Université de Montréal

Apprentissage des réseaux de neurones profonds et applications en traitement automatique de la langue naturelle

par Xavier Glorot

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

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

Novembre, 2014

Résumé

En apprentissage automatique, domaine qui consiste à utiliser des données pour apprendre une solution aux problèmes que nous voulons confier à la machine, le modèle des Réseaux de Neurones Artificiels (ANN) est un outil précieux. Il a été inventé voilà maintenant près de soixante ans, et pourtant, il est encore de nos jours le sujet d'une recherche active. Récemment, avec l'apprentissage profond, il a en effet permis d'améliorer l'état de l'art dans de nombreux champs d'applications comme la vision par ordinateur, le traitement de la parole et le traitement des langues naturelles.

La quantité toujours grandissante de données disponibles et les améliorations du matériel informatique ont permis de faciliter l'apprentissage de modèles à haute capacité comme les ANNs profonds. Cependant, des difficultés inhérentes à l'entraînement de tels modèles, comme les minima locaux, ont encore un impact important. L'apprentissage profond vise donc à trouver des solutions, en régularisant ou en facilitant l'optimisation. Le pré-entraînnement non-supervisé, ou la technique du "Dropout", en sont des exemples.

Les deux premiers travaux présentés dans cette thèse suivent cette ligne de recherche. Le premier étudie les problèmes de gradients diminuants/explosants dans les architectures profondes. Il montre que des choix simples, comme la fonction d'activation ou l'initialisation des poids du réseaux, ont une grande influence. Nous proposons l'initialisation normalisée pour faciliter l'apprentissage. Le second se focalise sur le choix de la fonction d'activation et présente le rectifieur, ou unité rectificatrice linéaire. Cette étude a été la première à mettre l'accent sur les fonctions d'activations linéaires par morceaux pour les réseaux de neurones profonds en apprentissage supervisé. Aujourd'hui, ce type de fonction d'activation est une composante essentielle des réseaux de neurones profonds.

Les deux derniers travaux présentés se concentrent sur les applications des ANNs en traitement des langues naturelles. Le premier aborde le sujet de l'adaptation de domaine pour l'analyse de sentiment, en utilisant des Auto-Encodeurs Débruitants. Celui-ci est encore l'état de l'art de nos jours. Le second traite de l'apprentissage de données multi-relationnelles avec un modèle à base d'énergie, pouvant être utilisé pour la tâche de désambiguation de sens.

Mots-clés: réseau de neurones artificiels, apprentissage profond, traitement automatique des langues naturelles, unités rectificatrices, adaptation de domaine, analyse de sentiment, données multi-relationnelles, désambiguation de sens, initialisation des poids.

Summary

Machine learning aims to leverage data in order for computers to solve problems of interest. Despite being invented close to sixty years ago, Artificial Neural Networks (ANN) remain an area of active research and a powerful tool. Their resurgence in the context of deep learning has led to dramatic improvements in various domains from computer vision and speech processing to natural language processing.

The quantity of available data and the computing power are always increasing, which is desirable to train high capacity models such as deep ANNs. However, some intrinsic learning difficulties, such as local minima, remain problematic. Deep learning aims to find solutions to these problems, either by adding some regularisation or improving optimisation. Unsupervised pre-training or Dropout are examples of such solutions.

The two first articles presented in this thesis follow this line of research. The first analyzes the problem of vanishing/exploding gradients in deep architectures. It shows that simple choices, like the activation function or the weights initialization, can have an important impact. We propose the normalized initialization scheme to improve learning. The second focuses on the activation function, where we propose the rectified linear unit. This work was the first to emphasise the use of linear by parts activation functions for deep supervised neural networks, which is now an essential component of such models.

The last two papers show some applications of ANNs to Natural Language Processing. The first focuses on the specific subject of domain adaptation in the context of sentiment analysis, using Stacked Denoising Auto-encoders. It remains state of the art to this day. The second tackles learning with multi-relational data using an energy based model which can also be applied to the task of word-sense disambiguation.

Keywords: artificial neural networks, deep learning, natural language processing, rectified linear units, domain adaptation, sentiment analysis, multi-relational data, word-sense disambiguation, weights initialization.

Table des matières

Table des figures

Liste des tableaux

Abbréviations et notations

Voici la liste des abbréviations avec la traduction des termes en anglais. Dans un soucis de clareté vis-à-vis de la littérature du domaine, nous allons utiliser les acronymes de la langue anglaise.

AI	Intelligence Artificielle (*Artificial Intelligence*)
AUC	Aire sous la courbe (*Area Under the Curve*)
ANN	Réseau de Neurones Artificiels (*Artificial Neural Network*)
DAE	Auto-Encodeur Débruitant (*Denoising Auto-Encoder*)
kNN	k Plus Proches Voisins (*k-Nearest Neighbors*)
LIF	Fonction Intègre-et-Tire avec Fuite (*Leaky Integrate-and-Fire function*)
MR	Représentation Sémantique (*Meaning Representation*)
NLP	Traitement Automatique des Langues Naturelles (*Natural Language Processing*)
POS	Partie du Discours (*Part Of Speech*)
RBF	Fonction à Base Radiale (*Radial Basis Function*)
RBM	Machine de Botlzmann Restreinte (*Restricted Boltzmann Machine*)
RMSE	Racine de l'Erreur Quadratique (*Root-Mean-Square Error*)
SRL	Identification de Rôles Syntaxiques (*Sémantique Role Labeling*)
SVD	Décomposition en valeurs singulières (*Singular value Decomposition*)
SVM	Machine à Vecteurs de Support (*Support Vector Machine*)
WSD	Désambiguation de sens (*Word-Sense Disambiguation*)

$\mathbf{x}$	Vecteur en gras, par défaut vecteur ligne
W	Matrice en majuscule
$W_{k,\bullet}$	Vecteur de la k-ième ligne de la matrice W
$^\intercal$	Transposée
$[\![n, m]\!]$	Nombres entiers ($\mathbb{N}$) compris entre n et m, inclus
$\arg\min_x f(x)$	Valeur de x minimisant $f(x)$
$P(X)$	Distribution de probabilité d'une variable aléatoire X
$P(Y\|X)$	Distribution de probabilité conditionnelle de Y sachant X
$\mathbb{E}_X[X]$	Espérance de la variable aléatoire X: $\int_x xP(X = x)dx$
$U\left]n, m\right[$	Distribution uniforme sur les nombres réels ($\mathbb{R}$) entre n et m
$\mathcal{N}(\cdot, \mu, \Sigma)$	Distribution de probabilité gaussienne de moyenne μ et de covariance Σ
XOR	Opérateur logique OU exclusif
L_1	Pénalité de régularisation suivant la norme l_1: $l_1(\mathbf{x}) = \sum_i \|x_i\|$

Remerciements

Je tiens, en tout premier lieu, à remercier Yoshua Bengio pour m'avoir donné l'opportunité de faire de la recherche dans le domaine passionnant de l'apprentissage profond, et ce, dans un cadre de travail idéal, entouré de personnes inspirantes et volontaires. Je tiens aussi à le remercier de m'avoir toujours soutenu, y compris dans les moments difficiles. J'ai beaucoup appris de sa vision de la recherche et j'en retiens un grand nombre de leçons inestimables.

Je tiens ensuite à remercier tous mes co-auteurs des articles présentés dans cette thèse: Yoshua Bengio, Jason Weston et Antoine Bordes ; avec une mention spéciale pour ce dernier à qui je dois également beaucoup. Je veux remercier aussi tous les co-auteurs des autres articles que j'ai réalisés pendant mon doctorat: Salah Rifai, Yann Dauphin, Grégoire Mesnil, Xavier Muller et Pascal Vincent ; ces travaux ont été très enrichissants.

Pour les nombreuses discussions de recherche, je tiens à remercier Guillaume Desjardins, Razvan Pascanu, Dumitru Erhan et Pascal Lamblin. N'oublions surtout pas Frédéric Bastien, pour son support technique primordial ainsi que pour sa contribution à Théano, avec James Bergstra et Olivier Breuleux, qui est la librairie que j'ai utilisée tout au long de ma thèse.

Je salue toute l'équipe du gamme, qui se reconnaîtra, pour la bonne ambiance et le support dans les moments difficiles. Je veux remercier mon père, Jacques Bayet et Christoph Wuersch pour m'avoir donné l'envie de persévérer en sciences, ainsi que mes parents et ma soeur pour avoir été derrière moi en toutes circonstances.

Enfin, car ils sont tout aussi importants, je veux remercier mes amis de (et pour) toujours: Philippe Bechet, Thierry Blanvarlet, Adrien Cascarino, Florence Célant, Benjamin Corbel, Emmanuel Gaspéri, Pierre Rivoire et Quentin Vernière. Pour me faire vivre pleinement tous les jours, je remercie Ariane Drapeau.

Je tiens à remercier Grégoire Mesnil, Emmanuelle Saulais, Yoshua Bengio, Yann Dauphin et Guillaume Desjardins pour leur aide sur la révision de ma thèse.

Enfin je remercie le DIRO, DARPA, la FESP et Yoshua Bengio pour m'avoir alloué des bourses qui ont été importantes pour la conduite de mon doctorat.

1 Apprentissage Automatique

Cette thèse est consacrée aux Réseaux de Neurones Artificiels (ANNs) profonds. Elle se concentrera plus particulièrement sur l'apprentissage de ce type de modèle et sur ses applications en Traitement des Langues Naturelles.

Ce chapitre et le suivant serviront d'introduction. Le premier présentera de manière globale l'apprentissage machine, et dans le second nous nous intéresserons en particulier aux ANNs, à l'apprentissage profond et au Traitement des Langues Naturelles.

Les deux premiers articles présentés dans cette thèse porterons sur l'apprentissage des ANNs profonds, ils étudieront certaines de ses difficultés et proposeront des solutions pour le faciliter.

Finalement, les deux derniers articles montreront des applications en Traitement des Langues Naturelles: en adaptation de domaine pour l'analyse de sentiment puis en apprentissage de données multi-relationnelles avec une application en désambiguation de sens.

1.1 Introduction à l'Apprentissage Automatique

L'Apprentissage Automatique est un sous-domaine de l'Intelligence Artificielle, il s'intéresse à la recherche de solutions pour résoudre des tâches que nous voulons confier à la machine, et ce par le biais d'un apprentissage. Par **apprentissage** nous désignons ici l'utilisation, de manière automatique, de données relatives à la tâche d'intérêt.

Il faut donc le mettre en opposition avec les approches de l'Intelligence Artificielle qui consistent à coder directement une solution. En effet, il est possible de rechercher un ensemble de règles qui permettent de résoudre la tâche, cela en

utilisant notre connaissance à priori ; alors que, dans le cadre de l'apprentissage machine, ces règles sont *apprises* automatiquement par le biais de données. Dans le premier cas, l'accent est donc mis sur la compréhension des processus nécessaires à la résolution de la tâche tandis que, dans l'autre, le sujet d'étude est l'apprentissage lui-même. Ces deux approches sont toutefois complémentaires. Il est en effet souvent difficile d'apprendre sans connaissances à priori et, inversement, la quantité toujours plus importante de données auxquelles nous avons accès contiennent des règles et des informations statistiques que nous ne savons pas formaliser directement. De ce fait, pour obtenir les meilleures performances, il est souvent nécessaires de mélanger les deux approches et de rajouter des connaissances à priori au modèle utilisé pour l'apprentissage.

Pour présenter le problème nous allons dans une première partie énumérer les différents paradigmes d'apprentissage, puis nous nous intéresseront au principal enjeu: la **généralisation**. Nous présenterons ensuite les différents types de modèles mathématiques utilisés en les illustrant avec des exemples concrets. Enfin, le contexte particulier d'apprentissage par transfert sera évoqué et plus précisément le cas de l'adaptation de domaine qui sera le sujet du chapitre 8.

1.2 Les paradigmes d'apprentissage

1.2.1 Apprentissage supervisé

Il s'agit de la forme d'apprentissage la plus intuitive. Dans ce paradigme les données sont composées d'entrées et de sorties (ou cibles). Nous connaissons donc la solution de la tâche pour un certain ensemble d'observations. On peut représenter les données sous la forme suivante: $d_i = (x_i, y_i)$ avec x l'entrée, y la cible associée et i l'indice de l'observation.

Lorsque la valeur de la sortie est une valeur discrète: $y \in [\![1, n]\!]$ représentant la classe (ou étiquette) de l'exemple x, on parle de tâche de **classification**. Il peut s'agir par exemple de la tâche de reconnaissance d'objets: $\mathbf{x}_i$ est un vecteur contenant la valeur des pixels d'une image représentant un objet en particulier et

y_i est un entier correspondant à cet objet. En pratique, on utilise un vecteur "one-hot" pour représenter la sortie. Il s'agit d'un vecteur de dimension n. La dimension correspondant à la classe de l'exemple a la valeur 1 et les autres 0. S'il n'y a que deux classes on parle de classification binaire.

Lorsque la valeur de sortie est continue, il s'agit d'une tâche de **régression**. Par exemple, imaginez que l'on ait un commentaire textuel d'un utilisateur vis-à-vis d'un produit (film, livre, objets...) et que la cible y_i est la note, par exemple sur une échelle entre 0 et 4, auquelle ce commentaire correspondrait. Cette tâche s'appelle l'analyse de sentiment [i]. Vous pouvez remarquer qu'il faut trouver un moyen de représenter le commentaire textuel. Une solution courante est d'utiliser les sacs de mots binaires: on fixe un vocabulaire $\mathcal{V}$ de n mots, $\mathbf{x}_i$ est un vecteur de n dimensions chacunes associées à un mot du vocabulaire, 1 indiquant la présence du mot dans le commentaire et 0 son absence.

1.2.2 Apprentissage non-supervisé

Parfois, les cibles ne sont pas disponibles: on a seulement $d_i = (x_i)$. Dans ce cas, un apprentissage est toujours possible. On peut en effet modéliser la distribution qui génère les entrées $P(x)$, on parle alors d'**estimation de densité de probabilité**. On peut aussi regrouper les données par similarité, il s'agit du problème de **partitionnement des données**. Enfin, il est possible d'apprendre une nouvelle représentation des données, et ce, dans deux contextes différents. Soit dans le but de compresser les données pour les manipuler ou les visualer plus facilement, il s'agit de la **réduction de dimensionnalité** et l'algorithme t-SNE, qui sera utilisé dans le chapitre 10, en est un exemple. Soit pour **extraire des caractéristiques** "intéressantes", c'est à dire à partir desquelles il est plus aisé d'effectuer un apprentissage ou de faire des prédictions. Nous nous intéresserons dans cette thèse principalement à ce dernier, notamment au chapitre 8.

i. L'analyse de sentiment n'est pas exclusivement une tâche de régression: la cible peut être, comme nous allons le voir plus tard, une valeur binaire correspondant au caractère positif (et à l'inverse négatif) du commentaire. Cette tâche n'est pas reservée aux commentaires concernant des produits, et peut concerner des sujets divers.

1.2.3 Apprentissage semi-supervisé

À la frontière de ces deux derniers paradigmes, il est aussi possible de n'avoir les cibles que pour une partie de l'ensemble des données. Il est en effet souvent coûteux d'obtenir des cibles pour un grand nombre d'observations. L'apprentissage semi-supervisé consiste à utiliser l'information présente dans les données non-étiquetées pour améliorer les performances de l'apprentissage supervisé. On peut par exemple faire un apprentissage non-supervisé sur toutes les données pour extraire des caractéristiques et effectuer l'apprentissage supervisé sur la nouvelle représentation, ou alors, on peut d'abord effectuer un apprentissage supervisé puis prédire les cibles sur les donnés non-étiquetées et, enfin, incorporer ces nouvelles données à l'entraînement supervisé.

1.2.4 Apprentissage par renforcement

Dans le cadre de l'apprentissage par renforcement, le but est d'apprendre quelles actions effectuer, étant donné un contexte, afin de maximiser une mesure de récompense qui peut, elle même, dépendre des actions passées. Des prédictions sont donc effectuées pendant l'apprentissage pour prendre des décisions et explorer l'espace des solutions. Cette approche fait intervenir le compromis exploration-exploitation, c'est à dire, soit essayer de nouvelles stratégies pour trouver une meilleure solution, quitte à faire des erreurs, soit maximiser la récompense avec la solution la plus performante actuellement apprise. C'est ce type de paradigme qui intervient pour l'apprentissage des tâches de contrôle, comme la marche, ou apprendre à jouer à des jeux de société. Il ne sera pas étudié dans cette thèse.

1.3 La généralisation

1.3.1 Le problème

Pour formaliser le problème, nous devons tout d'abord choisir une modélisation mathématique de la tâche que nous voulons automatiser. Cette dernière va définir l'espace des fonctions $\mathcal{F}$ dans lequel nous allons chercher une solution. Nous devons

aussi choisir une **fonction de coût** (ou objectif) $\mathcal{C}$ évaluant par un réel la performance d'une solution $f \in \mathcal{F}$ pour un échantillon des données. Si le coût est élevé, la performance est faible, et inversement. Si D est la variable aléatoire correspondant à l'ensemble des données relatives à la tâche en considération, alors le but ultime de l'apprentissage machine est de trouver une fonction f^* telle que:

$$f^* = \arg\min_{f \in \mathcal{F}} \mathbb{E}_D[\mathcal{C}(D, f)] \tag{1.1}$$

C'est à dire une une fonction qui minimise le **coût en généralisation**. Donc, si l'on a accès à l'ensemble de toutes les données possibles, le problème d'apprentissage automatique se réduit au problème d'optimisation ci-dessus (en addition au problème de modélisation: le choix de $\mathcal{F}$). Cela est rarement le cas: les données sont la plupart du temps limitées à un ensemble fini $\mathcal{D}$.

1.3.2 Le risque empirique et le sur-apprentissage

Envisageons de minimiser le risque empirique, c'est-à-dire la moyenne de la fonction de coût sur $\mathcal{D}$, un ensemble fini de données.

$$f' = \arg\min_{f \in \mathcal{F}} \frac{1}{|\mathcal{D}|} \sum_{d \in \mathcal{D}} [\mathcal{C}(d, f)] \tag{1.2}$$

Où $|\mathcal{D}|$ est le cardinal de $\mathcal{D}$. En élargissant l'ensemble $\mathcal{F}$ de manière appropriée, on peut remarquer qu'il est possible d'atteindre un risque empirique arbitrairement proche de la solution optimale sur $\mathcal{D}$. Généralement, une minimisation excessive du risque empirique sur $\mathcal{D}$ résulte en une augmentation du risque sur des données n'appartenant pas à l'ensemble d'entraînement, ce phénomène est appelé le **sur-apprentissage**. Or, justement, le but de l'apprentissage machine est de trouver une solution qui **généralise** sur des exemples que l'on a jamais rencontrés. Il est donc nécessaire d'isoler un sous-ensemble des données que l'on n'utilise pas pendant l'apprentissage afin d'estimer cette performance en généralisation.

1.3.3 La régularisation

Pour limiter le sur-apprentissage et donc améliorer les performances en généralisation, on peut limiter la **capacité** du modèle mathématique, conformément au *rasoir d'Ockham*. Cette procédure s'appelle la **régularisation**. On peut par exemple réduire l'espace des solutions $\mathcal{F}$ ou rajouter une **pénalité de régularisation** $\mathcal{R}$ à la fonction de coût qui dépend du modèle et/ou des données. Dans ce dernier cas, à celle-ci est associée un coefficient λ fixant le poids de la régularisation durant l'optimisation, ce dernier est un **hyper-paramètre**. Si le problème est trop contraint par la régularisation, on assiste, à l'inverse, au phénomène de **sous-apprentissage**: le modèle a trop peu de capacité pour apprendre la tâche convenablement. Il faut donc trouver un équilibre entre la capacité de l'espace des solutions $\mathcal{F}$, les contraintes apportés au problème d'optimisation par l'ensemble de données et celles apportées par la régularisation. Le manque de données nécessite de régulariser plus et inversement. On doit effectuer une **sélection de modèle** pour trouver cet équilibre.

1.3.4 La sélection de modèle

Pour faire la sélection de modèle et estimer la performance en généralisation, on divise l'ensemble $\mathcal{D}$ en trois sous-ensembles disjoints:

- L'*ensemble d'apprentissage*: $\mathcal{D}_{\text{app}}$, servant à la minimisation du risque empirique.

$$f^*_{\mathcal{F},\mathcal{R},\lambda} = \arg\min_{f\in\mathcal{F}} \frac{1}{|\mathcal{D}_{app}|} \sum_{d\in\mathcal{D}_{\text{app}}} \Big[\mathcal{C}(d,f) + \lambda\mathcal{R}(d,f)\Big] \tag{1.3}$$

- L'*ensemble de validation*: $\mathcal{D}_{\text{val}}$, servant à la sélection de modèle, c'est-à-dire au choix de l'espace des solutions $\mathcal{F}$, de la pénalité de régularisation $\mathcal{R}$ et des autres hyper-paramètres (que nous incluons ici dans λ) pour ajuster le compromis sur-apprentissage/sous-apprentissage.

$$(\mathcal{F}^*,\mathcal{R}^*,\lambda^*) = \arg\min_{(\mathcal{F},\mathcal{R},\lambda)} \frac{1}{|\mathcal{D}_{val}|} \sum_{d\in\mathcal{D}_{\text{val}}} \Big[\mathcal{C}(d, f^*_{\mathcal{F},\mathcal{R},\lambda})\Big] \tag{1.4}$$

– L'*ensemble de test*: $\mathcal{D}_{\text{tes}}$, servant à estimer la performance en généralisation.

$$\mathcal{G} = \frac{1}{|\mathcal{D}_{tes}|} \sum_{d \in \mathcal{D}_{\text{tes}}} \left[\mathcal{C}(d, f^*_{\mathcal{F}^*, \mathcal{R}^*, \lambda^*}) \right] \tag{1.5}$$

Notons que pour les équations 1.3 et 1.4, les minimisations exactes ne sont pas favorables computationnellement. En pratique, on utilise donc des minimisations approximatives.

Pour obtenir un meilleur estimé de la performance en généralisation ainsi qu'une indication de la variance de cette estimateur, on peut effectuer une validation croisée ("cross-validation" en anglais). On découpe $\mathcal{D}$ en K sous-ensembles de tailles similaires. On effectue K selections de modèle, où l'on utilise à chaque fois un sous-ensemble différent comme ensemble de test. On peut donc estimer la moyenne et la variance de l'erreur de généralisation. À l'extrême on peut fixer $K = |\mathcal{D}|$ (on ne retire qu'un exemple comme ensemble de test à chaque fois).

1.4 Les types de modèles

Il existe une grande variété de modèles en apprentissage machine. Nous allons évoquer ici quelques caractéristiques de ces derniers.

