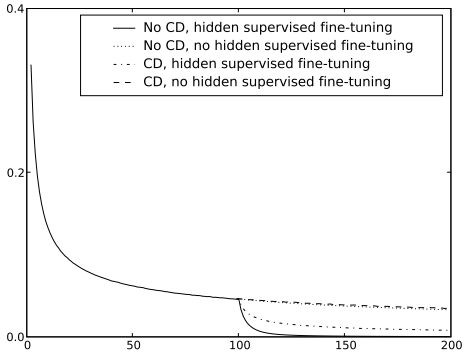
Figures 6.14(a) and 6.15(a) show examples of the progression of the test classification error for such an experiment with the SRBM and SAA networks respectively. As a baseline for the second phase, we also give the performance of the networks when unsupervised learning is stopped and only the parameters of the output layer are trained. These specific curves do not correspond to the best values of the hyperparameters, but are representative of the global picture we observed on several runs with different hyperparameter values.

We observe that the best option is to perform fine-tuning without layer-wise unsupervised learning, even when supervised learning is not introduced at the beginning. Also, though performing unsupervised and supervised learning at the same time outperforms unsupervised learning without fine-tuning, it appears to yield over-regularized networks, as indicated by the associated curves of the training negative log-likelihood of the target classes for both networks (see Figures 6.14(b) and 6.15(b)). Indeed, we see that by maintaining some unsupervised learning, the networks are not able to optimize as well their supervised training objective. From other runs with different learning rates, we have observed that this effect becomes less visible when the supervised learning rate gets larger, which reduces the relative importance of the unsupervised updates. But then the unsupervised updates usually bring significant instabilities in the learning process, making even the training cost oscillate.

Another interesting observation is that, when layer-wise unsupervised learning is performed, the classification error is less stable in an SRBM network than in an SAA network, as indicated by the dented learning curves in Figure 6.14(b), whereas the curves in Figure 6.15(b) are smoother. This may be related to the better performance of the SAA

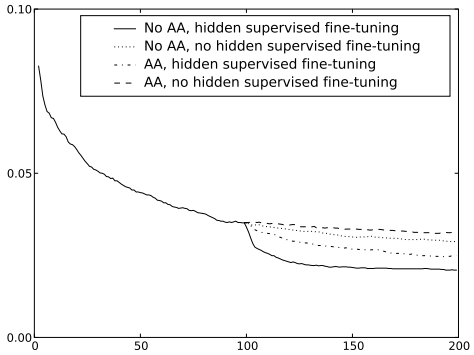


(a) SRBM network, test classification error curves

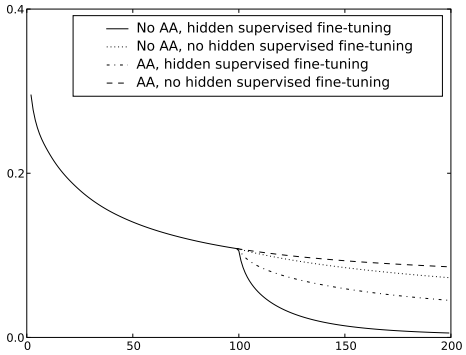


(b) SRBM network, train NLL error curves.

Figure 6.14: Example of learning curves of the 2-phase experiment of Section 6.9. During the first half of training, all hidden layers are trained according to CD and the output layer is trained according to the supervised objective, for all curves. In the second phase, all combinations of two possibilities are displayed: CD training is performed at all hidden layers ("CD") or not ("No CD"), and all hidden layers are fine-tuned according to the supervised objective ("hidden supervised fine-tuning") or not ("no hidden supervised fine-tuning").



(a) SAA network, test classification error curves



(b) SAA network, train NLL error curves.

Figure 6.15: Same as Figure 6.14, but with autoassociators ("AA") used for layer-wise unsupervised learning.

network (1.5%) versus the SRBM network (1.6%) when combining unsupervised and supervised gradients, in the experiment reported at the beginning of this section. Autoassociator learning might hence be more appropriate here, possibly because its training objective, i.e., the discovery of a representation that preserves the information in the input, is more compatible with the supervised training objective, which asks that the network discovers a representation that is predictive of the input's class. This hypothesis is related to the one presented at the end of Section 6.8 regarding the apparent improvement brought by minimizing the reconstruction error in addition to CD-1 updates.

These experiments show that one can eliminate the multiple unsupervised phases: each layer can be pre-trained in a way that simply ignores what the layer above are doing. However, it appears that a final phase involving only supervised gradient yields the best performance. A plausible explanation of these results, and in particular the quick improvement when the unsupervised updates are removed, is that the unsupervised pre-training brings the parameters near a good solution for the supervised criterion, but far enough from that solution to yield a significantly higher classification error. Note that in a setting where there is little labeled data but a lot of unlabelled examples, the additional regularization introduced by maintaining some unsupervised learning might be beneficial [132].

6.10 Conclusion

In this paper, we discussed in detail three principles for training deep neural networks, which are (1) pre-training one layer at a time in a greedy way (2) using unsupervised learning at each layer in a way that preserves information from the input and disentangles factors of variation and (3) fine-tuning the whole network with respect to the ultimate criterion of interest. We also presented experimental evidence that supports the claim that they are key ingredients for reaching good results. Moreover, we presented a series of experimental results that shed some light on many aspects of deep networks: confirming that the unsupervised procedure helps the optimization of the deep architecture, while initializing the parameters in a region near which a good solution of

the supervised task can be found. Our experiments showed cases where greater depth clearly helps, but too much depth could be slightly detrimental. We found that CD-1 can be improved by combining it with the gradient of reconstruction error, and that this is not just due to the use of a lower-variance update. We showed that the choice of input distribution in RBMs could be important for continuous-valued input and yielded different types of filters at the first layer. Finally we studied variants more amenable to online learning in which we show that if different training phases can be combined, the best results were obtained with a final fine-tuning phase involving only the supervised gradient.

There are many questions and issues that remain to be addressed and that we intend to investigate in future work. As pointed out in Section 6.8, the most successful unsupervised learning approach seems to fall in between generative and encoding approaches. This raises questions about what are the properties of a learning algorithm that learns good representations for deep networks. Finding good answers to these questions would have a direct positive impact on the performance of deep networks. Finally, better model selection techniques that would permit to reduce the number of hyperparameters would be beneficial and will need to be developed for deep network learning algorithms to become easier to use.

Acknowledgements

The author are particularly grateful for the inspiration from and constructive discussions with Dan Popovici, Aaron Courville, Olivier Delalleau, James Bergstra, and Dumitru Erhan. The authors also want to thank the editor and reviewers for their helpful comments and suggestions. This research was performed thanks to funding from NSERC, MITACS, and the Canada Research Chairs.

Appendix A: Pseudocode for Greedy Layer-Wise Training Paradigm

Input: training set $\mathcal{D} = \{(\mathbf{x}_t, y_t)\}_{t=1}^{T}$, pre-training learning rate $\epsilon_{\text{pre-train}}$ and fine-tuning learning rate $\epsilon_{\text{fine-tune}}$

Initialize weights $\mathbf{W}_{jk}^{i} \sim U(-a^{-0.5}, a^{-0.5})$ with $a = \max(|\widehat{\mathbf{h}}^{i-1}|, |\widehat{\mathbf{h}}^{i}|)$ and set biases $\mathbf{b}^{i}$ to 0

% Pre-training phase
for $i \in \{1, \ldots, l\}$ **do**
  while Pre-training stopping criterion is not met **do**
    Pick input example $\mathbf{x}_t$ from training set
    $\widehat{\mathbf{h}}^{0}(\mathbf{x}_t) \leftarrow \mathbf{x}_t$
    for $j \in \{1, \ldots, i-1\}$ **do**
      $\mathbf{a}^{j}(\mathbf{x}_t) = \mathbf{b}^{j} + \mathbf{W}^{j}\widehat{\mathbf{h}}^{j-1}(\mathbf{x}_t)$
      $\widehat{\mathbf{h}}^{j}(\mathbf{x}_t) = \text{sigmoid}\left(\mathbf{a}^{j}(\mathbf{x}_t)\right)$
    end for
    Using $\widehat{\mathbf{h}}^{i-1}(\mathbf{x}_t)$ as input example, update weights $\mathbf{W}^{i}$ and biases $\mathbf{b}^{i-1}$, $\mathbf{b}^{i}$ with learning rate $\epsilon_{\text{pre-train}}$ according to a layer-wise unsupervised criterion (see pseudocodes in appendices B and C)
  end while
end for

% Fine-tuning phase
while Fine-tuning stopping criterion is not met **do**
  Pick input example $(\mathbf{x}_t, y_t)$ from training set

  % Forward propagation
  $\widehat{\mathbf{h}}^{0}(\mathbf{x}_t) \leftarrow \mathbf{x}_t$
  for $i \in \{1, \ldots, l\}$ **do**

$\mathbf{a}^i(\mathbf{x}_t) = \mathbf{b}^i + \mathbf{W}^i \widehat{\mathbf{h}}^{i-1}(\mathbf{x}_t)$

$\widehat{\mathbf{h}}^i(\mathbf{x}_t) = \text{sigmoid}\left(\mathbf{a}^i(\mathbf{x}_t)\right)$

end for

$\mathbf{a}^{l+1}(\mathbf{x}_t) = \mathbf{b}^{l+1} + \mathbf{W}^{l+1} \widehat{\mathbf{h}}^l(\mathbf{x}_t)$

$\mathbf{o}(\mathbf{x}_t) = \widehat{\mathbf{h}}^{l+1}(\mathbf{x}_t) = \text{softmax}\left(\mathbf{a}^{l+1}(\mathbf{x}_t)\right)$

% Backward gradient propagation and parameter update

$\frac{\partial \log o_{y_t}(\mathbf{x}_t)}{\partial a_j^{l+1}(\mathbf{x}_t)} \leftarrow \mathbf{1}_{y_t=j} - o_j(\mathbf{x}_t)$ **for** $j \in \{1, \ldots, K\}$

$\mathbf{b}^{l+1} \leftarrow \mathbf{b}^{l+1} + \epsilon_{\text{fine-tune}} \frac{\partial \log o_{y_t}(\mathbf{x}_t)}{\partial \mathbf{a}^{l+1}(\mathbf{x}_t)}$

$\mathbf{W}^{l+1} \leftarrow \mathbf{W}^{l+1} + \epsilon_{\text{fine-tune}} \frac{\partial \log o_{y_t}(\mathbf{x}_t)}{\partial \mathbf{a}^{l+1}(\mathbf{x}_t)} \widehat{\mathbf{h}}^l(\mathbf{x}_t)^{\mathbf{T}}$

for $i \in \{1, \ldots, l\}$, in decreasing order **do**

$\frac{\partial \log o_{y_t}(\mathbf{x}_t)}{\partial \widehat{\mathbf{h}}^i(\mathbf{x}_t)} \leftarrow \left(\mathbf{W}^{i+1}\right)^{\mathbf{T}} \frac{\partial \log o_{y_t}(\mathbf{x}_t)}{\partial \mathbf{a}^{i+1}(\mathbf{x}_t)}$

$\frac{\partial \log o_{y_t}(\mathbf{x}_t)}{\partial a_j^i(\mathbf{x}_t)} \leftarrow \frac{\partial \log o_{y_t}(\mathbf{x}_t)}{\partial \widehat{h}_j^i(\mathbf{x}_t)} \widehat{h}_j^i(\mathbf{x}_t) \left(1 - \widehat{h}_j^i(\mathbf{x}_t)\right)$ **for** $j \in \{1, \ldots, |\widehat{\mathbf{h}}^i|\}$

$\mathbf{b}^i \leftarrow \mathbf{b}^i + \epsilon_{\text{fine-tune}} \frac{\partial \log o_{y_t}(\mathbf{x}_t)}{\partial \mathbf{a}^i}$

$\mathbf{W}^i \leftarrow \mathbf{W}^i + \epsilon_{\text{fine-tune}} \frac{\partial \log o_{y_t}(\mathbf{x}_t)}{\partial \mathbf{a}^i} \widehat{\mathbf{h}}^{i-1}(\mathbf{x}_t)^{\mathbf{T}}$

end for

end while

In the first step of the gradient computation, one has to be careful to compute the gradient of the cost with respect to $\mathbf{a}^{l+1}(\mathbf{x}_t)$ at once, in order not to lose numerical precision during the computation. In particular, computing $\frac{\partial \log o_{y_t}(\mathbf{x}_t)}{\partial \mathbf{o}(\mathbf{x}_t)}$ first, then $\frac{\partial \mathbf{o}(\mathbf{x}_t)}{\partial \mathbf{a}^{l+1}(\mathbf{x}_t)}$ and applying chain-rule, leads to numerical instability and sometimes parameter value explosion (NaN).

Appendix B: Restricted Boltzmann Machines and Deep Belief Networks

Restricted Boltzmann Machine

A restricted Boltzmann machine is an energy-based generative model defined over a visible layer $\mathbf{v}$ (sometimes called input) and a hidden layer $\mathbf{h}$ (sometimes called hidden factors or representation). Given an energy function $\text{energy}(\mathbf{v}, \mathbf{h})$ on the whole set of visible and hidden units, the joint probability is given by

$$p(\mathbf{v}, \mathbf{h}) = \frac{e^{-\text{energy}(\mathbf{v},\mathbf{h})}}{Z} \tag{6.6}$$

where Z ensures that $p(\mathbf{v}, \mathbf{h})$ is a valid distribution and sums to one. See Figure 6.3 for an illustration of an RBM.

Typically we take $h_i \in \{0, 1\}$, but other choices are possible. For now, we consider only binary units, i.e., $v_i \in \{0, 1\}$ (the continuous case will be discussed in Section 6.7), where the energy function has the form

$$\text{energy}(\mathbf{v}, \mathbf{h}) = -\mathbf{h}^\mathbf{T} W \mathbf{v} - c^\mathbf{T} \mathbf{v} - b^\mathbf{T} \mathbf{h} = -\sum_k c_k v_k - \sum_j b_j h_j - \sum_{jk} W_{jk} v_k h_j \ . \tag{6.7}$$

When considering the marginal distribution over $\mathbf{v}$, we obtain a mixture distribution

$$p(\mathbf{v}) = \sum_{\mathbf{h}} p(\mathbf{v}, \mathbf{h}) = \sum_{\mathbf{h}} p(\mathbf{v}|\mathbf{h}) p(\mathbf{h})$$

with a number of parameters linear in the number of hidden units H, while having a number of components exponential in H. This is because $\mathbf{h}$ can take as many as 2^H possible values. The 2^H distributions $p(\mathbf{v}|\mathbf{h})$ will in general be different, but they are tied. Though computing exactly the marginal $p(\mathbf{v})$ for large values of H is impractical, a good estimator of the log-likelihood gradient have been found with the contrastive divergence algorithm described in the next section. An important property of RBMs is that the posterior distribution over one layer given the other is tractable and fast to compute, as opposed to mixture models with very many components in general. Indeed

one can show that

$$p(\mathbf{v}|\mathbf{h}) = \prod_k p(v_k|\mathbf{h}) \quad \text{where} \quad p(v_k = 1|\mathbf{h}) = \text{sigmoid}(c_k + \sum_j W_{jk} h_j) \quad (6.8)$$

$$p(\mathbf{h}|\mathbf{v}) = \prod_j p(h_j|\mathbf{v}) \quad \text{where} \quad p(h_j = 1|\mathbf{v}) = \text{sigmoid}(b_j + \sum_k W_{jk} v_k)\ . \quad (6.9)$$

Because of the particular parametrization of RBMs, inference of the "hidden factors" $\mathbf{h}$ given the observed input vector $\mathbf{v}$ is very easy because those factors are conditionally independent given $\mathbf{v}$. On the other hand, unlike in many factor models (such as ICA [9, 38, 92] and sigmoidal belief networks [42, 79, 137]), these factors are generally not marginally independent (when we integrate $\mathbf{v}$ out). Notice the similarity between Equations 6.9 and 6.1, which makes it possible to relate the weights and biases of an RBM with those of a deep neural network.

Learning in a Restricted Boltzmann Machine

To train an RBM, we would like to compute the gradient of the negative log-likelihood of the data with respect to the RBM's parameters. However, given an input example $\mathbf{v}_0$, the gradient with respect to a parameter θ in an energy-based model

$$\frac{\partial}{\partial \theta} \left(-\log p(\mathbf{v}_0) \right) = E_{p(\mathbf{h}|\mathbf{v}_0)} \left[\frac{\partial \text{energy}(\mathbf{v}_0, \mathbf{h})}{\partial \theta} \right] - E_{p(\mathbf{v},\mathbf{h})} \left[\frac{\partial \text{energy}(\mathbf{v}, \mathbf{h})}{\partial \theta} \right] \quad (6.10)$$

necessitates a sum over all possible assignments for $\mathbf{h}$ (first expectation of Equation 6.10) and another sum over all assignments for $\mathbf{v}$ and $\mathbf{h}$ (second expectation). The first expectation is not problematic in an RBM because the posterior $p(\mathbf{h}|\mathbf{v}_0)$ and $\frac{\partial \text{energy}(\mathbf{v}_0,\mathbf{h})}{\partial \theta}$ factorize. However, the second expectation requires a prohibitive exponential sum over the possible configurations for $\mathbf{v}$ or $\mathbf{h}$.

Fortunately, there exists an approximation for this gradient given by the contrastive divergence (CD) algorithm [77], which has been shown to work well empirically [28]. There are two key elements in this approximation. First, consider that in order to estimate the second term of Equation 6.10, we could replace the expectation by a unique evalu-

ation of the gradient $\frac{\partial \text{energy}(\mathbf{v},\mathbf{h})}{\partial \theta}$ at a particular pair of values $(\mathbf{v}, \mathbf{h})$. This pair should ideally be sampled from the distribution $p(\mathbf{v}, \mathbf{h})$, which would make the estimation of the gradient unbiased. However, sampling exactly from an RBM distribution is not as easy as in a directed graphical model. Instead, we have to rely on sampling methods such as Markov Chain Monte Carlo methods. For an RBM, we can use Gibbs sampling based on the conditional distributions of Equations 6.8 and 6.9, but this method can be costly if the Markov chain mixes slowly. So the second key idea is to run only a few iterations of Gibbs sampling and use the data sample $\mathbf{v}_0$ as the initial state for the chain at the visible layer. It turns out that applying only one iteration of the Markov chain works well in practice. This corresponds to the following sampling procedure:

$$\mathbf{v}_0 \overset{p(\mathbf{h}_0|\mathbf{v}_0)}{\longrightarrow} \mathbf{h}_0 \overset{p(\mathbf{v}_1|\mathbf{h}_0)}{\longrightarrow} \mathbf{v}_1 \overset{p(\mathbf{h}_1|\mathbf{x}_1)}{\longrightarrow} \mathbf{h}_1$$

where $\overset{p(\mathbf{h}_i|\mathbf{v}_i)}{\longrightarrow}$ and $\overset{p(\mathbf{v}_{i+1}|\mathbf{h}_i)}{\longrightarrow}$ represent the operations of sampling from $p(\mathbf{h}_i|\mathbf{v}_i)$ and $p(\mathbf{v}_{i+1}|\mathbf{h}_i)$ respectively. Estimation of the gradient using the above sampling procedure is noted CD-1, with CD-k referring to the contrastive divergence algorithm, performing k iterations of the Markov chain up to $\mathbf{v}_k$. Training with CD-k has been shown to empirically approximate well training with the exact log-likelihood gradient [28]. Furthermore, it can be shown that the CD-k update is an unbiased estimator of the truncation of a series expansion of the log-likelihood gradient [11], where the truncated part of the series converges to 0 as k increases.

Now let us consider the estimation of the gradient on a weight W_{jk}. We have

$$\frac{\partial \text{energy}(\mathbf{v}, \mathbf{h})}{\partial W_{jk}} = -h_j v_k$$

which means that the CD-1 estimate of the gradient becomes

$$-E_{p(\mathbf{h}|\mathbf{v}_0)}\left[h_j v_{0k}\right] + E_{p(\mathbf{h}|\mathbf{v}_1)}\left[h_{1j} v_{1k}\right] = -p(h_j|\mathbf{v}_0) v_{0k} + p(h_j|\mathbf{v}_1) v_{1k} \; . \tag{6.11}$$

This is similar to what was presented in Hinton et al. [81] except that we clarify here that we take the expected value of $\mathbf{h}$ given $\mathbf{v}$ instead of averaging over samples. These

estimators have the same expected value because

$$E_{p(\mathbf{v},\mathbf{h})}\left[h_j v_k\right] = E_{p(\mathbf{v})}\left[E_{p(\mathbf{h}|\mathbf{v})}\left[h_j v_k\right]\right] = E_{p(\mathbf{v})}\left[p(h_j|\mathbf{v})v_k\right] \ .$$

Using $p(\mathbf{h}|\mathbf{v}_k)$ instead of $\mathbf{h}_k$ is also what is found in the Matlab code distributed with Hinton et Salakhutdinov [80]. Note that it is still necessary to sample $\mathbf{h}_0 \sim p(\mathbf{h}|\mathbf{v}_0)$ in order to sample $\mathbf{v}_1$, but it is not necessary to sample $\mathbf{h}_1$. The above gradient estimator can then be used to perform stochastic gradient descent by iterating through all vectors $\mathbf{v}_0$ of the training set and performing a parameter update using that gradient estimator in an on-line fashion. Gradient estimators for the biases b_k and c_j can as easily be derived from Equation 6.10.

Notice also that, even if $\mathbf{v}_0$ is not binary, the formula for the CD-1 estimate of the gradient does not change and is still computed essentially in the same way: only the sampling procedure for $p(\mathbf{v}|\mathbf{h})$ changes (see Section 6.7 for more details about dealing with continuous-valued inputs). The CD-1 training update for a given training input is detailed by the pseudocode in the next section.

In our implementation of the greedy layer-wise initialization phase, we use the deterministic sigmoidal outputs of the previous level as training vector for the next level RBM. By interpreting these real-valued components as probabilities, learning such a distribution for binary inputs can be seen as a crude "mean-field" way of dealing with probabilistic binary inputs (instead of summing or sampling across input configurations).

Pseudocode for Contrastive Divergence (CD-1) Training Update

Input: training input $\mathbf{x}$, RBM weights $\mathbf{W}^i$ and biases $\mathbf{b}^{i-1}, \mathbf{b}^i$ and learning rate ϵ

Notation: $a \sim p(\cdot)$ means set a equal to a random sample from $p(\cdot)$

% Set RBM parameters

$\mathbf{W} \leftarrow \mathbf{W}^i, \mathbf{b} \leftarrow \mathbf{b}^i, \mathbf{c} \leftarrow \mathbf{b}^{i-1}$

% Positive phase

$\mathbf{v}^0 \leftarrow \mathbf{x}$

$\widehat{\mathbf{h}}^0 \leftarrow \text{sigmoid}(\mathbf{b} + \mathbf{W}\mathbf{v}^0)$

% Negative phase

$\mathbf{h}^0 \sim p(\mathbf{h}|\mathbf{v}^0)$ according to Equation 6.9

$\mathbf{v}^1 \sim p(\mathbf{v}|\mathbf{h}^0)$ according to Equation 6.8

$\widehat{\mathbf{h}}^1 \leftarrow \text{sigmoid}(\mathbf{b} + \mathbf{W}\mathbf{v}^1)$

% Update

$\mathbf{W}^i \leftarrow \mathbf{W}^i + \epsilon \left(\widehat{\mathbf{h}}^0 \left(\mathbf{v}^0\right)^{\mathbf{T}} - \widehat{\mathbf{h}}^1 \left(\mathbf{v}^1\right)^{\mathbf{T}} \right)$

$\mathbf{b}^i \leftarrow \mathbf{b}^i + \epsilon \left(\widehat{\mathbf{h}}^0 - \widehat{\mathbf{h}}^1 \right)$

$\mathbf{b}^{i-1} \leftarrow \mathbf{b}^{i-1} + \epsilon \left(\mathbf{v}^0 - \mathbf{v}^1 \right)$

Deep Belief Network

We wish to make a quick remark on the distinction between the SRBM network and the more widely known deep belief network (DBN) [81], which is not a feed-forward neural network but a multi-layer generative model. The SRBM network was initially derived [81] from the DBN, for which stacking RBMs also provides a good initialization.