1.4.1 Paramétriques / Non-paramétriques

L'espace des fonctions dans lequel nous recherchons des solutions $\mathcal{F}$ peut soit:

– Être défini par un ensemble fini de paramètres θ, dans ce cas on parle de modèle paramétrique.

– Dans le cas contraire, si on ne peut pas représenter de cette façon toute fonction appartenant à $\mathcal{F}$, on parle de modèles non-paramétriques. La formulation de la solution dépend de l'ensemble des données.

Dans cette thèse, seulement des approches paramétriques seront explorées. Nous présenterons cependant deux méthodes non-paramétriques dans la section 1.5 à titre d'exemple: les k-plus-proches-voisins et les machines à vecteurs de support.

1.4.2 Probabilistes / Non-probabilistes

Les modèles en apprentissage machine peuvent être définis dans un cadre probabilistique ou non. Avec un modèle probabiliste, on peut utiliser la théorie des probabilités pour tirer des conclusions, obtenir des garanties et étendre ou manipuler le modèle. En contre-partie il faut respecter les contraintes mathématiques de ce cadre de travail.

1.4.3 Génératifs / Discriminatifs

Certaines approches capturent implicitement ou explicitement la distribution marginale des entrées, $P(X)$, ou jointe des entrées et sorties, $P(X, Y)$. Ces modèles sont dit génératifs: on peut générer des données synthétiques par échantillonnage. Les modèles qui capturent uniquement la distribution conditionnelle $P(Y|X)$ ou qui apprennent une fonction $f(x)$ qui relie directement l'espace d'entrée à celui des sorties, de façon à minimiser une fonction de coût: $\mathcal{C}(y, f(x))$, sont dit discriminatifs.

1.4.4 Linéaires / Non-linéaires

En classification, un modèle est un classifieur linéaire si la surface de séparation entre deux classes est définie par un hyperplan dans l'espace d'entrée. Cela est important pour déterminer le pouvoir discriminatif du modèle. Un exemple populaire: un classifieur linéaire ne peut apprendre l'opérateur XOR.

1.4.5 À fonction de coût convexe / non-convexe

Nous avons vu, avec l'équation 1.3, que l'apprentissage machine fait intervenir un problème d'optimisation. On veut minimiser une fonction de coût (plus éventuellement des termes de régularisation). Il y a deux possibilités. Soit la fonction que l'on veut minimiser est convexe, c'est à dire qu'elle satisfait la relation suivante:

$$\forall (a, b) \in \mathcal{X}, \forall t \in]0, 1[, f(t \times a + (1 - t) \times b) \leq t \times f(a) + (1 - t) \times f(b) \quad (1.6)$$

Dans ce cas, nous avons la garantie que tout minimum local est aussi un mini-

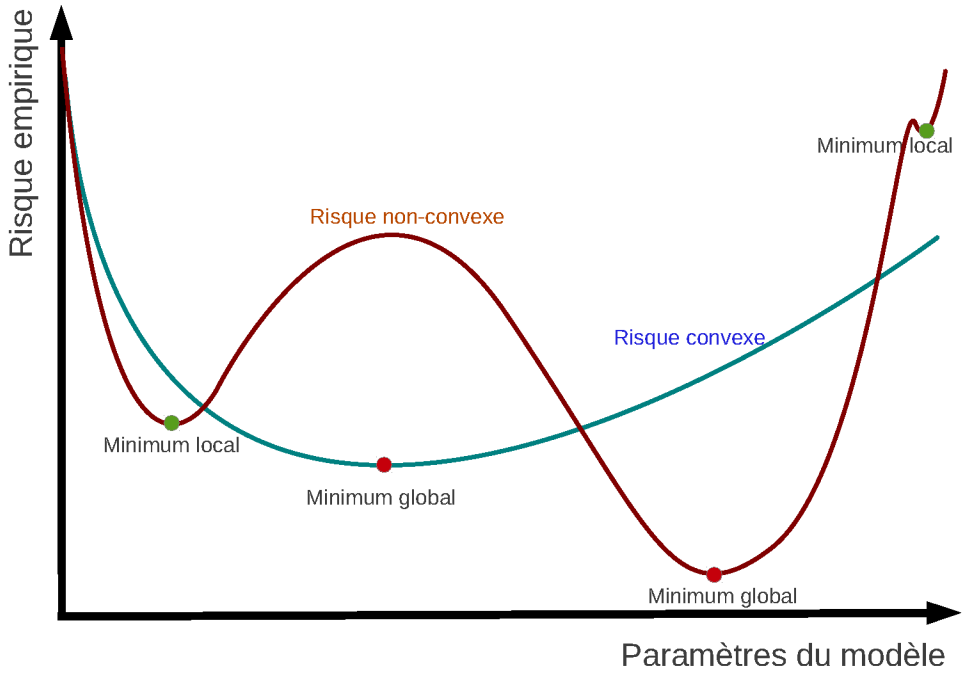

Figure 1.1 – Illustration de risques empiriques convexe et non-convexe.

mum global (et unique si la fonction est strictement convexe). Soit elle ne l'est pas et dans ce cas on a pas de garantie: on peut converger vers des solutions aux performances différentes suivant les conditions (initialisation, algorithme d'optimisation...). Cela revêtira une importance particulière dans cette thèse car les méthodes d'apprentissage profond font intervenir des problèmes d'optimisation non-convexes. Nous présenterons dans le chapitre 2 des solutions à ce problème. La figure 1.1 illustre ces deux cas de figure.

1.5 Exemples de modèles

1.5.1 Régression logistique multi-classes

La régression logistique est un modèle utilisé pour l'apprentissage supervisé en classification binaire. Nous allons nous intéresser ici à sa généralisation pour le

problème multi-classes. C'est un modèle paramétrique défini par une matrice de poids $W \in \mathbb{R}^{m \times n}$ et un vecteur de biais $\mathbf{b} \in \mathbb{R}^n$. L'entrée $\mathbf{x} \in \mathbb{R}^m$ est projetée de manière affine vers un espace de même dimension que le nombre de classes, n.

$$\mathbf{s} = \mathbf{x}W + \mathbf{b} \tag{1.7}$$

Il s'agit d'un modèle probabiliste. Afin d'obtenir des valeurs compatibles à des probabilités, une transformation de type "softmax" est appliquée.

$$\mathbf{y} = \frac{exp(\mathbf{s})}{\sum_j exp(s_j)} \tag{1.8}$$

Les valeurs de sortie sont comprises en 0 et 1 et leur somme vaut 1, elles peuvent donc représenter la probabilité conditionnelle d'appartenance à chaque classe $P(Y|X)$.

On trouve les paramètres $(W, \mathbf{b})$ en minimisant la log-vraissemblance négative d'appartenance à la bonne classe sur l'ensemble d'apprentissage. Il s'agit d'un modèle discriminatif. C'est un classifieur linéaire et son optimisation est convexe.

La couche de sortie des réseaux de neurones artificiels, que nous introduirons dans le chapitre suivant, est souvent une régression logistique multi-classe.

1.5.2 K-plus-proches-voisins (kNN)

Cet algorithme simple et intuitif peut-être utilisé pour la classification et la régression.

Il est défini par un opérateur mathématique de distance dans l'espace d'entrée (par exemple la distance euclidienne) et un entier strictement positif k (inférieur ou égal au nombre d'exemples dans l'ensemble d'apprentissage). Pour effectuer une prédiction sur un vecteur $\mathbf{x}$, on trouve les k exemples les plus proches de $\mathbf{x}$ dans l'ensemble d'entraînement dans le sens de la mesure de distance. La prédiction est la classe à vote majoritaire dans le cas de la classification, où la moyenne des sorties pour la régression, correspondant à ces k-plus-proches-voisins. Il s'agit d'un modèle non-paramétrique, discriminatif, non-linéaire, non-probabiliste. Il n'y a pas d'apprentissage en tant que tel (mis-à-part le stockage des données d'apprentissage).

On peut remarquer que k est un hyper-paramètre de régularisation, plus il est élevé plus la régularisation est forte. À l'extrême, lorsque k correspond au nombre d'exemples dans l'ensemble d'apprentissage, l'algorithme prédit toujours la classe majoritaire.

1.5.3 Fenêtres de Parzen

Afin de donner un exemple de modèle génératif, nous allons présenter les fenêtres de Parzen pour l'estimation de densité. Il s'agit d'un modèle non-paramétrique comme les kNN. La distribution générant les entrées est estimée en convoluant la distribution discrète en peigne de dirac de l'ensemble d'apprentissage avec un noyau continu.

Le noyau peut être choisi de manière arbitraire du moment qu'il respecte les contraintes d'une densité de probabilité:

$$\int_{\mathbf{x}} K(\mathbf{x}) d\mathbf{x} = 1 \tag{1.9}$$

$$\forall \mathbf{x}; K(\mathbf{x}) \geq 0 \tag{1.10}$$

Cela permet de donner une masse de probabilité à un volume défini par le noyau et ce autour de chaque exemple d'entraînement. Il est courant d'utiliser une gaussienne multivariée avec comme moyenne le vecteur nul et une variance isotropique.

$$K(\mathbf{x}) = \mathcal{N}(\mathbf{x}, 0, \sigma^2 \mathbf{I}) = \frac{1}{\sqrt{(2\pi\sigma^2)^d}} exp\left[- \frac{\mathbf{x}\mathbf{x}^\intercal}{2\sigma^2} \right] \tag{1.11}$$

Pour générer un exemple de cette distribution il suffit de choisir uniformément un exemple d'entraînement puis d'échantillonner à partir de la gaussienne centrée en cet exemple.

1.5.4 Machine à vecteurs de support

Transformations non-linéaires

L'utilisation de modèles linéaires limite le pouvoir discriminatif et donc le type de fonction que l'on peut apprendre. Une astuce pour ajouter de manière explicite de la **capacité** au modèle, est d'effectuer une transformation non-linéaire adaptée au type de données d'entrées. Par exemple, on peut rajouter des composantes polynomiales d'ordre supérieur dans le vecteur d'entrée. Dans le cas d'un espace de dimension 2 on a:

$$\Phi(\mathbf{x}) = (x_1, x_2, x_1^2, x_2^2, x_1 x_2) \tag{1.12}$$

Avec cette transformation, il est maintenant possible d'apprendre la fonction XOR qui est pourtant non-linéaire dans l'espace d'origine.

Espace dual

De nombreux modèles linéaires paramétriques peuvent être redéfinis dans un espace dit "dual". Les prédictions y sont calculées par une combinaison linéaire du produit scalaire des entrées avec les données d'entraînement, on appelle cela la fonction noyau. Avec une transformation non-linéaire des données $\Phi(\mathbf{x})$, la fonction noyau est donc définie comme suit:

$$K(\mathbf{x}, \mathbf{x}') = \Phi(\mathbf{x})\Phi(\mathbf{x}')^\intercal \tag{1.13}$$

On peut définir directement des fonctions noyaux et, sous certaines conditions (que la matrice de Gram de la fonction noyau soit définie semi-positive (Cristianini and Shawe-Taylor, 2000)), il peut être montré qu'il existe toujours une fonction $\Phi(\mathbf{x})$ satisfaisant l'équation 1.13. En pratique il n'est donc pas nécessaire de calculer explicitement $\Phi(\mathbf{x})$ pour évaluer la fonction noyau. Cette dernière peut même correspondre à une projection des données dans un espace de dimension infinie (c'est le cas du noyau Gaussien).

Maximisation de la marge

Dans le cas de données linéairement séparables, il existe généralement une infinité d'hyperplans permettant de classifier parfaitement les exemples. Pour régulariser le modèle, et donc réduire l'espace des solutions, on peut rajouter des contraintes. Une contrainte populaire consiste à maximiser la marge, c'est à dire maximiser la distance minimale entre la surface de séparation et n'importe lequel des exemples d'entraînement. Avec cette contrainte, l'hyperplan optimal ne dépend que d'un nombre réduit d'exemples d'entraînements (ceux les plus proches de la frontière de décision), on appelle ces derniers les vecteurs de support.

Machine à vecteurs de support

Les machines à vecteurs de support combinent ces différents aspects. Il s'agit de maximiser la marge d'un modèle linéaire avec ou sans transformation non-linéaire des données. On peut, pour cela, tirer partie de l'expression dans l'espace dual pour avoir une solution éparse et de la fonction noyau pour ne pas devoir calculer explicitement la transformation non-linéaire. Le problème d'optimisation a l'avantage d'être convexe et le modèle peut être non-linéaire. En général il s'agit d'un modèle non-paramétrique (la fonction noyau pouvant correspondre à une projection dans un espace de dimension infinie). Toutefois, avec une fonction noyau linéaire on peut écrire le modèle de manière paramétrique dans l'espace primal. Le principal inconvénient de ce type de modèle est que la complexité algorithmique du problème d'optimisation est quadratique avec le nombre d'exemples dans le cas général (avec un noyau gaussien par exemple). Cependant, toujours dans le cas linéaire, il est possible d'obtenir un algorithme d'apprentissage avec une complexité linéaire.

1.6 Un cas particulier en apprentissage: l'adaptation de domaine

1.6.1 Les différents types de transferts

Pour le moment, nous avons considéré uniquement le cas où les données sur lesquelles le système doit être déployé sont tirées de la même distribution que les

données d'entraînement. Dans ce paradigme, lorsque les distributions changent, il est nécessaire de re-collecter des données et de ré-entraîner le modèle. Cela n'est pas satisfaisant. En effet, il serait moins coûteux de pouvoir transférer le savoir appris sur l'ancienne distribution. Ce concept s'appelle le transfert de l'apprentissage. Une revue de littérature sur ce sujet a été proposée par Pan and Yang (2010).

Définissons un espace de représentation des données $\mathcal{X}$, et un espace des sorties $\mathcal{Y}$. Un domaine est défini par sa densité de probabilité: $P(X)$, et une tâche par la densité de probabilité conditionnelle: $P(Y|X)$. L'objectif du transfert de l'apprentissage est de généraliser sur une tâche et un domaine *cible* à partir de données collectées sur un domaine et une tâche *source*. Les principaux cas possibles de transfert sont:

- Le *transfert inductif*: les tâches source et cible sont différentes, les domaines source et cible pouvant être ou non différents. Des données supervisées pour la tâche cible sont disponibles. Il s'agit, soit d'utiliser des données sources non-supervisées ("self-taught learning"), soit supervisées ("multi-task learning") ; pour pouvoir améliorer les performances sur les données cibles.
- Le *transfert non-supervisé*: les domaines source et cible sont différents mais "reliés". L'objectif est de résoudre une tâche non-supervisée dans le domaine cible telle que la réduction de dimensionnalité.
- Le *transfert transductif*: celui-ci, et plus particulièrement l'adaptation de domaine, sera le sujet du chapitre 8. Les tâches source et cible sont identiques mais les domaines eux varient. Il existe alors deux cas particuliers. Soit les deux domaines partagent le même support, dans ce cas ce problème revient à celui du biais de sélection et, comme nous le verrons dans la section suivante, il existe une solution optimale. Soit les deux domaines partagent seulement de manière limité leurs supports. Dans ce dernier cas il s'agit de l'adaptation de domaine et la solution privilégiée est alors l'apprentissage d'une représentation partagée.

Nous allons donc maintenant étudier plus en détail le cas du transfert transductif.

Appelons P_S et P_C les densités de probabilités associées respectivement aux domaines source et cible. D'après la définition, si les tâches source et cible sont identiques, nous avons donc la relation suivante:

$$P_S(Y|X) = P_C(Y|X) \tag{1.14}$$

Cette hypothèse est forte mais peut être utilisée en pratique. Reprenons le cas de l'analyse de sentiment évoquée dans la section 1.2.1. Les domaines sont les différents types de produits. Par exemple, on veut apprendre sur des commentaires à propos de livres et utiliser le classifieur obtenu sur des commentaires de films. La polarité subjective des mots est généralement la même d'un domaine à un autre, allant dans le sens de l'équation 1.14.

Essayons maintenant d'obtenir des garanties sur la performance en transfert. Comme nous l'avons mentionné précédemment, il y a deux cas possibles.

1.6.2 Support partagé: biais de sélection

Comme à la section 1.3, écrivons la procédure d'optimisation de l'apprentissage automatique sur le domaine source:

$$f_S^* = \arg\min_{f \in \mathcal{F}} \mathbb{E}_S[\mathcal{C}((\mathbf{x}, y), f)] \tag{1.15}$$

Où $\mathbb{E}_S$ est défini comme suit:

$$f_S^* = \arg\min_{f \in \mathcal{F}} \int_{(\mathbf{x},y)\in(\mathcal{X},\mathcal{Y})} P_S(\mathbf{x}, y)\mathcal{C}((\mathbf{x}, y), f)d\mathbf{x}dy \tag{1.16}$$

Idéalement, si on avait directement accès à des données supervisées du domaine cible, la solution serait:

$$f_C^* = \arg\min_{f \in \mathcal{F}} \int_{(\mathbf{x},y)\in(\mathcal{X},\mathcal{Y})} P_C(\mathbf{x}, y)\mathcal{C}((\mathbf{x}, y), f)d\mathbf{x}dy \tag{1.17}$$

La définition du partage de support entre le domaine source et cible est (Shimodaira, 2000):

$$\forall \mathbf{x} \in \mathcal{X}, P_S(\mathbf{x}) \neq 0 \Leftrightarrow P_C(\mathbf{x}) \neq 0 \tag{1.18}$$

Cela veut dire que tous les vecteurs d'entrées ayant une probabilité non-nulle d'apparition dans un domaine ont une probabilité non-nulle d'apparition dans l'autre [i]. Notons $\mathcal{X}'$, l'espace des entrées tel que:

$$\forall \mathbf{x} \in \mathcal{X}, P_S(\mathbf{x}) \neq 0 \Leftrightarrow \mathbf{x} \in \mathcal{X}' \tag{1.19}$$

On peut alors écrire:

$$f_C^* = \arg\min_{f \in \mathcal{F}} \int_{(\mathbf{x},y)\in(\mathcal{X}',\mathcal{Y})} \frac{P_S(\mathbf{x},y)}{P_S(\mathbf{x},y)} P_C(\mathbf{x},y)\mathcal{C}((\mathbf{x},y),f)d\mathbf{x}dy \tag{1.20}$$

$$f_C^* = \arg\min_{f \in \mathcal{F}} \int_{(\mathbf{x},y)\in(\mathcal{X}',\mathcal{Y})} \frac{P_C(\mathbf{x},y)}{P_S(\mathbf{x},y)} P_S(\mathbf{x},y)\mathcal{C}((\mathbf{x},y),f)d\mathbf{x}dy \tag{1.21}$$

$$f_C^* = \arg\min_{f \in \mathcal{F}} \int_{(\mathbf{x},y)\in(\mathcal{X}',\mathcal{Y})} \frac{P_C(\mathbf{x})P_C(y|\mathbf{x})}{P_S(\mathbf{x})P_S(y|\mathbf{x})} P_S(\mathbf{x},y)\mathcal{C}((\mathbf{x},y),f)d\mathbf{x}dy \tag{1.22}$$

En utilisant la relation 1.14, on obtient:

$$f_C^* = \arg\min_{f \in \mathcal{F}} \int_{(\mathbf{x},y)\in(\mathcal{X}',\mathcal{Y})} \frac{P_C(\mathbf{x})}{P_S(\mathbf{x})} P_S(\mathbf{x},y)\mathcal{C}((\mathbf{x},y),f)d\mathbf{x}dy \tag{1.23}$$

En comparant les équations 1.16 et 1.23, on remarque qu'il est possible d'obtenir un transfert optimal, c'est-à-dire une performance égale à celle obtenue si on utilisait des données supervisées dans le domaine cible, en pondérant les exemples du domaine source suivant $\frac{P_C(\mathbf{x})}{P_S(\mathbf{x})}$ pendant l'apprentissage. C'est-à-dire:

$$\tilde{f}_C^* = \arg\min_{f \in \mathcal{F}} \frac{1}{|\mathcal{D}_S|} \sum_{(\mathbf{x},y)\in\mathcal{D}_S} \frac{P_C(\mathbf{x})}{P_S(\mathbf{x})} \mathcal{C}((\mathbf{x},y),f)d\mathbf{x}dy \tag{1.24}$$

Le problème peut donc se résoudre à estimer ce ratio de densités de probabilités.

i. Il est toutefois suffisant que $\forall \mathbf{x} \in \mathcal{X}, P_C(\mathbf{x}) \neq 0 \Rightarrow P_S(\mathbf{x}) \neq 0$ pour que les relations suivantes tiennes.

Nous n'allons pas explorer plus en avant cette voie, mais plutôt considérer le cas où le support est partiellement partagé.

1.6.3 Support partiellement partagé: adaptation de domaine

Reprenons encore l'exemple de la classification de sentiment, section 1.2.1. On peut remarquer que certains mots sont spécifiques à un domaine et apparaissent rarement dans les autres. Par exemple, "handy", pour des produits électroniques, a peu de chance d'apparaître dans des commentaires sur des films. L'hypothèse du support partagé n'est donc pas satisfaisante. Similairement, si la quantité de données disponibles est trop petite, il n'est pas possible d'estimer avec précision le ratio de densité $\frac{P_C(\mathbf{x})}{P_S(\mathbf{x})}$. Dans ces conditions, un transfert optimal n'est plus atteignable.

Une solution possible consiste à projeter les données dans un espace partagé par les deux domaines, avant d'entraîner le classifieur. Nous verrons au chapitre 8 que les méthodes non-supervisées utilisées pour l'apprentissage des réseaux de neurones profonds, qui seront présentées au chapitre suivant, sont particulièrement bien adaptées à ce problème et qu'elles sont d'ailleurs l'état de l'art dans le contexte de la classification de sentiment en traîtement des langues naturelles.

2 Apprentissage Profond

Après avoir présenté de manière globale l'apprentissage automatique, nous allons maintenant nous intéresser plus particulièrement à l'apprentissage profond. Pour illustrer la discussion, nous allons d'abord présenter en détail le type de modèle le plus utilisé dans ce contexte: les Réseaux de Neurones Artificiels (ANNs) ; ainsi qu'un algorithme pour les entraîner: la rétro-propagation du gradient. Avec ces exemples concrets en main, nous présenterons les avantages et les problèmes inhérents à l'apprentissage d'architectures profondes ; puis, nous étudierons les principales stratégies pour contourner ces problèmes. Finalement, nous traiterons de l'utilisation de tels modèles dans le cadre du traitement des langues naturelles.

2.1 Réseaux de Neurones Artificiels (ANNs)

S'inspirant originellement des observations en neurosciences vis-à-vis du fonctionnement du cerveau, ce type de modèle existe depuis bientôt soixante ans, avec l'invention du Perceptron (Rosenblatt, 1958). Sa formulation initiale comportait d'importantes limitations (Minsky and Papert, 1969), comme l'impossibilité de résoudre des problèmes non-linéaires. Il fut donc mis de côté jusqu'à l'invention des réseaux de neurones artificiels multi-couches et de la rétro-propagation du gradient (Rumelhart et al., 1986). Ces derniers étant encore sensibles au problème des minima locaux, ils furent une nouvelle fois relégués au second plan[i] avec la découverte des SVMs (Cortes and Vapnik, 1995), qui permettent de résoudre des problèmes non-linéaires, et ce, en utilisant une optimisation convexe. Avec l'apprentissage profond (Hinton et al., 2006; Ranzato et al., 2007; Bengio et al., 2007), les Réseaux de Neurones Artificiels connaissent de nouveau un essort. Nous allons présenter en détail le modèle, et ce, en partant des observations en neurosciences,

i. À l'exception des réseaux convolutionnels

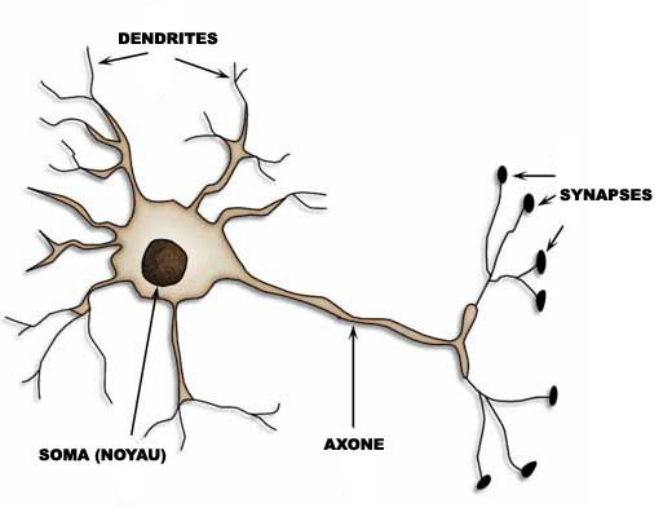

Figure 2.1 – Schéma d'un neurone biologique, *source:* http://goo.gl/Ah4cB7.

pour mettre en exergue les importantes simplifications des réseaux de neurones utilisés en apprentissage machine vis-à-vis de leurs homologues biologiques.

2.1.1 Le neurone biologique

L'unité de calcul principal du système nerveux, le neurone [i], est constitué de trois parties: l'arbre dentritique, le soma et l'axone ; une illustration est présentée à la figure 2.1. Au repos, l'intérieur du neurone est polarisé négativement par rapport à l'extérieur. Cela est dû à un ensemble de pompes contrôlant les concentrations relatives de différents ions, entre l'intérieur et l'extérieur du neurone. Les synapses sont les points de jonctions entre l'axone et les dendrites de deux neurones, un message y est transmis sous la forme de composés chimiques: les neuro-transmetteurs. Lorsque ceux-ci se lient aux récepteurs post-synaptiques, des canaux ioniques s'ouvrent ou se ferment sur la membrane et des courants post-synaptiques apparaissent. Ces courants sont ensuite intégrés de manière complexe et non-linéaire (Koch et al., 1983) dans l'arbre dendritique. Le potentiel du soma va changer et, si il dépasse un certain seuil, un potentiel d'action sera initié et propagé à travers l'axone, ce

i. Il est important de noter que d'autres types de cellules peuvent avoir une influence sur le traitement de l'information dans le cerveau: les cellules gliales et notamment les astrocytes. Il s'agit cependant d'un domaine de recherche récent et encore peu étudié (Araque and Navarrete, 2010).

qui libérera des neuro-transmetteurs aux synapses auxquelles il est relié. Les mécanismes de création et de propagation du potentiel d'action sont expliqués par le modèle d'Hodgkin-Huxley (Hodgkin and Huxley, 1952) basé sur un ensemble d'équations différentielles non-linéaires.

2.1.2 Réseaux de Neurones Artificiels

Le fonctionnement des neurones biologiques est très complexe, une modélisation fidèle requiert de monopoliser des ressources computationnelles importantes. Toutefois, il est possible de définir un modèle efficace inspiré du fonctionnement des neurones biologiques en admettant les (fausses) hypothèses suivantes:

- L'intégration des courants post-synaptiques est une transformation affine.
- L'information de phase des potentiels d'actions n'influe pas sur l'intégration des courants post-synaptiques.
- La fonction de réponse du neurone peut être assimilée à une fonction de notre choix $g(x)$.
- Tous les neurones peuvent être formalisés de la même manière.

La valeur de sortie d'un neurone en fonction de ses entrées ($\mathbf{x} = [x_1, ..., x_n]$) devient donc ; avec $\mathbf{w} = [w_1, ..., w_n]$ le vecteur de poids associés à chacune des entrées et b le biais, ou seuil, du neurone artificiel:

$$z(\mathbf{x}) = g\left(\sum_{i=1}^{n} \left(w_i \times x_i\right) + b\right) \tag{2.1}$$

Les fonctions d'activations g généralement utilisées sont: la sigmoïde et la tangente hyperbolique (LeCun et al., 1998) ; définies par les équations suivantes:

$$\text{sigmoid}(x) = \frac{1}{1 + e^{-x}} \tag{2.2}$$

$$\tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1} = 2 \times \text{sigmoid}(2x) - 1 \tag{2.3}$$

Le choix de la fonction d'activation est l'objet de plusieurs travaux réalisés dans le cadre de cette thèse et sera discuté aux chapitres 4 et 6.

Il est aussi possible de remplacer g par une fonction à base radiale, c'est à dire une fonction qui ne dépend que de la distance entre deux vecteurs, en l'occurence ici $\mathbf{x}$ et $\mathbf{w}$. La transformation affine est remplacée par cet opérateur de distance. Cela force une représentation locale: le neurone est activé seulement lorsque son entrée est au voisinage de $\mathbf{w}$. À l'inverse la transformation affine découpe l'espace en deux suivant l'hyperplan défini par l'équation: $\mathbf{x} \cdot \mathbf{w}^\intercal + b = 0$. Avec la fonction d'activation sigmoïde le neurone tend à être activé sur tout le demi-espace vérifiant $\mathbf{x} \cdot \mathbf{w}^\intercal + b < 0$ et à être désactivé sur tout le demi-espace vérifiant $\mathbf{x} \cdot \mathbf{w}^\intercal + b > 0$. On obtient une représentation *distribuée*.

Nous allons nous restreindre aux réseaux qui n'admettent pas de boucle dans leur graphe de calcul. Dans le cas contraire le réseau est dit récurrent et le modèle est dynamique. Ce type de modèle ne sera pas étudié dans cette thèse.

Rajoutons les contraintes suivantes:

- Les réseaux de neurones sont arrangés en couches.
- Les neurones d'une couche admettent comme entrées uniquement les neurones de la couche précédente.

Les couches, autres que celles de sortie et d'entrée du réseau, sont appelées couches cachées. Considérons $\mathbf{z}^0$ le vecteur d'entrée, $\mathbf{z}^i$ le vecteur des activations à la couche i, $\mathbf{s}^i$ le vecteur des valeurs avant la fonction d'activation g ainsi que W^i et $\mathbf{b}^i$ respectivement la matrice de poids et le vecteur de biais correspondants aux neurones artificiels $\mathbf{z}^i$. On obtient le modèle représenté à la figure 2.2 et on a:

$$\mathbf{s}^i = \mathbf{z}^{i-1} W^i + \mathbf{b}^i \tag{2.4}$$

$$\mathbf{z}^i = g(\mathbf{s}^i) \tag{2.5}$$

Enfin, on rajoute une couche de sortie, à laquelle nous voulons faire correspondre nos cibles. À cette dernière est associée une fonction de coût que nous voulons minimiser. Par exemple, il peut s'agir d'une couche similaire à la régression logistique, présentée à la section 1.5.1.

Malgré les simplifications admises vis-à-vis du fonctionnement des réseaux de neurones biologiques, les Réseaux de Neurones Artificiels ont une grande expressivité: avec une couche cachée, il s'agit d'un approximateur universel (Hornik et al.,

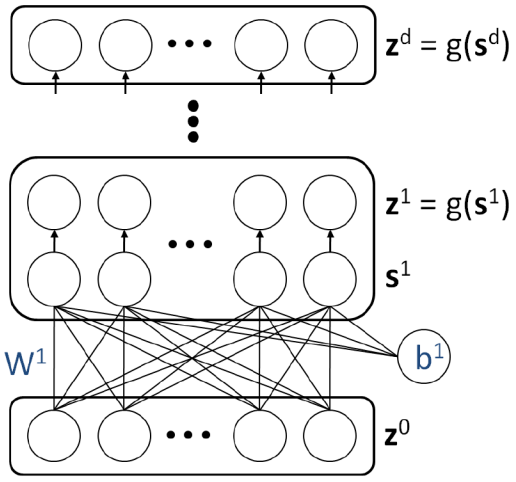

Figure 2.2 – Schéma de principe d'un Réseau de Neurones Artificiels.

1989) à condition que le nombre de neurones cachés soit assez grand. Cependant, l'optimisation d'un tel modèle n'est plus convexe et est sujet au problème des minima locaux, la régularisation prenant alors un rôle d'autant plus important.

2.1.3 Descente de gradient

Maintenant que nous avons défini l'architecture du modèle, comment pouvons-nous l'entraîner ? En neurosciences, à l'état actuel de nos connaissances, l'apprentissage s'effectue par le biais de règles d'apprentissages locales, comme par exemple avec l'apprentissage Hebbien (Hebb, 1949). Cependant, dans le cadre de l'apprentissage machine et des ANNs, ce type de règles n'eut qu'un succès limité. Il fallut attendre l'utilisation de règles globales, comme la descente de gradient que nous allons présenter dans cette section, pour obtenir de meilleures performances. Même s'il a été montré que sous certaines conditions la descente de gradient est équivalente à un apprentissage Hebbien (Xie and Seung, 2003), il est important de noter qu'il n'y a aucune évidence que cette dernière soit utilisée dans les réseaux de neurones biologiques.

Le principe de cette règle globale est le suivant: si la fonction d'activation et la fonction de coût sont dérivables, il est possible de définir le gradient de la fonction de coût par rapport aux paramètres du modèle. Ce gradient représente la direction locale de plus forte pente. Si on soustrait aux paramètres cette direction pondérée

par un scalaire: le *taux d'apprentissage*, et si ce dernier est assez petit pour ne pas diverger, on minimise alors itérativement la fonction objectif vers un minimum local.

La structure des réseaux de neurones permet de calculer ces gradients de manière efficace grâce à la méthode de rétro-propagation du gradient (Rumelhart et al., 1986). Celle-ci fonctionne comme suit:

Les paramètres du réseau sont tout d'abord initialisés: les biais à 0 et les poids à des valeurs aléatoires et faibles, l'objectif étant de propager un signal sans toutefois saturer les neurones. Une heuristique populaire consiste à initialiser les poids de la manière suivante:

$$w^i \sim U\left[-\frac{1}{\sqrt{n^{i-1}}}; \frac{1}{\sqrt{n^{i-1}}}\right] \tag{2.6}$$

Avec n^i le nombre de neurones de la couche i. Une meilleure stratégie concernant l'initialisation des poids sera le sujet de la section 4, travail réalisé dans le cadre de cette thèse.

À chaque itération:

- On effectue une propagation du signal vers le haut pour calculer la valeur des activations des neurones artificiels: $\mathbf{s}^i$, $\mathbf{z}^i$ et la valeur de la fonction objectif. Celles-ci sont gardées en mémoire.
- On effectue ensuite une rétro-propagation du gradient de la fonction de coût vers l'arrière en utilisant la règle de dérivation en chaine:

$$\frac{\partial Cost}{\partial s_k^i} = g'(s_k^i) W_{k,\bullet}^{i+1} \frac{\partial Cost}{\partial \mathbf{s}^{i+1}}^\intercal \tag{2.7}$$

  L'équation 2.7 montre le gradient rétro-propagé à la couche i qui correspond à la valeur du gradient pour le biais: b_k^i. On peut obtenir le gradient pour les poids de la couche i en utilisant la formule suivante:

$$\frac{\partial Cost}{\partial w_{l,k}^i} = z_l^{i-1} \frac{\partial Cost}{\partial s_k^i} \tag{2.8}$$

- Pour chaque paramètre θ du modèle, on soustrait le gradient associé pondéré

par le taux d'apprentissage α:

$$\theta \leftarrow \theta - \alpha \frac{\partial Cost}{\partial \theta} \tag{2.9}$$

Ces étapes peuvent être réalisées sur l'ensemble des données, ou sur un exemple (ou un sous-ensemble d'exemples) aléatoire, on parle alors de descente de gradient stochastique. Cette dernière est préférée, elle permet d'éviter de reproduire des calculs inutiles lorsque des exemples sont redondants et, surtout, elle agit en pratique comme un régularisateur grâce au bruit apporté par le processus aléatoire (Bourrely, 1989; Bottou and Gallinari, 1991).

Le taux d'apprentissage et le nombre d'itérations sont des hyper-paramètres que l'on doit également sélectionner avec l'erreur de validation. On arrête la procédure d'optimisation lorsque celle-ci remonte ou se stabilise, ceci s'appelle l'*arrêt précoce*. Les deux peuvent être vus comme des régularisateurs.

2.2 Pourquoi l'apprentissage profond

L'apprentissage profond consiste à entraîner des modèles composés de plusieurs transformations non-linéaires successives. Un ANN avec plusieurs couches cachées en est donc un exemple. Pourtant, nous avons vu qu'un ANN avec une couche cachée est déjà un approximateur universel, quel est alors l'avantage d'augmenter le nombre de couches ? C'est ce que nous allons voir dans cette section.

Il y a plusieurs arguments en faveur de tels réseaux. Premièrement, des résultats mathématiques montrent que certaines classes de fonctions, représentables de manière compacte avec un réseau de profondeur d, nécessiteraient un nombre exponentiel de paramètres avec un réseau de profondeur $d-1$ (Håstad and Goldmann, 1991; Bengio and Delalleau, 2011; **?**). Ensuite, on peut noter que l'architecture des réseaux profonds permet la réutilisation de paramètres ou de caractéristiques extraites, propriété désirable pour la modélisation de fonctions complexes. Et finalement, le cerveau humain, plus particulièrement le cortex visuel, est construit comme une architecture profonde, avec plusieurs sous-régions fonctionnelles (V1,V2,MT) arrangées en niveaux d'abstraction de plus en plus élevés.

Une des motivations est donc d'apprendre une hiérarchie de caractéristiques avec ces niveaux d'abstraction croissants. Par exemple, pour une tâche de reconnaissance d'objet, on aimerait que la première couche détecte des parties de bas niveau, telles que les bords ; que la suivante détecte des combinaisons de ses parties ; et ainsi de suite, pour représenter des concepts de plus en plus évolués. Ceci est observé expérimentalement avec les méthodes utilisées en apprentissage profond (Lee et al., 2009; Erhan et al., 2009; Zeiler and Fergus, 2014).

En plus d'extraire des caractéristiques de plus en plus abstraites, une autre propriété désirable serait de les démêler, afin que les différent facteurs qui expliquent les variations des données soient séparés. Ceci serait très interessant pour l'apprentissage multi-tâches, ainsi que pour l'adaptation de domaine, comme nous allons le voir au chapitre 8. Les travaux allant dans cette direction de recherche sont encore rares, mais il s'agit d'une voie prometteuse (Goodfellow et al., 2009; Rifai et al., 2012; Reed et al., 2014; Desjardins et al., 2012; Ozair and Bengio, 2014).

Avant 2006, l'entraînement des réseaux de neurones était limité aux réseaux avec une seule, voir deux, couches cachées, à l'exception des réseaux convolutionnels (LeCun and Bengio, 1994) ; les réseaux plus profonds donnant de moins bonnes performances. Ceci était dû aux raisons suivantes:

- Les performances computationnelles insuffisantes du matériel informatique.
- La quantité trop petite de données d'entraînement disponible.
- Les difficultés inhérentes à l'apprentissage profond.

Concernant les deux premiers points, la situation ne cesse de s'améliorer. Qu'en est-il du dernier ? Les principales difficultés sont les problèmes des minima locaux et celui des gradients diminuant/augmentant ("vanishing/exploding") (Bengio et al., 1994). Ce dernier concerne les réseaux récurrents, c'est-à-dire des réseaux admettant des boucles dans leurs connections. Une façon de les entraîner consiste à rétro-propager les gradients à travers le temps. Il peut être montré que la norme de ces gradients tend vers 0 ou vers l'infini à une vitesse exponentielle. Or, un réseau récurrent peut être vu comme un réseau profond avec une infinité de couches. Cela suggère que la rétro-propagation des erreurs est rendue difficile par la profondeur. Nous reviendrons sur ce point plus en détail à la section 4. Nous allons maintenant présenter plusieurs solutions qui ont permis de surpasser ces difficultés.

2.3 Les solutions

Les solutions utilisées pour apprendre de telles architectures sont variées, nous allons ici en mentionner quelques-unes notables.

Comme nous l'avons déjà mentionné, les réseaux convolutionnels (LeCun and Bengio, 1994) ont été les premières architectures profondes entrainées efficacement. Ils sont inspirés d'observations du traitement de l'information dans le cortex visuel où les neurones sont organisés en colonnes corticales. Le réseau n'admet que des connections locales et, de plus, les paramètres de ces connections sont partagés au sein d'une couche donnée. Cela réduit drastiquement le nombre de paramètres du modèle. Mathématiquement, la multiplication matricielle est remplacée par une convolution: les caractéristiques sont extraites sur tout le champ visuel plutôt qu'être spécifique à une région de ce dernier. En addition, une couche d'aggrégation, ou "pooling", est souvent utilisée pour réduire la taille des couches cachées (dans le cas convolutionnel appelées cartes de caractéristiques), cela permet aussi d'obtenir une invariance à des petites translations. La réduction du nombre de paramètres et le choix de l'architecture, appropriée au type de données à traiter, permet de faciliter l'apprentissage ; même dans le cas d'architectures profondes. Ce type de réseaux est très performant pour les tâches de traitement d'images (Szegedy et al., 2014), et est également utilisé pour le traitement de vidéo (Taylor et al., 2010), d'audio (Lee et al., 2009) et de texte (Collobert et al., 2011). Plus généralement, il est possible de l'appliquer pour toutes les données admettant une composante séquentielle et/ou temporelle et/ou spatiale.

Une autre voie de recherche pour faciliter l'entraînement des réseaux profonds consiste à améliorer l'algorithme d'optimisation, plutôt que d'utiliser la simple descente de gradient stochastique. Les méthodes de second ordre (Nocedal and Wright, 2006) utilisent l'information de courbure de la fonction de coût pour sélectionner automatiquement le taux d'apprentissage optimal (si la fonction objective est quadratique) pour chaque paramètre. Cela nécessite d'estimer la Hessienne de la fonction de coût, ce qui est coûteux computationnellement. De plus, cette dernière doit être définie positive, en d'autres termes, la fonction de coût doit être localement convexe, ce qui n'est pas toujours le cas. Un exemple d'application de cette méthode pour les réseaux de neurones profonds est l'algorithme "Hessian-free" (Martens, 2010). La méthode des gradients naturels, elle, utilise la géométrie

de la variété définie par les paramètres du modèle, cela correspond à remplacer la matrice Hessienne des méthodes de second ordre par la matrice d'information de Fisher (Amari, 1997; Pascanu and Bengio, 2013). Il est également possible d'utiliser une mesure d'incertitude sur les gradients, par exemple par le biais de la matrice de covariance de ces derniers, pour les normaliser (Le Roux et al., 2008). Ce ne sont que quelques exemples de méthodes utilisées pour améliorer l'optimisation de réseaux profonds, une énumération exhaustive nécessiterait un chapitre entier. Nous allons voir dans le chapitre 4 et 6, qu'il est possible de faciliter l'optimisation de réseaux profonds, même dans le cas de la descente de gradient stochastique simple, en faisant attention à l'initialisation aléatoire des paramètres et au choix de la fonction d'activation.

L'utilisation de régularisations appropriées est aussi une solution permettant de faciliter l'apprentissage des architectures profondes. Un exemple récent est la méthode dite "Dropout" (Srivastava et al., 2014; Krizhevsky et al., 2012). Il s'agit, pendant l'apprentissage, de masquer aléatoirement une fraction des unités cachées du réseau. Ce bruit permet de mieux utiliser la capacité du modèle car l'information doit être distribuée sur les unités pour résoudre la tâche. Cela peut-être vu comme l'apprentissage d'un ensemble de classifieurs qui partagent leurs paramètres. Pour la prédiction, on utilise le modèle complet, mais on multiplie les poids par la fraction de bruit utilisé pendant l'apprentissage, afin de conserver la valeur moyenne des activations des neurones identiques à celles durant l'apprentissage.

Hinton et al. (2006) et Bengio et al. (2007) ont montré qu'il était possible d'entraîner des réseaux profonds en utilisant un apprentissage non-supervisé vorace couche-par-couche, ou pré-entraînement, pour initialiser les paramètres. Une étude montre que l'apprentissage non-supervisé agit comme un régulariseur et qu'il permet de placer les paramètres dans un meilleur bassin d'attraction, associé à un minimum local avec une meilleure capacité de généralisation (Erhan et al., 2009). Ceci est à l'origine de la renaissance des architectures profondes. Cependant, avec les dernières avancées, comme la méthode "Dropout" et l'utilisation des unités rectificatrices qui seront, elles, présentées au chapitre 6, le pré-apprentissage ne semble plus nécessaire dans le cas supervisé avec beaucoup d'exemples d'entraînements. Cependant, dans le cas semi-supervisé ou en transfert de l'apprentissage, cette méthode reste centrale, comme nous le verrons au chapitres 6 et 8. La section suivante présente un exemple concret de modèle utilisé pour le pré-entraînement.

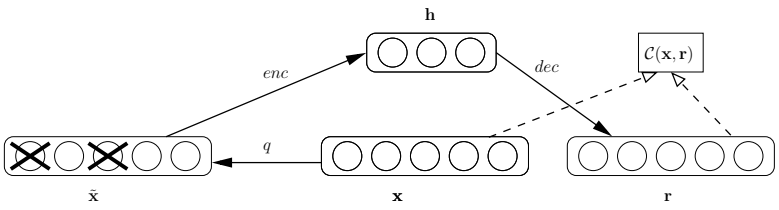

Figure 2.3 – Schéma de principe de l'Auto-Encodeur Débruitant, *source:* Vincent et al. (2010).