A DBN is a generative model with several layers of stochastic units. It actually corresponds to a sigmoid belief network [114] of $l - 1$ hidden layers, where the prior over its top hidden layer $\mathbf{h}^{l-1}$ (second factor of Equation 6.12) is an RBM, which itself

has a hidden layer $\mathbf{h}^l$. More precisely, it defines a distribution over an input layer $\mathbf{x}$ and l layers of binary stochastic units $\mathbf{h}^i$ as follows:

$$p(\mathbf{x}, \mathbf{h}^1, \ldots, \mathbf{h}^l) = \left(\prod_{i=1}^{l-1} p(\mathbf{h}^{i-1} | \mathbf{h}^i) \right) p(\mathbf{h}^{l-1}, \mathbf{h}^l) \tag{6.12}$$

where hidden units are conditionally independent given the units in the above layer

$$p(\mathbf{h}^{i-1} | \mathbf{h}^i) = \prod_k p(h_k^{i-1} | \mathbf{h}^i) \ .$$

To process binary values, Bernoulli layers can be used, which correspond to equations

$$p(h_k^{i-1} = 1 | \mathbf{h}^i) = \text{sigmoid} \left(b_k^{i-1} + \sum_j W_{jk}^i h_j^i \right)$$

where $\mathbf{h}^0 = \mathbf{x}$ is the input. We also have

$$p(\mathbf{h}^{l-1}, \mathbf{h}^l) \propto e^{\sum_j c_j^{l-1} h_j^{l-1} + \sum_k b_k^l h_k^l + \sum_{jk} W_{jk}^l h_j^{l-1} h_k^l} \tag{6.13}$$

for the top RBM. Note that Equation 6.13 can be obtained from Equations 6.6 and 6.7, by naming $\mathbf{v}$ as $\mathbf{h}^{l-1}$, and $\mathbf{h}$ as $\mathbf{h}^l$.

We emphasize the distinction between $\mathbf{h}^i$ and $\widehat{\mathbf{h}}^i(\mathbf{x})$, where the former is a random variable and the latter is the representation of an input $\mathbf{x}$ at the i-th hidden layer of the network obtained from the repeated application of Equation 6.1.

To train such a generative model, Hinton et al. [81] proposed the pre-training phase of the SRBM network. When a DBN has the same number of units in its even-numbered and odd-numbered layers (separately, i.e. not all layers need to have the same size), it was actually shown that this initialization improves a lower bound on the likelihood of the data as the DBN is made deeper. After this pre-training phase is over, Hinton et al. [81] propose a variant of the Wake-Sleep algorithm for sigmoid belief networks [79] to fine-tune the generative model.

By noticing the similarity between the process of approximating the posterior $p(\mathbf{h}^i | \mathbf{x})$

in a deep belief network and computing the hidden layer representation of an input $\mathbf{x}$ in a deep network, Hinton [78] then proposed the use of the greedy layer-wise pre-training procedure for deep belief networks to initialize a deep feed-forward neural network, which corresponds to the SRBM network described in this paper.

Appendix C: Pseudocode of Autoassociator Training Update

Input: training input $\mathbf{x}$, autoassociator weights $\mathbf{W}^i$ and biases $\mathbf{b}^{i-1}, \mathbf{b}^i$ and learning rate ϵ

% Set autoassociator parameters

$\mathbf{W} \leftarrow \mathbf{W}^i$, $\mathbf{b} \leftarrow \mathbf{b}^i$, $\mathbf{c} \leftarrow \mathbf{b}^{i-1}$

% Forward propagation

$\mathbf{a}(\mathbf{x}) \leftarrow \mathbf{b} + \mathbf{W}\mathbf{x}$

$\mathbf{h}(\mathbf{x}) \leftarrow \text{sigmoid}(\mathbf{a}(\mathbf{x}))$

$\widehat{\mathbf{a}}(\mathbf{x}) \leftarrow \mathbf{c} + \mathbf{W}^{\mathbf{T}}\mathbf{h}(\mathbf{x})$

$\widehat{\mathbf{x}} \leftarrow \text{sigmoid}(\widehat{\mathbf{a}}(\mathbf{x}))$

% Backward gradient propagation

$\frac{\partial C(\widehat{\mathbf{x}},\mathbf{x})}{\partial \widehat{\mathbf{a}}(\mathbf{x})} \leftarrow \widehat{\mathbf{x}} - \mathbf{x}$

$\frac{\partial C(\widehat{\mathbf{x}},\mathbf{x})}{\partial \widehat{h}(\mathbf{x})} \leftarrow \mathbf{W}\frac{\partial C(\widehat{\mathbf{x}},\mathbf{x})}{\partial \widehat{\mathbf{a}}(\mathbf{x})}$

$\frac{\partial C(\widehat{\mathbf{x}},\mathbf{x})}{\partial a_j(\mathbf{x})} \leftarrow \frac{\partial C(\widehat{\mathbf{x}},\mathbf{x})}{\partial \widehat{h}_j(\mathbf{x})}\widehat{h}_j(\mathbf{x})\left(1 - \widehat{h}_j(\mathbf{x})\right)$ **for** $j \in \{1, \ldots, |\widehat{\mathbf{h}}(\mathbf{x})|\}$

% Update

$\mathbf{W}^i \leftarrow \mathbf{W}^i - \epsilon\left(\frac{\partial C(\widehat{\mathbf{x}},\mathbf{x})}{\partial \mathbf{a}(\mathbf{x})}\mathbf{x}^{\mathbf{T}} + \widehat{\mathbf{h}}(\mathbf{x})\frac{\partial C(\widehat{\mathbf{x}},\mathbf{x})}{\partial \widehat{\mathbf{a}}(\mathbf{x})}^{\mathbf{T}}\right)$

$\mathbf{b}^i \leftarrow \mathbf{b}^i - \epsilon\frac{\partial C(\widehat{\mathbf{x}},\mathbf{x})}{\partial \mathbf{a}(\mathbf{x})}$

$\mathbf{b}^{i-1} \leftarrow \mathbf{b}^{i-1} - \epsilon\frac{\partial C(\widehat{\mathbf{x}},\mathbf{x})}{\partial \widehat{\mathbf{a}}(\mathbf{x})}$

CHAPITRE 7

PRÉSENTATION DU TROISIÈME ARTICLE

7.1 Détails de l'article

An Empirical Evaluation of Deep Architectures on Problems with Many Factors of Variation

Hugo Larochelle, Dumitru Erhan, Aaron Courville, James Bergstra et Yoshua Bengio

Publié dans *Proceedings of the 24th Annual International Conference on Machine Learning, Omnipress* en 2007.

7.2 Contexte

Au chapitre précédent, on a vu différentes approches basées sur un apprentissage non-supervisé et capables d'améliorer l'entraînement d'un réseau de neurones profond. On y a étudié empiriquement l'impact du choix de différentes topologies pour un réseau profond et de l'utilisation de différentes variantes de l'approche vorace couche par couche. Cependant, les expériences en question ont porté sur relativement peu de jeux de données, soit principalement la base de données MNIST ainsi qu'une variante contenant des rotations d'image.

7.3 Contributions

Dans l'article suivant, plutôt que de varier le fonctionnement des algorithmes d'apprentissage pour réseaux profonds, on varie le problème d'apprentissage à résoudre. Plus précisément, on démontre que pour des problèmes dont l'entrée est générée selon plusieurs **facteurs de variation** indépendants, l'avantage en terme de performance de généralisation apporté par les réseaux profonds (entraînés selon les algorithmes décrits au chapitre 6) devient particulièrement important. On désigne par facteurs de variation l'ensemble des variables latentes déterminant le processus de génération des entrées.

Plus spécifiquement, supposons que le modèle générateur des données puisse être écrit comme suit :

$$p(\mathbf{x}) = \sum_{h_1 \in \mathcal{H}_1} \dots \sum_{h_l \in \mathcal{H}_l} p(\mathbf{x}|h_1, \dots, h_l) p(h_1) \dots p(h_l) \ ,$$

où les l variables latentes $h_1 \dots h_l$ correspondent chacune à un facteur de variation. Par exemple, dans le cas du jeu de données MNIST auquel on a appliqué des rotations, $\mathbf{h}$ inclura une variable h_i correspondant à l'angle de la rotation.

Ainsi, on démontre empiriquement dans cet article que les réseaux de neurones, en particulier ceux entraînés à l'aide de machines de Boltzmann restreintes, ont en général une meilleure performance de généralisation que d'autres modèles «plats» (i.e., non profonds). On y souligne aussi une difficulté appréhendée dans l'utilisation des réseaux profonds, dans le cas où la distribution des entrées contient plusieurs facteurs de variation n'étant pas corrélés avec la cible y mais étant suffisamment complexes pour nécessiter de la capacité de la part du réseau.

7.4 Commentaires

Les résultats publiés à l'origine pour les jeux de données incluant des rotations (*mnist-rot* et *mnist-rot-back-image*) ont dû être regénérés, suite à la découverte d'une erreur dans le code générant ces rotations. Ainsi, les nouveaux résultats ont remplacé les anciens dans la version de cet article transcrite dans cette thèse. Le texte de l'analyse des résultats a aussi été mis à jour afin de mieux refléter ces nouveaux résultats. À noter que ces nouveaux résultats ont été générés en 2007, et ont aussitôt été ajoutés au site Web mis sur pied dans le cadre de ces travaux, à l'adresse `http://www.iro.umontreal.ca/~lisa/icml2007`. De plus, ces nouveaux résultats n'ont rien changé aux conclusions qualitatives générales de l'article.

De plus, les réseaux profonds entraînés à l'aide de machines de Boltzmann restreintes pour cet article ne correspondent pas exactement aux réseaux SRBM du chapitre précédent. Effectivement, on a ici tenté de respecter le plus possible la définition d'un *Deep*

Belief Network (DBN) en utilisant une procédure d'inférence pour $p(y|\mathbf{x})$ qui soit la moins approximative possible. Ainsi, à partir de la valeur de l'avant-dernière couche cachée, une formule différente a été appliquée afin de calculer la sortie du réseau. Effectivement, dans cet article, la dernière couche cachée correspond à la couche cachée d'une machine de Boltzmann restreinte pour la classification[1] décrite au chapitre 4. La prédiction de y à partir de l'avant-dernière couche cachée prend alors la même forme que l'équation 4.4 (où $\mathbf{x}$ est remplacé par la valeur de l'avant-dernière couche cachée), plutôt que de suivre le calcul standard d'un réseau de neurones (voir l'équation 6.1). Cependant, mon expérience avec ces réseaux m'indique que cette petite différence n'a vraisemblablement pas eu d'impact important sur les résultats obtenus et que les conclusions dérivées de cet article auraient été les mêmes si un réseau SRBM avait été utilisé.

[1]Plus précisément, les neurones liés à la prédicion de y font alors partie de la couche visible de la machine de Boltzmann.

CHAPTER 8

AN EMPIRICAL EVALUATION OF DEEP ARCHITECTURES ON PROBLEMS WITH MANY FACTORS OF VARIATION

8.1 Abstract

Recently, several learning algorithms relying on models with deep architectures have been proposed. Though they have demonstrated impressive performance, to date, they have only been evaluated on relatively simple problems such as digit recognition in a controlled environment, for which many machine learning algorithms already report reasonable results. Here, we present a series of experiments which indicate that these models show promise in solving harder learning problems that exhibit many factors of variation. These models are compared with well-established algorithms such as Support Vector Machines and single hidden-layer feed-forward neural networks.

8.2 Introduction

Several recent empirical and theoretical results have brought deep architectures to the attention of the machine learning community: they have been used, with good results, for dimensionality reduction [80, 132], and classification of digits from the MNIST data set [14, 81]. A core contribution of this body of work is the training strategy for a family of computational models that is similar or identical to traditional multilayer perceptrons with sigmoidal hidden units. Traditional gradient-based optimization strategies are not effective when the gradient must be propagated across multiple non-linearities. Hinton [78] gives empirical evidence that a sequential, greedy, optimization of the weights of each layer using the generative training criterion of a Restricted Boltzmann Machine tends to initialize the weights such that global gradient-based optimization can work. Bengio et al. [14] showed that this procedure also worked using the autoassociator unsupervised training criterion and empirically studied the sequential, greedy layer-wise strategy. However, to date, the only empirical comparison on classification problems

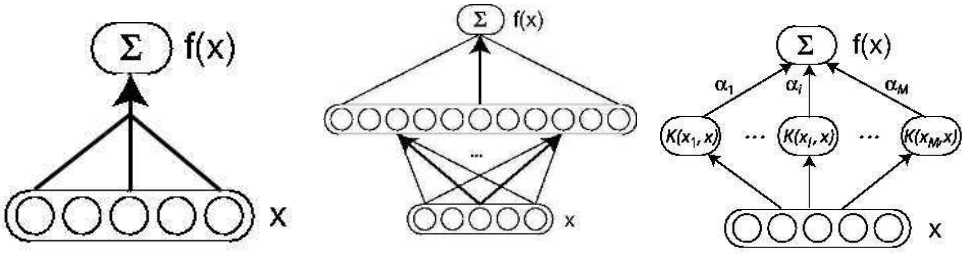
between these deep training algorithms and the state-of-the-art has been on MNIST, on which many algorithms are relatively successful and in which the classes are known to be well separated in the input space. It remains to be seen whether the advantages seen in the MNIST dataset are observed in other more challenging tasks.

Ultimately, we would like algorithms with the capacity to capture the complex structure found in language and vision tasks. These problems are characterized by many factors of variation that interact in nonlinear ways and make learning difficult. For example, the NORB dataset introduced by LeCun et al. [102] features toys in real scenes, in various lighting, orientation, clutter, and degrees of occlusion. In that work, they demonstrate that existing general algorithms (Gaussian SVMs) perform poorly. In this work, we propose a suite of datasets that spans some of the territory between MNIST and NORB–starting with MNIST, and introducing multiple factors of variation such as rotation and background manipulations. These toy datasets allow us to test the limits of current state-of-the-art algorithms, and explore the behavior of the newer deep-architecture training procedures, *with architectures not tailored to machine vision*. In a very limited but significant way, we believe that these problems are closer to "real world" tasks, and can serve as milestones on the road to AI.

8.2.1 Shallow and Deep Architectures

We define a *shallow model* as a model with very few layers of composition, e.g. linear models, one-hidden-layer neural networks and kernel SVMs (see figure 8.1). On the other hand, *deep architecture models* are such that their output is the result of the composition of some number of computational units, commensurate with the amount of data one can possibly collect, i.e. not exponential in the characteristics of the problem such as the number of factors of variation or the number of inputs. These units are generally organized in layers so that the many levels of computation can be composed.

A function may appear complex from the point of view of a local non-parametric learning algorithm such as a Gaussian kernel machine, because it has many variations, such as the *sine* function. On the other hand, the Kolmogorov complexity of that function could be small, and it could be representable efficiently with a deep architecture.



(a) Linear model architecture (b) Single layer neural network architecture (c) Kernel SVM architecture

Figure 8.1: Examples of models with shallow architectures.

See Bengio et LeCun [15] for more discussion on this subject, and pointers to the circuit complexity theory literature showing that shallow circuits can require exponentially more components than deeper circuits.

However, optimizing deep architectures is computationally challenging. It was believed until recently impractical to train deep neural networks (except Convolutional Neural Networks [98]), as iterative optimization procedures tended to get stuck near poor local minima. Fortunately, effective optimization procedures using unsupervised learning have recently been proposed and have demonstrated impressive performance for deep architectures.

8.2.2 Scaling to Harder Learning Problems

Though there are benchmarks to evaluate generic learning algorithms (e.g. the UCI Machine Learning Repository) many of these proposed learning problems do not possess the kind of complexity we address here.

We are interested in problems for which the underlying data distribution can be thought as the product of factor distributions, which means that a sample corresponds to a combination of particular values for these factors. For example, in a digit recognition task, the factors might be the scaling, rotation angle, deviation from the center of the image and the background of the image. Note how some of these factors (such as the background) may be very high-dimensional. In natural language processing, factors which influence the distribution over words in a document include topic, style and

various characteristics of the author. In speech recognition, potential factors can be the gender of the speaker, the background noise and the amount of echo in the environment. In these important settings, it is not feasible to collect enough data to cover the input space effectively; especially when these factors vary independently.

Research in incorporating factors of variation into learning procedures has been abundant. A lot of the published results refer to learning invariance in the domain of digit recognition and most of these techniques are engineered for a specific set of invariances. For instance, Decoste et Scholkopf [44] present a thorough review that discusses the problem of incorporating prior knowledge into the training procedure of kernel-based methods. More specifically, they discuss prior knowledge about invariances such as translations, rotations etc. Three main methods are described:

1. hand-engineered kernel functions,

2. artificial generation of transformed examples (the so-called *Virtual SV* method),

3. and a combination of the two: engineered kernels that generate artificial examples (e.g. *kernel jittering*).

The main drawback of these methods, from our point of view, is that domain experts are required to explicitly identify the types of invariances that need to be modeled. Furthermore these invariances are highly problem-specific. While there are cases for which manually crafted invariant features are readily available, it is difficult in general to construct invariant features.

We are interested in learning procedures and architectures that would *automatically* discover and represent such invariances (ideally, in an efficient manner). We believe that one good way of achieving such goals is to have procedures that learn high-level features ("abstractions") that build on lower-level features. One of the main goals of this paper is thus to examine empirically the link between high-level feature extraction and different types of invariances. We start by describing two architectures that are designed for extracting high-level features.

8.3 Learning Algorithms with Deep Architectures

Hinton et al. [81] introduced a greedy layer-wise *unsupervised* learning algorithm for Deep Belief Networks (DBN). This training strategy for such networks was subsequently analyzed by Bengio et al. [14] who concluded that it is an important ingredient in effective optimization and training of deep networks. While lower layers of a DBN extract "low-level features" from the input observation $\mathbf{x}$, the upper layers are supposed to represent more "abstract" concepts that explain $\mathbf{x}$.

8.3.1 Deep Belief Networks and Restricted Boltzmann Machines

For classification, a DBN model with ℓ layers models the joint distribution between target y, observed variables x_j and i hidden layers $\mathbf{h}^k$ made of all binary units h_i^k, as follows:

$$P(\mathbf{x}, \mathbf{h}^1, \ldots, \mathbf{h}^\ell) = \left(\prod_{k=1}^{\ell-2} P(\mathbf{h}^k | \mathbf{h}^{k+1}) \right) P(y, \mathbf{h}^{\ell-1}, \mathbf{h}^\ell)$$

where $\mathbf{x} = \mathbf{h}^0$, $P(\mathbf{h}^k | \mathbf{h}^{k+1})$ has the form given by equation 8.1 and $P(y, \mathbf{h}^{\ell-1}, \mathbf{h}^\ell)$ is a Restricted Boltzmann Machine (RBM), with the bottom layer being the concatenation of y and $\mathbf{h}^{\ell-1}$ and the top layer is $\mathbf{h}^\ell$.

An RBM with n hidden units is a parametric model of the joint distribution between hidden variables h_i and observed variables x_j of the form:

$$P(\mathbf{x}, \mathbf{h}) \propto e^{\mathbf{h}' W \mathbf{x} + b' \mathbf{x} + c' \mathbf{h}}$$

with parameters $\theta = (W, b, c)$. If we restrict h_i and x_j to be binary units, it is straightforward to show that

$$P(\mathbf{x}|\mathbf{h}) = \prod_i P(x_i|\mathbf{h}) = \prod_i \text{sigmoid}(b_i + \sum_j W_{ji} h_j) \tag{8.1}$$

where $\mathrm{sigmoid}$ is the logistic sigmoid function, and $P(\mathbf{h}|\mathbf{x})$ also has a similar form:

$$P(\mathbf{h}|\mathbf{x}) = \prod_j P(h_j|\mathbf{x}) = \prod_j \mathrm{sigmoid}(c_j + \sum_i W_{ji} x_i) \tag{8.2}$$

The RBM form can be generalized to other conditional distributions besides the binomial, including continuous variables. See Welling et al. [162] for a generalization of RBM models to conditional distributions from the exponential family.

RBM models can be trained by gradient descent. Although $P(\mathbf{x})$ is not tractable in an RBM, the Contrastive Divergence gradient [77] is a good stochastic approximation of $\frac{\partial \log P(\mathbf{x})}{\partial \theta}$. The contrastive divergence stochastic gradient can be used to initialize each layer of a DBN as an RBM. The number of layers can be increased greedily, with the newly added top layer trained as an RBM to model the output of the previous layers. When initializing the weights to $\mathbf{h}^\ell$, an RBM is trained to model the concatenation of y and $\mathbf{h}^{\ell-1}$. This iterative pre-training procedure is illustrated in figure 8.2.

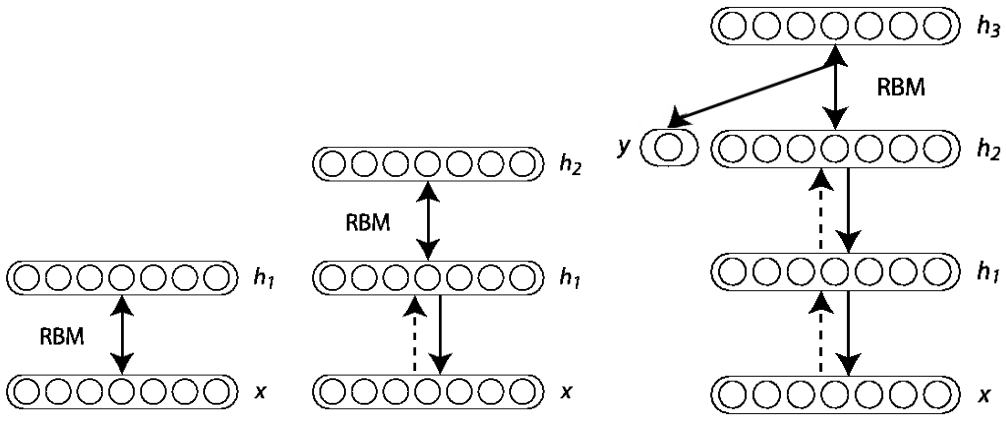
Using a mean-field approximation of the conditional distribution of layer $\mathbf{h}^{\ell-1}$, we can compute a representation $\widehat{\mathbf{h}}^{\ell-1}$ for the input by setting $\widehat{\mathbf{h}}^0 = \mathbf{x}$ and iteratively computing $\widehat{\mathbf{h}}^k = P(\mathbf{h}^k|\widehat{\mathbf{h}}^{k-1})$ using equation 8.2. We then compute the probability of all classes given the approximately inferred value $\widehat{\mathbf{h}}^{\ell-1}$ for $\mathbf{h}^{\ell-1}$ using the following expression:

$$P(y|\widehat{\mathbf{h}}^{\ell-1}) = \sum_{\mathbf{h}^\ell} P(y, \mathbf{h}^\ell|\widehat{\mathbf{h}}^{\ell-1})$$

which can be calculated efficiently. The network can then be fine-tuned according to this estimation of the class probabilities by maximizing the log-likelihood of the class assignments in a training set using standard back-propagation.

8.3.2 Stacked Autoassociators

As demonstrated by Bengio et al. [14], the idea of successively extracting non-linear features that "explain" variations of the features at the previous level can be applied not only to RBMs but also to *autoassociators*. An autoassociator is simply a model (usually a one-hidden-layer neural network) trained to reproduce its input by forcing the



(a) RBM for $\mathbf{x}$ (b) RBM for $\mathbf{h}^1$ (c) RBM for $\mathbf{h}^2$ and y

Figure 8.2: Iterative pre-training construction of a Deep Belief Network.

computations to flow through a "bottleneck" representation. Here we used the following architecture for autoassociators. Let $\mathbf{x}$ be the input of the autoassociator, with $x_i \in [0, 1]$, interpreted as the probability for the bit to be 1. For a layer with weight matrix $\mathbf{W}$, hidden biases column vector $\mathbf{c}$ and input biases column vector $\mathbf{b}$, the reconstruction probability for bit i is $p_i(\mathbf{x})$, with the vector of probabilities:

$$p(\mathbf{x}) = \text{sigmoid}(\mathbf{b} + \mathbf{W}^{\mathbf{T}}\text{sigmoid}(\mathbf{c} + \mathbf{W}\mathbf{x})).$$

The training criterion for the layer is the average of negative log-likelihoods for predicting $\mathbf{x}$ from $p(\mathbf{x})$. For example, if $\mathbf{x}$ is interpreted either as a sequence of bits or a sequence of bit probabilities, we minimize the reconstruction cross-entropy:

$$R = -\sum_i x_i \log p_i(\mathbf{x}) + (1 - x_i) \log(1 - p_i(\mathbf{x})).$$

See Bengio et al. [14] for more details. Once an autoassociator is trained, its internal "bottleneck" representation (here, $\text{sigmoid}(\mathbf{c}+\mathbf{W}^{\mathbf{T}}\mathbf{x})$) can be used as the input for training a second autoassociator etc. Figure 8.3 illustrates this iterative training procedure. The stacked autoassociators can then be fine-tuned with respect to a supervised training criterion (adding a predictive output layer on top), using back-propagation to compute

gradient on parameters of all layers.

8.4 Benchmark Tasks

In order to study the capacity of these algorithms to scale to learning problems with many factors of variation, we have generated datasets where we can identify some of these factors of variation explicitly. We focused on vision problems, mostly because they are easier to generate and analyze. In all cases, the classification problem has a balanced class distribution.

8.4.1 Variations on Digit Recognition

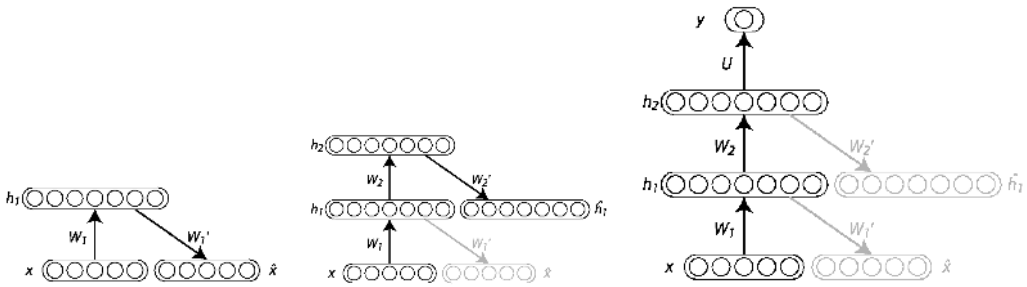
Models with deep architectures have been shown to perform competitively on the MNIST digit recognition dataset [14, 81, 132]. In this series of experiments, we construct new datasets by adding additional factors of variation to the MNIST images. The generative process used to generate the datasets is as follows:

1. Pick sample $(x, y) \in \mathcal{X}$ from the digit recognition dataset;
2. Create a perturbed version $\widehat{x}$ of x according to some factors of variation;
3. Add $(\widehat{x}, y)$ to a new dataset $\widehat{\mathcal{X}}$;
4. Go back to 1 until enough samples are generated.

Introducing multiple factors of variation leads to the following benchmarks:

mnist-rot: the digits were rotated by an angle generated uniformly between 0 and 2π radians. Thus the factors of variation are the rotation angle and those already contained in MNIST, such as hand writing style;

mnist-back-rand: a random background was inserted in the digit image. Each pixel value of the background was generated uniformly between 0 and 255;



(a) Reconst. $\mathbf{x}$ (b) Reconst. $\mathbf{h}^1$ (c) Predict y

Figure 8.3: Iterative training construction of the Stacked Autoassociators model.

mnist-back-image: a random patch from a black and white image was used as the background for the digit image. The patches were extracted randomly from a set of 20 images downloaded from the internet. Patches which had low pixel variance (i.e. contained little texture) were ignored;

mnist-rot-back-image: the perturbations used in *mnist-rot* and *mnist-back-image* were combined.

These 4 databases have 10000, 2000 and 50000 samples in their training, validation and test sets respectively. Figure 8.4 shows samples from these datasets.

8.4.2 Discrimination between Tall and Wide Rectangles

In this task, a learning algorithm needs to recognize whether a rectangle contained in an image has a larger width or length. The rectangle can be situated anywhere in the 28×28 pixel image. We generated two datasets for this problem:

rectangles: the pixels corresponding to the border of the rectangle has a value of 255, 0 otherwise. The height and width of the rectangles were sampled uniformly, but when their difference was smaller than 3 pixels the samples were rejected. The top left corner of the rectangles was also sampled uniformly, constrained so that the whole rectangle would fit in the image;

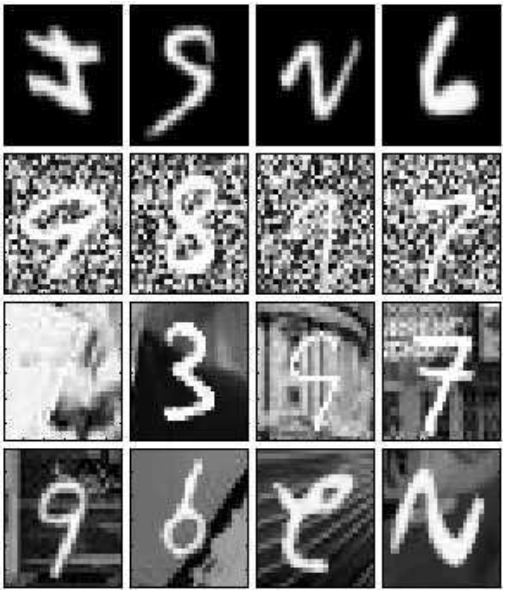


Figure 8.4: From top to bottom, samples from *mnist-rot*, *mnist-back-rand*, *mnist-back-image*, *mnist-rot-back-image*.

*rectangles-image***:** the border and inside of the rectangles corresponds to an image patch and a background patch is also sampled. The image patches are extracted from one of the 20 images used for *mnist-back-image*. Sampling of the rectangles is essentially the same as for *rectangles*, but the area covered by the rectangles was constrained to be between 25% and 75% of the total image, the length and width of the rectangles were forced to be of at least 10 and their difference was forced to be of at least 5 pixels.

We generated training sets of size 1000 and 10000 and validation sets of size 200 and 2000 for *rectangles* and *rectangles-image* respectively. The test sets were of size 50000 in both cases. Samples for these two tasks are displayed in figure 8.5.

8.4.3 Recognition of Convex Sets

The task of discriminating between tall and wide rectangles was designed to exhibit the learning algorithms' ability to process certain image shapes and learn their properties. Following the same principle, we designed another learning problem which consists in

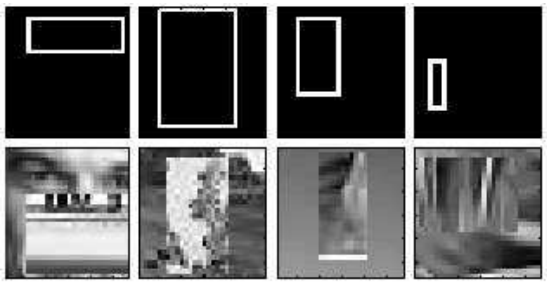


Figure 8.5: From top to bottom, samples from *rectangles* and *rectangles-image*.

indicating if a set of pixels forms a convex set.

Like the MNIST dataset, the convex and non-convex datasets both consist of images of 28×28 pixels. The convex sets consist of a single convex region with pixels of value 255 (white). Candidate convex images were constructed by taking the intersection of a random number of half-planes whose location and orientation were chosen uniformly at random.

Candidate non-convex images were constructed by taking the union of a random number of convex sets generated as above. The candidate non-convex images were then tested by checking a convexity condition for every pair of pixels in the non-convex set. Those sets that failed the convexity test were added to the dataset. The parameters for generating the convex and non-convex sets were balanced to ensure that the mean number of pixels in the set is the same.

The generated training, validation and test sets are of size 6000, 2000 and 50000 respectively. Samples for this tasks are displayed in figure 8.6.

8.5 Experiments

We performed experiments on the proposed benchmarks in order to compare the performance of models with deep architectures with other popular generic classification algorithms.

In addition to the Deep Belief Network (denoted DBN-3) and Stacked Autoassociators (denoted SAA-3) models, we conducted experiments with a single hidden-layer DBN (DBN-1), a single hidden-layer neural network (NNet), SVM models with Gaus-

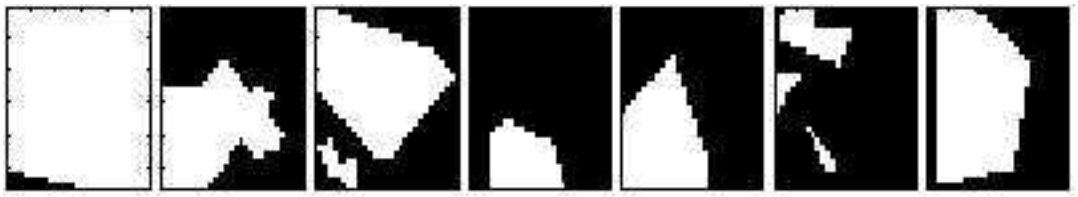


Figure 8.6: Samples from *convex*, where the first, fourth, fifth and last samples correspond to convex white pixel sets.

sian ($\mathbf{SVM}_{rbf}$) and polynomial ($\mathbf{SVM}_{poly}$) kernels.

In all cases, model selection was performed using a validation set. For NNet, the best combination of number of hidden units (varying from 25 to 700), learning rate (from 0.0001 to 0.1) and decrease constant (from 0 to 10^{-6}) of stochastic gradient descent and weight decay penalization (from 0 to 10^{-5}) was selected using a grid search.

For DBN-3 and SAA-3, both because of the large number of hyper-parameters and because these models can necessitate more than a day to train, we could not perform a full grid search in the space of hyper-parameters. For both models, the number of hidden units per layer must be chosen, in addition to all other optimization parameters (learning rates for the unsupervised and supervised phases, stopping criteria of the unsupervised phase, etc.). The hyper-parameter search procedure we used alternates between fixing a neural network architecture and searching for good optimization hyper-parameters in a manner similar to coordinate descent[1]. In general, we tested from 50 to 150 different configurations of hyper-parameters for DBN-3 and SAA-3. The layer sizes varied in the intervals $[500, 3000]$, $[500, 4000]$ and $[1000, 6000]$ respectively for the first, second and third layer and the learning rates varied between 0.0001 and 0.1. In the case of the single hidden layer DBN-1 model, we allowed ourselves to test for much larger hidden layer sizes, in order to balance the number of parameters between it and the DBN-3 models we tested.

For all neural networks, we used early stopping based on the classification error of the model on the validation set. However during the initial unsupervised training of DBN-

[1]See `http://www.iro.umontreal.ca/~lisa/icml2007` for more details about this procedure

3, the intractability of the RBM training criterion precluded the use of early stopping. Instead, we tested 50 or 100 unsupervised learning epochs for each layer and selected the best choice based on the final accuracy of the model on the validation set.

The experiments with the NNet, DBN-1, DBN-3 and SAA-3 models were conducted using the `PLearn`[2] library, an Open Source C++ library for machine learning which was developed and is actively used in our lab.

In the case of SVMs with Gaussian kernels, we performed a two-stage grid search for the width of the kernel and the soft-margin parameter. In the first stage, we searched through a coarse logarithmic grid ranging from $\sigma = 10^{-7}$ to 1 and $C = 0.1$ to 10^5. In the second stage, we performed a more fine-grained search in the vicinity of that tuple (σ, C) that gave the best validation error. In the case of the polynomial kernel, the strategy was the same, except that we searched through all possible degrees of the polynomial up to 20, rendering the fine-grained search on this parameter useless. Conforming to common practice, we also allowed the SVM models to be retrained on the concatenation of the training and validation set using the selected hyper-parameters. Throughout the experiments we used the publicly available library `libSVM` [32], version 2.83.

For all datasets, the input was normalized to have values between 0 and 1. When the input was binary (i.e. for *rectangles* and *convex*), the Deep Belief Network model used binary input units and when the input was in $[0, 1]^n$ (i.e. for *mnist-rot*, *mnist-back-rand*, *mnist-back-imag*, *mnist-rot-back-image* and *rectangles-image*) it used truncated exponential input units [14].

8.5.1 Benchmark Results

The classification performances for the different learning algorithms on the different datasets of the benchmark are reported in table 8.1. As a reference for the variations on digit recognition experiments, we also include the algorithms' performance on the original MNIST database, with training, validation and test sets of size 10000, 2000 and 50000 respectively. Note that the training set size is significantly smaller than that typically used.

[2] See `http://www.plearn.org/`

Dataset	SVM_{rbf}	SVM_{poly}	NNet	DBN-1	SAA-3	DBN-3
mnist-basic	**3.03±0.15**	3.69±0.17	4.69±0.19	3.94±0.17	3.46±0.16	**3.11±0.15**
mnist-rot	11.11±0.28	15.42±0.32	18.11±0.34	14.69±0.31	**10.30±0.27**	**10.30±0.27**
mnist-back-rand	14.58±0.31	16.62±0.33	20.04±0.35	9.80±0.26	11.28±0.28	**6.73±0.22**
mnist-back-image	22.61±0.37	24.01±0.37	27.41±0.39	**16.15±0.32**	23.00±0.37	**16.31±0.32**
mnist-rot-back-image	55.18±0.44	56.41±0.43	62.16±0.43	52.21±0.44	51.93±0.44	**47.39±0.44**
rectangles	**2.15±0.13**	**2.15±0.13**	7.16±0.23	4.71±0.19	**2.41±0.13**	2.60±0.14
rectangles-image	24.04±0.37	24.05±0.37	33.20±0.41	23.69±0.37	24.05±0.37	**22.50±0.37**
convex	19.13±0.34	19.82±0.35	32.25±0.41	19.92±0.35	**18.41±0.34**	**18.63±0.34**

Table 8.1: Results on the benchmark for problems with factors of variation (in percentages). The best performance as well as those with overlapping confidence intervals are marked in bold.

Taken together, these results show that deep architecture models globally have the best performance. In all cases, either DBN-3 or SAA-3 are among the best performing models (within the confidence intervals). Five times out of eight the best accuracy is obtained with a deep architecture model (either DBN-3 or SAA-3). This is especially true in two cases, *mnist-back-rand* and *mnist-rot-back-image*, where they perform better by a large margin. Also, deep architecture models consistently improve on NNet, which is basically a shallow and totally supervised version of the deep architecture models.

8.5.2 Impact of Background Pixel Correlation

Looking at the results obtained on *mnist-back-rand* and *mnist-back-image* by the different algorithms, it seems that pixel correlation contained in the background images is the key element that worsens the performances. To explore the disparity in performance of the learning algorithms between MNIST with independent noise and MNIST on a background image datasets, we made a series of datasets of MNIST digits superimposed on a background of correlated noisy pixel values.

Correlated pixel noise was sampled from a zero-mean multivariate Gaussian distribution of dimension equal to the number of pixels: $s \sim \mathcal{N}(0, \Sigma)$. The covariance matrix, Σ, is specified by a convex combination of an identity matrix and a Gaussian kernel function (with bandwidth $\sigma = 6$) with mixing coefficient γ. The Gaussian kernel induced a neighborhood correlation structure among pixels such that nearby pixels are more correlated than pixels further apart. For each sample from $\mathcal{N}(0, \Sigma)$, the pixel values p (ranging from 0 to 255) were determined by passing elements of s through the standard error function $p_i = erf(s_i/\sqrt{2})$ and multiplying by 255. We generated six datasets with varying degrees of neighborhood correlation by setting the mixture weight γ to the values $\{0, 0.2, 0.4, 0.6, 0.8, 1\}$. The marginal distributions for each pixel p_i is uniform[0,1] for each value of γ. Figure 8.7 shows some samples from the 6 different tasks.

We ran experiments on these 6 datasets, in order to measure the impact of background pixel correlation on the classification performance. Figure 8.8 shows a comparison of the results obtained by DBN-3, SAA-3 and SVM_{rbf}. In the case of the deep models,

we used the same layer sizes for all six experiments. The selected layer sizes had good performance on both *mnist-back-image* and *mnist-back-rand*. However, we did vary the hyper-parameters related to the optimization of the deep networks and chose the best ones for each problems based on the validation set performance. All hyper-parameters of SVM_{rbf} were chosen according to the same procedure.

It can be seen that, as the amount of background pixel correlation increases, the classification performance of all three algorithms degrade. This is coherent with the results obtained on *mnist-back-image* and *mnist-back-rand*. This also indicates that, as the factors of variation become more complex in their interaction with the input space, the relative advantage brought by DBN-3 and SAA-3 diminishes. This observation is preoccupying and implies that learning algorithms such as DBN-3 and SAA-3 will eventually need to be adapted in order to scale to harder, potentially "real life" problem.

One might argue that it is unfair to maintain the same layer sizes of the deep architecture models in the previous experiment, as it is likely that the model will need more capacity as the input distribution becomes more complex. This is a valid point, but given that, in the case of DBN-3 we already used a fairly large network (the first, second and third layers had respectively 3000, 2000 and 2000 hidden units), scaling the size of the network to even bigger hidden layers implies serious computational issues. Also, for even more complex datasets such as the NORB dataset [102], which consists in 108×108 stereo images of objects from different categories with many factors of variation such as lighting conditions, elevation, azimuth and background, the size of the deep models becomes too large to even fit in memory. In our preliminary experiments where we subsampled the images to be 54×54 pixels, the biggest models we were able to train only reached 51.6% (DBN-3) and 48.0% (SAA-3), whereas SVM_{rbf} reached 43.6% and NNet reached 43.2%. Hence, a natural next step for learning algorithms for deep architecture models would be to find a way for them to use their capacity to more directly model features of the data that are more predictive of the target value.

Further details of our experiments and links to downloadable versions of the datasets are available online at: `http://www.iro.umontreal.ca/~lisa/icml2007`

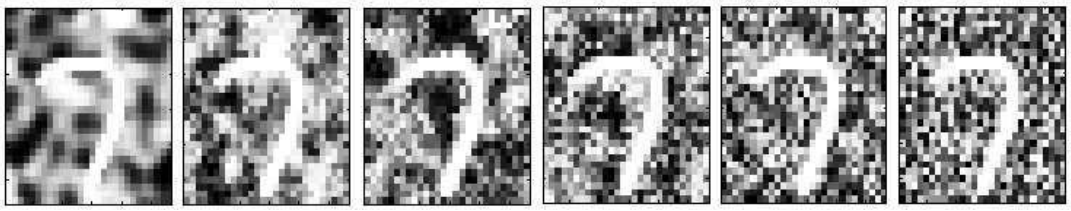


Figure 8.7: From left to right, samples with progressively less pixel correlation in the background.

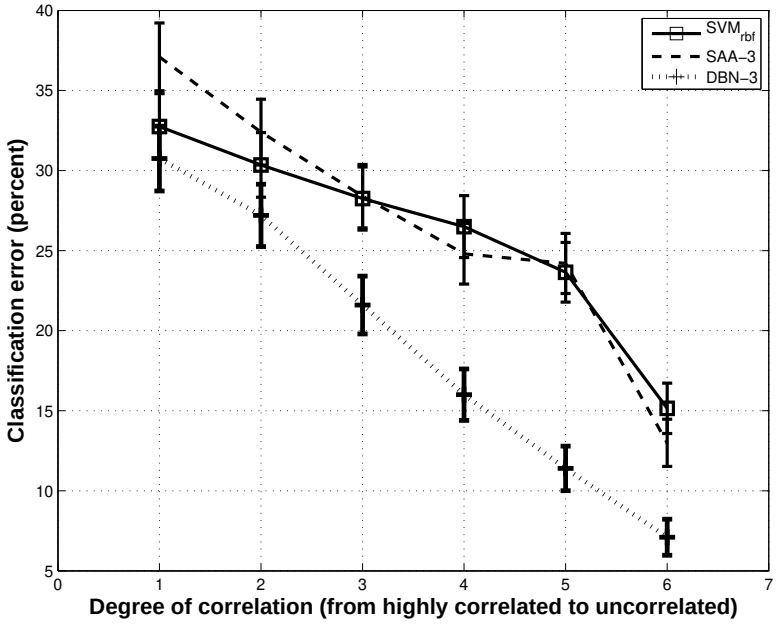


Figure 8.8: Classification error of SVM_{rbf}, **SAA-3** and **DBN-3** on MNIST examples with progressively less pixel correlation in the background.

8.6 Conclusion and Future Work

We presented a series of experiments which show that deep architecture models tend to outperform other shallow models such as SVMs and single hidden-layer feed-forward neural networks. We also analyzed the relationships between the performance of these learning algorithms and certain properties of the problems that we considered. In particular, we provided empirical evidence that they compare favorably to other state-of-the-art learning algorithms on learning problems with many factors of variation, but only up to a certain point where the data distribution becomes too complex and computational

constraints become an important issue.

Acknowledgments

We would like to thank Yann LeCun for suggestions and discussions. We thank the anonymous reviewers who gave useful comments that improved the paper. This work was supported by NSERC, MITACS and the Canada Research Chairs.

CHAPITRE 9

PRÉSENTATION DU QUATRIÈME ARTICLE

9.1 Détails de l'article

Extracting and Composing Robust Features with Denoising Autoencoders
Pascal Vincent, Hugo Larochelle, Yoshua Bengio et Pierre-Antoine Manzagol
Publié dans *Proceedings of the 24th Annual International Conference on Machine Learning, Omnipress* en 2008.

9.2 Contexte

Les deux articles précédents donnent plusieurs comparaisons empiriques entre l'entraînement d'un réseau profond à l'aide de machines de Boltzmann restreintes et d'autoencodeurs (ou autoassociateurs). On remarque que dans plusieurs cas, les autoencodeurs ne permettent pas d'obtenir une aussi bonne erreur de généralisation que les machines de Boltzmann restreintes. Par contre, étant donné la flexibilité de la modélisation par autoencodeurs, cet écart de performance est malheureux.

Une raison possiblement derrière cet écart proviendrait de l'existence de solutionss inintéressantes[1] à l'entraînement d'un autoencodeur lorsque celui-ci contient autant de neurones cachés que d'éléments d'entrée. Ainsi, il serait intéressant d'adapter la procédure d'entraînement afin de s'assurer que l'optimisation ne soit pas attirée vers ce genre de solution. L'article suivant propose une variante simple d'autoencodeur qui règle précisément ce problème.

9.3 Contributions

L'article suivant présente l'autoencodeur de débruitage ou *denoising autoencoder* qui, à partir d'une entrée à laquelle on a ajouté du « bruit », est entraîné à reconstuire

[1] On fait ici références aux solutions où les rangées de la matrice de connexions ne contient qu'un seul élément non nul, tel que discuté à la section 6.5.

l'entrée originale sans bruit. Plusieurs types différents de bruit pourraient être considérés. Dans l'article en question, on propose un bruit qui fixe à 0 une fraction fixe des éléments de l'entrée. Les éléments à fixer à 0 sont choisis aléatoirement. Ainsi, cet autoencodeur est entraîné à être robuste à l'absence de certains éléments. Puisque la fonction identité ($f(\mathbf{x}) = \mathbf{x}$) n'est pas une bonne solution pour ce critère d'entraînement, on réussit alors à éviter les solutions inintéressantes mentionnées ci-haut.

Dans cet article, on présente donc les résultats obtenus à l'aide de ce nouvel autoencodeur lorsqu'utilisé pour l'initialisation d'un réseau profond, dans le cadre d'expériences basées sur les mêmes jeux de données de l'article du chapitre 8. On remarque alors que les autoencodeurs de débruitage permettent d'atteindre et même dépasser dans plusieurs cas ceux obtenus à l'aide d'autoencodeurs standards ou de machines de Boltzmann restreintes. On y décrit aussi plusieurs motivations pour la dérivation de cette procédure d'entraînement des autoencodeurs, basées chacune sur l'apprentissage de variétés, la théorie de l'information et l'apprentissage d'un modèle génératif, respectivement.

9.4 Commentaires

Suite à la publication de cet article, les travaux de Seung [140] présentant une approche similaire à celle présentée ici ont été portés à mon attention. Seung [140] propose d'entraîner un réseau récurrent à reconstruire, en un nombre fixe d'itérations, une entrée à partir d'une version bruitée de celle-ci. Le bruit ajouté dans une image d'entrée correspond à la mise à 0 des composantes situées dans une petite région rectangulaire de cette image. L'approche présentée dans l'article de cette thèse correspond alors à celle de Seung [140] avec une seule itération, mais où le bruit ajouté n'est pas contraint à être structuré en une région rectangulaire de l'image et est ainsi plus générique. De façon plus importante, l'article de cette thèse vise à démontrer que cette façon d'entraîner un autoencodeur est bénéfique dans le contexte de l'initialisation d'un réseau profond, ce qui n'est pas l'objectif des travaux de Seung [140].