2.4 Auto-Encodeur Débruitant

Un auto-encodeur est un réseau de neurones que l'on entraîne à reconstruire son entrée. Il est constitué d'un encodeur qui transforme les données en un code h et d'un décodeur chargé de reconstruire l'entrée à partir du code. Si les fonctions d'activations sont linéaires et que le coût de reconstruction est l'erreur quadratique, l'Auto-Encodeur est similaire à une analyse en composante principale, à une rotation près. Il s'agit donc d'une méthode de réduction de dimensionnalité, le nombre d'unités de la couche cachée ne devant pas dépasser le nombre d'entrées sinon la fonction identité est une solution triviale. Il est donc important de régulariser le modèle.

L'Auto-Encodeur Débruitant (Vincent et al., 2008), ou DAE, illustré à la figure 2.3, est une de ses différentes formes de régularisation. On commence par bruiter aléatoirement l'entrée $\mathbf{x}$ avec un processus de corruption q. Il s'agit la plupart du temps, soit d'un bruit gaussien avec une variance σ, soit d'une substitution des entrées par 0 avec une probabilité p. L'entrée corrompue $\tilde{x}$ est ensuite donnée à l'Auto-Encodeur que l'on entraîne pour reconstruire l'entrée originale. Expérimentalement, ce modèle permet d'obtenir des réseaux profonds avec des performances discriminatoires similaires ou supérieures à celles de modèles pré-entraînés avec des modèles génératifs à fonction d'énergie comme la Machine de Boltzmann Restreinte. On peut d'ailleurs voir un DAE comme un modèle génératif (Bengio et al., 2013)

Notons qu'il existe d'autres formes de régularisation, par exemple la pénalité contractive: l'objectif n'est plus de débruiter l'entrée mais d'obtenir une représentaton qui varie peu en fonction des variations locales de l'entrée (Rifai et al., 2011).

2.5 Traitement des Langues Naturelles

Le Traitement Automatique des Langues Naturelles, ou NLP, est l'ensemble des tâches informatiques manipulant les languages humains. Ces dernières sont très variées. À titre d'exemple, il peut s'agir:

- de la modélisation du language, qui vise à assigner une probabilité d'apparition à une séquence de mots ;
- de la désambiguation de sens, qui consiste à identifier, suivant le contexte, quel sens d'un mot est utilisé ;
- de la traduction automatique d'un language à l'autre ;
- de l'analyse syntactique de surface, dont l'objectif est d'identifier les éléments constituant d'une phrase, comme les groupes nominaux ou les verbes ;
- de l'identification de rôle sémantique, qui attribue des relations entre ces constituants...

La plupart des solutions abordées dans ce domaine sont élaborées de manière spécifique à chacune des tâches et nécessitent l'apport de nombreuses connaissances à priori du domaine.

L'utilisation des réseaux de neurones est toutefois possible. En effet, Bengio et al. (2003) présente un modèle de language probabilistique neuronal. L'idée de base étant de représenter les mots par un vecteur de dimension fixée, dont les coordonnées sont apprises, appelé "embedding". Cela permet de représenter les mots de manière distribuée. Par la suite, il a été montré que ce type de modèle peut être utilisé sur 4 différentes tâches du traitement des langues naturelles, en obtenant des performances similaires à l'état de l'art (Collobert et al., 2011), et ce, en injectant des connaissances à priori minimales. Depuis, l'utilisation des modèles à base d'embeddings s'est développée, comprenant les réseaux de neurones réccurents (Mikolov et al., 2011) et récursifs (Socher et al., 2011) pour une grande variété de tâches.

Dans cette thèse nous allons illustrer l'utilisation de modèles à base d'embeddings sans réccurence ni récursion pour la tâche d'analyse de sentiment aux chapitres 6 et 8, ainsi que pour des données multi-relationelles au chapitre 10, qui dans ce dernier cas permettra l'encodage de base de données et la désambiguation de sens dans le contexte de l'analyse sémantique.

3 Introduction de l'article 1

X. Glorot et Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *JMLR W&CP: Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS 2010)*, pages 249-256, 2010

3.1 Contexte

Ce travail a été réalisé peu de temps après les premières études montrant qu'il est possible d'améliorer les performances des réseaux de neurones profonds en les initialisant à l'aide d'un pré-entraînement non-supervisé (Hinton et al., 2006; Bengio et al., 2007). Des minima locaux avec de meilleures performances en généralisation sont donc atteignables, mais, avec une initialisation aléatoire des paramètres et une descente de gradient stochastique, ils ne semblaient pas accessibles, notamment à cause des problèmes de points citiques (Dauphin et al., 2014). Plusieurs expériences montraient que l'apprentissage de réseaux profonds sans pré-entraînement était impossible dans certains cas, par exemple avec 5 couches cachées sur MNIST (Erhan et al., 2009).

Dans le cadre de ce travail, nous avons voulu étudier la dynamique de l'apprentissage des réseaux profonds initialisés aléatoirement, en analysant les valeurs des activations et de celles des gradients rétro-propagés. L'objectif était d'analyser l'influence du choix de la fonction d'activation et de l'initialisation des paramètres, pour mieux comprendre les difficultés inhérentes à l'apprentissage de ces modèles et d'identifier les choix qui permettent de réduire celles-ci.

Nous avons utilisé, pour ce faire, plusieurs tâches de reconnaissance d'objets avec des données réelles et synthétiques.

3.2 Contribution de l'étudiant

Ce projet a été réalisé au tout début de mon doctorat. Je voulais, en visualisant la dynamique de l'apprentissage, améliorer mes intuitions. Yoshua Bengio m'a guidé à travers mes découvertes. J'ai réalisé l'intégralité des expériences et j'ai aussi développé le code pour générer Shapeset-3 $\times$ 2, à partir d'une version simplifiée créé par James Bergstra. L'idée de l'initialisation normalisée et les développements mathématiques sont de moi. Étant nouveau dans le domaine de recherche, Yoshua Bengio m'a beaucoup aidé pour la rédaction de l'article.

3.3 Impact

Cet article a permis de montrer que des choix simples, comme la fonction d'activation et l'initialisation des poids, peuvent avoir une grande influence sur les performances des réseaux de neurones profonds. Contrairement à ce qui avait été montré précédemment, ce travail montre que l'apprentissage des réseaux de neurones très profonds, initialisés aléatoirement, est possible. Même si les résultats restent encore inférieurs aux méthodes d'optimisation plus évoluées, comme "Hessian-free" (Martens, 2010), les performances atteintes sont proches (Chapelle and Erhan, 2011). Dans la même voie de recherche, Sutskever et al. (2013) montre que des initialisations éparses donnent également de bon résultats, de plus la différence de performance avec "Hessian Free" est encore réduite en utilisant le "momentum" pendant l'apprentissage.

4 Understanding the difficulty of training deep feedforward neural networks

Abstract

Whereas before 2006 it appears that deep multi-layer neural networks were not successfully trained, since then several algorithms have been shown to successfully train them, with experimental results showing the superiority of deeper vs less deep architectures. All these experimental results were obtained with new initialization or training mechanisms. Our objective here is to understand better why standard gradient descent from random initialization is doing so poorly with deep neural networks, to better understand these recent relative successes and help design better algorithms in the future. We first observe the influence of the non-linear activations functions. We find that the logistic sigmoid activation is unsuited for deep networks with random initialization because of its mean value, which can drive especially the top hidden layer into saturation. Surprisingly, we find that saturated units can move out of saturation by themselves, albeit slowly, and explaining the plateaus sometimes seen when training neural networks. We find that a new non-linearity that saturates less can often be beneficial. Finally, we study how activations and gradients vary across layers and during training, with the idea that training may be more difficult when the singular values of the Jacobian associated with each layer are far from 1. Based on these considerations, we propose a new initialization scheme that brings substantially faster convergence.

4.1 Deep Neural Networks

Deep learning methods aim at learning feature hierarchies with features from higher levels of the hierarchy formed by the composition of lower level features. They include learning methods for a wide array of *deep architectures*, including

neural networks with many hidden layers (Vincent et al., 2008) and graphical models with many levels of hidden variables (Hinton et al., 2006), among others (Zhu et al., 2009; Weston et al., 2008). Much attention has recently been devoted to them (see (Bengio, 2009) for a review), because of their theoretical appeal, inspiration from biology and human cognition, and because of empirical success in vision (Ranzato et al., 2007; Larochelle et al., 2007; Vincent et al., 2008) and natural language processing (NLP) (Collobert and Weston, 2008; Mnih and Hinton, 2009). Theoretical results reviewed and discussed by Bengio (2009), suggest that in order to learn the kind of complicated functions that can represent high-level abstractions (e.g. in vision, language, and other AI-level tasks), one may need *deep architectures*.

Most of the recent experimental results with deep architecture are obtained with models that can be turned into deep supervised neural networks, but with initialization or training schemes different from the classical feedforward neural networks (Rumelhart et al., 1986). Why are these new algorithms working so much better than the standard random initialization and gradient-based optimization of a supervised training criterion ? Part of the answer may be found in recent analyses of the effect of unsupervised pre-training (Erhan et al., 2009), showing that it acts as a regularizer that initializes the parameters in a "better" basin of attraction of the optimization procedure, corresponding to an apparent local minimum associated with better generalization. But earlier work (Bengio et al., 2007) had shown that even a purely supervised but greedy layer-wise procedure would give better results. So here instead of focusing on what unsupervised pre-training or semi-supervised criteria bring to deep architectures, we focus on analyzing what may be going wrong with good old (but deep) multi-layer neural networks.

Our analysis is driven by investigative experiments to monitor activations (watching for saturation of hidden units) and gradients, across layers and across training iterations. We also evaluate the effects on these of choices of activation function (with the idea that it might affect saturation) and initialization procedure (since unsupervised pre-training is a particular form of initialization and it has a drastic impact).

4.2 Experimental Setting and Datasets

Code to produce the new datasets introduced in this section is available from: http://goo.gl/3fZm6c.

4.2.1 Online Learning on an Infinite Dataset: Shapeset-3×2

Recent work with deep architectures (see Figure 7 in Bengio (2009)) shows that even with very large training sets or online learning, initialization from unsupervised pre-training yields substantial improvement, which does not vanish as the number of training examples increases. The online setting is also interesting because it focuses on the optimization issues rather than on the small-sample regularization effects, so we decided to include in our experiments a synthetic images dataset inspired from Larochelle et al. (2007) and Larochelle et al. (2009), from which as many examples as needed could be sampled, for testing the online learning scenario.

We call this dataset the **Shapeset-**3×2 dataset, with example images in Figure 4.1 (left). **Shapeset-**3×2 contains images of 1 or 2 two-dimensional objects, each taken from 3 shape categories (triangle, parallelogram, ellipse), and placed with random shape parameters (relative lengths and/or angles), scaling, rotation, translation and grey-scale.

We noticed that for only one shape present in the image the task of recognizing it was too easy. We therefore decided to sample also images with two objects, with the constraint that the second object does not overlap with the first by more than fifty percent of its area, to avoid hiding it entirely. The task is to predict the objects present (e.g. triangle + ellipse, parallelogram + parallelogram, triangle alone, etc.) without having to distinguish between the foreground shape and the background shape when they overlap. This therefore defines nine configuration classes.

The task is fairly difficult because we need to discover invariances over rotation, translation, scaling, object color, occlusion and relative position of the shapes. In parallel we need to extract the factors of variability that predict which object shapes are present.

The size of the images are arbitrary but we fixed it to 32×32 in order to work with deep dense networks efficiently.

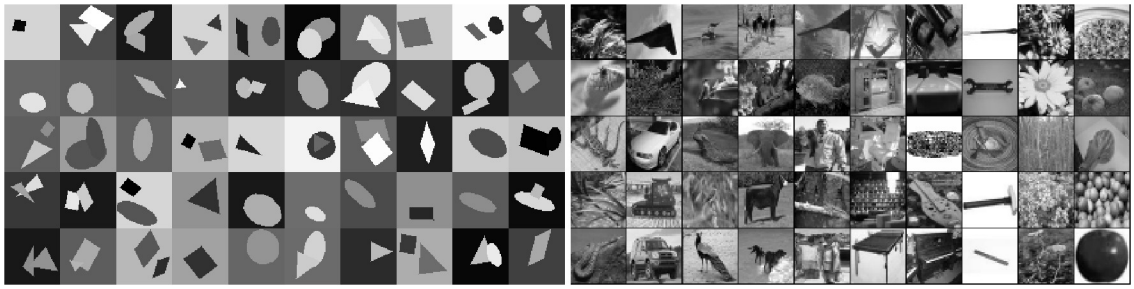

Figure 4.1 – *Left: Shapeset-3×2 images at 64×64 resolution. The examples we used are at 32×32 resolution. The learner tries to predict which objects (parallelogram, triangle, or ellipse) are present, and 1 or 2 objects can be present, yielding 9 possible classifications. Right: Small-ImageNet images at full resolution.*

4.2.2 Finite Datasets

The MNIST digits (LeCun et al., 1998), dataset has 50,000 training images, 10,000 validation images (for hyper-parameter selection), and 10,000 test images, each showing a 28×28 grey-scale pixel image of one of the 10 digits.

CIFAR-10 (Krizhevsky and Hinton, 2009) is a labelled subset of the tiny-images dataset that contains 50,000 training examples (from which we extracted 10,000 as validation data) and 10,000 test examples. There are 10 classes corresponding to the main object in each image: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, or truck. The classes are balanced. Each image is in color, but is just 32×32 pixels in size, so the input is a vector of $32 \times 32 \times 3 = 3072$ real values.

Small-ImageNet which is a set of tiny 37×37 gray level images dataset computed from the higher-resolution and larger set at `http://www.image-net.org`, with labels from the WordNet noun hierarchy. We have used 90,000 examples for training, 10,000 for the validation set, and 10,000 for testing. There are 10 balanced classes: reptiles, vehicles, birds, mammals, fish, furniture, instruments, tools, flowers and fruits Figure 4.1 (right) shows randomly chosen examples.

4.2.3 Experimental Setting

We optimized feedforward neural networks with one to five hidden layers, with one thousand hidden units per layer, and with a softmax logistic regression for the output layer. The cost function is the negative log-likelihood $-\log P(y|\mathbf{x})$, where $(\mathbf{x}, y)$ is the (input image, target class) pair. The neural networks were optimized

with stochastic back-propagation on mini-batches of size ten, i.e., the average $\mathbf{g}$ of $\frac{\partial -\log P(y|\mathbf{x})}{\partial \boldsymbol{\theta}}$ was computed over 10 consecutive training pairs $(\mathbf{x}, y)$ and used to update parameters $\boldsymbol{\theta}$ in that direction, with $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon \mathbf{g}$. The learning rate ϵ is a hyper-parameter that is optimized based on validation set error after a large number of updates (5 million).

We varied the type of non-linear activation function in the hidden layers: the sigmoid $1/(1+e^{-x})$, the hyperbolic tangent $\tanh(x)$, and a newly proposed activation function (Bergstra et al., 2009) called the softsign, $x/(1+|x|)$. The softsign is similar to the hyperbolic tangent (its range is -1 to 1) but its tails are quadratic polynomials rather than exponentials, i.e., it approaches its asymptotes much slower.

In the comparisons, we search for the best hyper-parameters (learning rate and depth) separately for each model. Note that the best depth was always five for Shapeset-3 $\times$ 2, except for the sigmoid, for which it was four.

We initialized the biases to be 0 and the weights W_{ij} at each layer with the following commonly used heuristic:

$$W_{ij} \sim U\Big[-\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}}\Big], \tag{4.1}$$

where $U[-a, a]$ is the uniform distribution in the interval $(-a, a)$ and n is the size of the previous layer (the number of columns of W).

4.3 Effect of Activation Functions and Saturation During Training

Two things we want to avoid and that can be revealed from the evolution of activations is excessive saturation of activation functions on one hand (then gradients will not propagate well), and overly linear units (they will not compute something interesting).

4.3.1 Experiments with the Sigmoid

The sigmoid non-linearity has been already shown to slow down learning because of its none-zero mean that induces important singular values in the Hessian (LeCun

et al., 1998). In this section we will see another symptomatic behavior due to this activation function in deep feedforward networks.

We want to study possible saturation, by looking at the evolution of activations during training, and the figures in this section show results on the **Shapeset-**3×2 data, but similar behavior is observed with the other datasets. Figure 4.2 shows the evolution of the activation values (after the non-linearity) at each hidden layer during training of a deep architecture with sigmoid activation functions. Layer 1 refers to the output of first hidden layer, and there are four hidden layers. The graph shows the means and standard deviations of these activations. These statistics along with histograms are computed at different times during learning, by looking at activation values for a fixed set of 300 test examples.

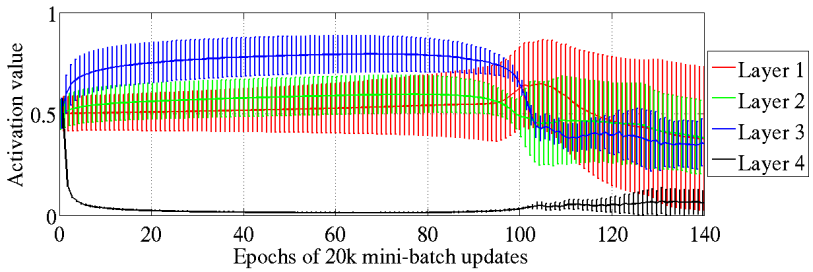

Figure 4.2 – *Mean and standard deviation (vertical bars) of the activation values (output of the sigmoid) during supervised learning, for the different hidden layers of a deep architecture. The top hidden layer quickly saturates at 0 (slowing down all learning), but then slowly desaturates around epoch 100.*

We see that very quickly at the beginning, all the sigmoid activation values of the last hidden layer are pushed to their lower saturation value of 0. Inversely, the others layers have a mean activation value that is above 0.5, and decreasing as we go from the output layer to the input layer. We have found that this kind of saturation can last very long in deeper networks with sigmoid activations, e.g., the depth-five model never escaped this regime during training. The big surprise is that for intermediate number of hidden layers (here four), the saturation regime may be escaped. At the same time that the top hidden layer moves out of saturation, the first hidden layer begins to saturate and therefore to stabilize.

We hypothesize that this behavior is due to the combination of random initialization and the fact that an hidden unit output of 0 corresponds to a saturated sigmoid. Note that deep networks with sigmoids but initialized from unsupervised

pre-training (e.g. from RBMs) do not suffer from this saturation behavior. Our proposed explanation rests on the hypothesis that the transformation that the lower layers of the randomly initialized network computes initially is not useful to the classification task, unlike the transformation obtained from unsupervised pre-training. The logistic layer output softmax($b + Wh$) might initially rely more on its biases b (which are learned very quickly) than on the top hidden activations h derived from the input image (because h would vary in ways that are not predictive of y, maybe correlated mostly with other and possibly more dominant variations of x). Thus the error gradient would tend to push Wh towards 0, which can be achieved by pushing h towards 0. In the case of symmetric activation functions like the hyperbolic tangent and the softsign, sitting around 0 is good because it allows gradients to flow backwards. However, pushing the sigmoid outputs to 0 would bring them into a saturation regime which would prevent gradients to flow backward and prevent the lower layers from learning useful features. Eventually but slowly, the lower layers move toward more useful features and the top hidden layer then moves out of the saturation regime. Note however that, even after this, the network moves into a solution that is of poorer quality (also in terms of generalization) then those found with symmetric activation functions, as can be seen in figure 4.11.

4.3.2 Experiments with the Hyperbolic tangent

As discussed above, the hyperbolic tangent networks do not suffer from the kind of saturation behavior of the top hidden layer observed with sigmoid networks, because of its symmetry around 0. However, with our standard weight initialization $U\left[-\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}}\right]$, we observe a sequentially occurring saturation phenomenon starting with layer 1 and propagating up in the network, as illustrated in Figure 4.3. Why this is happening remains to be understood.

4.3.3 Experiments with the Softsign

The softsign $x/(1 + |x|)$ is similar to the hyperbolic tangent but might behave differently in terms of saturation because of its smoother asymptotes (polynomial instead of exponential). We see on Figure 4.3 that the saturation does not occur one layer after the other like for the hyperbolic tangent. It is faster at the beginning and then slow, and all layers move together towards larger weights.

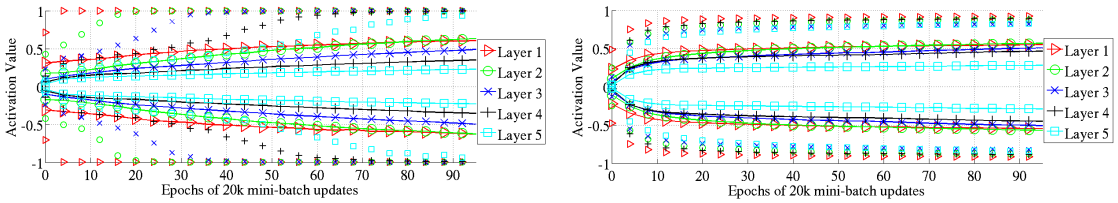

Figure 4.3 – *Left:98 percentiles (markers alone) and standard deviation (solid lines with markers) of the distribution of the activation values for the hyperbolic tangent networks in the course of learning. We see the first hidden layer saturating first, then the second, etc. Right: 98 percentiles (markers alone) and standard deviation (solid lines with markers) of the distribution of activation values for the softsign during learning. Here the different layers saturate less and do so together.*

We can also see at the end of training that the histogram of activation values is very different from that seen with the hyperbolic tangent (Figure 4.4). Whereas the latter yields modes of the activations distribution mostly at the extremes (asymptotes -1 and 1) or around 0, the softsign network has modes of activations around its knees (between the linear regime around 0 and the flat regime around -1 and 1). These are the areas where there is substantial non-linearity but where the gradients would flow well.

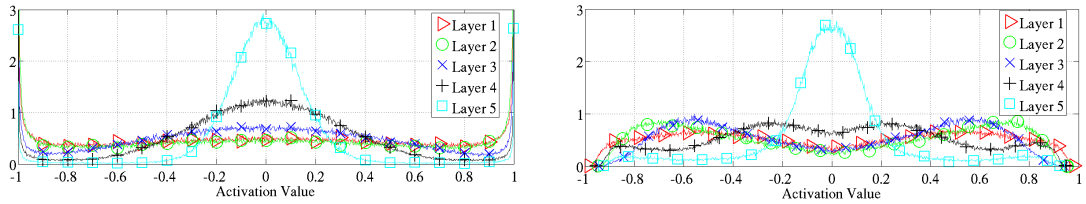

Figure 4.4 – *Activation values normalized histogram at the end of learning, averaged across units of the same layer and across 300 test examples. Left: activation function is hyperbolic tangent, we see important saturation of the lower layers. Right: activation function is softsign, we see many activation values around (-0.6,-0.8) and (0.6,0.8) where the units do not saturate but are non-linear.*

4.4 Studying Gradients and their Propagation

4.4.1 Effect of the Cost Function

We have found that the logistic regression or conditional log-likelihood cost function ($-\log P(y|\mathbf{x})$) coupled with softmax outputs) worked much better (for

classification problems) than the quadratic cost which was traditionally used to train feedforward neural networks (Rumelhart et al., 1986). This is not a new observation (Solla et al., 1988) but we find it important to stress here. We found that the plateaus in the training criterion (as a function of the parameters) are less present with the log-likelihood cost function. We can see this on Figure 4.5, which plots the training criterion as a function of two weights for a two-layer network (one hidden layer) with hyperbolic tangent units, and a random input and target signal. There are clearly more severe plateaus with the quadratic cost.