CHAPTER 10

EXTRACTING AND COMPOSING ROBUST FEATURES WITH DENOISING AUTOENCODERS

10.1 Abstract

Previous work has shown that the difficulties in learning deep generative or discriminative models can be overcome by an initial unsupervised learning step that maps inputs to useful intermediate representations. We introduce and motivate a new training principle for unsupervised learning of a representation based on the idea of making the learned representations robust to partial corruption of the input pattern. This approach can be used to train autoencoders, and these denoising autoencoders can be stacked to initialize deep architectures. The algorithm can be motivated from a manifold learning and information theoretic perspective or from a generative model perspective. Comparative experiments clearly show the surprising advantage of corrupting the input of autoencoders on a pattern classification benchmark suite.

10.2 Introduction

Recent theoretical studies indicate that deep architectures [10, 15] may be needed to *efficiently* model complex distributions and achieve better generalization performance on challenging recognition tasks. The belief that additional levels of functional composition will yield increased representational and modeling power is not new [76, 111, 150]. However, in practice, learning in deep architectures has proven to be difficult. One needs only to ponder the difficult problem of inference in deep directed graphical models, due to "explaining away". Also looking back at the history of multi-layer neural networks, their difficult optimization [10, 14] has long prevented reaping the expected benefits of going beyond one or two hidden layers. However this situation has recently changed with the successful approach of [14, 80, 81, 103, 125] for training Deep Belief Networks and stacked autoencoders.

One key ingredient to this success appears to be the use of an unsupervised training criterion to perform a layer-by-layer initialization: each layer is at first trained to produce a higher level (hidden) representation of the observed patterns, based on the representation it receives as input from the layer below, by optimizing a local unsupervised criterion. Each level produces a representation of the input pattern that is more abstract than the previous level's, because it is obtained by composing more operations. This initialization yields a starting point, from which a global fine-tuning of the model's parameters is then performed using another training criterion appropriate for the task at hand. This technique has been shown empirically to avoid getting stuck in the kind of poor solutions one typically reaches with random initializations. While unsupervised learning of a mapping that produces "good" intermediate representations of the input pattern seems to be key, little is understood regarding what constitutes "good" representations for initializing deep architectures, or what explicit criteria may guide learning such representations. We know of only a few algorithms that seem to work well for this purpose: Restricted Boltzmann Machines (RBMs) trained with contrastive divergence on one hand, and various types of autoencoders on the other.

The present research begins with the question of what explicit criteria a good intermediate representation should satisfy. Obviously, it should at a minimum retain a certain amount of "information" about its input, while at the same time being constrained to a given form (e.g. a real-valued vector of a given size in the case of an autoencoder). A supplemental criterion that has been proposed for such models is sparsity of the representation [103, 122]. Here we hypothesize and investigate an additional specific criterion: **robustness to partial destruction of the input**, i.e., partially destroyed inputs should yield almost the same representation. It is motivated by the following informal reasoning: a good representation is expected to capture stable structures in the form of dependencies and regularities characteristic of the (unknown) distribution of its observed input. For high dimensional redundant input (such as images) at least, such structures are likely to depend on evidence gathered from a combination of many input dimensions. They should thus be recoverable from partial observation only. A hallmark of this is our human ability to recognize partially occluded or corrupted images. Further evidence is

our ability to form a high level concept associated to multiple modalities (such as image and sound) and recall it even when some of the modalities are missing.

To validate our hypothesis and assess its usefulness as one of the guiding principles in learning deep architectures, we propose a modification to the autoencoder framework to explicitly integrate robustness to partially destroyed inputs. Section 10.3 describes the algorithm in details. Section 10.4 discusses links with other approaches in the literature. Section 10.5 is devoted to a closer inspection of the model from different theoretical standpoints. In section 10.6 we verify empirically if the algorithm leads to a difference in performance. Section 10.7 concludes the study.

10.3 Description of the Algorithm

10.3.1 Notation and Setup

Let X and Y be two random variables with joint probability density $p(X, Y)$, with marginal distributions $p(X)$ and $p(Y)$. Throughout the text, we will use the following notation:

- Expectation: $\mathbb{E}_{p(X)}[f(X)] = \int p(\mathbf{x}) f(\mathbf{x}) \mathrm{d}\mathbf{x}$.
- Entropy: $\mathbb{H}(X) = \mathbb{H}(p) = \mathbb{E}_{p(X)}[-\log p(X)]$.
- Conditional entropy: $\mathbb{H}(X|Y) = \mathbb{E}_{p(X,Y)}[-\log p(X|Y)]$.
- Kullback-Leibler divergence: $\mathbb{D}_{\mathsf{KL}}(p\|q) = \mathbb{E}_{p(X)}[\log \frac{p(X)}{q(X)}]$.
- Cross-entropy: $\mathbb{H}(p\|q) = \mathbb{E}_{p(X)}[-\log q(X)] = \mathbb{H}(p) + \mathbb{D}_{\mathsf{KL}}(p\|q)$.
- Mutual information: $\mathbf{I}(X; Y) = \mathbb{H}(X) - \mathbb{H}(X|Y)$.
- Sigmoid: $s(x) = \frac{1}{1+e^{-x}}$ and $s(\mathbf{x}) = (s(\mathbf{x}_1), \ldots, s(\mathbf{x}_d))^{\mathbf{T}}$.
- Bernoulli distribution with mean μ: $\mathcal{B}_\mu(x)$
  and by extension $\mathcal{B}_\mu(\mathbf{x}) = (\mathcal{B}_{\mu_1}(\mathbf{x}_1), \ldots, \mathcal{B}_{\mu_d}(\mathbf{x}_d))$.

The setup we consider is the typical supervised learning setup with a training set of n (input, target) pairs $D_n = \{(\mathbf{x}^{(1)}, t^{(1)}) \ldots, (\mathbf{x}^{(n)}, t^{(n)})\}$, that we suppose to be an i.i.d. sample from an unknown distribution $q(X, T)$ with corresponding marginals $q(X)$ and $q(T)$.

10.3.2 The Basic Autoencoder

We begin by recalling the traditional autoencoder model such as the one used in [14] to build deep networks. An autoencoder takes an input vector $\mathbf{x} \in [0, 1]^d$, and first maps it to a hidden representation $\mathbf{y} \in [0, 1]^{d'}$ through a deterministic mapping $\mathbf{y} = f_\theta(\mathbf{x}) = s(\mathbf{W}\mathbf{x} + \mathbf{b})$, parameterized by $\theta = \{\mathbf{W}, \mathbf{b}\}$. $\mathbf{W}$ is a $d' \times d$ weight matrix and $\mathbf{b}$ is a bias vector. The resulting latent representation $\mathbf{y}$ is then mapped back to a "reconstructed" vector $\mathbf{z} \in [0, 1]^d$ in input space $\mathbf{z} = g_{\theta'}(\mathbf{y}) = s(\mathbf{W}'\mathbf{y} + \mathbf{b}')$ with $\theta' = \{\mathbf{W}', \mathbf{b}'\}$. The weight matrix $\mathbf{W}'$ of the reverse mapping may optionally be constrained by $\mathbf{W}' = \mathbf{W}^\mathbf{T}$, in which case the autoencoder is said to have *tied weights*. Each training $\mathbf{x}^{(i)}$ is thus mapped to a corresponding $\mathbf{y}^{(i)}$ and a reconstruction $\mathbf{z}^{(i)}$. The parameters of this model are optimized to minimize the *average reconstruction error*:

$$\begin{aligned} \theta^\star, \theta'^\star = & \qquad \arg\min_{\theta,\theta'} \frac{1}{n} \sum_{i=1}^{n} L\left(\mathbf{x}^{(i)}, \mathbf{z}^{(i)}\right) \\ = & \qquad \arg\min_{\theta,\theta'} \frac{1}{n} \sum_{i=1}^{n} L\left(\mathbf{x}^{(i)}, g_{\theta'}(f_\theta(\mathbf{x}^{(i)}))\right) \end{aligned} \tag{10.1}$$

where L is a loss function such as the traditional *squared error* $L(\mathbf{x}, \mathbf{z}) = \|\mathbf{x} - \mathbf{z}\|^2$. An alternative loss, suggested by the interpretation of $\mathbf{x}$ and $\mathbf{z}$ as either bit vectors or vectors of bit probabilities (Bernoullis) is the *reconstruction cross-entropy*:

$$\begin{aligned} L_{\mathbb{H}}(\mathbf{x}, \mathbf{z}) = & \qquad \mathbb{H}(\mathcal{B}_\mathbf{x} \| \mathcal{B}_\mathbf{z}) \\ = & \qquad -\sum_{k=1}^{d} [\mathbf{x}_k \log \mathbf{z}_k + (1 - \mathbf{x}_k) \log(1 - \mathbf{z}_k)] \end{aligned} \tag{10.2}$$

Note that if $\mathbf{x}$ is a binary vector, $L_{\mathbb{H}}(\mathbf{x}, \mathbf{z})$ is a negative log-likelihood for the example $\mathbf{x}$, given the Bernoulli parameters $\mathbf{z}$. Equation 10.1 with $L = L_{\mathbb{H}}$ can be written

$$\theta^\star, \theta'^\star = \arg\min_{\theta,\theta'} \mathbb{E}_{q^0(X)} \left[L_{\mathbb{H}} \left(X, g_{\theta'}(f_\theta(X)) \right) \right] \tag{10.3}$$

where $q^0(X)$ denotes the empirical distribution associated to our n training inputs. This optimization will typically be carried out by stochastic gradient descent.

10.3.3 The Denoising Autoencoder

To test our hypothesis and enforce robustness to partially destroyed inputs we modify the basic autoencoder we just described. We will now train it to reconstruct a clean "repaired" input from a *corrupted*, partially destroyed one. This is done by first corrupting the initial input $\mathbf{x}$ to get a partially destroyed version $\tilde{\mathbf{x}}$ by means of a stochastic mapping $\tilde{\mathbf{x}} \sim q_{\mathcal{D}}(\tilde{\mathbf{x}}|\mathbf{x})$. In our experiments, we considered the following corrupting process, parameterized by the desired proportion ν of "destruction": for each input $\mathbf{x}$, a fixed number νd of components are chosen at random, and their value is forced to 0, while the others are left untouched. All information about the chosen components is thus removed from that particuler input pattern, and the autoencoder will be trained to "fill-in" these artificially introduced "blanks". Note that alternative corrupting noises could be considered[1]. The corrupted input $\tilde{\mathbf{x}}$ is then mapped, as with the basic autoencoder, to a hidden representation $\mathbf{y} = f_\theta(\tilde{\mathbf{x}}) = s(\mathbf{W}\tilde{\mathbf{x}} + \mathbf{b})$ from which we reconstruct a $\mathbf{z} = g_{\theta'}(\mathbf{y}) = s(\mathbf{W}'\mathbf{y} + \mathbf{b}')$ (see figure 10.1 for a schematic representation of the process). As before the parameters are trained to minimize the average reconstruction error $L_{\mathbb{H}}(\mathbf{x}, \mathbf{z}) = \mathbb{H}(\mathcal{B}_{\mathbf{x}} \| \mathcal{B}_{\mathbf{z}})$ over a training set, i.e. to have $\mathbf{z}$ as close as possible to the uncorrupted input $\mathbf{x}$. But the key difference is that $\mathbf{z}$ is now a deterministic function of $\tilde{\mathbf{x}}$ rather than $\mathbf{x}$ and thus the result of a stochastic mapping of $\mathbf{x}$.

Let us define the joint distribution

$$q^0(X, \widetilde{X}, Y) = q^0(X) q_{\mathcal{D}}(\widetilde{X}|X) \delta_{f_\theta(\widetilde{X})}(Y) \tag{10.4}$$

[1]The approach we describe and our analysis is not specific to a particular kind of corrupting noise.

where $\delta_u(v)$ puts mass 0 when $u \neq v$. Thus Y is a deterministic function of $\widetilde{X}$. $q^0(X, \widetilde{X}, Y)$ is parameterized by θ. The objective function minimized by stochastic gradient descent becomes:

$$\arg\min_{\theta,\theta'} \mathbb{E}_{q^0(X,\widetilde{X})} \left[L_{\mathbb{H}} \left(X, g_{\theta'}(f_\theta(\widetilde{X})) \right) \right]. \tag{10.5}$$

So from the point of view of the stochastic gradient descent algorithm, in addition to picking an input sample from the training set, we will also produce a random corrupted version of it, and take a gradient step towards reconstructing the uncorrupted version from the corrupted version. Note that in this way, the autoencoder cannot learn the identity, unlike the basic autoencoder, thus removing the constraint that $d' < d$ or the need to regularize specifically to avoid such a trivial solution.

10.3.4 Layer-wise Initialization and Fine Tuning

The basic autoencoder has been used as a building block to train deep networks [14], with the representation of the k-th layer used as input for the $(k+1)$-th, and the $(k+1)$-th layer trained after the k-th has been trained. After a few layers have been trained, the parameters are used as initialization for a network optimized with respect to a supervised training criterion. This greedy layer-wise procedure has been shown to yield significantly better local minima than random initialization of deep networks , achieving better generalization on a number of tasks [95].

The procedure to train a deep network using the denoising autoencoder is similar. The only difference is how each layer is trained, i.e., to minimize the criterion in eq. 10.5 instead of eq. 10.3. Note that the corruption process $q_{\mathcal{D}}$ is only used during training, but not for propagating representations from the raw input to higher-level representations. Note also that when layer k is trained, it receives as input the uncorrupted output of the previous layers.

10.4 Relationship to Other Approaches

Our training procedure for the denoising autoencoder involves learning to recover a clean input from a corrupted version, a task known as *denoising*. The problem of image denoising, in particular, has been extensively studied in the image processing community and many recent developments rely on machine learning approaches (see e.g. Elad et Aharon [50], Hammond et Simoncelli [68], Roth et Black [129]). A particular form of gated autoencoders has also been used for denoising in Memisevic [112]. Denoising using autoencoders was actually introduced much earlier [58, 99], as an alternative to Hopfield models [83]. Our objective however is fundamentally different from that of developing a competitive image denoising algorithm. We investigate explicit robustness to corrupting noise as a novel criterion guiding the learning of suitable intermediate representations to initialize a deep network. Thus our corruption+denoising procedure is applied not only on the input, but also recursively to intermediate representations.

The approach also bears some resemblance to the well known technique of augmenting the training data with stochastically "transformed" patterns. E.g. augmenting a training set by transforming original bitmaps through small rotations, translations, and scalings is known to improve final classification performance. In contrast to this technique our approach does not use any prior knowledge of image topology, nor does it produce extra labeled examples for supervised training. We use corrupted patterns in a generic (i.e. *not* specific to images) *unsupervised* initialization step, while the supervised training phase uses the unmodified original data.

There is a well known link between "training with noise" and regularization: they are equivalent for small additive noise [20]. By contrast, our corruption process is a large, non-additive, destruction of information. We train autoencoders to "fill in the blanks", not merely be smooth functions (regularization). Also in our experience, regularized autoencoders (i.e. with weight decay) do not yield the quantitative jump in performance and the striking qualitative difference observed in the filters that we get with denoising autoencoders.

There are also similarities with the work of [47] on robust coding over noisy chan-

nels. In their framework, a linear encoder is to encode a clean input for optimal transmission over a noisy channel to a decoder that reconstructs the input. This work was later extended to robustness to noise in the input, in a proposal for a model of retinal coding [48]. Though some of the inspiration behind our work comes from neural coding and computation, our goal is not to account for experimental data of neuronal activity as in [48]. Also, the non-linearity of our denoising autoencoder is crucial for its use in initializing a deep neural network.

It may be objected that, if our goal is to handle *missing values* correctly, we could have more naturally defined a proper latent variable generative model, and infer the posterior over the latent (hidden) representation in the presence of missing inputs. But this usually requires a costly marginalization[2] which has to be carried out for each new example. By contrast, our approach tries to learn a fast and robust deterministic mapping f_θ from examples of *already corrupted* inputs. The burden is on learning such a constrained mapping during training, rather than on unconstrained inference at use time. We expect this may force the model to capture implicit invariances in the data, and result in interesting features. Also note that in section 10.5.2 we will see how our learning algorithm for the denoising autoencoder can be viewed as a form of variational inference in a particular generative model.

The resulting training procedure for denoising autoencoder also bears some similarity with the pseudolikelihood criterion [19], which requires that a generative model be good at predicting a "missing" input component given the value of all others. The same training criterion has been proposed for training Dependency Networks [**?**], which are more general in that they are not necessarily consistent with any positive distribution of the input. The disadvantage of the pseulikelihood approach is that the relationship between the value of any input component with all others might be too simple for the model to extract any meaningfull or complex statistical structure from the input distribution. A worse-case scenario would consist in an input vector where each component is a copy of another component in that vector. Denoising autoencoders can easily overcome this issue by setting more input components to 0 so as to force the model to discover

[2] as for RBMs, where it is exponential in the number of missing values

relationships involving more input components (see Figure 10.3 for an illustration).

10.5 Analysis of Denoising Autoencoders

The above intuitive motivation for the denoising autoencoder was given with the perspective of discovering robust representations. In the following, which can be skipped without hurting the remainder of the paper, we propose alternative perspectives on the algorithm.

10.5.1 Manifold Learning Perspective

The process of mapping a corrupted example to an uncorrupted one can be visualized in Figure 10.2, with a low-dimensional manifold near which the data concentrate. We learn a stochastic operator $p(X|\widetilde{X})$ that maps an $\widetilde{X}$ to an X, $p(X|\widetilde{X}) = \mathcal{B}_{g_{\theta'}(f_\theta(\widetilde{X}))}(X)$. The corrupted examples will be much more likely to be outside and farther from the manifold than the uncorrupted ones. Hence the stochastic operator $p(X|\widetilde{X})$ learns a map that tends to go from lower probability points $\widetilde{X}$ to high probability points X, generally on or near the manifold. Note that when $\widetilde{X}$ is farther from the manifold, $p(X|\widetilde{X})$ should learn to make bigger steps, to reach the manifold. At the limit we see that the operator should map even far away points to a small volume near the manifold.

The denoising autoencoder can thus be seen as a way to define and learn a manifold. The intermediate representation $Y = f(X)$ can be interpreted as a coordinate system for points on the manifold (this is most clear if we force the dimension of Y to be smaller than the dimension of X). More generally, one can think of $Y = f(X)$ as a representation of X which is well suited to capture the main variations in the data, i.e., on the manifold. When additional criteria (such as sparsity) are introduced in the learning model, one can no longer directly view $Y = f(X)$ as an explicit low-dimensional coordinate system for points on the manifold, but it retains the property of capturing the main factors of variation in the data.

10.5.2 Top-down, Generative Model Perspective

In this section we recover the training criterion for our denoising autoencoder (eq. 10.5) from a generative model perspective. Specifically we show that training the denoising autoencoder as described in section 10.3.3 is equivalent to maximizing a variational bound on a particular generative model.

Consider the generative model $p(X, \widetilde{X}, Y) = p(Y)p(X|Y)p(\widetilde{X}|X)$ where $p(X|Y) = \mathcal{B}_{s(\mathbf{W}'Y+\mathbf{b}')}$ and $p(\widetilde{X}|X) = q_{\mathcal{D}}(\widetilde{X}|X)$. $p(Y)$ is a uniform prior over $Y \in [0,1]^{d'}$. This defines a generative model with parameter set $\theta' = \{\mathbf{W}', \mathbf{b}'\}$. We will use the previously defined $q^0(X, \widetilde{X}, Y) = q^0(X)q_{\mathcal{D}}(\widetilde{X}|X)\delta_{f_\theta(\widetilde{X})}(Y)$ (equation 10.4) as an auxiliary model in the context of a variational approximation of the log-likelihood of $p(\widetilde{X})$. Note that we abuse notation to make it lighter, and use the same letters X, $\widetilde{X}$ and Y for different sets of random variables representing the same quantity under different distributions: p or q^0. Keep in mind that whereas we had the dependency structure $X \to \widetilde{X} \to Y$ for q or q^0, we have $Y \to X \to \widetilde{X}$ for p.

Since p contains a corruption operation at the last generative stage, we propose to fit $p(\widetilde{X})$ to corrupted training samples. Performing maximum likelihood fitting for samples drawn from $q^0(\widetilde{X})$ corresponds to minimizing the cross-entropy, or maximizing

$$\begin{aligned}\mathcal{H} &= \max_{\theta'}\{-\mathbb{H}(q^0(\widetilde{X})\|p(\widetilde{X}))\} \\ &= \max_{\theta'}\{\mathbb{E}_{q^0(\widetilde{X})}[\log p(\widetilde{X})]\}. \end{aligned} \tag{10.6}$$

Let $q^\star(X, Y|\widetilde{X})$ be a conditional density, the quantity

$$\mathcal{L}(q^\star, \widetilde{X}) = \mathbb{E}_{q^\star(X,Y|\widetilde{X})}\left[\log \frac{p(X, \widetilde{X}, Y)}{q^\star(X, Y|\widetilde{X})}\right]$$

is a lower bound on $\log p(\widetilde{X})$ since the following can be shown to be true for any $q^\star$:

$$\log p(\widetilde{X}) = \mathcal{L}(q^\star, \widetilde{X}) + \mathbb{D}_{\mathsf{KL}}(q^\star(X, Y|\widetilde{X})\|p(X, Y|\widetilde{X}))$$

Also it is easy to verify that the bound is tight when $q^\star(X, Y|\widetilde{X}) = p(X, Y|\widetilde{X})$, where the $\mathbb{D}_{\mathsf{KL}}$ becomes 0. We can thus write $\log p(\widetilde{X}) = \max_{q^\star} \mathcal{L}(q^\star, \widetilde{X})$, and consequently rewrite equation 10.6 as

$$\begin{aligned}\mathcal{H} &= \max_{\theta'}\{\mathbb{E}_{q^0(\widetilde{X})}[\max_{q^\star} \mathcal{L}(q^\star, \widetilde{X})]\} \\ &= \max_{\theta', q^\star}\{\mathbb{E}_{q^0(\widetilde{X})}[\mathcal{L}(q^\star, \widetilde{X})]\} \end{aligned} \tag{10.7}$$

where we moved the maximization outside of the expectation because an unconstrained $q^\star(X, Y|\widetilde{X})$ can in principle perfectly model the conditional distribution needed to maximize $\mathcal{L}(q^\star, \widetilde{X})$ for any $\widetilde{X}$. Now if we replace the maximization over an unconstrained $q^\star$ by the maximization over the parameters θ of our q^0 (appearing in f_θ that maps an $\mathbf{x}$ to a $\mathbf{y}$), we get a lower bound on $\mathcal{H}$: $\mathcal{H} \geq \max_{\theta', \theta}\{\mathbb{E}_{q^0(\widetilde{X})}[\mathcal{L}(q^0, \widetilde{X})]\}$ Maximizing this lower bound, we find

$$\begin{aligned}&\arg\max_{\theta, \theta'}\{\mathbb{E}_{q^0(\widetilde{X})}[\mathcal{L}(q^0, \widetilde{X})]\} \\ &= \arg\max_{\theta, \theta'} \mathbb{E}_{q^0(X, \widetilde{X}, Y)}\left[\log \frac{p(X, \widetilde{X}, Y)}{q^0(X, Y|\widetilde{X})}\right] \\ &= \arg\max_{\theta, \theta'} \mathbb{E}_{q^0(X, \widetilde{X}, Y)}\Big[\log p(X, \widetilde{X}, Y)\Big] \\ &\qquad + \mathbb{E}_{q^0(\widetilde{X})}\left[\mathbb{H}[q^0(X, Y|\widetilde{X})]\right] \\ &= \arg\max_{\theta, \theta'} \mathbb{E}_{q^0(X, \widetilde{X}, Y)}\left[\log p(X, \widetilde{X}, Y)\right].\end{aligned}$$

Note that θ only occurs in $Y = f_\theta(\widetilde{X})$, and θ' only occurs in $p(X|Y)$. The last line is therefore obtained because $q^0(X|\widetilde{X}) \propto q_{\mathcal{D}}(\widetilde{X}|X)q^0(X)$ (none of which depends on (θ, θ')), and $q^0(Y|\widetilde{X})$ is deterministic, i.e., its entropy is constant, irrespective of (θ, θ'). Hence the entropy of $q^0(X, Y|\widetilde{X}) = q^0(Y|\widetilde{X})q^0(X|\widetilde{X})$, does not vary with (θ, θ'). Fi-

nally, following from above, we obtain our training criterion (eq. 10.5):

$$\begin{aligned}
&\arg\max_{\theta,\theta'} \mathbb{E}_{q^0(\widetilde{X})}[\mathcal{L}(q^0, \widetilde{X})] \\
&= \arg\max_{\theta,\theta'} \mathbb{E}_{q^0(X,\widetilde{X},Y)}[\log[p(Y)p(X|Y)p(\widetilde{X}|X)]] \\
&= \arg\max_{\theta,\theta'} \mathbb{E}_{q^0(X,\widetilde{X},Y)}[\log p(X|Y)] \\
&= \arg\max_{\theta,\theta'} \mathbb{E}_{q^0(X,\widetilde{X})}[\log p(X|Y = f_\theta(\widetilde{X}))] \\
&= \arg\min_{\theta,\theta'} \mathbb{E}_{q^0(X,\widetilde{X})} \left[L_{\mathbb{H}} \left(X, g_{\theta'}(f_\theta(\widetilde{X})) \right) \right]
\end{aligned}$$

where the third line is obtained because (θ, θ') have no influence on $\mathbb{E}_{q^0(X,\widetilde{X},Y)}[\log p(Y)]$ because we chose $p(Y)$ uniform, i.e. constant, nor on $\mathbb{E}_{q^0(X,\widetilde{X})}[\log p(\widetilde{X}|X)]$, and the last line is obtained by inspection of the definition of $L_{\mathbb{H}}$ in eq. 10.2, when $p(X|Y = f_\theta(\widetilde{X}))$ is a $\mathcal{B}_{g_{\theta'}(f_\theta(\widetilde{X}))}$.