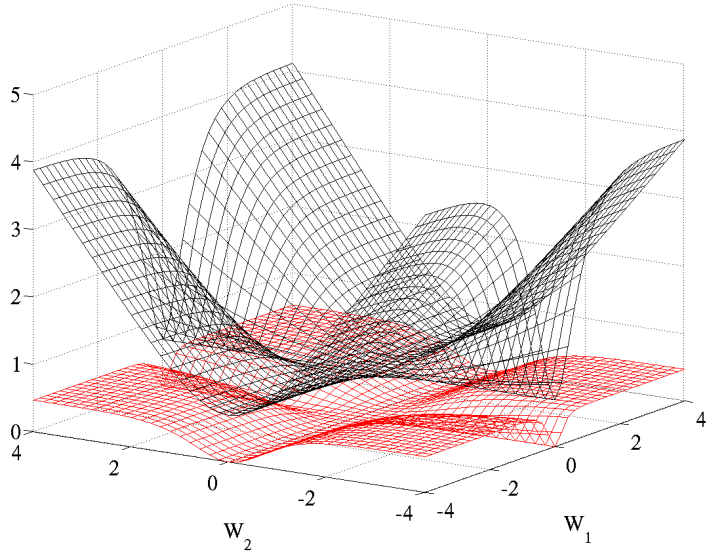

Figure 4.5 – *Cross entropy (black, surface on top) and quadratic (red, bottom surface) cost as a function of two weights (one at each layer) of a network with two layers, W_1 respectively on the first layer and W_2 on the second, output layer.*

4.4.2 Gradients at initialization

Theoretical Considerations and a New Normalized Initialization

We study the back-propagated gradients, or equivalently the gradient of the cost function on the inputs biases at each layer. Bradley (2009) found that back-propagated gradients were smaller as one moves from the output layer towards the input layer, just after initialization. He studied networks with linear activation at

each layer, finding that the variance of the back-propagated gradients decreases as we go backwards in the network. We will also start by studying the linear regime.

For a dense artificial neural network using symmetric activation function f with unit derivative at 0 (i.e. $f'(0) = 1$), if we write $\mathbf{z}^i$ for the activation vector of layer i, and $\mathbf{s}^i$ the argument vector of the activation function at layer i, we have $\mathbf{s}^i = \mathbf{z}^i W^i + \mathbf{b}^i$ and $\mathbf{z}^{i+1} = f(\mathbf{s}^i)$. From these definitions we obtain the following:

$$\frac{\partial Cost}{\partial s_k^i} = f'(s_k^i) W_{k,\bullet}^{i+1} \frac{\partial Cost}{\partial \mathbf{s}^{i+1}}^\intercal \tag{4.2}$$

$$\frac{\partial Cost}{\partial w_{l,k}^i} = z_l^i \frac{\partial Cost}{\partial s_k^i} \tag{4.3}$$

The variances will be expressed with respect to the input, outpout and weights initialization randomness. Consider the hypothesis that we are in a linear regime at the initialization, that the weights are initialized independently and that the inputs features variances are the same ($= Var[x]$). If the input and weights random variables have zero mean then we can say that, with n_i the size of layer i and x the network input,

$$f'(s_k^i) \approx 1, \tag{4.4}$$

$$Var[z^i] = Var[x] \prod_{i'=0}^{i-1} n_{i'} Var[W^{i'}], \tag{4.5}$$

We write $Var[W^{i'}]$ for the shared scalar variance of all weights at layer i'. Similarly, if the gradients of the cost with respect to the output layer have zero mean and equal variances, then for a network with d layers,

$$Var\Big[\frac{\partial Cost}{\partial s^i}\Big] = Var\Big[\frac{\partial Cost}{\partial s^d}\Big] \prod_{i'=i}^{d} n_{i'+1} Var[W^{i'}], \tag{4.6}$$

$$\begin{aligned} Var\Big[\frac{\partial Cost}{\partial w^i}\Big] = &\prod_{i'=0}^{i-1} n_{i'} Var[W^{i'}] \prod_{i'=i}^{d-1} n_{i'+1} Var[W^{i'}] \\ &\times Var[x] Var\Big[\frac{\partial Cost}{\partial s^d}\Big]. \end{aligned} \tag{4.7}$$

From a forward-propagation point of view, to keep information flowing we would

like that

$$\forall(i,i'), Var[z^i] = Var[z^{i'}]. \tag{4.8}$$

From a back-propagation point of view we would similarly like to have

$$\forall(i,i'), Var\Big[\frac{\partial Cost}{\partial s^i}\Big] = Var\Big[\frac{\partial Cost}{\partial s^{i'}}\Big]. \tag{4.9}$$

These two conditions transform to:

$$\forall i, \quad n_i Var[W^i] = 1 \tag{4.10}$$

$$\forall i, \quad n_{i+1} Var[W^i] = 1 \tag{4.11}$$

As a compromise between these two constraints, we might want to have

$$\forall i, \quad Var[W^i] = \frac{2}{n_i + n_{i+1}} \tag{4.12}$$

Note how both constraints are satisfied when all layers have the same width. If we also have the same initialization for the weights we could get the following interesting properties:

$$\forall i, Var\Big[\frac{\partial Cost}{\partial s^i}\Big] = \Big[nVar[W]\Big]^{d-i} Var\Big[\frac{\partial Cost}{\partial s^d}\Big] \tag{4.13}$$

$$\forall i, Var\Big[\frac{\partial Cost}{\partial w^i}\Big] = \Big[nVar[W]\Big]^{d} Var[x] Var\Big[\frac{\partial Cost}{\partial s^d}\Big] \tag{4.14}$$

We can see that the variance of the gradient on the weights is the same for all layers, but the variance of the back-propagated gradient might still vanish or explode as we consider deeper networks. Note how this is reminiscent of issues raised when studying recurrent neural networks (Bengio et al., 1994), which can be seen as very deep networks when unfolded through time.

The standard initialization that we have used (eq.4.1) gives rise to variance with the following property:

$$nVar[W] = \frac{1}{3} \tag{4.15}$$

where n is the layer size (assuming all layers of the same size). This will cause the variance of the back-propagated gradient to be dependent on the layer (and decreasing).

The normalization factor may therefore be important when initializing deep networks because of the multiplicative effect through layers, and we suggest the following initialization procedure to approximately satisfy our objectives of maintaining activation variances and back-propagated gradients variance as one moves up or down the network. We call it the **normalized initialization**:

$$W \sim U\Big[-\frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}, \frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}\Big] \tag{4.16}$$

Gradient Propagation Study

To empirically validate the above theoretical ideas, we have plotted some normalized histograms of activation values, weight gradients and of the back-propagated gradients at initialization with the two different initialization methods. The results displayed (Figures 4.6, 4.7 and 4.8) are from experiments on **Shapeset-**3×2, but qualitatively similar results were obtained with the other datasets.

We monitor the singular values of the Jacobian matrix associated with layer i:

$$J^i = \frac{\partial \mathbf{z}^{i+1}}{\partial \mathbf{z}^i} \tag{4.17}$$

When consecutive layers have the same dimension, the average singular value corresponds to the average ratio of infinitesimal volumes mapped from $\mathbf{z}^i$ to $\mathbf{z}^{i+1}$, as well as to the ratio of average activation variance going from $\mathbf{z}^i$ to $\mathbf{z}^{i+1}$. With our normalized initialization, this ratio is around 0.8 whereas with the standard initialization, it drops down to 0.5.

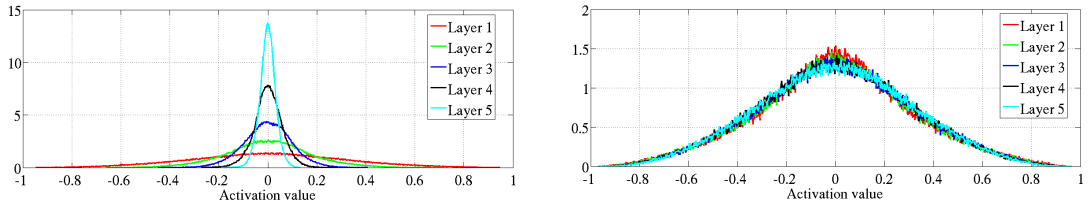

Figure 4.6 – *Activation values normalized histograms with hyperbolic tangent activation, with standard (left) vs normalized initialization (right). Left: 0-peak increases for higher layers.*

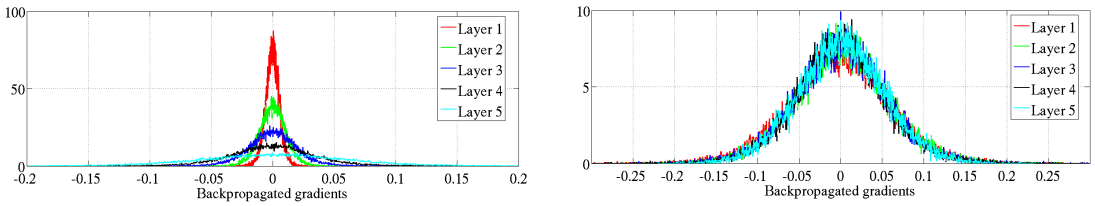

Figure 4.7 – *Back-propagated gradients normalized histograms with hyperbolic tangent activation, with standard (left) vs normalized (right) initialization. Left: 0-peak decreases for higher layers.*

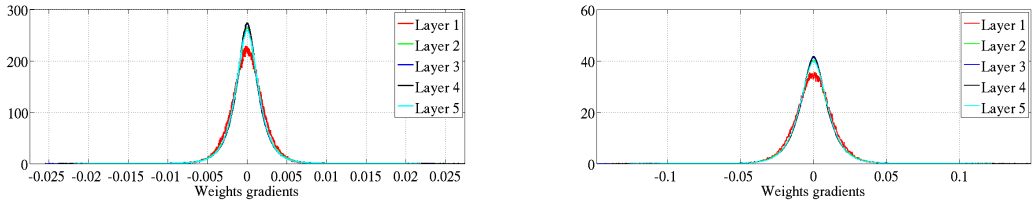

Figure 4.8 – *Weight gradient normalized histograms with hyperbolic tangent activation just after initialization, with standard initialization (left) and normalized initialization (right), for different layers. Even though with standard initialization the back-propagated gradients get smaller, the weight gradients do not!*

4.4.3 Back-propagated Gradients During Learning

The dynamic of learning in such networks is complex and we would like to develop better tools to analyze and track it. In particular, we cannot use simple variance calculations in our theoretical analysis because the weights values are not anymore independent of the activation values and the linearity hypothesis is also violated.

As first noted by Bradley (2009), we observe (Figure 4.7) that at the beginning of training, after the standard initialization (eq. 4.1), the variance of the back-propagated gradients gets smaller as it is propagated downwards. However we find that this trend is reversed very quickly during learning. Using our normalized initialization we do not see such decreasing back-propagated gradients (left of Figure 4.7).

What was initially really surprising is that even when the back-propagated gradients become smaller (standard initialization), the variance of the weights gradients is roughly constant across layers, as shown on Figure 4.8. However, this is explained by our theoretical analysis above (eq. 4.14). Interestingly, as shown in Figure 4.9, these observations on the weight gradient of standard and normalized

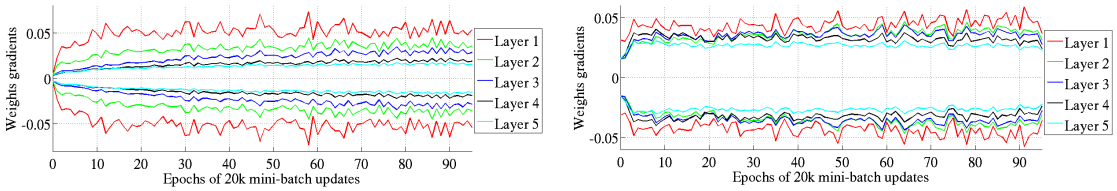

Figure 4.9 – *Standard deviation intervals of the weights gradients with hyperbolic tangent with standard initialization (left) and normalized (right) during training. We see that the normalization allows to keep the same variance of the weights gradient across layers, during training (left: smaller variance for higher layers).*

initialization change during training (here for a tanh network). Indeed, whereas the gradients have initially roughly the same magnitude, they diverge from each other (with larger gradients in the lower layers) as training progresses, especially with the standard initialization. Note that this might be one of the advantages of the normalized initialization, since having gradients of very different magnitudes at different layers may yield to ill-conditioning and slower training.

Finally, we observe that the softsign networks share similarities with the tanh networks with normalized initialization, as can be seen by comparing the evolution of activations in both cases (resp. Figure 4.3-left and Figure 4.10).

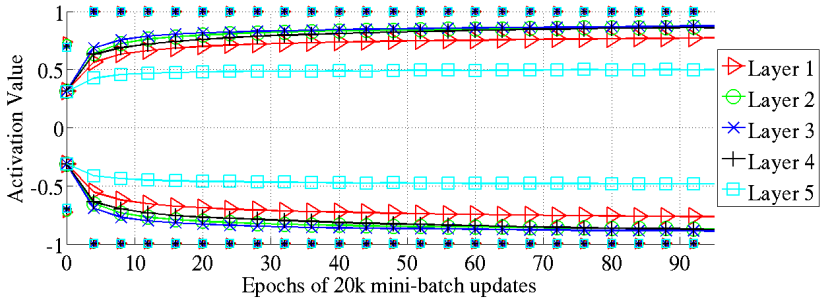

Figure 4.10 – *98 percentile (markers alone) and standard deviation (solid lines with markers) of the distribution of activation values for hyperbolic tangent with normalized initialization during learning.*

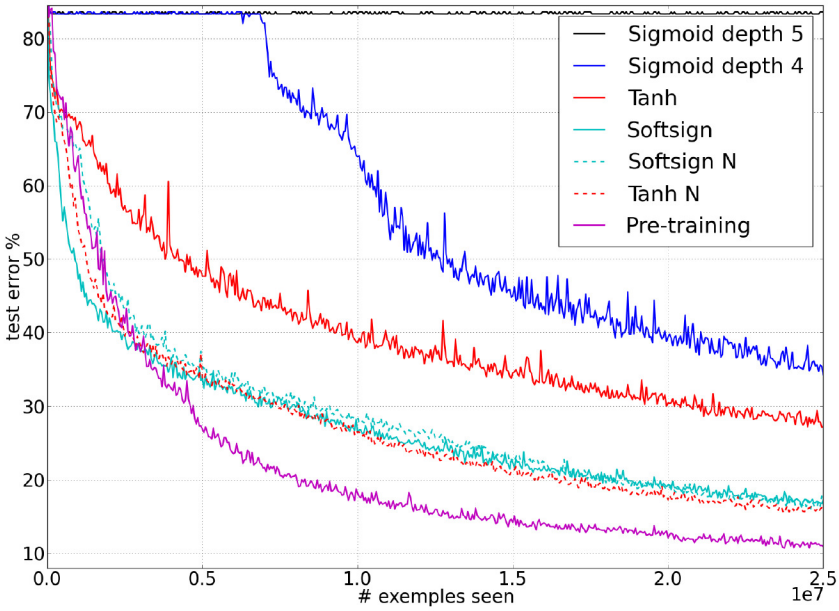

Figure 4.11 – *Test error during online training on the* **Shapeset-**3×2 *dataset, for various activation functions and initialization schemes (ordered from top to bottom in decreasing final error). N after the activation function name indicates the use of normalized initialization.*

4.5 Error Curves and Conclusions

The final consideration that we care for is the success of training with different strategies, and this is best illustrated with error curves showing the evolution of test error as training progresses and asymptotes. Figure 4.11 shows such curves with online training on **Shapeset-**3×2, while Table 4.1 gives final test error for all the datasets studied (**Shapeset-**3×2, MNIST, CIFAR-10, and Small-ImageNet). As a baseline, we optimized RBF SVM models on one hundred thousand Shapeset examples and obtained 59.47% test error, while on the same set we obtained 50.47% with a depth five hyperbolic tangent network with normalized initialization.

These results illustrate the effect of the choice of activation and initialization. As a reference we include in Figure 4.11 the error curve for the supervised fine-tuning from the initialization obtained after unsupervised pre-training with denoising auto-encoders (Vincent et al., 2008). For each network the learning rate is separately chosen to minimize error on the validation set. We can remark that on **Shapeset-**3×2, because of the task difficulty, we observe important saturations during learning, this might explain that the normalized initialization or the softsign

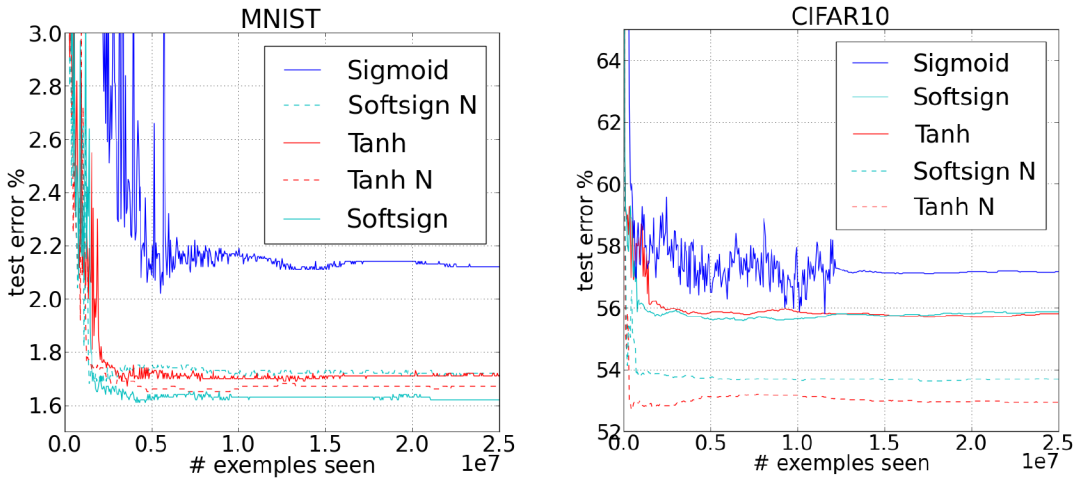

Figure 4.12 – *Test error curves during training on MNIST and CIFAR10, for various activation functions and initialization schemes (ordered from top to bottom in decreasing final error). N after the activation function name indicates the use of normalized initialization.*

effects are more visible.

Several conclusions can be drawn from these error curves:

- The more classical neural networks with sigmoid or hyperbolic tangent units and standard initialization fare rather poorly, converging more slowly and apparently towards ultimately poorer local minima.
- The softsign networks seem to be more robust to the initialization procedure than the tanh networks, presumably because of their gentler non-linearity.
- For tanh networks, the proposed normalized initialization can be quite helpful, presumably because the layer-to-layer transformations maintain magnitudes of activations (flowing upward) and gradients (flowing backward).

Others methods can alleviate discrepancies between layers during learning, e.g., exploiting second order information to set the learning rate separately for each parameter. For example, we can exploit the diagonal of the Hessian (LeCun et al., 1998) or a gradient variance estimate. Both those methods have been applied for **Shapeset**-3×2 with hyperbolic tangent and standard initialization. We observed a gain in performance but not reaching the result obtained from normalized initialization. In addition, we observed further gains by combining normalized initialization with second order methods: the estimated Hessian might then focus on discrepancies between units, not having to correct important initial discrepancies between

Table 4.1 – *Test error with different activation functions and initialization schemes for deep networks with 5 hidden layers. N after the activation function name indicates the use of normalized initialization. Results in bold are statistically different from non-bold ones under the null hypothesis test with* $p = 0.005$.

TYPE	Shapeset	MNIST	CIFAR-10	ImageNet
Softsign	**16.27**	**1.64**	55.78	**69.14**
Softsign N	**16.06**	**1.72**	**53.8**	**68.13**
Tanh	27.15	**1.76**	55.9	70.58
Tanh N	**15.60**	**1.64**	**52.92**	**68.57**
Sigmoid	82.61	2.21	57.28	70.66

layers.

In all reported experiments we have used the same number of units per layer. However, we verified that we obtain the same gains when the layer size increases (or decreases) with layer number.

The other conclusions from this study are the following:

- Monitoring activations and gradients across layers and training iterations is a powerful investigative tool for understanding training difficulties in deep nets.
- Sigmoid activations (not symmetric around 0) should be avoided when initializing from small random weights, because they yield poor learning dynamics, with initial saturation of the top hidden layer.
- Keeping the layer-to-layer transformations such that both activations and gradients flow well (i.e. with a Jacobian around 1) appears helpful, and allows to eliminate a good part of the discrepancy between purely supervised deep networks and ones pre-trained with unsupervised learning.
- Many of our observations remain unexplained, suggesting further investigations to better understand gradients and training dynamics in deep architectures.

5 Introduction de l'article 2

Xavier Glorot, Antoine Bordes and Yoshua Bengio. Deep Sparse Rectifier Neural Networks, in: *JMLR W&CP: Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2011

5.1 Contexte

Ce travail s'inscrit dans la continuité de celui présenté au chapitre 4 et partage le même contexte. L'objectif est toujours de faciliter l'apprentissage des réseaux de neurones profonds par des choix simples. Ici, nous nous intéressons spécifiquement à la fonction d'activation. Mon intérêt pour les neurosciences m'a poussé à étudier les fonctions d'activations utilisées pour modéliser le comportement des neurones. En partant de la nature éparse des représentations corticales et du succès récent de l'utilisation des unités rectificatrices bruitées pour l'apprentissage de RBM (Nair and Hinton, 2010), nous avons décidé d'utiliser des unités rectificatrices pour l'apprentissage supervisé des réseaux profonds.

5.2 Contribution

L'idée initiale d'utiliser les unités rectificatrices est venue de Yoshua Bengio. De mon côté j'ai effectué la revue de littérature en neurosciences, développé les intuitions mathématiques, réalisé les expériences et, enfin, rédigé l'article initial, soumis à NIPS. Ce dernier a été refusé et Antoine Bordes a rejoint le projet pour m'aider à l'améliorer pour qu'il soit ensuite accepté à AISTATS.

5.3 Impact

Cet article a eu un impact très important. Il est le premier à présenter l'avantage d'utiliser les unités rectificatrices pour les réseaux profonds, dans un contexte supervisé. Comme nous pouvons le voir à la figure 6.4, avec une grande quantité de données supervisées, l'utilisation d'un pré-entraînement non-supervisé n'est plus cruciale pour trouver un "bon" minimum local.