10.5.3 Other Theoretical Perspectives

Information Theoretic Perspective: Consider $X \sim q(X)$, q unknown, $Y = f_\theta(\widetilde{X})$. It can easily be shown [155] that minimizing the expected reconstruction error amounts to *maximizing a lower bound on mutual information* $\mathbf{I}(X;Y)$. Denoising autoencoders can thus be justified by the objective that Y captures as much information as possible about X *even as* Y *is a function of corrupted input.*

Stochastic Operator Perspective: Extending the manifold perspective, the denoising autoencoder can also be seen as corresponding to a semi-parametric model from which we can sample [155]:

$$p(X) = \frac{1}{n} \sum_{i=1}^{n} \sum_{\tilde{\mathbf{x}}} p(X|\widetilde{X} = \tilde{\mathbf{x}}) q_{\mathcal{D}}(\tilde{\mathbf{x}}|\mathbf{x}_i),$$

where $\mathbf{x}_i$ is one of the n training examples.

10.6 Experiments

We performed experiments with the proposed algorithm on the same benchmark of classification problems used in [95][3]. It contains different variations of the MNIST digit classification problem (input dimensionality $d = 28 \times 28 = 784$), with added factors of variation such as rotation (*rot*), addition of a background composed of random pixels (*bg-rand*) or made from patches extracted from a set of images (*bg-img*), or combinations of these factors (*rot-bg-img*). These variations render the problems particularly challenging for current generic learning algorithms. Each problem is divided into a training, validation, and test set (10000, 2000, 50000 examples respectively). A subset of the original MNIST problem is also included with the same example set sizes (problem *basic*). The benchmark also contains additional binary classification problems: discriminating between convex and non-convex shapes (*convex*), and between wide and long rectangles (*rect*, *rect-img*).

[3] All the datasets for these problems are available at http://www.iro.umontreal.ca/∼lisa/icml2007.

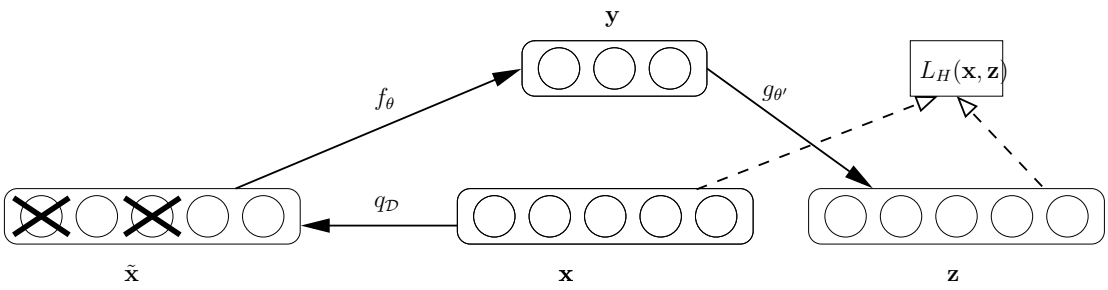


Figure 10.1: An example $\mathbf{x}$ is corrupted to $\tilde{\mathbf{x}}$. The autoencoder then maps it to $\mathbf{y}$ and attempts to reconstruct $\mathbf{x}$.

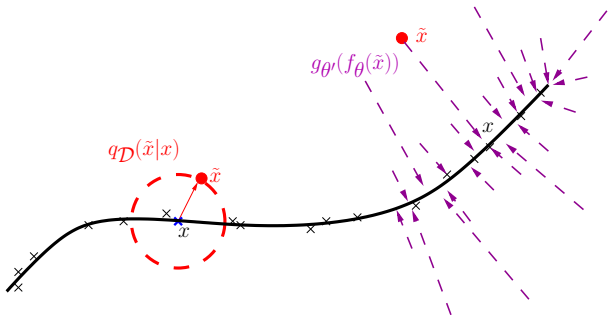
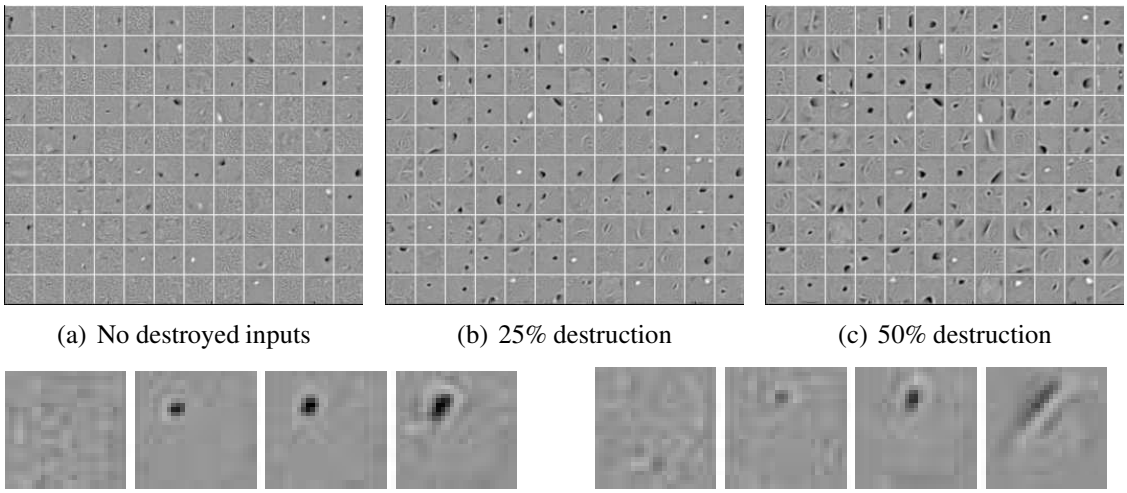


Figure 10.2: **Manifold learning perspective.** Suppose training data ($\times$) concentrate near a low-dimensional manifold. Corrupted examples ($\cdot$) obtained by applying corruption process $q_\mathcal{D}(\widetilde{X}|X)$ will lie farther from the manifold. The model learns with $p(X|\widetilde{X})$ to "project them back" onto the manifold. Intermediate representation Y can be interpreted as a coordinate system for points on the manifold.



(a) No destroyed inputs (b) 25% destruction (c) 50% destruction

(d) Neuron A (0%, 10%, 20%, 50% destruction) (e) Neuron B (0%, 10%, 20%, 50% destruction)

Figure 10.3: **Filters obtained after training the first denoising autoencoder.**
(a-c) show some of the filters obtained after training a denoising autoencoder on MNIST samples, with increasing destruction levels ν. The filters at the same position in the three images are related only by the fact that the autoencoders were started from the same random initialization point.
(d) and **(e)** zoom in on the filters obtained for two of the neurons, again for increasing destruction levels.
As can be seen, with no noise, many filters remain similarly uninteresting (undistinctive almost uniform grey patches). As we increase the noise level, denoising training forces the filters to differentiate more, and capture more distinctive features. Higher noise levels tend to induce less local filters, as expected. One can distinguish different kinds of filters, from local blob detectors, to stroke detectors, and some full character detectors at the higher noise levels.

Dataset	SVM_{rbf}	SVM_{poly}	DBN-1	SAA-3	DBN-3	SdA-3 (ν)
basic	**3.03±0.15**	3.69±0.17	3.94±0.17	3.46±0.16	3.11±0.15	**2.80±0.14** (10%)
rot	11.11±0.28	15.42±0.32	14.69±0.31	**10.30±0.27**	**10.30±0.27**	**10.29±0.27** (10%)
bg-rand	14.58±0.31	16.62±0.33	9.80±0.26	11.28±0.28	**6.73±0.22**	10.38±0.27 (40%)
bg-img	22.61±0.37	24.01±0.37	**16.15±0.32**	23.00±0.37	**16.31±0.32**	**16.68±0.33** (25%)
rot-bg-img	55.18±0.44	56.41±0.43	52.21±0.44	51.93±0.44	47.39±0.44	**44.49±0.44** (25%)
rect	**2.15±0.13**	**2.15±0.13**	4.71±0.19	2.41±0.13	2.60±0.14	**1.99±0.12** (10%)
rect-img	24.04±0.37	24.05±0.37	23.69±0.37	24.05±0.37	22.50±0.37	**21.59±0.36** (25%)
convex	19.13±0.34	19.82±0.35	19.92±0.35	**18.41±0.34**	**18.63±0.34**	**19.06±0.34** (10%)

Table 10.1: **Comparison of stacked denoising autoencoders (SdA-3) with other models.**
Test error rate on all considered classification problems is reported together with a 95% confidence interval. Best performer is in bold, as well as those for which confidence intervals overlap. SdA-3 appears to achieve performance superior or equivalent to the best other model on all problems except *bg-rand*. For SdA-3, we also indicate the fraction ν of destroyed input components, as chosen by proper model selection. Note that SAA-3 is equivalent to SdA-3 with $\nu = 0\%$.

Neural networks with 3 hidden layers initialized by stacking *denoising autoencoders* (SdA-3), and fine tuned on the classification tasks, were evaluated on all the problems in this benchmark. Model selection was conducted following a similar procedure as Larochelle et al. [95]. Several values of hyper parameters (destruction fraction ν, layer sizes, number of unsupervised training epochs) were tried, combined with early stopping in the fine tuning phase. For each task, the best model was selected based on its classification performance on the validation set.

Table 10.1 reports the resulting classification error on the test set for the new model (SdA-3), together with the performance reported in Larochelle et al. [95][4] for SVMs with Gaussian and polynomial kernels, 1 and 3 hidden layers deep belief network (DBN-1 and DBN-3) and a 3 hidden layer deep network initialized by stacking basic autoencoders (SAA-3). Note that SAA-3 is equivalent to a SdA-3 with $\nu = 0\%$ destruction. As can be seen in the table, the corruption+denoising training works remarkably well as an initialization step, and in most cases yields significantly better classification performance than basic autoencoder stacking with no noise. On all but one task the SdA-3 algorithm performs on par or better than the best other algorithms, including deep belief nets. Due to space constraints, we do not report all selected hyper-parameters in the table (only showing ν). But it is worth mentioning that, for the majority of tasks, the model selection procedure chose best performing models with an *overcomplete first hidden layer representation* (typically of size 2000 for the 784-dimensional MNIST-derived tasks). This is very different from the traditional "bottleneck" autoencoders, and made possible by our denoising training procedure. All this suggests that the proposed procedure was indeed able to produce more useful feature detectors.

Next, we wanted to understand qualitatively the effect of the corruption+denoising training. To this end we display the filters obtained after initial training of the first denoising autoencoder on MNIST digits. Figure 10.3 shows a few of these filters as little image patches, for different noise levels. Each patch corresponds to a row of the learnt weight matrix $\mathbf{W}$, i.e. the incoming weights of one of the hidden layer neurons. The

[4]Except that *rot* and *rot-bg-img*, as reported on the website from which they are available, have been regenerated since Larochelle et al. [95], to fix a problem in the initial data generation process. We used the updated data and corresponding benchmark results given on this website.

beneficial effect of the denoising training can clearly be seen. Without the denoising procedure, many filters appear to have learnt no interesting feature. They look like the filters obtained after random initialization. But when increasing the level of destructive corruption, an increasing number of filters resemble sensible feature detectors. As we move to higher noise levels, we observe a phenomenon that we expected: filters become less local, they appear sensitive to larger structures spread out across more input dimensions.

10.7 Conclusion and Future Work

We have introduced a very simple training principle for autoencoders, based on the objective of undoing a corruption process. This is motivated by the goal of learning representations of the input that are robust to small irrelevant changes in input. We also motivated it from a manifold learning perspective and gave an interpretation from a generative model perspective.

This principle can be used to train and stack autoencoders to initialize a deep neural network. A series of image classification experiments were performed to evaluate this new training principle. The empirical results support the following conclusions: unsupervised initialization of layers with an explicit denoising criterion helps to capture interesting structure in the input distribution. This in turn leads to intermediate representations much better suited for subsequent learning tasks such as supervised classification. It is possible that the rather good experimental performance of Deep Belief Networks (whose layers are initialized as RBMs) is partly due to RBMs encapsulating a similar form of robustness to corruption in the representations they learn, possibly because of their stochastic nature which introduces noise in the representation during training. Future work inspired by this observation should investigate other types of corruption process, not only of the input but of the representation itself as well.

Acknowledgments

We thank the anonymous reviewers for their useful comments that helped improved the paper. We are also very grateful for financial support of this work by NSERC, MITACS and CIFAR.

CHAPITRE 11

PRÉSENTATION DU CINQUIÈME ARTICLE

11.1 Détails de l'article

Deep Learning using Robust Interdependent Codes
Hugo Larochelle, Dumitru Erhan and Pascal Vincent
Publié dans *Proceedings of the 12th International Conference on Artificial Intelligence and Statistics,* en 2009.

11.2 Contexte

Jusqu'à maintenant, l'amélioration de la performance de généralisation d'un réseau de neurones que l'on a réussi à obtenir est basée sur deux principes :

- la régularisation basée sur l'apprentissage non-supervisé ;
- l'imposition d'une architecture profonde à plusieurs couches cachées.

Le premier principe est plutôt générique et est fondamental au succès du réseau de neurones. Le second est plus spécifiquement inspiré du genre de problèmes que l'on cherche à résoudre, soit des problèmes associés à l'intelligence humaine. On peut ainsi se demander s'il est possible d'aller un peu plus loin dans la spécification du genre de calculs que l'on croit nécessaire pour bien résoudre ce genre de problème.

On a déjà vu à la section 2.3 qu'il y a des raisons de croire que le système visuel humain serait organisé selon une architecture profonde. Il est aussi connu qu'en plus de connexions neuronales entre chacune des couches ou régions du système visuel, il existe aussi des connexions dites « horizontales » entre les neurones d'une même région. On s'attend ainsi à ce que ces connexions puissent implémenter des interactions d'inhibition et d'excitation entre différents neurones. Cette observation apporte donc une source d'inspiration naturelle pour tenter d'améliorer encore plus la performance d'un réseau de neurones profond.

11.3 Contributions

L'article suivant décrit une approche simple et efficace pour introduire des interactions d'inhibition et d'excitation entre les neurones d'une couche cachée. Cette approche, basée sur le modèle d'autoencodeur de débruitage présenté dans l'article précédent, est efficace dans la mesure où elle ne nécessite pas l'atteinte du point d'équilibre d'un système d'équations récursif, contrairement aux autres méthodes ayant été proposées dans la littérature. Plus précisément, cette approche correspond à un autoencodeur de débruitage contenant deux couches cachées de même taille. Les connexions entre ces deux couches jouent alors le rôle de connexions horizontales pouvant implémenter des relations d'inhibition et d'excitation. Conceptuellement, les première et deuxième couches cachées correspondent alors à la valeur des neurones avant et après l'application des interactions liées à ces connexions, respectivement.

On démontre dans cet article que cette approche réussit bien à apprendre des interactions d'inhibition et d'excitation à l'aide d'apprentissage non-supervisé, et que l'ajout de telles interactions à un réseau profond permet d'obtenir de meilleures performances dans le cadre de problèmes de classification de caractères. On y présente aussi des expériences qui indiquent que les représentations extraites de cette façon sont au moins aussi utiles que celles calculées par une approche basée sur un système d'équations récursif, tout en étant calculées beaucoup plus efficacement.

11.4 Commentaires

On remarquera que les résultats rapportés dans cet article pour les autoencodeurs de débruitage et les machines de Boltzmann restreintes sont légèrement meilleurs que ceux précédemment donnés. Cette différence est simplement due à la recherche d'hyperparamètres plus exhaustive effectuée dans le cadre de cet article, comparativement aux expériences des chapitres 8 et 10 qui ont été faites sur beaucoup plus de jeux de données et rendaient difficile une exploration aussi approfondie. Plus précisément, les résultats de cet article ont été obtenus en utilisant 2000 neurones cachés par couche, ce qui a nécessité un temps de calcul significatif.

CHAPTER 12

DEEP LEARNING USING ROBUST INTERDEPENDENT CODES

We investigate a simple yet effective method to introduce inhibitory and excitatory interactions between units in the layers of a deep neural network classifier. The method is based on the greedy layer-wise procedure of deep learning algorithms and extends the denoising autoencoder [154] by adding asymmetric lateral connections between its hidden coding units, in a manner that is much simpler and computationally more efficient than previously proposed approaches. We present experiments on two character recognition problems which show for the first time that lateral connections can significantly improve the classification performance of deep networks.

12.1 Introduction

Recently, an increasing amount of work in the machine learning literature has been addressing the difficult issue of training neural networks with many layers of hidden neurons. The motivation behind introducing several intermediate layers between the input of a neural network and its output is that hard AI-related learning problems, such as those addressing vision and language, require discovering complex high-level abstractions, which can be represented more efficiently by models with a deep architecture [15]. While deep networks are not novel, the discovery of techniques able to train them successfully and deliver superior generalization performance is recent. This new class of algorithms, *deep learning* algorithms, have proved successful at leveraging the power of deep networks in several contexts such as image classification [95], object recognition [123], regression [134], dimensionality reduction [80] and document retrieval [133].

Current deep learning algorithms are based on a greedy layer-wise training procedure [14, 81] which decouples the algorithm in two phases. The *pre-training phase* initializes a deep network with a set of *greedy modules* by training them sequentially in an unsupervised manner. Each is trained on the representation produced by the greedy module below, with the goal to discover a higher-level representation of it, so that the

representations become more abstract as we move up the network. This is followed by a *fine-tuning phase* which aims at globally adjusting all the parameters of the network according to some (often supervised) criterion related to the ultimate task of interest.

Most recent research has been focusing on the development of good greedy modules, which play a decisive role in the quality of the representations learned by deep networks. A variety of greedy modules have been proposed: Restricted Boltzmann Machines (RBMs) [81], autoassociators or autoencoders [14], sparse autoencoders [122], denoising autoencoders [154] and non-linear embedding algorithms [163]. These greedy modules leverage unlabeled data to discover meaningful representations and their training objectives span a vast variety of motivations and properties of representations.

All these previous greedy modules however share one characteristic about the way they transform their input into a new representation: given an input pattern, all elements of the representation are computed independently and cannot interact in an inhibitory or excitatory fashion. However, there is a growing body of work on introducing pairwise interactions between the hidden units of models with latent representations [59, 73, 86, 117], which show that they can be beneficial in modeling data such as patches of natural images.

In this paper, we extend the basic denoising autoencoder [154] by introducing lateral connections between coding elements, which permit simple yet useful interactions between codes. We show experimentally that the lateral connections learn to implement inhibitory and excitatory interactions which allow discrimination between visually overlapping patterns. We also demonstrate that such a **denoising autoencoder with interdependent codes (DA-IC)** outperforms the basic denoising autoencoder as well as RBMs in training deep neural network classifiers on two character recognition problems. Finally, we show that interdependent codes tend to extract a richer set of features which are less likely to be linearly predictable from each other (i.e. less correlated), leaving it to upper layers to account for the remaining non-linear dependencies between these features.

12.2 Denoising Autoencoder

The present work builds on the **denoising autoencoder** [154] as a greedy module for deep learning. Denoising autoencoders are motivated by the idea that a good representation $\mathbf{enc}(\mathbf{x})$ for some input vector $\mathbf{x}$ should be informative of $\mathbf{x}$ and invariant to induction of noise in the input. Given a *corrupted* version $\widetilde{\mathbf{x}}$ of the input, such a *robust* representation should make it possible to recover $\mathbf{x}$ from $\mathbf{enc}(\widetilde{\mathbf{x}})$, through a decoding function $\mathbf{dec}(\cdot)$.

A denoising autoencoder thus requires the following:

- $\mathbf{enc}(\cdot)$: an encoder function which computes a new representation for its input. This function's parameters should be adjustable given an error gradient.
- $\mathbf{dec}(\cdot)$: a decoder function which decodes a representation and gives a prediction for the original input. This function's parameters should also be adjustable.
- $p(\widetilde{\mathbf{x}}|\mathbf{x})$: a conditional distribution used to generate corrupted versions $\widetilde{\mathbf{x}}$ of an input $\mathbf{x}$.
- $C(\cdot,\cdot)$: a differentiable cost function that computes the dissimilarity between two vectors or representations.

The corruption process $p(\widetilde{\mathbf{x}}|\mathbf{x})$ used originally [154] sets to zero (i.e. destroys all information from) a random subset of the elements of $\mathbf{x}$, corresponding to a fraction α of all elements. This means that the autoencoder must learn to compute a representation that is informative of the original input even when some of its elements are missing. This technique was inspired by the ability of humans to have an appropriate understanding of their environment even in situations where the available information is incomplete (e.g. when looking at an object that is partly occluded).

Training a denoising autoencoder is as simple as training a standard autoencoder through backpropagation, with the additional step of corrupting the input. Given a training input pattern $\mathbf{x}_t$, first we generate a noisy version $\widetilde{\mathbf{x}}_t$, compute its representation $\mathbf{enc}(\widetilde{\mathbf{x}}_t)$, compute a reconstruction $\mathbf{dec}(\mathbf{enc}(\widetilde{\mathbf{x}}_t))$ and compare it to the **original input**

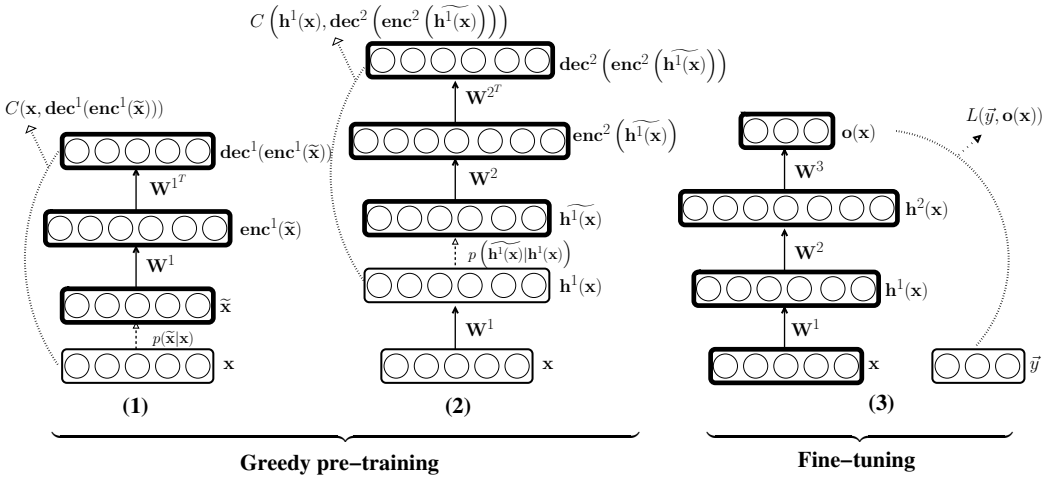


Figure 12.1: Illustration of the greedy layer-wise procedure for training a 2 hidden layer neural network with denoising autoencoders. To avoid clutter, biases $\mathbf{b}^i$ and $\mathbf{c}^i$ are not represented in the figures.

$\mathbf{x}_t$ using the cost function $C(\mathbf{x}_t, \mathbf{dec}(\mathbf{enc}(\widetilde{\mathbf{x}}_t))$. Then we compute the error gradient $\frac{\partial}{\partial \theta_k} C(\mathbf{x}_t, \mathbf{dec}(\mathbf{enc}(\widetilde{\mathbf{x}}_t))$ for all parameters θ_k of the encoder and decoder functions, and update all parameters using stochastic gradient descent.

We consider the same corruption process $p(\widetilde{\mathbf{x}}|\mathbf{x})$ and encoder/decoder pair as proposed originally:

$$\mathbf{enc}(\tilde{\mathbf{x}}) = \text{sigmoid}(\mathbf{b} + \mathbf{W}\tilde{\mathbf{x}}) \tag{12.1}$$

$$\mathbf{dec}(\mathbf{enc}(\tilde{\mathbf{x}})) = \text{sigmoid}(\mathbf{c} + \mathbf{W}^T \mathbf{enc}(\tilde{\mathbf{x}})) \tag{12.2}$$

and use the same cross-entropy cost function:

$$C(\mathbf{x}, \mathbf{y}) = -\sum_i \left(x_i \log y_i + (1 - x_i) \log(1 - y_i)\right)$$

where the elements of $\mathbf{x}$ and $\mathbf{y}$ are assumed to be in $[0, 1]$.

We wish to use denoising autoencoders to train a deep neural network classifier. In a network with l hidden layers, we compute the activity $\mathbf{h}^i(\mathbf{x})$ of the ith hidden layer given some input $\mathbf{x}$ as follows:
$\mathbf{h}^i(\mathbf{x}) = \text{sigmoid}(\mathbf{b}^i + \mathbf{W}^i \mathbf{h}^{i-1}(\mathbf{x}))\ \forall i \in \{1, \ldots, l\}$, with

$\mathbf{h}^0(\mathbf{x}) = \mathbf{x}$.

Class assignments probabilities are computed at the output layer as follows:
$\mathbf{o}(\mathbf{x}) = \text{softmax}(\mathbf{b}^{l+1} + \mathbf{W}^{l+1}\mathbf{h}^l(\mathbf{x}))$ with

$$\text{softmax}(\mathbf{a}) = \left(\frac{\exp(a_i)}{\sum_k \exp(a_k)} \right)_{i=1}^{m}$$

To use denoising autoencoders for deep learning, we follow the general greedy layer-wise procedure [14, 81] and pre-train each layer of a deep neural network as a denoising autoencoder. The procedure is depicted in Fig. 12.1. During the greedy pre-training phase, when training the ith layer, each input is mapped to its hidden representation $\mathbf{h}^{i-1}(\mathbf{x})$ and is used as a training sample to a denoising autoencoder with biases $\mathbf{b} = \mathbf{b}^i$, $\mathbf{c} = \mathbf{b}^{i-1}$ and weights $\mathbf{W} = \mathbf{W}^i$. Note that this requires the corruption of $\mathbf{h}^{i-1}(\mathbf{x})$ into $\widetilde{\mathbf{h}^{i-1}(\mathbf{x})}$. A layer is pre-trained for a fixed number of updates, after which the new representation it learned is stored to be used as input for the next layer. Greedy pre-training then moves on to the next hidden layer. Once all layers have thus been initialized, the whole network is fine-tuned[1] by stochastic gradient descent using backpropagation and the class assignment negative log-likelihood cost $L(\vec{y}, \mathbf{o}(\mathbf{x})) = -\sum_k \vec{y}_k \log o(\mathbf{x})_k$ where $\vec{y} = (1_{k=y})_{k=1}^m$.

12.3 Denoising Autoencoder with Interdependent Codes (DA-IC)

As mentioned earlier, a denoising autoencoder is one example of a deep network greedy module among others in the literature where the elements of the hidden representations (or codes) are computed independently. By this, we mean that the activation of a hidden layer neuron is a simple direct function of its input pattern only, and is not influenced by what other neurons in its layer do. They are therefore unable to implement interactions between these codes, such as inhibitory and excitatory interactions. Lateral connections between elements of hidden representations have been used successfully to model natural images in sparse coding [59], ICA [86] and energy-based [117] models.

In this work, we investigate whether such interactions can also be useful in learning a

[1] without any data corruption

deep neural network classifier. One approach to introduce interactions between the units of a layer is to express their effect in a recursive equation [118, 141]:

$$\mathbf{enc}(\widetilde{\mathbf{x}})_j = \text{sigmoid}\left(b_j + \sum_k W_{jk}\tilde{x}_k + \sum_{k\neq j} V_{jk}\mathbf{enc}(\widetilde{\mathbf{x}})_k\right) \tag{12.3}$$

where each V_{jk} induces an interaction between hidden neuron j and k, if $V_{jk} \neq 0$.

To compute an encoding, its elements are updated recursively according to Equation 12.3 for a number of iterations or until convergence. There are two disadvantages to this approach. First, computing the encoding becomes expensive for large layers or number of iterations. Second, optimizing this encoding through gradient descent is also expensive and hard. For these reasons, we decided to take a different approach which, while being much simpler conceptually and computationally, is able to implement the type of lateral interactions that are expected from Equation 12.3. We simply view the inhibitory and excitatory lateral connections as performing an extra non-linear processing step on the regular encoding, and model this step by a standard linear+sigmoid layer. Thus our approach is akin to simply adding a hidden layer to the encoding function, ensuring that all computations will be fast. The presence of simple constraints on the autoencoder, specifically the encoding/decoding functions sharing the same (transposed) weights, ensures that the role of the additional set of weights $\mathbf{V}$ can be interpreted as that of lateral connections, just like in Equation 12.3.

We extend the denoising autoencoder model by taking into account such lateral connections *in the encoder function only*, and propose to study their effect, and verify that they indeed behave according to what we expect from lateral connections. Introducing such richer interactions only in the encoder function can be motivated by the view of the decoder function as a generative model for which the encoder performs a crude variational "inference" [154]. It is well known that even very simple generative models can yield a complicated posterior over the hidden representation, due to "explaining away" effects. From this perspective, the mapping from visible to hidden is often more complex than the mapping from hidden to visible. So it makes sense to have a higher capacity encoder, with the ability to learn a more complex non-linear mapping, than the decoder.

Formally, the denoising autoencoder is modified by adding asymmetric lateral connections, whose strengths are stored in a square matrix $\mathbf{V}$, as follows: given a pre-encoding of a corrupted input

$$\widehat{\mathbf{enc}(\widetilde{\mathbf{x}})} = \text{sigmoid}(\mathbf{b} + \mathbf{W}\tilde{\mathbf{x}})$$

a final encoding is computed by using the following interaction between hidden units:

$$\mathbf{enc}(\widetilde{\mathbf{x}})_j = \text{sigmoid}\left(d_j + V_{jj}\widehat{\mathbf{enc}(\widetilde{\mathbf{x}})}_j + \sum_{k \neq j} V_{jk}\widehat{\mathbf{enc}(\widetilde{\mathbf{x}})}_k\right)$$

where $V_{jj} > 0$. The same decoding function of Equation 12.2 is used. Though the constraint of a positive diagonal for $\mathbf{V}$ could have required special attention, using the same weight matrix $\mathbf{W}$ in the pre-encoding and decoding implicitly favors this situation, a fact that was observed to hold empirically. We also find the diagonal elements of $\mathbf{V}$ to be usually larger than other elements on the same column or row. This DA-IC architecture is illustrated in Fig. 12.2.

To perform deep learning, we use a greedy layer-wise procedure to pre-train all layers. In this case, each layer $\mathbf{h}^i(\mathbf{x})$ also has lateral connections $\mathbf{V}^i$ as well as the additional set of biases $\mathbf{d}^i$:

$$\mathbf{h}^i(\mathbf{x}) = \text{sigmoid}\left(\mathbf{d}^i + \mathbf{V}^i\text{sigmoid}(\mathbf{b}^i + \mathbf{W}^i\tilde{\mathbf{x}})\right)$$

Thus, for each layer, pre-training is using the previous layer representations $\mathbf{h}^{i-1}(\mathbf{x})$ as training samples to a DA-IC with biases $\mathbf{b} = \mathbf{b}^i$, $\mathbf{c} = \mathbf{b}^{i-1}$, $\mathbf{d} = \mathbf{d}^i$ and weights $\mathbf{W} = \mathbf{W}^i$, $\mathbf{V} = \mathbf{V}^i$.

12.4 Related Work

The idea of introducing pairwise connections between elements in a hidden representation for unsupervised learning is not new. They have been used in an information maximization framework to allow overcomplete representations [141]. One important

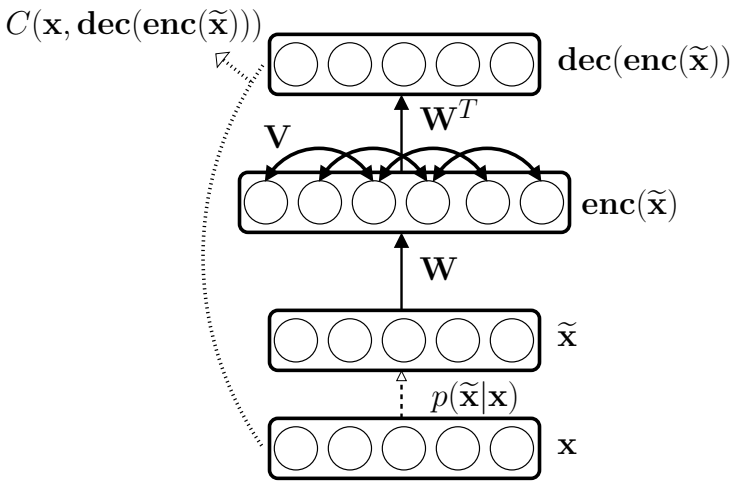


Figure 12.2: Illustration of the denoising autoencoder with interdependent codes.

difference in our approach is that the computation of the elements of the representation requires only one quick pass through the lateral connections instead of several recursive passes; the latter would render their use in a deep network much more computationally expensive.

Lateral connections have also been used previously in models with several layers of hidden representation [73, 118]. However, these connections are only used in the top-down generative process of the model and approximate bottom-up inference is done independently for each element of a hidden layer given the previous one. Interpreting the decoding function as the deterministic equivalent of a top-down generative process, the DA-IC takes the inverse perspective, where inference is complicated and generation (reconstruction) is simple.

Several models of the primary visual cortex have also integrated the concept of pairwise interactions, including sparse coding [59], ICA [86] and energy-based models [117]. One motivation often cited for using such connections is that they permit to better capture higher-order dependencies that would not be modeled otherwise.

Our work is aimed at leveraging the use of lateral connections in multi-layer neural networks for building competitive classifiers, in contrast to modeling the distribution of images. To our knowledge, none of the previously published approaches on introducing lateral connections in *deep networks* has studied if they did indeed yield a performance

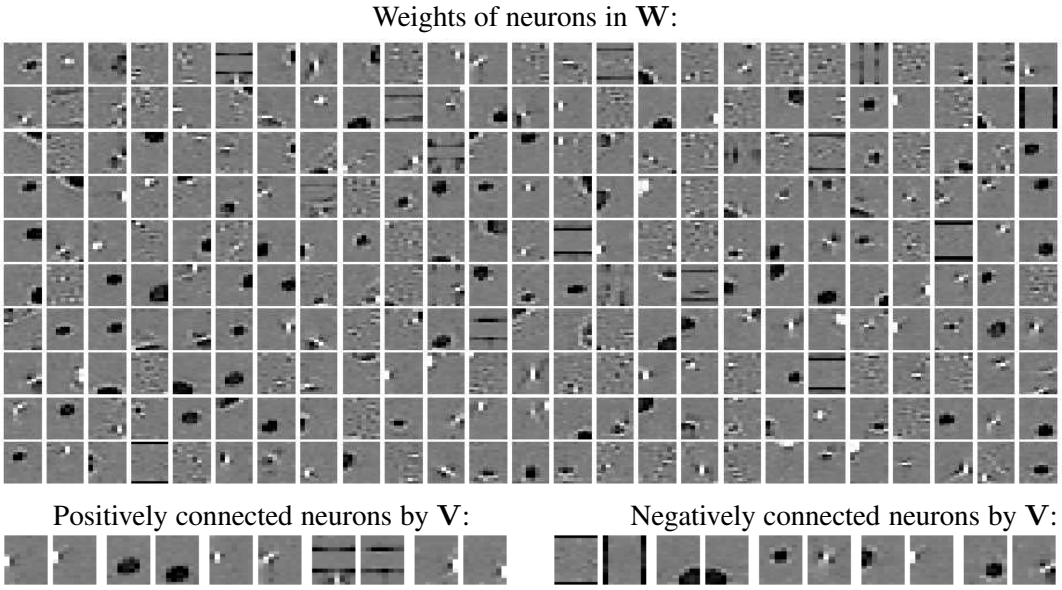


Figure 12.3: **Top:** visualization of the input weights of the hidden units, corresponding to the rows of $\mathbf{W}$. A variety of filters were learned, including small pen strokes and empty background detectors. **Bottom:** visualization of a subset of excitatory and inhibitory connections in $\mathbf{V}$. Positively connected neurons have overlapping filters, often shifted by few pixels. Negatively connected neurons detect aspects of the input which are mutually exclusive, such as empty background versus pen strokes.

gain when used to build a classifier. The discriminative power of sparse codes (whose inference exhibit inhibitory interactions, though without explicit lateral connections) has been investigated previously [120], however they are not applicable directly to deep learning, since fine-tuning such representations according to a global task presents a technical challenge. Moreover, though the Sparse Encoding Symmetric Machine [122] approach to sparse coding is appropriate for deep learning, as mentioned earlier, the encoding function in that case still computes the codes independently given an input, a situation we try to improve on here. Our simple approach for introducing interdependent codes in denoising autoencoders could however easily be adapted to that framework.

Dataset	SVM_{rbf}	DBN-3	SDA-3	SDA-6	SDAIC-3
MNIST-rot	11.11	9.01	9.53	9.68	**8.07**
OCR-letters (fold 1)	9.70	9.68	9.69	10.15	**9.60**
OCR-letters (fold 2)	9.36	9.68	9.92	9.92	**9.31**
OCR-letters (fold 3)	9.94	10.07	10.29	10.32	**9.46**
OCR-letters (fold 4)	10.32	10.46	10.42	10.51	**9.92**
OCR-letters (fold 5)	10.19	10.58	9.93	10.58	**9.50**
OCR-letters (all)	9.90	10.09	10.05	10.30	**9.56**

Table 12.1: Classification performance of deep networks and gaussian kernel SVMs for two character recognition problems. The deep networks with interdependent codes statistically significantly outperform other models on both problems. We report the results on each fold of the *OCR-letters* experiment to show that the improvement in performance of interdependent codes is consistent.

12.5 Experiments

We performed experiments on two character recognition problems where the input patterns from different classes are likely to be overlapping visually. This is a setting where lateral connections ought to be useful by using inhibitory connections to discern similar but mutually exclusive features of the input. The first problem, noted *MNIST-rot* [95], consists in classifying images of rotated digits[2]. The second classification dataset, noted *OCR-letters* [3] corresponds to an English character recognition problem where 16×8 binary pixel images must be classified into 26 classes, corresponding to the 26 letters of the English alphabet (see Fig. 12.4).

12.5.1 Comparison of classification performance

We evaluated the performance of the DA-IC as a greedy module for deep learning by comparing it with two other greedy modules: basic denoising autoencoders and RBMs. For each type of greedy module, deep neural network classifiers were initialized by

[2]This dataset has been regenerated since its publication and can be downloaded here: http://www.iro.umontreal.ca/∼lisa/icml2007. The set of 28×28 pixel images was generated using random rotations of digit images taken from the MNIST dataset, and was divided into training, validation and test splits of 10000, 2000 and 50000 examples each.

[3]This dataset is publicly available at http://ai.stanford.edu/∼btaskar/ocr/. For our experiments, we took the original dataset and generated 5 folds with mutually exclusive test sets of 10000 examples each.

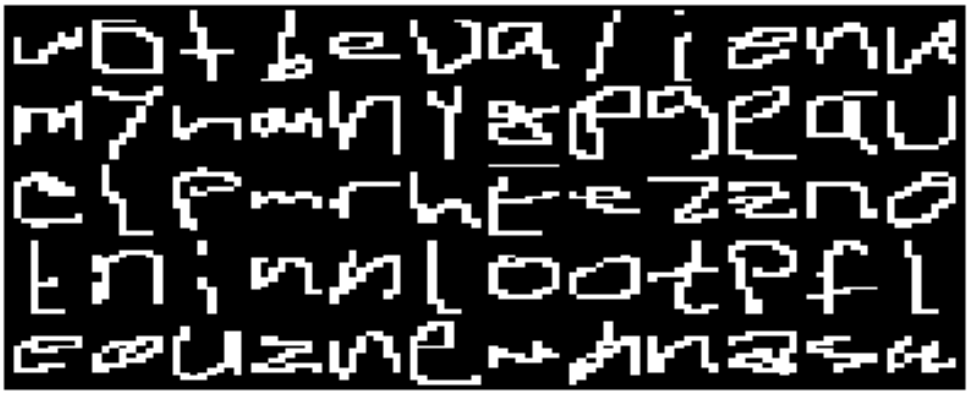


Figure 12.4: Input samples from the *OCR-letters* dataset of binary character images.

stacking three such greedy modules before fine-tuning the whole network by stochastic gradient descent[4]. The deep networks initialized with DA-ICs had 1000 hidden units in each layer. For fairness, since RBMs and basic denoising autoencoders have fewer parameters (hence less capacity) for the same size of hidden layer, we also considered deep networks with larger layers of up to 2000 hidden units in model selection. We chose networks with the same number of hidden units at each layer, as we found this topology to work well. Another fair comparison with a network with similar number of parameters, is to stack **6** layers of either RBMs or denoising autoencoders: both achieved about the same performance, so we report results on denoising autoencoders only. We denote by **DBN-**l, **SDA-**l and **SDAIC-**l deep networks initialized by stacking l modules of RBMs, denoising autoencoders, and DA-IC, respectively. As a general baseline, we also report the performance of a kernel SVM with Gaussian kernel (noted SVM_{rbf}), which often achieves state-of-the-art performance.

The results, reported in Table 12.1, confirm that the **interdependent codes are able to improve the discriminative performance of a deep network classifier.** The addition of lateral connections also enables deep networks to outperform an SVM classifier.

[4]Model selection, based on the classification error obtained on the validation set, was done over the number of iterations of greedy pre-training as well as the value of the learning rates for greedy pre-training and fine-tuning. For denoising autoencoders, the fraction of masked or destroyed inputs α also had to be chosen by model selection; we compared $\alpha = 0.1$ and 0.25. Early-stopping based on the validation set error determined the number of fine-tuning iterations.

The fact that SDAIC-3 outperforms SDA-6 shows that it is not simply the additional capacity of SDAIC-3 with respect to SDA-3 and DBN-3 that explains these performance differences. We also tried to add a phase of global *unsupervised* fine-tuning[5] before the supervised fine-tuning of SDA-6, but it at best improved only slightly its performance, not reaching the performance of SDAIC-3. This confirms the primary importance of pre-training with a DA-IC greedy module.

12.5.2 Qualitative analysis of learnt parameters

To get a better idea of the type of interactions the lateral connections are able to capture, we display in Fig. 12.3 the values of the weights or filters learned for each neuron, as well as the weights for pairs of neurons which have strong positive or negative lateral connections. Black, mid-gray and white pixels in the filters correspond to weights of -3, 0, and 3 respectively, with intermediate values corresponding to intermediate shades. The DA-IC was trained for 2.5 million updates on samples from the *OCR-letters* dataset, with a learning rate of 0.005, $\alpha = 0.25$ and a small L1 weight decay of 0.0001. The learned filters detect various aspects of the input, such as small pen strokes, which have localized positive weights and negative biases[6] (thus will be active only if a pen stroke is present), and regions of empty backgrounds, which have localized negative weights and positive biases (thus will only be active if no pen stroke is present). There are also filters that can determine whether the width and height of a character is smaller than a certain number of pixels (see filters with wide horizontal or vertical bars).

The lateral connections also model interesting interactions between these filters. Pairs of neurons that are positively connected often have visually similar filters. Also, pairs of neurons that are negatively connected are sensitive to mutually exclusive patterns in the input. For instance, pairs of pen-stroke and empty background detectors in the same region of the image usually inhibit each other. Another example is two filters that detect whether the sides or the top and bottom of the image are empty (see the first negatively connected pair in Fig. 12.3), two events that cannot be true simultaneously

[5]Global unsupervised fine-tuning consists in optimizing reconstruction error after a full up and down pass through all the layers.

[6]To simplify the visualization, the value of the biases are not shown in Fig. 12.3

since all characters touch at least one border of the image.

Next, we wanted to examine more closely the effect of $\mathbf{V}$. We presented a number of input patterns to a DA-IC trained on OCR-letters and considered pairs of neurons in the hidden layer with inhibitory lateral connections between them (corresponding to a negative weight in $\mathbf{V}$). We measured the activity of these neurons before applying $\mathbf{V}$ and after. Fig. 12.5 shows two examples, together with the filters associated to the considered neurons. A typical inhibitory behaviour can be observed: after applying $\mathbf{V}$ and a nonlinearity, a clear winner emerges within pairs of negatively connected neurons that have equally strong activities before applying $\mathbf{V}$. In the **e** example, the competition is between detecting a vertical segments on the left edge, or detecting it one pixel to the right. These are unlikely to occur together. In the **o** example, the choice is between detecting an empty spot in the lower right corner or seeing a vertical segment on the right edge that continues nearly to the bottom of the corner. Again, the two are contradictory. In both cases, inhibitory connections appear crucial in choosing the feature that better describes the input pattern. This disambiguation between two conflicting aspects in the input would not be possible with a simple layer that does not correct for interdependencies.

12.5.3 Comparison with alternative techniques for learning lateral interactions

Next, we wanted to see how our simple method for learning lateral interactions (DA-IC) compared to alternatives based on iterating a recursive equation, as previously proposed. Due to these alternatives being very time consuming, we focused on unsupervised training of a single layer (greedy module) to learn a representation (code)[7]. We then measured the classification performance obtained by a linear least squares classifier that uses that learned code as input. We specifically considered the following greedy modules:

- **RBM:** Restricted Boltzmann Machine with no lateral connections.
- **DA:** Ordinary Denoising Autoencoder, no lateral connections.

[7]We tested using both 10 and 30 iterations through Equation 12.3. Notice that computing $\mathbf{enc}(\widetilde{\mathbf{x}})$ with these alternative models requires **10 and 30 times** (respectively) as many multiply-add operations involving the $H^2 - H$ lateral connections V_{jk}, where H is the number of hidden units (the diagonal of $\mathbf{V}$ is not used in Equation 12.3).

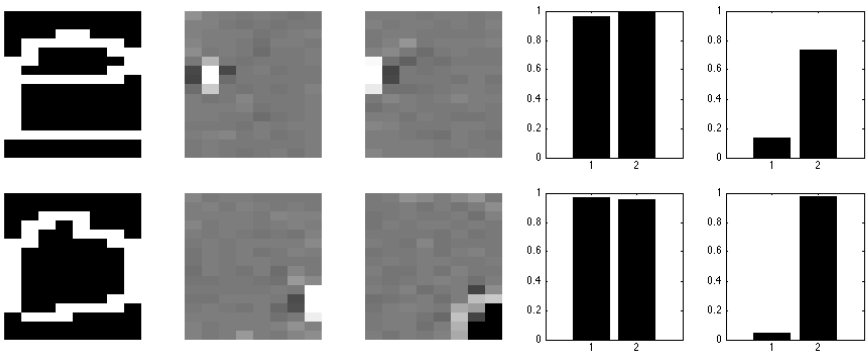


Figure 12.5: Illustration of inhibitory behaviour. Two examples are shown: **e** and **o**. In each, from left to right: the input pattern, the filters for two neurons of the first hidden layer, the values taken by these neurons before taking into account lateral connection weights $\mathbf{V}$, and their values after applying $\mathbf{V}$ and a sigmoid. As can be seen, lateral connections allow to disambiguate situations in which we have equally strong initial responses from the two neurons.