D'ailleurs, en conjonction avec la technique du "Dropout" (Srivastava et al., 2014), qui est particulièrement bien adaptée aux unités rectificatrices (Warde-Farley et al., 2013), ce type de fonction d'activation a permis d'améliorer l'état de l'art de manière notable dans plusieurs domaines, comme en vision pour la reconnaissance d'objets (Krizhevsky et al., 2012; Szegedy et al., 2014) ou en traitement de la parole (Sainath et al., 2013).

De nos jours, ce type de fonction d'activation est de plus en plus utilisé et étudié. **?** a montré comment le nombre de régions linéaires représentables dans un réseau de neurones avec unités linéaires par morceaux augmente exponentiellement avec le nombre de couches et polynomialement avec le nombre d'unités par couche. Goodfellow et al. (2013) propose une fonction d'activation, elle aussi linéaire par parties, qui est une généralisation des unités rectificatrices: les unités "maxout". Celles-ci aggrègent les sorties de plusieurs unités linéaires en prenant le maximum et donnent des résultats prometteurs.

6 Deep Sparse Rectifier Neural Networks

Abstract

While logistic sigmoid neurons are more biologically plausible than hyperbolic tangent neurons, the latter work better for training multi-layer neural networks. This paper shows that rectifying neurons are an even better model of biological neurons and yield equal or better performance than hyperbolic tangent networks in spite of the hard non-linearity and non-differentiability at zero, creating sparse representations with true zeros, which seem remarkably suitable for naturally sparse data. Even though they can take advantage of semi-supervised setups with extra-unlabeled data, deep rectifier networks can reach their best performance without requiring any unsupervised pre-training on purely supervised tasks with large labeled datasets. Hence, these results can be seen as a new milestone in the attempts at understanding the difficulty in training deep but purely supervised neural networks, and closing the performance gap between neural networks learnt with and without unsupervised pre-training.

6.1 Introduction

Many differences exist between the neural network models used by machine learning researchers and those used by computational neuroscientists. This is in part because the objective of the former is to obtain computationally efficient learners, that generalize well to new examples, whereas the objective of the latter is to abstract out neuroscientific data while obtaining explanations of the principles involved, providing predictions and guidance for future biological experiments. Areas where both objectives coincide are therefore particularly worthy of investigation, pointing towards computationally motivated principles of operation in the brain

that can also enhance research in artificial intelligence. In this paper we show that two common gaps between computational neuroscience models and machine learning neural network models can be bridged by using the following linear by part activation : $\max(0, x)$, called the rectifier (or hinge) activation function. Experimental results will show engaging training behavior of this activation function, especially for *deep architectures* (see Bengio (2009) for a review), i.e., where the number of hidden layers in the neural network is 3 or more.

Recent theoretical and empirical work in statistical machine learning has demonstrated the importance of learning algorithms for deep architectures. This is in part inspired by observations of the mammalian visual cortex, which consists of a chain of processing elements, each of which is associated with a different representation of the raw visual input. This is particularly clear in the primate visual system (Serre et al., 2007), with its sequence of processing stages: detection of edges, primitive shapes, and moving up to gradually more complex visual shapes. Interestingly, it was found that the features learned in deep architectures resemble those observed in the first two of these stages (in areas V1 and V2 of visual cortex) (Lee et al., 2008), and that they become increasingly invariant to factors of variation (such as camera movement) in higher layers (Goodfellow et al., 2009).

Regarding the training of deep networks, something that can be considered a breakthrough happened in 2006, with the introduction of Deep Belief Networks (Hinton et al., 2006), and more generally the idea of initializing each layer by unsupervised learning (Bengio et al., 2007; Ranzato et al., 2007). Some authors have tried to understand why this unsupervised procedure helps (Erhan et al., 2010) while others investigated why the original training procedure for deep neural networks failed (Glorot and Bengio, 2010). From the machine learning point of view, this paper brings additional results in these lines of investigation.

We propose to explore the use of rectifying non-linearities as alternatives to the hyperbolic tangent or sigmoid in deep artificial neural networks, in addition to using an L_1 regularizer on the activation values to promote sparsity and prevent potential numerical problems with unbounded activation. Nair and Hinton (2010) present promising results of the influence of such units in the context of Restricted Boltzmann Machines compared to logistic sigmoid activations on image classification tasks. Our work extends this for the case of pre-training using denoising auto-encoders (Vincent et al., 2008) and provides an extensive empirical compar-

ison of the rectifying activation function against the hyperbolic tangent on image classification benchmarks as well as an original derivation for the text application of sentiment analysis.

Our experiments on image and text data indicate that training proceeds better when the artificial neurons are either off or operating mostly in a linear regime. Surprisingly, rectifying activation allows deep networks to achieve their best performance without unsupervised pre-training. Hence, our work proposes a new contribution to the trend of understanding and merging the performance gap between deep networks learnt with and without unsupervised pre-training (Erhan et al., 2010; Glorot and Bengio, 2010). Still, rectifier networks can benefit from unsupervised pre-training in the context of semi-supervised learning where large amounts of unlabeled data are provided. Furthermore, as rectifier units naturally lead to sparse networks and are closer to biological neurons' responses in their main operating regime, this work also bridges (in part) a machine learning / neuroscience gap in terms of activation function and sparsity.

This paper is organized as follows. Section 6.2 presents some neuroscience and machine learning background which inspired this work. Section 6.3 introduces rectifier neurons and explains their potential benefits and drawbacks in deep networks. Then we propose an experimental study with empirical results on image recognition in Section 6.4.1 and sentiment analysis in Section 10.6.4. Section 10.7 presents our conclusions.

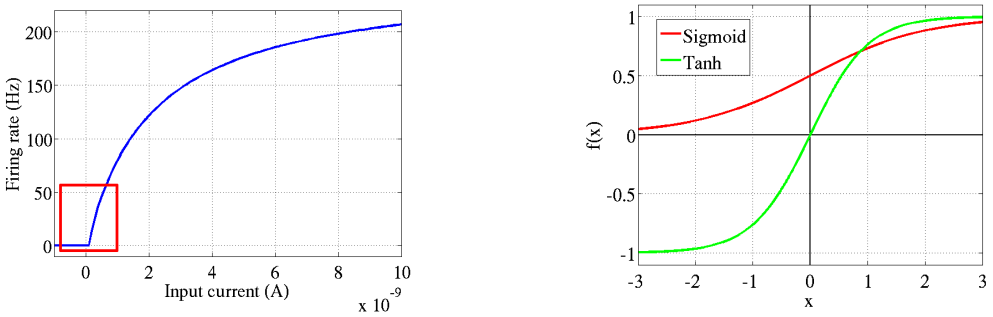

Figure 6.1 – *Left:* **Common neural activation function motivated by biological data.** *Right:* **Commonly used activation functions** in neural networks literature: logistic sigmoid and hyperbolic tangent (*tanh*).

6.2 Background

6.2.1 Neuroscience Observations

For models of biological neurons, the activation function is the expected firing rate as a function of the total input currently arising out of incoming signals at synapses (Dayan and Abott, 2001). An activation function is termed, respectively *antisymmetric* or *symmetric* when its response to the opposite of a strongly excitatory input pattern is respectively a strongly inhibitory or excitatory one, and *one-sided* when this response is zero. The main gaps that we wish to consider between computational neuroscience models and machine learning models are the following:

- Studies on brain energy expense suggest that neurons encode information in a sparse and distributed way (Attwell and Laughlin, 2001), estimating the percentage of neurons active at the same time to be between 1 and 4% (Lennie, 2003). This corresponds to a trade-off between richness of representation and small action potential energy expenditure. Without additional regularization, such as an L_1 penalty, ordinary feedforward neural nets do not have this property. For example, the sigmoid activation has a steady state regime around $\frac{1}{2}$, therefore, after initializing with small weights, all neurons fire at half their saturation regime. This is biologically implausible *and* hurts gradient-based optimization (LeCun et al., 1998; Glorot and Bengio, 2010).
- Important divergences between biological and machine learning models concern non-linear activation functions. A common biological model of neuron, the leaky integrate-and-fire (or *LIF*) (Dayan and Abott, 2001), gives the following relation between the firing rate and the input current, illustrated in Figure 6.1 (left):

$$f(I) = \begin{cases} \left[\tau \log\left(\frac{E+RI-V_r}{E+RI-V_{th}}\right) + t_{ref}\right]^{-1}, & \\ \qquad\qquad \text{if } E + RI > V_{th} & \\ 0\,, \qquad \text{if } E + RI \leq V_{th} & \end{cases}$$

where t_{ref} is the refractory period (minimal time between two action po-

tentials), I the input current, V_r the resting potential and V_{th} the threshold potential (with $V_{th} > V_r$), and R, E, τ the membrane resistance, potential and time constant. The most commonly used activation functions in the deep learning and neural networks literature are the standard *logistic sigmoid* and the *hyperbolic tangent* (see Figure 6.1, right), which are equivalent up to a linear transformation. The hyperbolic tangent has a steady state at 0, and is therefore preferred from the optimization standpoint (LeCun et al., 1998; Glorot and Bengio, 2010), but it forces an antisymmetry around 0 which is absent in biological neurons.

6.2.2 Advantages of Sparsity

Sparsity has become a concept of interest, not only in computational neuroscience and machine learning but also in statistics and signal processing (Candes and Tao, 2005). It was first introduced in computational neuroscience in the context of sparse coding in the visual system (Olshausen and Field, 1997). It has been a key element of deep convolutional networks exploiting a variant of auto-encoders (Ranzato et al., 2007, 2008; Mairal et al., 2009) with a sparse distributed representation, and has also become a key ingredient in Deep Belief Networks (Lee et al., 2008). A sparsity penalty has been used in several computational neuroscience (Olshausen and Field, 1997; Doi et al., 2006) and machine learning models (Lee et al., 2007; Mairal et al., 2009), in particular for deep architectures (Lee et al., 2008; Ranzato et al., 2007, 2008). However, in the latter, the neurons end up taking small but non-zero activation or firing probability. We show here that using a rectifying non-linearity gives rise to real zeros of activations and thus truly sparse representations. From a computational point of view, such representations are appealing for the following reasons:

– **Information disentangling**. One of the claimed objectives of deep learning algorithms (Bengio, 2009) is to disentangle the factors explaining the variations in the data. A dense representation is highly entangled because almost any change in the input modifies most of the entries in the representation vector. Instead, if a representation is both sparse and robust to small input changes, the set of non-zero features is almost always roughly conserved by small changes of the input.

– **Efficient variable-size representation.** Different inputs may contain different amounts of information and would be more conveniently represented using a variable-size data-structure, which is common in computer representations of information. *Varying the number of active neurons allows a model to control the effective dimensionality of the representation for a given input and the required precision.*
– **Linear separability**. Sparse representations are also more likely to be linearly separable, or more easily separable with less non-linear machinery, simply because the information is represented in a high-dimensional space. Besides, this can reflect the original data format. In text-related applications for instance, the original raw data is already very sparse (see Section 10.6.4).
– **Distributed but sparse**. Dense distributed representations are the richest representations, being potentially exponentially more efficient than purely local ones (Bengio, 2009). Sparse representations' efficiency is still exponentially greater, with the power of the exponent being the number of non-zero features. They may represent a good trade-off with respect to the above criteria.

Nevertheless, forcing too much sparsity may hurt predictive performance for an equal number of neurons, because it reduces the effective capacity of the model.

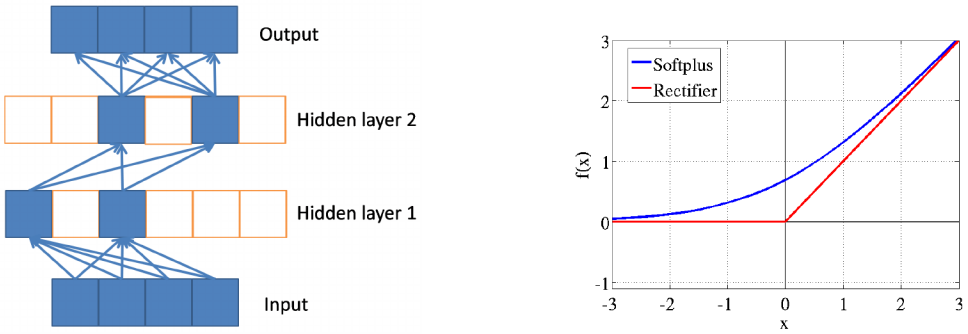

Figure 6.2 – *Left:* **Sparse propagation of activations and gradients in a network of rectifier units.** The input selects a subset of active neurons and computation is linear in this subset. *Right:* **Rectifier and softplus activation functions.** The second one is a smooth version of the first.

6.3 Deep Rectifier Networks

6.3.1 Rectifier Neurons

The neuroscience literature (Bush and Sejnowski, 1995; Douglas and al., 2003) indicates that *cortical neurons are rarely in their maximum saturation regime*, and suggests that their activation function can be approximated by a rectifier. Most previous studies of neural networks involving a rectifying activation function concern recurrent networks (Salinas and Abbott, 1996; Hahnloser, 1998).

The rectifier function $\text{rectifier}(x) = \max(0, x)$ is one-sided and therefore does not enforce a sign symmetry[i] or antisymmetry[i]: instead, the response to the opposite of an excitatory input pattern is 0 (no response). However, we can obtain symmetry or antisymmetry by combining two rectifier units sharing parameters.

Advantages The rectifier activation function allows a network to easily obtain sparse representations. For example, after uniform initialization of the weights, around 50% of hidden units continuous output values are real zeros, and this fraction can easily increase with sparsity-inducing regularization. Apart from being more biologically plausible, sparsity also leads to mathematical advantages (see previous section).

As illustrated in Figure 6.2 (left), the only non-linearity in the network comes from the path selection associated with individual neurons being active or not. For a given input *only a subset of neurons are active*. Computation is *linear* on this subset: once this subset of neurons is selected, the output is a linear function of the input (although a large enough change can trigger a discrete change of the active set of neurons). The function computed by each neuron or by the network output in terms of the network input is thus linear by parts. We can see the model as an *exponential number of linear models that share parameters* (Nair and Hinton, 2010). Because of this linearity, gradients flow well on the active paths of neurons (there is no gradient vanishing effect due to activation non-linearities of sigmoid or tanh units), and mathematical investigation is easier. Computations are also cheaper: there is no need for computing the exponential function in activations, and sparsity can be exploited.

i. The hyperbolic tangent absolute value non-linearity $|\tanh(x)|$ used by Jarrett et al. (2009) enforces sign symmetry. A $\tanh(x)$ non-linearity enforces sign antisymmetry.

Potential Problems One may hypothesize that the hard saturation at 0 may hurt optimization by blocking gradient back-propagation. To evaluate the potential impact of this effect we also investigate the softplus activation: $softplus(x) = log(1 + e^x)$ (Dugas et al., 2001), a smooth version of the rectifying non-linearity. We lose the exact sparsity, but may hope to gain easier training. However, experimental results (see Section 6.4.1) tend to contradict that hypothesis, suggesting that hard zeros can actually help supervised training. We hypothesize that the hard non-linearities do not hurt *so long as the gradient can propagate along some paths*, i.e., that some of the hidden units in each layer are non-zero. With the *credit and blame assigned to these ON units* rather than distributed more evenly, we hypothesize that optimization is easier. Another problem could arise due to the unbounded behavior of the activations; one may thus want to use a regularizer to prevent potential numerical problems. Therefore, we use the L_1 penalty on the activation values, which also promotes additional sparsity. Also recall that, in order to efficiently represent symmetric/antisymmetric behavior in the data, a rectifier network would need twice as many hidden units as a network of symmetric/antisymmetric activation functions.

Finally, rectifier networks are subject to ill-conditioning of the parametrization. Biases and weights can be scaled in different (and consistent) ways while preserving the same overall network function. More precisely, consider for each layer of depth i of the network a scalar α_i, and scaling the parameters as $\mathbf{W}'_i = \frac{\mathbf{W}_i}{\alpha_i}$ and $\mathbf{b}'_i = \frac{\mathbf{b}_i}{\prod_{j=1}^{i} \alpha_j}$. The output units values then change as follow: $\mathbf{s}' = \frac{\mathbf{s}}{\prod_{j=1}^{n} \alpha_j}$. Therefore, as long as $\prod_{j=1}^{n} \alpha_j$ is 1, the network function is identical.

6.3.2 Unsupervised Pre-training

This paper is particularly inspired by the sparse representations learned in the context of auto-encoder variants, as they have been found to be very useful in training deep architectures (Bengio, 2009), especially for unsupervised pre-training of neural networks (Erhan et al., 2010).

Nonetheless, certain difficulties arise when one wants to introduce rectifier activations into stacked denoising auto-encoders (Vincent et al., 2008). First, the hard saturation below the threshold of the rectifier function is not suited for the reconstruction units. Indeed, whenever the network happens to reconstruct a zero in place

of a non-zero target, the reconstruction unit can not backpropagate any gradient. [i] Second, the unbounded behavior of the rectifier activation also needs to be taken into account. In the following, we denote $\tilde{\mathbf{x}}$ the corrupted version of the input $\mathbf{x}$, $\sigma()$ the logistic sigmoid function and θ the model parameters $(W_{enc}, b_{enc}, W_{dec}, b_{dec})$, and define the linear recontruction function as:

$$f(\mathbf{x}, \theta) = W_{dec} \max(\mathbf{x} W_{enc} + \mathbf{b}_{enc}, 0) + \mathbf{b}_{dec} \ .$$

Here are the several strategies we have experimented:

1. Use a softplus activation function for the reconstruction layer, along with a quadratic cost:

$$L(\mathbf{x}, \theta) = ||\mathbf{x} - \log(1 + \exp(f(\tilde{\mathbf{x}}, \theta)))||^2 \ .$$

2. Scale the rectifier activation values coming from the previous encoding layer to bound them between 0 and 1, then use a sigmoid activation function for the reconstruction layer, along with a cross-entropy reconstruction cost.

$$L(\mathbf{x}, \theta) \ = \ -\mathbf{x} \log(\sigma(f(\tilde{\mathbf{x}}, \theta))) - (1 - \mathbf{x}) \log(1 - \sigma(f(\tilde{\mathbf{x}}, \theta))) \ .$$

3. Use a linear activation function for the reconstruction layer, along with a quadratic cost. We tried to use input unit values either before or after the rectifier non-linearity as reconstruction targets. (For the first layer, raw inputs are directly used.)
4. Use a rectifier activation function for the reconstruction layer, along with a quadratic cost.

The first strategy has proven to yield better generalization on image data and the second one on text data. Consequently, the following experimental study presents results using those two.

i. Why is this not a problem for hidden layers too ? we hypothesize that it is because gradients can still flow through the active (non-zero), possibly helping rather than hurting the assignment of credit.

6.4 Experimental Study

This section discusses our empirical evaluation of rectifier units for deep networks. We first compare them to hyperbolic tangent and softplus activations on image benchmarks with and without pre-training, and then apply them to the text task of sentiment analysis.

6.4.1 Image Recognition

Experimental setup We considered the image datasets detailed below. Each of them has a training set (for tuning parameters), a validation set (for tuning hyper-parameters) and a test set (for reporting generalization performance). They are presented according to their number of training/validation/test examples, their respective image sizes, as well as their number of classes:

- MNIST (LeCun et al., 1998): 50k/10k/10k, 28×28 digit images, 10 classes.
- CIFAR10 (Krizhevsky and Hinton, 2009): 50k/5k/5k, $32 \times 32 \times 3$ RGB images, 10 classes.
- NISTP: 81,920k/80k/20k, 32×32 character images from the NIST database 19, with randomized distortions (Bengio and al, 2010), 62 classes. This dataset is much larger and more difficult than the original NIST (Grother, 1995).
- NORB: 233,172/58,428/58,320, taken from Jittered-Cluttered NORB (LeCun et al., 2004). Stereo-pair images of toys on a cluttered background, 6 classes. The data has been preprocessed similarly to (Nair and Hinton, 2010): we subsampled the original $2 \times 108 \times 108$ stereo-pair images to $2 \times 32 \times 32$ and scaled linearly the image in the range $[-1,1]$. We followed the procedure used by Nair and Hinton (2010) to create the validation set.

For all experiments except on the NORB data (LeCun et al., 2004), the models we used are stacked denoising auto-encoders (Vincent et al., 2008) with three hidden layers and 1000 units per layer. The architecture of Nair and Hinton (2010) has been used on NORB: two hidden layers with respectively 4000 and 2000 units. We used a cross-entropy reconstruction cost for tanh networks and a quadratic cost over a softplus reconstruction layer for the rectifier and softplus networks. We chose masking noise as the corruption process: each pixel has a probability of 0.25 of being artificially set to 0. The unsupervised learning rate is constant, and the following values have been explored: $\{.1, .01, .001, .0001\}$. We select the model with

Table 6.1 – Test error on networks of depth 3. Bold results represent statistical equivalence between similar experiments, with and without pre-training, under the null hypothesis of the pairwise test with $p = 0.05$.

Neuron	MNIST	CIFAR10	NISTP	NORB
***With* unsupervised pre-training**				
Rectifier	**1.20%**	**49.96%**	**32.86%**	**16.46%**
Tanh	**1.16%**	**50.79%**	35.89%	17.66%
Softplus	**1.17%**	**49.52%**	**33.27%**	19.19%
***Without* unsupervised pre-training**				
Rectifier	**1.43%**	**50.86%**	**32.64%**	**16.40%**
Tanh	1.57%	52.62%	36.46%	19.29%
Softplus	1.77%	53.20%	35.48%	17.68%

the lowest reconstruction error. For the supervised fine-tuning we chose a constant learning rate in the same range as the unsupervised learning rate with respect to the supervised validation error. The training cost is the negative log likelihood $-\log P(\text{correct class}|\text{input})$ where the probabilities are obtained from the output layer (which implements a softmax logistic regression). We used stochastic gradient descent with mini-batches of size 10 for both unsupervised and supervised training phases.

To take into account the potential problem of rectifier units not being symmetric around 0, we use a variant of the activation function for which half of the units output values are multiplied by -1. This serves to cancel out the mean activation value for each layer and can be interpreted either as inhibitory neurons or simply as a way to equalize activations numerically. Additionally, an L_1 penalty on the activations with a coefficient of 0.001 was added to the cost function during pre-training and fine-tuning in order to increase the amount of sparsity in the learned representations.

Main results Table 6.1 summarizes the results on networks of 3 hidden layers of 1000 hidden units each, comparing all the neuron types[i] on all the datasets, with

i. We also tested a rescaled version of the LIF and $max(tanh(x), 0)$ as activation functions. We obtained worse generalization performance than those of Table 6.1, and chose not to report them.