- **SRBM:** Semi-Restricted Boltzmann Machines [118], but with lateral connections between hidden units, instead of visible units as originally proposed.
- **DA-settling:** Denoising Autoencoder with "settling" lateral connections in the encoder: i.e. we iterate several times through Eq. 12.3.
- **DA-IC:** Our proposed Denoising Autoencoder with Interdependent Codes.

Fig. 12.6 gives the resulting classification performance as a function of the size of the code (the number of hidden units). We emphasize that the codes were learned in an entirely unsupervised fashion[8]. We observe that DA-IC systematically outperforms both RBM and DA (differences are statistically significant, except for 250 units on OCR-letters). When compared to the alternative techniques for introducing lateral interactions, DA-IC outperforms them on *MNIST-rot* (differences are statistically significant), and is also best (statistically equivalent to SRBM) on OCR-letters. We want to emphasize here that, contrary to the alternative techniques involving iterating a recursive equation, DA-IC is very simple and computationally very cheap (no iteration involved).

[8]Only the number of unsupervised training iterations and the learning rate were selected based on classification performance on the validation set

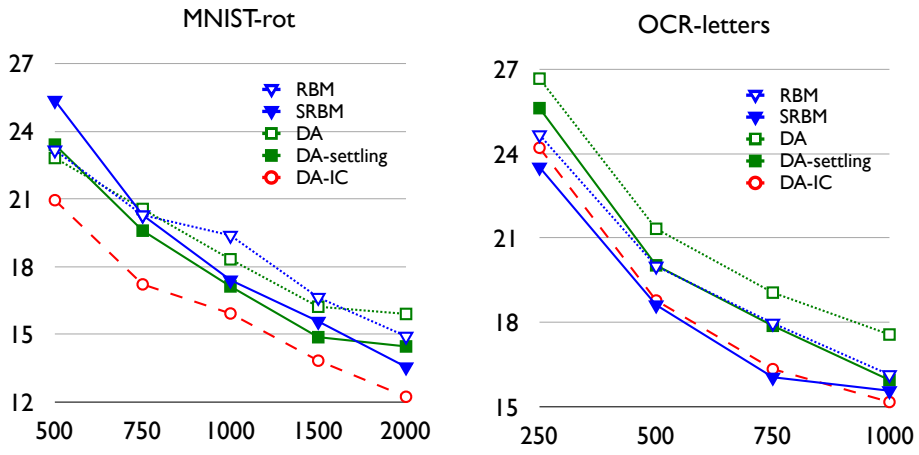


Figure 12.6: Test classification error (%) of a linear classifier using the codes learned by different types of greedy modules, for increasing size of hidden layer.

12.5.4 Analysis of correlation

Finally, we provide a possible explanation as to why DA-ICs are better suited for deep learning. The performance of deep networks with 1, 2 and 3 stacked DA-ICs is 10.33%, 8.91% and 8.07% respectively on the *MNIST-rot* dataset, which confirms that the DA-IC can leverage the addition of layers. Intuitively, a necessary condition for a greedy module to be appropriate for deep learning is that it should compute representations which, while being informative of the input, are not too linearly correlated. Otherwise, some of the coding elements would be easily predictable by others and therefore essentially useless. Since denoising autoencoders use a log-linear decoder function, training implicitly discourages highly correlated hidden units, which would waste some of the capacity of the encoder. However, as the size of the hidden layer grows, it is likely that adding uncorrelated units requires more non-linear computations from the encoder. So, by adding lateral connections to the encoder function, we would expect the encoder to be better able to reduce the correlation in its code units. To verify this claim, we computed the mean of pairwise absolute correlations between the activities of the hidden units of a denoising autoencoder and of a DA-IC for several large sizes of hidden layers, on the *MNIST-rot* dataset. Model selection was performed based on the mean absolute

correlations obtained on the validation set. The result, reported in Fig. 12.7, confirms that interdependent codes exhibit less correlation between their elements.

12.6 Conclusion

We presented a simple extension of denoising autoencoders which allows learning inhibitory and excitatory interactions between the hidden code units and demonstrated their usefulness as greedy modules for deep learning. Experiments on two character recognition problems showed that using Denoising Autoencoder with Interdependent Codes (DA-IC) **outperforms state-of-the-art learning algorithms for deep networks classifiers and kernel SVMs**. While the technique we use for taking into account lateral interactions is both **simpler and computationally much more efficient than previously proposed alternative techniques** (based on a recursive update equation) we showed it does learn codes that yield **equivalent or better classification performance** than these more cumbersome alternatives.

Acknowledgements

The authors thank Yoshua Bengio for constructive discussions. This research was supported by MITACS.

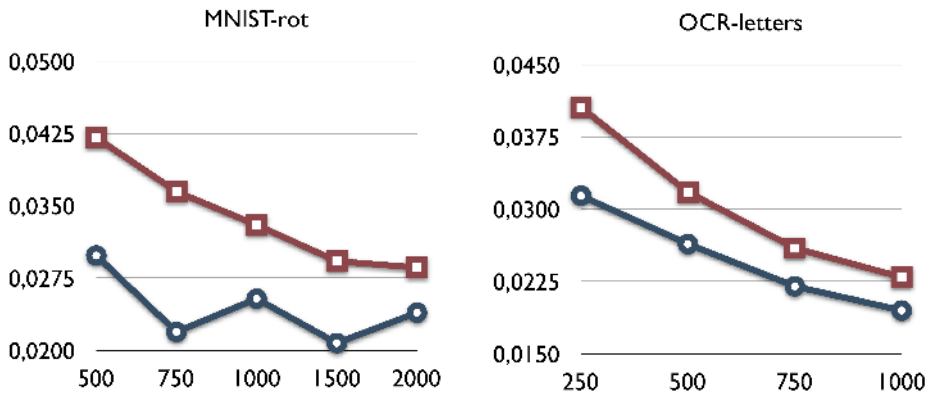


Figure 12.7: Mean pairwise absolute correlation between the coding elements of a basic denoising autoencoder (squares) and a denoising autoencoder with interdependent codes (circles), for different layer sizes.

CHAPITRE 13

CONCLUSION

Les travaux de cette thèse ont débuté dans un contexte de recherche où la popularité des réseaux de neurones artificiels était en pente descendante et où l'approche dominante pour la résolution d'un problème de classification était plutôt d'utiliser une machine à noyau. Certains chercheurs [12, 15] mettaient cependant en doute la capacité des machines à noyau à résoudre les problèmes liés à l'intelligence artificielle et, parallèlement, Hinton et al. [81] faisaient une percée par le développement d'un algorithme d'apprentissage efficace pour des réseaux à plusieurs couches cachées.

Les travaux de cette thèse visent donc à présenter des algorithmes d'apprentissage qui permettent aux réseaux de neurones artificiels de s'approcher un peu plus de la résolution de problèmes liés à l'intelligence artificielle. Par le fait même, les expériences présentées dans cette thèse ont démontré plus d'une fois la difficulté éprouvée par les machines à noyau à atteindre les mêmes performances de généralisation. Ces nouveaux algorithmes d'apprentissage tirent tous profit d'une forme d'apprentissage non-supervisé, afin de découvrir une représentation abstraite et utile des données. Les travaux de cette thèse ont aussi permis de mettre en évidence et de mieux comprendre les rôles joués par l'apprentissage non-supervisé et l'utilisation d'une architecture profonde dans l'amélioration de la performance d'un réseau de neurones artificiel.

13.1 Synthèse des articles

Voici donc une synthèse des contributions apportées par les différents articles de cette thèse.

13.1.1 Classification using Discriminative Restricted Boltzmann Machines

Dans cet article, une méthode simple est présentée pour améliorer la performance de généralisation d'un type spécifique de réseau de neurones, la machine de Boltzmann

restreinte. Cette méthode tire profit de la régularisation obtenue par l'apprentissage hybride génératif/discriminant, afin d'entraîner une couche de neurones cachés plus large qu'il n'aurait été possible sans une telle régularisation. Les expériences présentées dans cet article permettent de souligner l'avantage de l'utilisation d'un réseau de neurones plutôt qu'une machine à noyau, et ce, de façon marquée dans le cadre de l'apprentissage semi-supervisé.

En général, les travaux de cette thèse tendent à démontrer qu'une seule couche cachée n'est pas suffisante pour résoudre certains problèmes complexes. Bien que cela demeure une réalité, il existe tout de même plusieurs autres problèmes où une seule couche cachée suffit. Un avantage importante de la méthode présentée dans cet article est qu'elle nécessite en général peu d'hyper-paramètres, contrairement à l'approche où une machine de Boltzmann restreinte est utilisée pour l'initialisation d'un réseau. Ainsi, la facilité d'utilisation de cette méthode en fait une contribution importante.

13.1.2 Exploring Strategies for Training Deep Neural Networks

Cet article met en évidence l'importance de l'apprentissage non-supervisé dans l'apprentissage d'un réseau de neurones à plusieurs couches, selon la procédure vorace de Hinton et al. [81]. Les expériences qui y sont présentées démontrent qu'en plus d'apporter une régularisation utile au réseau, l'aspect vorace de la procédure d'apprentissage non-supervisé peut aussi faciliter l'optimisation du réseau.

De plus, on y a démontré que des performances similaires sont atteignables en utilisant des réseaux autoassociateurs plutôt que des machines de Boltzmann restreintes pour l'initialisation d'un réseau profond. Cette observation implique que l'apprentissage spécifique de la machine de Boltzmann restreinte n'est pas l'unique clé du succès de la procédure vorace. C'est plutôt l'utilisation d'une forme générale d'apprentissage non-supervisé qui est importante. Ainsi, cet article a ouvert la porte au développement d'autres algorithmes d'apprentissage non-supervisé pour l'apprentissage de réseaux profonds.

13.1.3 An Empirical Evaluation of Deep Architectures on Problems with Many Factors of Variation

Dans cet article, on présente une évaluation empirique des réseaux profonds sur une vaste gamme de problèmes générés à partir de plusieurs facteurs de variation. On y confirme l'avantage apporté par l'utilisation d'une architecture profonde, et ce sur beaucoup plus de jeux de données que dans l'article précédent. D'ailleurs, ces jeux de données forment une contribution additionnelle importante de ces travaux, et sont d'ailleurs disponibles pour téléchargement sur le Web pour quiconque souhaiterait comparer un nouvel algorithme à ceux utilisés dans cet article.

13.1.4 Extracting and Composing Robust Features with Denoising Autoencoders

Le second article de cette thèse a démontré que la procédure de Hinton et al. [81] n'avait pas à se restreindre aux machines de Boltzmann restreintes pour bien fonctionner. Ainsi, dans ce quatrième article, on démontre qu'une variante simple des réseaux autoassociateurs permet d'atteindre une performance de généralisation qui est en général au moins aussi bonne que celle obtenue par les machines de Boltzmann restreintes. Cette contribution est importante, puisqu'elle permet de combiner la flexibilité des autoassociateurs avec la bonne performance des machines de Boltzmann restreintes.

13.1.5 Deep Learning using Robust Interdependent Codes

Dans ce cinquième et dernier article, on améliore encore plus les réseaux autoassociateurs en introduisant de façon simple et efficace des interactions d'inhibition et d'excitation entre les neurones de leur couche cachée. Cette technique fonctionne par l'ajout d'une couche cachée de la même taille que la couche cachée originale, à laquelle elle est connectée. La valeur de cette nouvelle couche cachée correspond au résultat de l'application des interactions d'inhibition et d'excitation entre les neurones cachés. En utilisant ce nouvel autoassociateur pour initialiser un réseau profond, on réussit alors à obtenir de meilleures performances qu'auparavant sur des problèmes de classification de caractères. De plus, on observe que la méthode employée pour implémenter de telles

interactions est plus efficace et fonctionne au moins aussi bien qu'une méthode plus classique basée sur la solution d'un système d'équations récursif.

13.2 Conclusion

Les travaux de cette thèse visaient à apporter certaines contributions à la recherche sur l'entraînement des réseaux de neurones. Le choix de se concentrer sur les réseaux de neurones est motivé par leur capacité à généraliser de façon non-locale. Afin d'augmenter leur performance de généralisation, on a tiré profit de deux stratégies. La première est d'utiliser une procédure d'apprentissage non-supervisé afin de bien régulariser un réseau (profond) et découvrir des concepts statistiques utiles de l'entrée. La deuxième est d'utiliser une architecture profonde, afin de pouvoir extraire des concepts statistiques plus abstraits et complexes. Cette dernière stratégie est partiellement inspirée du genre de calculs effectués par le système visuel humain et de la structure hiérarchique naturellement retrouvée dans des données visuelles, textuelles et sonores. Ces deux stratégies s'étant avérées utiles, on peut se demander s'il serait bénéfique de les approfondir un peu plus. Mes travaux m'incitent à répondre par l'affirmative.

Considérons l'utilisation de l'apprentissage non-supervisé. On remarquera que dans le cas d'un réseau à plusieurs couches, l'apprentissage non-supervisé se fait de façon vorace, sans coordination entre l'entraînement des connexions entre différentes couches. La seule exception serait possiblement les travaux sur les autoassociateurs avec interaction d'inhibition et d'excitation, et cette approche a justement apporté une amélioration des performances. Ainsi, des progrès sont certainement possibles dans la procédure d'apprentissage non-supervisé des réseaux de neurones. De plus, il n'est pas clair que le même type d'apprentissage non-supervisé (i.e., le même algorithme) doive absolument être utilisé pour l'initialisation de toutes les couches d'un même réseau. Par exemple, Karklin et Lewicki [93] proposent d'entraîner une deuxième couche cachée modélisant bien la covariance (plutôt que la valeur moyenne) des éléments de la première couche cachée, dans le cadre d'un modèle hiérarchique de codage par représentation creuse (*sparse coding*). Les expériences présentées par Karklin et Lewicki [93] indiquent que

ce type de critère d'entraînement permet d'obtenir un second niveau de représentation informatif quand au type de texture présent dans l'image.

Pour ce qui est de l'utilisation d'une architecture profonde, j'estime probable que ce ne soit que le début de l'inspiration que puisse offrir la recherche sur le fonctionnement du cerveau. Entre autres, l'existence de connexions latérales entre les neurones d'une même région du système visuel humain aura aussi été une inspiration utile pour les travaux du cinquième article. De plus, la topologie de réseau profond où des connexions n'existent qu'entre deux couches cachées successives n'est pas vraiment compatible avec certaines observations expérimentales, qui laissent croire à l'existence de connexions entre couches non-successives. Ainsi, il n'est pas impossible que la topologie ayant été considérée dans les travaux de cette thèse puisse être raffinée de façon à améliorer encore plus les performances.

Finalement, des aspects plus pratiques devront aussi être considérés dans la recherche à venir sur les réseaux profonds. Par exemple, le défi que pose la sélection de modèle pour ces réseaux profonds a été mentionné à quelques reprises dans les travaux de cette thèse. Ainsi, des progrès de ce côté devront probablement être faits afin d'assurer une meilleure facilité d'utilisation de cette technologie. Ceci permettrait d'accélérer la recherche sur les réseaux profonds et de la rendre plus accessible aux chercheurs ayant une moins grande expérience avec ce type de modèle. Un autre obstacle à la recherche est la taille importante de ces réseaux, ce qui augmente le temps et la quantité de mémoire que nécessite leur utilisation. Il y a au moins deux approches possibles pour la réduction de ces inconvénients. La première est d'augmenter les ressources informatiques disponibles lors de l'utilisation d'un même réseau profond, par exemple en parallélisant les calculs à effectuer et en les distribuant sur plusieurs microprocesseurs. La seconde approche consiste à utiliser de façon parcimonieuse les ressources disponibles, en utilisant une paramétrisation creuse (*sparse*) du réseau. Cette paramétrisation pourrait être imposée, comme dans un réseau à convolution [100], ou apprise, comme c'est le cas lorsqu'on utilise une régularisation ℓ^1 [149]. Évidemment, l'implantation du réseau profond devrait utiliser une telle paramétrisation de façon à en tirer profit dans la quantité de calcul et de mémoire qui en résulte.

J'ai donc la conviction que la recherche sur l'entraînement de réseaux profonds présente des perspectives prometteuses dans la résolution de problèmes complexes liés à l'intelligence artificielle. Les travaux de cette thèse sont d'ailleurs en quelque sorte un signe de cette tendance. Ainsi, en plus de progrès dans le cadre de problèmes de classification statiques, je crois que des progrès dans des problèmes de nature dynamique ou séquentielle peuvent être considérés et sont à anticiper.

Index

BIBLIOGRAPHIE

[1] A. K. Agrawala. Learning with a probabilistic teacher. *IEEE Transactions on Information Theory*, 16:373–379, 1970.

[2] M. A. Aizerman, E. M. Braverman et L. I. Rozonoer. Theoretical foundations of the potential function method in pattern recognition learning. *Automation and Remote Control*, 25:821–837, 1964.

[3] S. Amari, N. Murata, K. R. Muller, M. Finke et H. H. Yang. Asymptotic statistical theory of overtraining and cross-validation. *IEEE Transactions on Neural Networks*, 8(5):985–996, 1997.

[4] Rie Kubota Ando et Tong Zhang. A framework for learning predictive structures from multiple tasks and unlabeled data. *Journal of Machine Learning Research*, 6:1817–1853, 2005.

[5] Peter Auer, Mark Herbster et Manfred K. Warmuth. Exponentially many local minima for single neurons. Dans M. Mozer, D. S. Touretzky et M. Perrone, éditeurs, *Advances in Neural Information Processing System 8*, pages 315–322. MIT Press, Cambridge, MA, 1996.

[6] Pierre Baldi et Kurt Hornik. Neural networks and principal component analysis : Learning from examples without local minima. *Neural Networks*, 2:53–58, 1989.

[7] Jonathan Baxter. Learning model bias. Dans M. Mozer, D. S. Touretzky et M. Perrone, éditeurs, *Advances in Neural Information Processing Systems*, volume 8, pages 169–175, Cambridge, MA, 1996. MIT Press.

[8] Mikhail Belkin, Partha Niyogi et Vikas Sindhwani. Manifold regularization : A geometric framework for learning from labeled and unlabeled examples. *Journal of Machine Learning Research.*, 7:2399–2434, 2006. ISSN 1533-7928.

[9] Anthony J. Bell et Terrence J. Sejnowski. An information maximisation approach to blind separation and blind deconvolution. *Neural Computation*, 7(6):1129–1159, 1995.

[10] Yoshua Bengio. Learning deep architectures for AI. Rapport technique 1312, Université de Montréal, dept. IRO, 2007.

[11] Yoshua Bengio et Olivier Delalleau. Justifying and generalizing contrastive divergence. Rapport technique 1311, Dept. IRO, Université de Montréal, 2007.

[12] Yoshua Bengio, Olivier Delalleau et Nicolas Le Roux. The curse of highly variable functions for local kernel machines. Dans Y. Weiss, B. Schölkopf et J. Platt, éditeurs, *Advances in Neural Information Processing Systems 18*, pages 107–114. MIT Press, Cambridge, MA, 2006.

[13] Yoshua Bengio, Olivier Delalleau et Nicolas Le Roux. Label propagation and quadratic criterion. Dans Olivier Chapelle, Bernhard Schölkopf et Alexander Zien, éditeurs, *Semi-Supervised Learning*, pages 193–216. MIT Press, 2006.

[14] Yoshua Bengio, Pascal Lamblin, Dan Popovici et Hugo Larochelle. Greedy layer-wise training of deep networks. Dans B. Schölkopf, J. Platt et T. Hoffman, éditeurs, *Advances in Neural Information Processing Systems 19*, pages 153–160. MIT Press, 2007.

[15] Yoshua Bengio et Yann LeCun. Scaling learning algorithms towards AI. Dans L. Bottou, O. Chapelle, D. DeCoste et J. Weston, éditeurs, *Large Scale Kernel Machines*. MIT Press, 2007.

[16] Yoshua Bengio, Martin Monperrus et Hugo Larochelle. Non-local estimation of manifold structure. *Neural Computation*, 18(10):2509–2528, 2006.

[17] Yoshua Bengio, Jean-Franç Paiement, Pascal Vincent, Olivier Delalleau, Nicolas Le Roux et Marie Ouimet. Out-of-sample extensions for LLE, Isomap, MDS, Eigenmaps, and Spectral Clustering. Dans S. Thrun, L. Saul et B. Schölkopf, éditeurs, *Advances in Neural Information Processing Systems 16*. MIT Press, 2004.

[18] A. Berger, S. Della Pietra et V. Della Pietra. A maximum entropy approach to natural language processing. *Computational Linguistics*, 22:39–71, 1996.

[19] Julian Besag. Statistical analysis of non-lattice data. *The Statistician*, 24(3):179–195, 1975.

[20] Chris M. Bishop. Training with noise is equivalent to tikhonov regularization. *Neural Computation*, 7(1):108–116, 1995.

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

[22] Avrim Blum et Shuchi Chawla. Learning from labeled and unlabeled data using graph mincuts. Dans *Proc. 18th International Conf. on Machine Learning*, pages 19–26. Morgan Kaufmann, San Francisco, CA, 2001.

[23] Avrim Blum et Tom Mitchell. Combining labeled and unlabeled data with co-training. Dans *COLT : Proceedings of the Workshop on Computational Learning Theory, Morgan Kaufmann Publishers*, pages 92–100, 1998.

[24] B. Boser, I. Guyon et V. Vapnik. A training algorithm for optimal margin classifiers. Dans *Fifth Annual Workshop on Computational Learning Theory*, pages 144–152, Pittsburgh, 1992.

[25] Léon Bottou. Online algorithms and stochastic approximations. Dans David Saad, éditeur, *Online Learning in Neural Networks*. Cambridge University Press, Cambridge, UK, 1998.

[26] Guillaume Bouchard. Bias-variance tradeoff in hybrid generative-discriminative models. Dans *ICMLA '07 : Proceedings of the Sixth International Conference on Machine Learning and Applications*, pages 124–129, Washington, DC, USA, 2007. IEEE Computer Society. ISBN 0-7695-3069-9.

[27] Guillaume Bouchard et Bill Triggs. The tradeoff between generative and discriminative classifiers. Dans *IASC International Symposium on Computational Statistics (COMPSTAT)*, pages 721–728, Prague, août 2004.

[28] Miguel A. Carreira-Perpiñan et Geoffrey E. Hinton. On contrastive divergence learning. Dans Robert G. Cowell et Zoubin Ghahramani, éditeurs, *Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics, Jan 6-8, 2005, Savannah Hotel, Barbados*, pages 33–40. Society for Artificial Intelligence and Statistics, 2005.

[29] Rich Caruana. Multitask connectionist learning. Dans *Proceedings of the 1993 Connectionist Models Summer School*, pages 372–379, 1993.

[30] Rich Caruana. Multitask learning. *Machine Learning*, 28(1):41–75, 1997.

[31] Zehra Cataltepe, Yaser S. Abu-mostafa et Malik Magdon-ismail. No free lunch for early stopping. *Neural Computation*, 11:995–1009, 1999.

[32] Chih-Chung Chang et Chih-Jen Lin. *LIBSVM : a library for support vector machines*, 2001. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[33] O. Chapelle, B. Schölkopf et A. Zien. *Semi-Supervised Learning*. MIT Press, Cambridge, MA, 2006.

[34] O. Chapelle et A. Zien. Semi-supervised classification by low density separation. Dans Z. Ghahramani Cowell, R., éditeur, *Tenth International Workshop on Artificial Intelligence and Statistics*, pages 57–64, 01 2005.

[35] Hsin Chen et Alan F. Murray. A continuous restricted Boltzmann machine with an implementable training algorithm. *IEE Proceedings of Vision, Image and Signal Processing*, 150(3):153–158, 2003.

[36] Ronan Collobert et Samy Bengio. Links between perceptrons, MLPs and SVMs. Dans *ICML '04 : Twenty-first international conference on Machine learning*, New York, NY, USA, 2004. ACM Press.

[37] Ronan Collobert et Jason Weston. A unified architecture for natural language processing : Deep neural networks with multitask learning. Dans *Proceedings of the Twenty-fifth International Conference on Machine Learning (ICML 2008)*, pages 160–167, 2008.

[38] Pierre Comon. Independent component analysis - a new concept ? *Signal Processing*, 36:287–314, 1994.

[39] C. Cortes et V. Vapnik. Support vector networks. *Machine Learning*, 20:273–297, 1995.

[40] G. W. Cottrell, P. Munro et D. Zipser. Learning internal representations from gray-scale images : An example of extensional programming. Dans *Ninth Annual Conference of the Cognitive Science Society*, pages 462–473, Seattle 1987, 1987. Lawrence Erlbaum, Hillsdale.