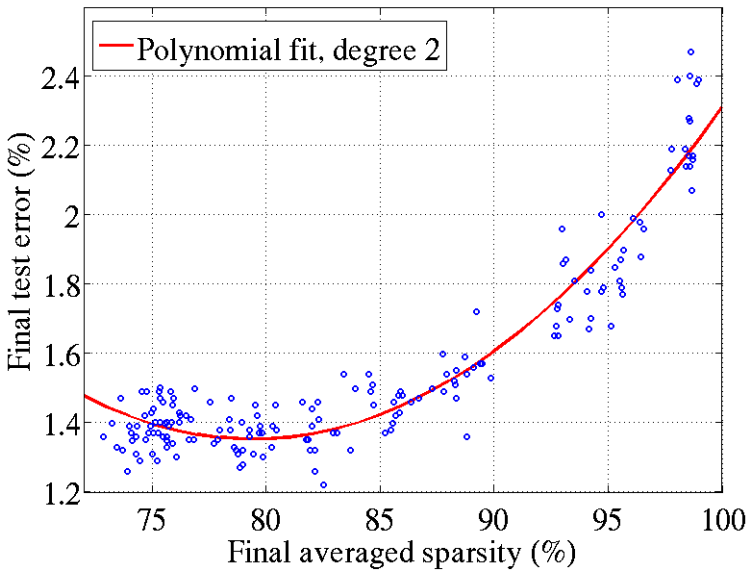

Figure 6.3 – Influence of final sparsity on accuracy. 200 randomly initialized deep rectifier networks were trained on MNIST with various L_1 penalties (from 0 to 0.01) to obtain different sparsity levels. Results show that enforcing sparsity of the activation does not hurt final performance until around 85% of true zeros.

or without unsupervised pre-training. In the latter case, the supervised training phase has been carried out using the same experimental setup as the one described above for fine-tuning. The main observations we make are the following:

- Despite the hard threshold at 0, networks trained with the rectifier activation function can find local minima of greater or equal quality than those obtained with its smooth counterpart, the softplus. On NORB, we tested a rescaled version of the softplus defined by $\frac{1}{\alpha}softplus(\alpha x)$, which allows to interpolate in a smooth manner between the softplus ($\alpha = 1$) and the rectifier ($\alpha = \infty$). We obtained the following α/test error couples: 1/17.68%, 1.3/17.53%, 2/16.9%, 3/16.66%, 6/16.54%, ∞/16.40%. There is no trade-off between those activation functions. Rectifiers are not only biologically plausible, they are also computationally efficient.
- There is almost no improvement when using unsupervised pre-training with rectifier activations, contrary to what is experienced using tanh or softplus. Purely supervised rectifier networks remain competitive on all 4 datasets, even against the pretrained tanh or softplus models.

– Rectifier networks are truly deep sparse networks. There is an average exact sparsity (fraction of zeros) of the hidden layers of 83.4% on MNIST, 72.0% on CIFAR10, 68.0% on NISTP and 73.8% on NORB. Figure 6.3 provides a better understanding of the influence of sparsity. It displays the MNIST test error of deep rectifier networks (without pre-training) according to different average sparsity obtained by varying the L_1 penalty on the activations. Networks appear to be quite robust to it as models with 70% to almost 85% of true zeros can achieve similar performances.

With labeled data, deep rectifier networks appear to be attractive models. They are biologically credible, and, compared to their standard counterparts, do not seem to depend as much on unsupervised pre-training, while ultimately yielding sparse representations.

This last conclusion is slightly different from those reported in (Nair and Hinton, 2010) in which is demonstrated that unsupervised pre-training with Restricted Boltzmann Machines and using rectifier units is beneficial. In particular, the paper reports that pre-trained rectified Deep Belief Networks can achieve a test error on NORB below 16%. However, we believe that our results are compatible with those: we extend the experimental framework to a different kind of models (stacked denoising auto-encoders) and different datasets (on which conclusions seem to be different). Furthermore, note that our rectified model without pre-training on NORB is very competitive (16.4% error) and outperforms the 17.6% error of the non-pretrained model from Nair and Hinton (2010), which is basically what we find with the non-pretrained softplus units (17.68% error).

Semi-supervised setting Figure 6.4 presents results of semi-supervised experiments conducted on the NORB dataset. We vary the percentage of the original labeled training set which is used for the supervised training phase of the rectifier and hyperbolic tangent networks and evaluate the effect of the unsupervised pre-training (using the whole training set, unlabeled). Confirming conclusions of Erhan et al. (2010), the network with hyperbolic tangent activations improves with unsupervised pre-training for any labeled set size (even when all the training set is labeled).

However, the picture changes with rectifying activations. In semi-supervised setups (with few labeled data), the pre-training is highly beneficial. But the more the

labeled set grows, the closer the models with and without pre-training. Eventually, when all available data is labeled, the two models achieve identical performance. Rectifier networks can maximally exploit labeled and unlabeled information.

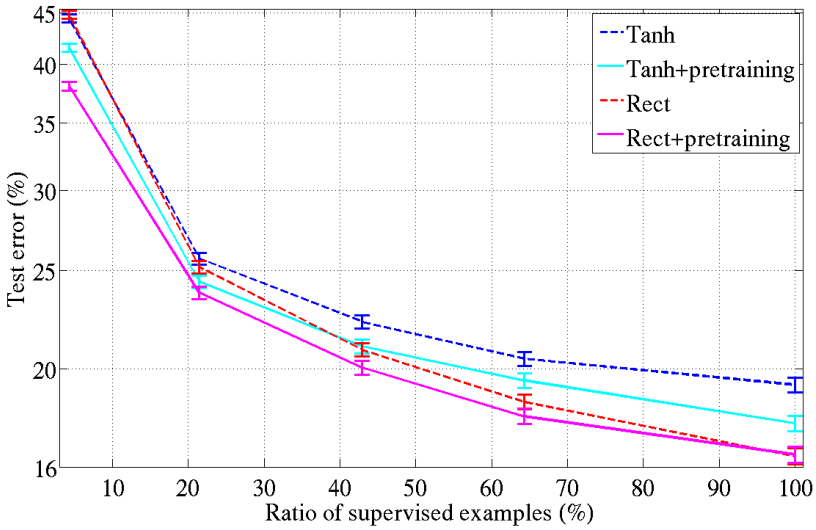

Figure 6.4 – Effect of unsupervised pre-training. On NORB, we compare hyperbolic tangent and rectifier networks, with or without unsupervised pre-training, and fine-tune only on subsets of increasing size of the training set.

6.4.2 Sentiment Analysis

Nair and Hinton (2010) also demonstrated that rectifier units were efficient for image-related tasks. They mentioned the intensity equivariance property (i.e. without bias parameters the network function is linearly variant to intensity changes in the input) as argument to explain this observation. This would suggest that rectifying activation is mostly useful to image data. In this section, we investigate on a different modality to cast a fresh light on rectifier units.

A recent study (Zhou et al., 2010) shows that Deep Belief Networks with binary units are competitive with the state-of-the-art methods for sentiment analysis. This indicates that deep learning is appropriate to this text task which seems therefore ideal to observe the behavior of rectifier units on a different modality, and provide a data point towards the hypothesis that rectifier nets are particarly appropriate for sparse input vectors, such as found in NLP. Sentiment analysis is a text mining area which aims to determine the judgment of a writer with respect to a given topic

Table 6.2 – Examples of restaurant reviews from www.opentable.com dataset. The learner must predict the related rating on a 5 star scale (right column).

Customer's review:	**Rating**
"*Overpriced, food small portions, not well described on menu.*"	⋆
"*Food quality was good, but way too many flavors and textures going on in every single dish. Didn't quite all go together.*"	⋆⋆
"*Calameri was lightly fried and not oily—good job—they need to learn how to make desserts better as ours was frozen.*"	⋆⋆⋆
"*The food was wonderful, the service was excellent and it was a very vibrant scene. Only complaint would be that it was a bit noisy.*"	⋆⋆⋆⋆
"*We had a great time there for Mother's Day. Our server was great! Attentive, funny and really took care of us!*"	⋆⋆⋆⋆⋆

(see (Pang and Lee, 2008) for a review). The basic task consists in classifying the polarity of reviews either by predicting whether the expressed opinions are positive or negative, or by assigning them star ratings on either 3, 4 or 5 star scales.

Following a task originally proposed by Snyder and Barzilay (2007), our data consists of restaurant reviews which have been extracted from the restaurant review site www.opentable.com. We have access to 10,000 labeled and 300,000 unlabeled training reviews, while the test set contains 10,000 examples. The goal is to predict the rating on a 5 star scale and performance is evaluated using Root Mean Squared Error (RMSE).[i] Table 6.2 displays some samples of the dataset. The review text is treated as a bag of words and transformed into binary vectors encoding the presence/absence of terms. For computational reasons, only the 5000 most frequent terms of the vocabulary are kept in the feature set.[ii] The resulting preprocessed data is very sparse: 0.6% of non-zero features on average. Unsupervised pre-training of the networks employs both labeled and unlabeled training reviews while the supervised fine-tuning phase is carried out by 10-fold cross-validation on the labeled

i. Even though our tasks are identical, our database is much larger than the one of (Snyder and Barzilay, 2007).

ii. Preliminary experiments suggested that larger vocabulary sizes did not markedly change results.

training examples.

The model are stacked denoising auto-encoders, with 1 or 3 hidden layers of 5000 hidden units and rectifier or tanh activation, which are trained in a greedy layer-wise fashion. Predicted ratings are defined by the expected star value computed using multiclass (multinomial, softmax) logistic regression output probabilities. For rectifier networks, when a new layer is stacked, activation values of the previous layer are scaled within the interval [0,1] and a sigmoid reconstruction layer with a cross-entropy cost is used. We also add an L_1 penalty to the cost during pre-training and fine-tuning. Because of the binary input, we use a "salt and pepper noise" (i.e. masking some inputs by zeros and others by ones) for unsupervised training of the first layer. A zero masking (as in (Vincent et al., 2008)) is used for the higher layers. We selected the noise level based on the classification performance, other hyperparameters are selected according to the reconstruction error.

Table 6.3 – Test RMSE and sparsity level obtained by 10-fold cross-validation on OpenTable data.

Network	**RMSE**	**Sparsity**
No hidden layer	0.885 ± 0.006	99.4% ± 0.0
Rectifier (1-layer)	0.807 ± 0.004	28.9% ± 0.2
Rectifier (3-layers)	**0.746** ± 0.004	53.9% ± 0.7
Tanh (3-layers)	0.774 ± 0.008	00.0% ± 0.0

Results are displayed in Table 6.3. Interestingly, the RMSE significantly decreases as we add hidden layers to the rectifier neural net. These experiments confirm that rectifier networks improve after an unsupervised pre-training phase in a semi-supervised setting: with no pre-training, the 3-layers model can not obtain a RMSE lower than 0.833. Additionally, although we can not replicate the original very high degree of sparsity of the training data, the 3-layers network can still attain an overall sparsity of more than 50%. Finally, on data with these particular properties (binary, high sparsity), the 3-layers network with tanh activation function (which has been learnt with the exact same pre-training+fine-tuning setup) is clearly outperformed. The sparse behavior of the deep rectifier network seems particularly suitable in this case, because the raw input is very sparse and *varies in its number of non-zeros*. The latter can also be achieved with sparse internal

representations, not with dense ones.

Since no result has ever been published on the OpenTable data, we applied our model on the Amazon sentiment analysis benchmark (Blitzer et al., 2007) in order to assess the quality of our network with respect to literature methods. This dataset proposes reviews of 4 kinds of Amazon products, for which the polarity (positive or negative) must be predicted. We followed the experimental setup defined by Zhou et al. (2010). In their paper, the best model achieves a test accuracy of 73.72% (on average over the 4 kinds of products) where our 3-layers rectifier network obtains 78.95%.

6.5 Conclusion

Sparsity and neurons operating mostly in a linear regime can be brought together in more biologically plausible deep neural networks. Rectifier units help to bridge the gap between unsupervised pre-training and no pre-training, which suggests that they may help in finding better minima during training. This finding has been verified for four image classification datasets of different scales and all this in spite of their inherent problems, such as zeros in the gradient, or ill-conditioning of the parametrization. Rather sparse networks are obtained (from 50 to 80% sparsity for the best generalizing models, whereas the brain is hypothesized to have 95% to 99% sparsity), which may explain some of the benefit of using rectifiers.

Furthermore, rectifier activation functions have shown to be remarkably adapted to sentiment analysis, a text-based task with a very large degree of data sparsity. This promising result tends to indicate that deep sparse rectifier networks are not only beneficial to image classification tasks and might yield powerful text mining tools in the future.

7 Introduction de l'article 3

Xavier Glorot, Antoine Bordes and Yoshua Bengio. Domain Adaptation for Large-Scale Sentiment Classification: A Deep Learning Approach, in: *Proceedings of the Twenty-eight International Conference on Machine Learning (ICML)*, pages 97-110, 2011

7.1 Contexte

Dans le contexte du traitement des langues naturelles, la notion de domaine, que nous avons présenté à la section 1.6.3, revêt une importance particulière. En effet, suivant la langue, le sujet ou l'auteur du texte, des phrases ayant le même sens peuvent prendre des formes différentes. L'étiquetage des données étant coûteux, il est primordial de pouvoir transférer l'apprentissage d'un domaine à un autre.

Dans le cas général d'un support partiellement partagé, l'apprentissage d'une représentation commune aux deux domaines est une solution privilégiée. Cependant, les solutions qui existaient au moment de l'écriture de l'article utilisaient des modèles linéaires et étaient déployées sur des petits ensembles de données (de l'ordre de 20k exemples) alors qu'en pratique, il est possible d'accéder à beaucoup plus.

Il était donc intéressant de développer une solution permettant d'utiliser une grande quantité de données non-supervisées, provenant possiblement de plusieurs domaines différents (et pas seulement des domaines source et cible) pour améliorer les performances en transfert et nous approcher du contexte d'une application industrielle réelle. Nous voulions également nous affranchir d'une capacité linéaire en utilisant les modèles non-linéaires utilisés pour le pré-entraînement des architectures profondes.

7.2 Contribution

Suite au succès des DAEs sur la tâche de l'analyse de sentiment, présenté au chapitre 6, nous avons décidé avec Antoine Bordes de tester ces représentations pour l'adaptation de domaine. Ce dernier a pré-traité les données et effectué l'évaluation des T-SVM. De mon côté, j'ai réalisé toutes les autres expériences. L'hypothèse du démêlement des facteurs de variations du domaine et du sentiment, ainsi que l'expérience pour le mettre en évidence sont ma contribution. Il en est de même des différentes mesures pour quantifier les performances en transfert. L'écriture de l'article s'est faite conjointement avec Antoine Bordes. Yoshua Bengio a participé au développement des idées.

7.3 Impact

À notre connaissance, ce papier est encore de nos jours l'état-de-l'art sur cet ensemble de données. Suite à cette étude, plusieurs travaux ont utilisés des méthodes similaires pour l'adaptation de domaine pour d'autres modalitées, comme le traitement de la parole (Deng et al., 2014; Zhang and Wu, 2013) et la reconnaissance d'objets (Ghifary et al., 2014).

Chen et al. (2012) a reproduit les résultats de notre étude en simplifiant de manière importante l'algorithme d'apprentissage. Afin de démontrer la force de notre méthode, nous avons effectué une expérience supplémentaire en entraînant un modèle utilisant 25000 entrées sur le grand Amazon, nous avons ensuite déployé ce dernier sur le petit Amazon et nous avons constaté que le modèle pouvait tirer parti d'encore plus de données comme le montre la figure 7.1, obtenant même un ratio de transfert proche de 1.

L'avantage du démêlement partiel des facteurs de variations pour l'adaptation de domaine est aussi un résultat important. Sa mise en évidence pour des algorithmes d'entraînement non-supervisé génériques nous encourage dans cette voie.

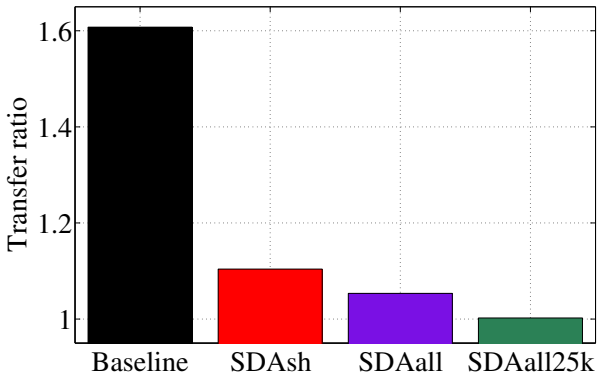

Figure 7.1 – Ratio de transfert sur le petit Amazon, SDA_{sh} est la solution de l'article, SDA_{all} est entraîné sur le grand Amazon avec 5000 dimensions, et SDA_{all25k} avec 25000 dimensions.

8 Domain Adaptation for Large-Scale Sentiment Classification: A Deep Learning Approach

Abstract

The exponential increase in the availability of online reviews and recommendations makes sentiment classification an interesting topic in academic and industrial research. Reviews can span so many different domains that it is difficult to gather annotated training data for all of them. Hence, this paper studies the problem of domain adaptation for sentiment classifiers, hereby a system is trained on labeled reviews from one source domain but is meant to be deployed on another. We propose a deep learning approach which learns to extract a meaningful representation for each review in an unsupervised fashion. Sentiment classifiers trained with this high-level feature representation clearly outperform state-of-the-art methods on a benchmark composed of reviews of 4 types of Amazon products. Furthermore, this method scales well and allowed us to successfully perform domain adaptation on a larger industrial-strength dataset of 22 domains.

8.1 Introduction

With the rise of social media such as blogs and social networks, reviews, ratings and recommendations are rapidly proliferating; being able to automatically filter them is a current key challenge for businesses looking to sell their wares and identify new market opportunities. This has created a surge of research in sentiment classification (or sentiment analysis), which aims to determine the judgment of a writer with respect to a given topic based on a given textual comment. Sentiment analysis is now a mature machine learning research topic, as illustrated with this review Pang and Lee (2008). Applications to many different domains have been presented, ranging from movie reviews Pang et al. (2002) and congressional floor

debates Thomas et al. (2006) to product recommendations Snyder and Barzilay (2007); Blitzer et al. (2007).

This large variety of data sources makes it difficult and costly to design a robust sentiment classifier. Indeed, reviews deal with various kinds of products or services for which vocabularies are different. For instance, consider the simple case of training a system analyzing reviews about only two sorts of products: *kitchen appliances* and *DVDs*. One set of reviews would contain adjectives such as "malfunctioning", "reliable" or "sturdy", and the other "thrilling", "horrific" or "hilarious", etc. Therefore, data distributions are different across domains. One solution could be to learn a different system for each domain. However, this would imply a huge cost to annotate training data for a large number of domains and prevent us from exploiting the information shared across domains. An alternative strategy, evaluated here, consists in learning a single system from the set of domains for which labeled and unlabeled data are available and then apply it to any target domain (labeled or unlabeled). This only makes sense if the system is able to discover intermediate abstractions that are shared and meaningful across domains. This problem of training and testing models on different distributions is known as domain adaptation (Daumé III and Marcu, 2006).

In this paper, we propose a Deep Learning approach for the problem of domain adaptation of sentiment classifiers. The promising new area of Deep Learning has emerged recently ; see Bengio (2009) for a review. Deep Learning is based on algorithms for **discovering intermediate representations** built in a hierarchical manner. Deep Learning relies on the discovery that unsupervised learning could be used to set each level of a hierarchy of features, one level at a time, based on the features discovered at the previous level. These features have successfully been used to initialize deep neural networks Hinton and Salakhutdinov (2006); Hinton et al. (2006); Bengio et al. (2007). Imagine a probabilistic graphical model in which we introduce latent variables which correspond to the true explanatory factors of the observed data. It is likely that answering questions and learning dependencies in the space of these latent variables would be easier than answering questions about the raw input. A simple linear classifier or non-parametric predictor trained from as few as one or a few examples might be able to do the job. The key to achieving this is learning better representations, mostly from unlabeled data: how this is done is what differentiates Deep Learning algorithms.

The Deep Learning system we introduce in Section 8.3 is designed to use unlabeled data to extract high-level features from reviews. We show in Section 10.5 that sentiment classifiers trained with these learnt features can: (i) surpass state-of-the-art performance on a benchmark of 4 kinds of products and (ii) successfully perform domain adaptation on a large-scale data set of 22 domains, beating all of the baselines we tried.

8.2 Domain Adaptation

Domain adaptation considers the setting in which the training and testing data are sampled from different distributions. Assume we have two sets of data: a *source* domain S providing labeled training instances and a *target* domain T providing instances on which the classifier is meant to be deployed. We do not make the assumption that these are drawn from the same distribution, but rather that S is drawn from a distribution p_S and T from a distribution p_T. The learning problem consists in finding a function realizing a good *transfer* from S to T i.e. it is trained on data drawn from p_S and generalizes well on data drawn from p_T.

Deep Learning algorithms learns intermediate concepts between raw input and target. Our intuition for using it in this setting is that these intermediate concepts could yield better transfer across domains. Suppose for example that these intermediate concepts indirectly capture things like product quality, product price, customer service, etc. Some of these concepts are general enough to make sense across a wide range of domains (corresponding to products or services, in the case of sentiment analysis). Because the same words or tuples of words may be used across domains to indicate the presence of these higher-level concepts, it should be possible to discover them. Furthermore, because Deep Learning exploits unsupervised learning to discover these concepts, one can exploit the large amounts of unlabeled data across all domains to learn these intermediate representations. Here, as in many other Deep Learning approaches, we do not engineer what these intermediate concepts should be, but instead use generic learning algorithms to discover them.

8.2.1 Related Work

Learning setups relating to domain adaptation have been proposed before and published under different names. Daumé III and Marcu (2006) formalized the problem and proposed an approach based on a mixture model. A general way to address domain adaptation is through instance weighting, in which instance-dependent weights are added to the loss function Jiang and Zhai (2007). Another solution to domain adaptation can be to transform the data representations of the source and target domains so that they present the same joint distribution of observations and labels. Ben-David et al. (2007) formally analyze the effect of representation change for domain adaptation while Blitzer et al. (2006) propose the Structural Correspondence Learning (SCL) algorithm that makes use of the unlabeled data from the target domain to find a low-rank joint representation of the data.

Finally, domain adaptation can be simply treated as a standard semi-supervised problem by ignoring the domain difference and considering the source instances as labeled data and the target ones as unlabeled data Dai et al. (2007). In that case, the framework is very close to that of self-taught learning (Raina et al., 2007), in which one learns from labeled examples of some categories as well as unlabeled examples from a larger set of categories. The approach of Raina et al. (2007) relies crucially on the unsupervised learning of a representation, like the approach proposed here.