[41] T. Cox et M. Cox. *Multidimensional Scaling*. Chapman & Hall, London, 1994.

[42] Peter Dayan, Geoffrey Hinton, Radford Neal et Rich Zemel. The Helmholtz machine. *Neural Computation*, 7:889–904, 1995.

[43] Virginia R. de Sa. Learning classification with unlabeled data. Dans Jack D. Cowan, Gerald Tesauro et Joshua Alspector, éditeurs, *Proc. NIPS'93, Neural Information Processing Systems*, pages 112–119, San Francisco, CA, 1993. Morgan Kaufmann Publishers.

[44] D. Decoste et B. Scholkopf. Training invariant support vector machines. *Machine Learning*, 46:161–190, 2002.

[45] Olivier Delalleau, Yoshua Bengio et Nicolas Le Roux. Efficient non-parametric function induction in semi-supervised learning. Dans R. G. Cowell et Z. Ghah-

ramani, éditeurs, *Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics, Jan 6-8, 2005, Savannah Hotel, Barbados*, pages 96–103. Society for Artificial Intelligence and Statistics, 2005.

[46] David DeMers et Garrison W. Cottrell. Non-linear dimensionality reduction. Dans C.L. Giles, S.J. Hanson et J.D. Cowan, éditeurs, *Advances in Neural Information Processing Systems 5*, pages 580–587, San Mateo CA, 1993. Morgan Kaufmann.

[47] Eizaburo Doi, Doru C. Balcan et Michael S. Lewicki. A theoretical analysis of robust coding over noisy overcomplete channels. Dans Y. Weiss, B. Schölkopf et J. Platt, éditeurs, *Advances in Neural Information Processing Systems 18*, pages 307–314. MIT Press, Cambridge, MA, 2006.

[48] Eizaburo Doi et Michael S. Lewicki. A theory of retinal population coding. Dans Bernhard Schölkopf, John Platt et Thomas Hoffman, éditeurs, *NIPS*, pages 353–360. MIT Press, 2007.

[49] Gregory Druck, Chris Pal, Andrew Mccallum et Xiaojin Zhu. Semi-supervised classification with hybrid generative/discriminative methods. Dans *KDD '07 : Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 280–289, New York, NY, USA, 2007. ACM.

[50] M. Elad et M. Aharon. Image denoising via sparse and redundant representations over learned dictionaries. *IEEE Transactions on Image Processing*, 15(12):3736–3745, décembre 2006.

[51] Theodoros Evgeniou et Massimiliano Pontil. Regularized multi–task learning. Dans *KDD '04 : Proceedings of the 2004 ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 109–117, New York, NY, USA, 2004. ACM Press.

[52] Scott E. Fahlman et Christian Lebiere. The cascade-correlation learning architecture. Dans D.S. Touretzky, éditeur, *Advances in Neural Information Processing Systems 2*, pages 524–532, Denver, CO, 1990. Morgan Kaufmann, San Mateo.

[53] S. C. Fralick. Learning to recognize patterns without a teacher. *IEEE Transactions on Information Theory*, 13:57–64, 1967.

[54] Ildiko E. Frank et Jerome H. Friedman. A statistical view of some chemometrics regression tools. *Technometrics*, 35(2):109–148, 1993.

[55] Yoav Freund et David Haussler. Unsupervised learning of distributions on binary vectors using two layer networks. Rapport technique UCSC-CRL-94-25, University of California, Santa Cruz, 1994.

[56] Brendan J. Frey. *Graphical models for machine learning and digital communication*. MIT Press, 1998.

[57] Kenji Fukumizu et Shun-ichi Amari. Local minima and plateaus in hierarchical structures of multilayer perceptrons. *Neural Networks*, 13(3):317–327, 2000.

[58] Patrick Gallinari, Yann LeCun, Sylvie Thiria et Francoise Fogelman-Soulie. Memoires associatives distribuees. Dans *Proceedings of COGNITIVA 87*, Paris, La Villette, 1987.

[59] Pierre Garrigues et Bruno Olshausen. Learning horizontal connections in a sparse coding model of natural images. Dans J. C. Platt, D. Koller, Y. Singer et S. Roweis, éditeurs, *Advances in Neural Information Processing Systems 20*, pages 505–512. MIT Press, Cambridge, MA, 2008.

[60] Peter V. Gehler, Alex D. Holub et Max Welling. The rate adapting poisson model for information retrieval and object recognition. Dans *ICML '06 : Proceedings of the 23rd international conference on Machine learning*, pages 337–344, New York, NY, USA, 2006. ACM. ISBN 1-59593-383-2.

[61] Amir Globerson et Sam Roweis. Metric learning by collapsing classes. Dans Y. Weiss, B. Schölkopf et J. Platt, éditeurs, *Advances in Neural Information Processing Systems 18*, pages 451–458. MIT Press, Cambridge, MA, 2006.

[62] J. Goldberger, S. Roweis, G. Hinton et R. Salakhutdinov. Neighbourhood components analysis. Dans L.K. Saul, Y. Weiss et L. Bottou, éditeurs, *Advances in Neural Information Processing Systems 17*. MIT Press, 2005.

[63] Yves Grandvalet et Yoshua Bengio. Semi-supervised Learning by Entropy Minimization. Dans L.K. Saul, Y. Weiss et L. Bottou, éditeurs, *Advances in Neural Information Processing Systems 17*, Cambridge, MA, décembre 2004. MIT Press.

[64] Yves Grandvalet et Yoshua Bengio. Entropy regularization. Dans Olivier Chapelle, Bernhard Schölkopf et Alexander Zien, éditeurs, *Semi-Supervised Learning*, pages 151–168. MIT Press, 2006.

[65] Isabelle Guyon et Andre Elisseeff. An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3:1157–1182, 2003. ISSN 1533-7928.

[66] Raia Hadsell, Ayse Erkan, Pierre Sermanet, Marco Scoffier, Urs Muller et Yann LeCun. Deep belief net learning in a long-range vision system for autonomous off-road driving. Dans *Proc. Intelligent Robots and Systems (IROS'08)*, 2008.

[67] Katsuyuki Hagiwara et Kazuhiro Kuno. Regularization learning and early stopping in linear networks. Dans *IJCNN '00 : Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks (IJCNN'00)-Volume 4*, page 4511, Washington, DC, USA, 2000. IEEE Computer Society. ISBN 0-7695-0619-4.

[68] D. K. Hammond et E. P. Simoncelli. A machine learning framework for adaptive combination of signal denoising methods. Dans *2007 International Conference on Image Processing*, volume 6, pages 29–32, 2007.

[69] W. Härdle, M. Müller, S. Sperlich et A. Werwatz. *Nonparametric and Semiparametric Models*. Springer, http ://www.xplore-stat.de/ebooks/ebooks.html, 2004.

[70] Johan Håstad. Almost optimal lower bounds for small depth circuits. Dans *Proceedings of the 18th annual ACM Symposium on Theory of Computing*, pages 6–20, Berkeley, California, 1986. ACM Press.

[71] Johan Hastad et M. Goldmann. On the power of small-depth threshold circuits. *Computational Complexity*, 1:113–129, 1991.

[72] J. Hegdé et D. C. Van Essen. Selectivity for complex shapes in primate visual area v2. *Journal of Neuroscience*, 20(5), March 2000. ISSN 1529-2401.

[73] G. Hinton, S. Osindero et K. Bao. Learning causally linked markov random fields. Dans *AISTATS'05*, 2005.

[74] Geoff Hinton. To recognize shapes, first learn to generate images. Dans Paul Cisek, Trevor Drew et John Kalaska, éditeurs, *Computational Neuroscience : Theoretical Insights into Brain Function*. Elsevier, 2007.

[75] Geoffrey E. Hinton. Learning distributed representations of concepts. Dans *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*, pages 1–12, Amherst 1986, 1986. Lawrence Erlbaum, Hillsdale.

[76] Geoffrey E. Hinton. Connectionist learning procedures. *Artificial Intelligence*, 40:185–234, 1989.

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

[78] Geoffrey E. Hinton. To recognize shapes, first learn to generate images. Rapport technique UTML TR 2006-003, University of Toronto, 2006.

[79] Geoffrey E. Hinton, Peter Dayan, Brendan J. Frey et Radford M. Neal. The wake-sleep algorithm for unsupervised neural networks. *Science*, 268:1558–1161, 1995.

[80] Geoffrey E. Hinton et Ruslan R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, juillet 2006.

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

[82] Alex Holub et Pietro Perona. A discriminative framework for modelling object classes. Dans *CVPR '05 : Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 1*, pages 664–671, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7695-2372-2.

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

[84] Kurt Hornik, Maxwell Stinchcombe et Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366, 1989.

[85] David H. Hubel et Torsten N. Wiesel. Receptive fields of single neurons in the cat's striate cortex. *Journal of Physiology*, 148:574–591, 1959.

[86] A. Hyvärinen, P. O. Hoyer et M. O. Inki. Topographic independent component analysis. *Neural Computation*, 13(7):1527–1558, 2001.

[87] Minami Ito et Hidehiko Komatsu. Representation of angles embedded within contour stimuli in area v2 of macaque monkeys. *Journal of Neuroscience*, 24 (13):3313–3324, March 2004.

[88] Tommi S. Jaakkola et David Haussler. Exploiting generative models in discriminative classifiers. Dans M.S. Kearns, S.A. Solla et D.A. Cohn, éditeurs, *Advances in Neural Information Processing Systems 11*. MIT Press, Cambridge, MA, 1999.

[89] Tony Jebara. *Machine Learning : Discriminative and Generative (The Kluwer International Series in Engineering and Computer Science)*. Springer, décembre 2003.

[90] Thorsten Joachims. Transductive inference for text classification using support vector machines. Dans Ivan Bratko et Saso Dzeroski, éditeurs, *Proceedings of ICML-99, 16th International Conference on Machine Learning*, pages 200–209, Bled, SL, 1999. Morgan Kaufmann Publishers, San Francisco, US.

[91] I. T. Jolliffe. *Principal Component Analysis*. Springer-Verlag, New York, 1986.

[92] Christian Jutten et Jeanny Herault. Blind separation of sources, part I : an adaptive algorithm based on neuromimetic architecture. *Signal Processing*, 24:1–10, 1991.

[93] Yan Karklin et Michael S. Lewicki. A model for learning variance components of natural images. Dans S. Becker, S. Thrun et K. Obermayer, éditeurs, *Advances in Neural Information Processing Systems 15*, pages 1367–1374. MIT Press, 2003.

[94] Hugo Larochelle et Yoshua Bengio. Classification using discriminative restricted Boltzmann machines. Dans Andrew McCallum et Sam Roweis, éditeurs, *Proceedings of the 25th Annual International Conference on Machine Learning (ICML 2008)*, pages 536–543. Omnipress, 2008.

[95] Hugo Larochelle, Dumitru Erhan, Aaron Courville, James Bergstra et Yoshua Bengio. An empirical evaluation of deep architectures on problems with many factors of variation. Dans Zoubin Ghahramani, éditeur, *ICML 2007 : Proceedings of the Twenty-fourth International Conference on Machine Learning*, pages 473–480. Omnipress, 2007.

[96] Julia A. Lasserre, Christopher M. Bishop et Thomas P. Minka. Principled hybrids of generative and discriminative models. Dans *CVPR '06 : Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 87–94, Washington, DC, USA, 2006. IEEE Computer Society. ISBN 0-7695-2597-0.

[97] Nicolas Le Roux et Yoshua Bengio. Representational power of restricted Boltzmann machines and deep belief networks. *Neural Computation*, 20(6):1631–1649, 2008.

[98] 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.

[99] Yann LeCun. *Modèles connexionistes de l'apprentissage*. Thèse de doctorat, Université de Paris VI, 1987.

[100] Yann LeCun, Leon Bottou, Yoshua Bengio et Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, novembre 1998.

[101] Yann LeCun, Léon Bottou, Genevieve B. Orr et Klaus-Robert Müller. Efficient backprop. Dans *Neural Networks, Tricks of the Trade*, Lecture Notes in Computer Science LNCS 1524. Springer Verlag, 1998.

[102] Yann LeCun, Fu-Jie Huang et Léon Bottou. Learning methods for generic object recognition with invariance to pose and lighting. Dans *Proceedings of CVPR'04*. IEEE Press, 2004.

[103] Honglak Lee, Chaitanya Ekanadham et Andrew Ng. Sparse deep belief net model for visual area V2. Dans J.C. Platt, D. Koller, Y. Singer et S. Roweis, éditeurs, *Advances in Neural Information Processing Systems 20*. MIT Press, Cambridge, MA, 2008.

[104] Régis Lengellé et Thierry Denoeux. Training MLPs layer by layer using an objective function for internal representations. *Neural Networks*, 9:83–97, 1996.

[105] Percy Liang et Michael I. Jordan. An asymptotic analysis of generative, discriminative, and pseudolikelihood estimators. Dans *ICML '08 : Proceedings of the 25th international conference on Machine learning*, pages 584–591, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-205-4.

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

[107] D. J. C. MacKay. Bayesian interpolation. *Neural Computation*, 4(3):415–447, 1992.

[108] D. J. C. MacKay. A practical Bayesian framework for backpropagation networks. *Neural Computation*, 4(3):448–472, 1992.

[109] J. MacQueen. Some methods for classification and analysis of multivariate observations. Dans *Proceedings of the Fifth Berkeley Symposium on Mathematics, Statistics and Probability, Vol. 1*, pages 281–296, 1967.

[110] Andrew McCallum, Chris Pal, Gregory Druck et Xuerui Wang. Multi-conditional learning : Generative/discriminative training for clustering and classification. Dans *Twenty-first National Conference on Artificial Intelligence (AAAI-06)*. AAAI Press, 2006.

[111] J. L. McClelland, D. E. Rumelhart et the PDP Research Group. *Parallel Distributed Processing : Explorations in the Microstructure of Cognition*, volume 2. MIT Press, Cambridge, 1986.

[112] Roland Memisevic. *Non-linear latent factor models for revealing structure in high-dimensional data*. Thèse de doctorat, Departement of Computer Science, University of Toronto, Toronto, Ontario, Canada, 2007.

[113] Javier R. Movellan, Paul Mineiro et R. J. Williams. A monte-carlo EM approach for partially observable diffusion processes : theory and applications to neural networks. *Neural Computation*, 14:1501–1544, 2002.

[114] Radford M. Neal. Connectionist learning of belief networks. *Artificial Intelligence*, 56:71–113, 1992.

[115] Andrew Y. Ng et Michael I. Jordan. On discriminative vs. generative classifiers : A comparison of logistic regression and naive bayes. Dans *NIPS*, pages 841–848, 2001.

[116] B. A. Olshausen et D. J. Field. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 381:607–609, 1996.

[117] S. Osindero, M. Welling et G. E. Hinton. Topographic product models applied to natural scene statistics. *Neural Computation*, 18(2):381–414, 2006.

[118] Simon Osindero et Geoffrey E. Hinton. Modeling image patches with a directed hierarchy of markov random field. Dans *Neural Information Processing Systems Conference (NIPS) 20*, 2008.

[119] E. Parzen. On the estimation of a probability density function and mode. *Annals of Mathematical Statistics*, 33:1064–1076, 1962.

[120] Rajat Raina, Alexis Battle, Honglak Lee, Benjamin Packer et Andrew Y. Ng. Self-taught learning : transfer learning from unlabeled data. Dans *ICML*, pages 759–766, 2007.

[121] Rajat Raina, Yirong Shen, Andrew Y. Ng et Andrew Mccallum. Classification with hybrid generative/discriminative models. Dans *In Advances in Neural Information Processing Systems 16*. MIT Press, 2003.

[122] Marc'Aurelio Ranzato, Y-Lan Boureau et Yann LeCun. Sparse feature learning for deep belief networks. Dans J.C. Platt, D. Koller, Y. Singer et S. Roweis, éditeurs, *Advances in Neural Information Processing Systems 20*. MIT Press, Cambridge, MA, 2008.

[123] Marc'Aurelio Ranzato, Fu-Jie Huang, Y-Lan Boureau et Yann LeCun. Unsupervised learning of invariant feature hierarchies with applications to object recognition. Dans *Proc. Computer Vision and Pattern Recognition Conference (CVPR'07)*. IEEE Press, 2007.

[124] Marc'Aurelio Ranzato, Christopher Poultney, Sumit Chopra et Yann LeCun. Efficient learning of sparse representations with an energy-based model. Dans B. Schölkopf, J. Platt et T. Hoffman, éditeurs, *Advances in Neural Information Processing Systems 19*. MIT Press, 2007.

[125] Marc'Aurelio Ranzato, Christopher Poultney, Sumit Chopra et Yann LeCun. Efficient learning of sparse representations with an energy-based model. Dans B. Schölkopf, J. Platt et T. Hoffman, éditeurs, *Advances in Neural Information Processing Systems 19*. MIT Press, 2007.

[126] C. Rasmussen. The infinite gaussian mixture model. Dans S.A. Solla, T.K. Leen et K-R. Müller, éditeurs, *Advances in Neural Information Processing Systems 12*, 2000.

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

[128] F. Rosenblatt. *Principles of Neurodynamics*. Spartan, New York, 1962.

[129] S. Roth et M. J. Black. Fields of experts : a framework for learning image priors. Dans *IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages 860–867, 2005.

[130] S. Roweis et L. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326, décembre 2000.

[131] D. E. Rumelhart, G. E. Hinton et R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

[132] Ruslan Salakhutdinov et Geoffrey Hinton. Learning a nonlinear embedding by preserving class neighbourhood structure. Dans *Proceedings of AISTATS 2007*, San Juan, Porto Rico, 2007. Omnipress.

[133] Ruslan Salakhutdinov et Geoffrey Hinton. Semantic hashing. Dans *Proceedings of the 2007 Workshop on Information Retrieval and applications of Graphical Models (SIGIR 2007)*, Amsterdam, 2007. Elsevier.

[134] Ruslan Salakhutdinov et Geoffrey Hinton. Using deep belief nets to learn covariance kernels for gaussian processes. Dans J. C. Platt, D. Koller, Y. Singer et

S. Roweis, éditeurs, *Advances in Neural Information Processing Systems 20*. MIT Press, Cambridge, MA, 2008.

[135] Ruslan Salakhutdinov, Andriy Mnih et Geoffrey Hinton. Restricted Boltzmann machines for collaborative filtering. Dans *ICML '07 : Proceedings of the 24th international conference on Machine learning*, pages 791–798, New York, NY, USA, 2007. ACM.

[136] Ruslan Salakhutdinov et Iain Murray. On the quantitative analysis of deep belief networks. Dans *Proceedings of the International Conference on Machine Learning*, volume 25, 2008.

[137] Lawrence K. Saul, Tommi Jaakkola et Michael I. Jordan. Mean field theory for sigmoid belief networks. *Journal of Artificial Intelligence Research*, 4:61–76, 1996.

[138] Eric Saund. Dimensionality-reduction using connectionist networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(3):304–314, 1989.

[139] H. J. Scudder. Probability of error of some adaptive pattern-recognition machines. *IEEE Transactions on Information Theory*, 11:363–371, 1965.

[140] H. Sebastian Seung. Learning continuous attractors in recurrent networks. Dans M.I. Jordan, M.J. Kearns et S.A. Solla, éditeurs, *Advances in Neural Information Processing Systems 10*, pages 654–660. MIT Press, 1998.

[141] O. Shriki, H. Sompolinsky et D. D. Lee. An information maximization approach to overcomplete and recurrent representations. Dans *NIPS 13*, 2001.

[142] B. W. Silverman. *Density Estimation for Statistics and Data Analysis*. Chapman and Hall, London, 1986.

[143] J. Siöberg et L. Ljung. Overtraining, regularization, and searching for minimum in neural networks. Rapport technique, Linköping University, S-581 83 Linköping, Sweden, 1992.

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

[145] H. Steinhaus. Sur la division des corp matériels en parties. *Bulletin L'Académie Polonaise des Sciences*, 4:801–804, 1956.

[146] Ilya Sutskever et Geoffrey Hinton. Learning multilevel distributed representations for high-dimensional sequences. Dans *Proceedings of the Eleventh International Conference on Artificial Intelligence and Statistics, March 21-24, 2007, Porto-Rico*, 2007.

[147] Graham Taylor, Geoffrey Hinton et Sam Roweis. Modeling human motion using binary latent variables. Dans *Advances in Neural Information Processing Systems 20*. MIT Press, 2006.

[148] J. Tenenbaum, V. de Silva et J. C. L. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, décembre 2000.

[149] R. J. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society B*, 58:267–288, 1995.

[150] P. E. Utgoff et D. J. Stracuzzi. Many-layered learning. *Neural Computation*, 14: 2497–2539, 2002.

[151] L. J. P. van der Maaten et G. E. Hinton. Visualizing high-dimensional data using t-sne. *Journal of Machine Learning Research*, 2008.

[152] V. Vapnik. *Statistical Learning Theory*. Wiley, Lecture Notes in Economics and Mathematical Systems, volume 454, 1998.

[153] V. N. Vapnik et A. Lerner. Pattern recognition using generalized portrait method. *Automation and Remote Control*, 24, 1963.

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

[155] Pascal Vincent, Hugo Larochelle, Yoshua Bengio et Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. Rapport technique 1316, Université de Montréal, dept. IRO, 2008.

[156] Changfeng Wang, Santosh S. Venkatesh et J. Stephen Judd. Optimal stopping and effective machine complexity in learning. Dans *Advances in Neural Information Processing Systems 6*, pages 303–310. Morgan Kaufmann, 1994.

[157] Ingo Wegener. *The Complexity of Boolean Functions*. John Wiley & Sons, 1987.

[158] K. Q. Weinberger, F. Sha et L. K. Saul. Learning a kernel matrix for nonlinear dimensionality reduction. Dans *Proceedings of the Twenty First International Conference on Machine Learning (ICML-04)*, pages 839–846, Banff, Canada, 2004.

[159] Kilian Weinberger, John Blitzer et Lawrence Saul. Distance metric learning for large margin nearest neighbor classification. Dans Y. Weiss, B. Schölkopf et J. Platt, éditeurs, *Advances in Neural Information Processing Systems 18*, pages 1473–1480. MIT Press, Cambridge, MA, 2006.

[160] K.Q. Weinberger et G. Tesauro. Metric learning for kernel regression. Dans *Proc. of the 11 thInternational Conference on Artificial Intelligence and Statistics*, 2007.

[161] Max Welling et Geoffrey E. Hinton. A new learning algorithm for mean field Boltzmann machines. Dans *ICANN '02 : Proceedings of the International Conference on Artificial Neural Networks*, pages 351–357, London, UK, 2002. Springer-Verlag. ISBN 3-540-44074-7.

[162] Max Welling, Michal Rosen-Zvi et Geoffrey E. Hinton. Exponential family harmoniums with an application to information retrieval. Dans L.K. Saul, Y. Weiss et

L. Bottou, éditeurs, *Advances in Neural Information Processing Systems 17*. MIT Press, 2005.

[163] Jason Weston, Frédéric Ratle et Ronan Collobert. Deep learning via semi-supervised embedding. Dans *Proceedings of the Twenty-fifth International Conference on Machine Learning (ICML 2008)*, 2008.

[164] C. K. I. Williams et G. E. Hinton. Mean field networks that learn to discriminate temporally distorted strings. Dans *Connectionist Models : Proceedings of the 1990 Connectionist Summer School*, San Mateo, CA, 1990.

[165] Eric P. Xing, Rong Yan et Alexander G. Hauptmann. Mining associated text and images with dual-wing harmoniums. Dans *UAI*, pages 633–641. AUAI Press, 2005. ISBN 0-9749039-1-4.

[166] Andrew Yao. Separating the polynomial-time hierarchy by oracles. Dans *Proceedings of the 26th Annual IEEE Symposium on Foundations of Computer Science*, pages 1–10, 1985.

[167] David Yarowsky. Unsupervised word sense disambiguation rivaling supervised methods. Dans *33rd Annual Meeting of the ACL*, pages 189–196, Cambridge, MA, 1995.

[168] X. Zhu et Z. Ghahramani. Towards semisupervised classification with markov random fields. Rapport technique, Carnegie Mellon University, 2002.

[169] X. Zhu, Z. Ghahramani et J. Lafferty. Semi-supervised learning using Gaussian fields and harmonic functions. Dans *ICML 2003*, 2003.