8.2.2 Applications to Sentiment Classification

Sentiment analysis and domain adaptation are closely related in the literature, and many works have studied domain adaptation exclusively for sentiment analysis. Among those, a large majority propose experiments performed on the benchmark made of reviews of Amazon products gathered by Blitzer et al. (2007).

Amazon data The data set proposes more than 340,000 reviews regarding 22 different product types[i] and for which reviews are labeled as either positive or negative. As detailed in Table 8.1 (top), there is a vast disparity between domains in the total number of instances and in the proportion of negative examples.

i. The data are available from http://www.cs.jhu.edu/~mdredze/datasets/sentiment/. It is actually composed of 25 domains but we removed 3 of them which were very small (less than 400 instances in total).

Since this data set is heterogeneous, heavily unbalanced and large-scale, a smaller and more controlled version has been released. The reduced data set contains 4 different domains: *Books*, *DVDs*, *Electronics* and *Kitchen* appliances. There are 1000 positive and 1000 negative instances for each domain, as well as a few thousand unlabeled examples. The positive and negative examples are also exactly balanced (see the bottom section of Table 8.1 for details). This latter version is used as a benchmark in the literature. To the best of our knowledge, this paper will contain the first published results on the large Amazon dataset.

Compared Methods In the original paper regarding the smaller 4-domain benchmark dataset, Blitzer et al. (2007) adapt Structural Correspondence Learning (SCL) for sentiment analysis. Li and Zong (2008) propose the Multi-label Consensus Training (MCT) approach which combines several base classifiers trained with SCL. Pan et al. (2010) first use a Spectral Feature Alignment (SFA) algorithm to align words from different source and target domains to help bridge the gap between them. These 3 methods serve as comparisons in our empirical evaluation.

8.3 Deep Learning Approach

8.3.1 Background

If Deep Learning algorithms are able to capture, to some extent, the underlying generative factors that explain the variations in the input data, what is really needed to exploit that ability is for the learned representations to help in **disentangling the underlying factors of variation**. The simplest and most useful way this could happen is if some of the features learned (the individual elements of the learned representation) are mostly related to only some of these factors, perhaps only one. Conversely, it would mean that such features would have *invariant properties*, i.e., they would be highly specific in their response to a subset (maybe only one) of these factors of variation and insensitive to the others. This hypothesis was tested by Goodfellow et al. (2009), for images and geometric invariances associated with movements of the camera.

It is interesting to evaluate Deep Learning algorithms on sentiment analysis for several reasons. First, if they can extract features that somewhat disentangle the underlying factors of variation, this would likely help to perform transfer across domains, since we expect that there exist generic concepts that characterize product reviews across many domains. Second, for our Amazon datasets, we know some of these factors (such as whether or not a review is about a particular product, or is a positive appraisal for that product), so we can use this knowledge to quantitatively check to what extent they are disentangled in the learned representation: domain adaptation for sentiment analysis becomes a medium for better understanding deep architectures. Finally, even though Deep Learning algorithms have not yet been evaluated for domain adaptation of sentiment classifiers, several very interesting results have been reported on other tasks involving textual data, beating the previous state-of-the-art in several cases (Salakhutdinov and Hinton, 2007; Collobert and Weston, 2008; Ranzato et al., 2007).

8.3.2 Stacked Denoising Auto-encoders

The basic framework for our models is the Stacked Denoising Auto-encoder (Vincent et al., 2008). An auto-encoder is comprised of an encoder function $h(\cdot)$ and a decoder function $g(\cdot)$, typically with the dimension of $h(\cdot)$ smaller than that of its argument. The reconstruction of input $\mathbf{x}$ is given by $r(\mathbf{x}) = g(h(\mathbf{x}))$, and auto-encoders are typically trained to minimize a form of reconstruction error $loss(\mathbf{x}, r(\mathbf{x}))$. Examples of reconstruction error include the squared error, or like here, when the elements of $\mathbf{x}$ or $r(\mathbf{x})$ can be considered as probabilities of a discrete event, the Kullback-Liebler divergence between elements of $\mathbf{x}$ and elements of $r(\mathbf{x})$. When the encoder and decoder are linear and the reconstruction error is quadratic, one recovers in $h(\mathbf{x})$ the space of the principal components (PCA) of $\mathbf{x}$. Once an auto-encoder has been trained, one can stack another auto-encoder on top of it, by training a second one which sees the encoded output of the first one as its training data. Stacked auto-encoders were one of the first methods for building deep architectures (Bengio et al., 2007), along with Restricted Boltzmann Machines (RBMs) (Hinton et al., 2006). Once a stack of auto-encoders or RBMs has been trained, their parameters describe multiple levels of representation for x and can be used to initialize a supervised deep neural network Bengio (2009) or

directly feed a classifier, as we do in this paper.

An interesting alternative to the ordinary auto-encoder is the Denoising Auto-encoder (Vincent et al., 2008) or DAE, in which the input vector $\mathbf{x}$ is stochastically corrupted into a vector $\tilde{\mathbf{x}}$, and the model is trained to *denoise*, i.e., to minimize a denoising reconstruction error $loss(\mathbf{x}, r(\tilde{\mathbf{x}}))$. Hence the DAE cannot simply copy its input $\tilde{\mathbf{x}}$ in its code layer $h(\tilde{\mathbf{x}})$, even if the dimension of $h(\tilde{\mathbf{x}})$ is greater than that of $\tilde{\mathbf{x}}$. The denoising error can be linked in several ways to the likelihood of a generative model of the distribution of the uncorrupted examples x Vincent (2011).

8.3.3 Proposed Protocol

In our setting we have access to unlabeled data from various domains, and to the labels for one source domain only. We tackle the problem of domain adaptation for sentiment classifiers with a two-step procedure.

First, a higher-level feature extraction is learnt in an unsupervised fashion from the text reviews of all the available domains using a Stacked Denoising Auto-encoder (SDA) with rectifier units (i.e. $max(0, x)$) for the code layer. RBMs with (soft) rectifier units have been introduced in Nair and Hinton (2010). We have used such units because they have been shown to outperform other non-linearities on a sentiment analysis task (Glorot et al., 2011). The SDA is learnt in a greedy layer-wise fashion using stochastic gradient descent. For the first layer, the non-linearity of the decoder is the logistic sigmoid, the corruption process is a masking noise (i.e. each active input has a probability P to be set to 0) [i] and the training criterion is the Kullback-Liebler divergence. The rectifier non-linearity is too hard to be used on "output" units: reconstruction error gradients would not flow if the reconstruction was 0 (argument of the rectifier is negative) when the target is positive. For training the DAEs of upper layers, we use the softplus activation function (i.e. $\log(1 + \exp(x))$, a smooth version of the rectifier) as non-linearity for the decoder output units. We also use the squared error as reconstruction error criterion and a Gaussian corruption noise, which is added *before* the rectifier non-linearity of the input layer in order to keep the sparsity of the representation. The code layer activations (after the rectifier), at different depths, define the new representations

i. We also tried to set inactive inputs to 1 with a different probability but we did not observe any improvement.

In a second step, a linear classifier is trained on the transformed labeled data of the source domain. Support Vector Machines (SVM) being known to perform well on sentiment classification (Pang et al., 2002), we use a linear SVM with squared hinge loss. This classifier is eventually tested on the target domain(s).

8.3.4 Discussion

The previous protocol exhibits appealing properties for domain adaptation of sentiment classifiers.

Existing domain adaptation methods for sentiment analysis focus on the information from the source and target distributions, whereas the SDA unsupervised learning can use data from other domains, *sharing the representation across all those domains*. This also reduces the computation required to transfer to several domains because a single round of unsupervised training is required, and allows us to *scale well with large amount of data* and consider real-world applications.

The code learned by the SDA is a *non-linear mapping of the input* and can therefore encode complex data variations. To the best of our knowledge, existing domain adaptation methods for sentiment analysis map inputs into a new or an augmented space using only linear projections. Furthermore, rectifier non-linearities have the the nice ability to naturally provide *sparse representations* (with exact zeros) for the code layer, which are well suited to linear classifiers and are efficient with respect to computational cost and memory use.

8.4 Empirical Evaluation

8.4.1 Experimental Setup

For both data sets, the preprocessing corresponds to the setting of (Blitzer et al., 2007): each review text is treated as a bag-of-words and transformed into binary vectors encoding the presence/absence of unigrams and bigrams. For computational reasons, only the 5000 most frequent terms of the vocabulary of unigrams and bigrams are kept in the feature set. We use the train/test splits given in Table 8.1. For all experiments, the *baseline* is a linear SVM trained on the raw data whereas

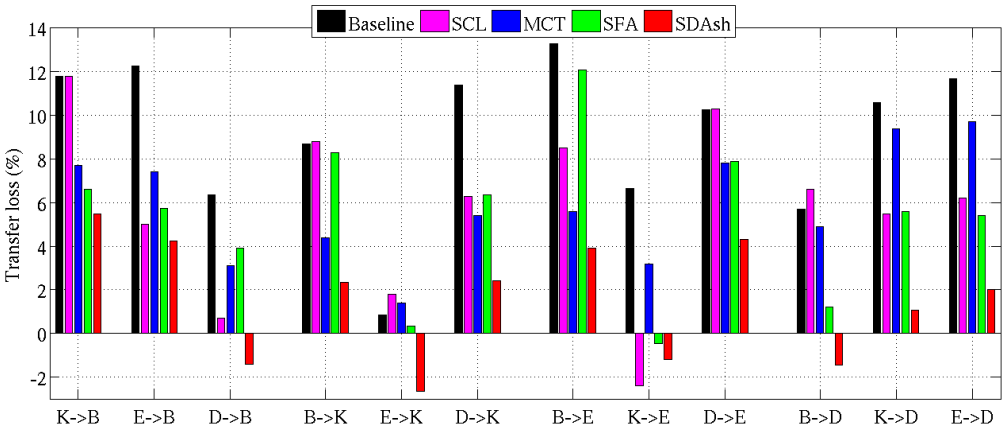

Figure 8.1 – **Transfer losses on the Amazon benchmark** of 4 domains: *Kitchen*(K), *Electronics*(E), *DVDs*(D) and *Books*(B). All methods are trained on the labeled set of one domain and evaluated on the test sets of the others. SDA_{sh} outperforms all others on 11 out of 12 cases.

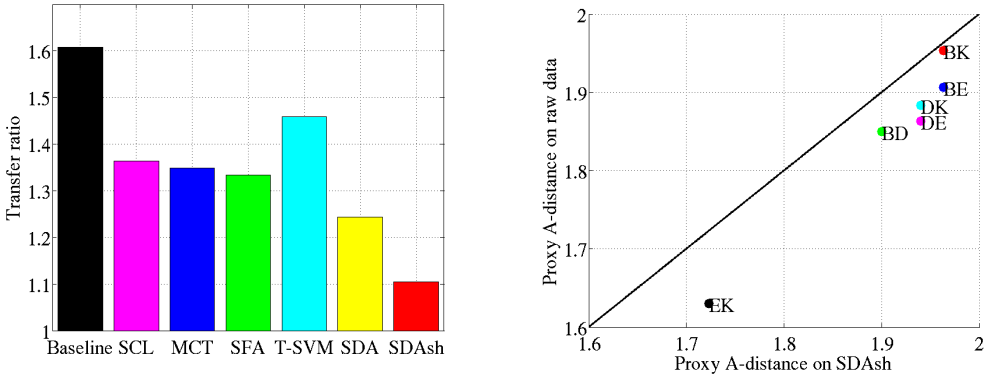

Figure 8.2 – *Left:* **Transfer ratios** on the Amazon benchmark. Both SDA-based systems outperforms the rest even if SDA_{sh} is better. *Right:* **Proxy A-distances** between domains of the Amazon benchmark for the 6 different pairs. Transforming data with SDA_{sh} increases the proxy A-distance.

our method, denoted SDA$_{sh}$, corresponds to the same kind of SVM but trained and tested on data for which features have been transformed by the system described in Section 8.3. The hyper-parameters of all SVMs are chosen by cross-validation on the training set.

For SDA$_{sh}$, we explored an extensive set of hyper-parameters: a masking noise probability in $\{0.0, 0.3, 0.5, 0.6, 0.7, 0.8, 0.9\}$, (its optimal value was usually high: 0.8) ; a Gaussian noise standard deviation for upper layers in $\{0.01, 0.1, 0.25, 0.5, 1\}$; a size of hidden layers in $\{1000, 2500, 5000\}$, (5000 always gave the best performance) ; a L_1 regularization penalty on the activations in $\{0.0, 10^{-8}, 10^{-5}, 10^{-3}, 10^{-2}\}$; a learning rate in $\{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}\}$. All values were selected w.r.t. the averaged in-domain validation error. All algorithms were implemented using the Theano library (Bergstra et al., 2010).

8.4.2 Metrics

We denote by $e(S, T)$, the *transfer error*, defined as the test error obtained by a method trained on the source domain S and tested on the target domain T ($e(T, T)$ is termed the *in-domain error*). The main point of comparison in domain adaptation is the *baseline in-domain error*, denoted $e_b(T, T)$, which corresponds to the test error obtained by the baseline method, i.e. a linear SVM on raw features trained and tested on the raw features of the target domain.

With these definitions, we can define the standard domain adaptation metric: the **transfer loss** t. It is the difference between the transfer error and the in-domain baseline error i.e. $t(S, T) = e(S, T) - e_b(T, T)$ for a source domain S and a target domain T.

Unfortunately, when one deals with a large number of heterogeneous domains with different difficulties (as with the large Amazon data), the transfer loss is not satisfactory. In addition, taking its mean over all possible couples of source-target domains is uninformative. Hence, we also introduce the following metrics:

– **Transfer ratio** $\mathcal{Q}$: it also characterizes the transfer but is defined by replacing the difference by a quotient in t because this is less sensitive to important variations of in-domain errors, and thus more adapted to averaging. We report its mean over all source-target couples of the data set: $\mathcal{Q} = \frac{1}{n} \sum_{(S,T)_{S \neq T}} \frac{e(S,T)}{e_b(T,T)}$ (with n the number of couples (S, T) with $S \neq T$).

– **In-domain ratio** $\mathcal{I}$: some domain adaptation methods, like ours, transform the feature representation of all the domains, including the source. Thus in-domain errors of such methods are different from those of the baseline. The in-domain ratio measures this and is defined by: $\mathcal{I} = \frac{1}{m} \sum_S \frac{e(T,T)}{e_b(T,T)}$ (with m the total number of domains).

8.4.3 Benchmark Experiments

On the benchmark of 4 domains, we compare our domain adaptation protocol with the 3 methods from the literature introduced in Section 8.2.2: SCL, SFA and MCT. We report the results from the original papers, which have been obtained using the whole feature vocabulary and on different splits, but of identical sizes as ours. From our experience, results are consistent whatever the train/test splits as long as set sizes are preserved. Hence, one can check that all baselines achieve similar performances. We also report results obtained by a Transductive SVM (Sindhwani and Keerthi, 2006) trained in a standard semi-supervised setup: the training set of the source domain is used as labeled set, and the training set of the other domains as the unlabeled set. [i] On this data set with a relatively small number of training instances, our unsupervised feature extractor is made of a single layer of 5000 units.

Main results Figure 8.1 depicts the transfer loss for all methods and for all source-target domain pairs. The best transfer is achieved by the SVM trained on our transformed features in 11 out of 12 cases (SCL is only slightly better in *Kitchen* $\rightarrow$ *Electronics*) and significantly better for 8 cases. Interestingly, for each target domain, there is one case of negative transfer loss for SDA_{sh}: an SVM trained on a different domain can outperform an SVM trained on the target domain because of the quality of our features.

Figure 8.2 (left) depicts the transfer ratio for the same methods plus the transductive SVM (T-SVM) and a second version of our system denoted SDA. Contrary to SDA_{sh}, the unsupervised training of SDA has not been performed on all the available domains but on couples, as does SCL: for instance, to transfer from *Books* to *DVD*, the feature extractor of SDA is trained on reviews from these 2 domains

i. Non-linear models (MLPs) were also investigated, with a similar protocol as presented in Section 8.4.4, but they did not reach the performance in transfer of the baseline. We believe this is due to the small training set size.

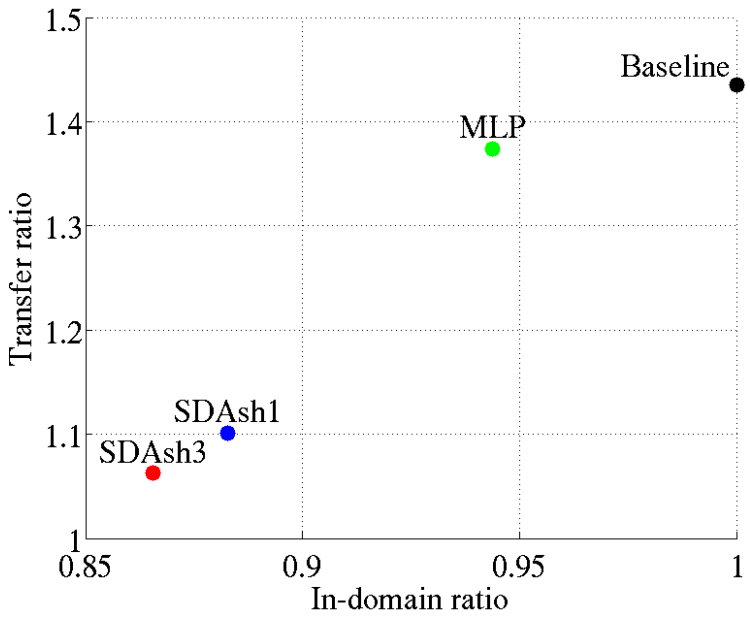

Figure 8.3 – Transfer ratios according to in-domain ratios on the large-scale Amazon data. Systems based on SDA$_{sh}$ are better for both metrics and depth helps.

only. The transfer ratio for SDA being higher than for SDA$_{sh}$, we can conclude that sharing the unsupervised pre-training across all domains (even on those which are not directly concerned) is beneficial, as expected. Figure 8.2 also shows that the combination of an unsupervised and a supervised phase performed by SDA$_{sh}$ (and SDA) outperforms the pure semi-supervised T-SVM. In term of absolute classification performance, we obtained the following averaged transfer generalization errors: Baseline - 24.3%, SFA - 21.3%, SDA$_{sh}$ - 16.7%.

A-distance The A-distance is a measure of similarity between two probability distributions. Ben-David et al. (2007) showed that the A-distance between the source and target distributions is a crucial part of an upper generalization bound for domain adaptation. They hypothesized that it should be difficult to discriminate between the source and target domains in order to have a good transfer between them, because this would imply similar feature distributions. In practice, computing the exact A-distance is impossible and one has to compute a proxy. Hence, we measured the generalization error ϵ of a linear SVM classifier trained to discriminate

between two domains. Our proxy for the A-distance is then defined as $\hat{d}_A = 2(1 - 2\epsilon)$.

Figure 8.2 (right) reports the results for each pair of domains. Surprisingly, $\hat{d}_A$ is *increased* in the new feature space: domain recognition is improved by the unsupervised feature extraction of SDA$_{sh}$. Consequently, following Ben-David et al. (2007), the representation of SDA$_{sh}$ should hurt transfer, but we also observe an improvement (see Figure 8.1). An explanation could be that the unsupervised feature extraction disentangles domain specific and sentiment polarity information.

To test this hypothesis, we trained an L_1-regularized SVM to select the most relevant features on 6 domain recognition tasks (one per domain pair), and 5 sentiment analysis tasks (one per domain plus all domains together). Figure 8.4 shows a histogram of the number of tasks associated with individual features, separated into the number of domain vs sentiment tasks. The color level at coordinate (n, m) indicates the number of features that have been re-used for n sentiment analysis tasks and for m domain recognition tasks. Comparing graphs obtained using raw data and data transformed by the SDA$_{sh}$ confirms our hypothesis: relevant features for domain recognition and sentiment analysis are far less overlapping in the latter case. Indeed, a complete feature disentangling would lead to a graph for which only the first column and the bottom line would be colored, indicating that each feature is either used for domain recognition *or* for sentiment classification, but not both. Transforming raw data with SDA$_{sh}$ brings features closer to that pattern.

8.4.4 Large-Scale Experiments

We now present results obtained on the larger version of the data. These conditions are more realistic and a better representation of the real-world than those of the previous experiments: more domains with different and larger sizes, different ratios between positive and negative examples, etc. We compare 3 methods in addition to the baseline: our feature extractor with either one (SDA$_{sh}$1) or 3 layers (SDA$_{sh}$3) of 5000 units, and a multi-layer perceptron (MLP) with the following architecture: a softmax logistic regression on top of one hidden layer with 5000 hyperbolic tangent units.

Figure 8.3 presents the transfer ratio of each model according to their in-domain ratio. Those results correspond to the following averaged transfer generalization

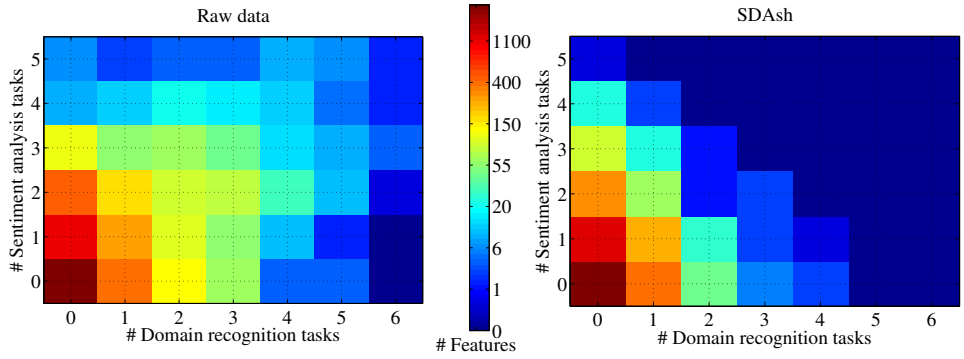

Figure 8.4 – L_1 **feature selection on the Amazon benchmark.** Both graphs depict the number of tasks of domain recognition (x-axis) and sentiment analysis (y-axis) in which a feature is re-used by L1-classifiers trained on raw features (*left*) or features transformed by SDA$_{sh}$. (*right*). See Section 8.4.3 for details.

errors: (Baseline) - 14.5%, (MLP) - 13.9%, (SDA$_{sh}$1) - 11.5% and (SDA$_{sh}$3) - 10.9%. Despite the large number of domains and their heterogeneity, there is a significant improvement for both SDA systems. The performance of the MLP shows that the non-linearity helps but is not sufficient to gather all necessary information from data: one needs an unsupervised phase which can encompass data from all domains. One can also verify that, on this large-scale problem, a single layer is not enough to reach optimal performance. Stacking 3 layers yields the best representation from the data. It is worth noting that the improvement of SDA$_{sh}$3 compared to the baseline is higher on the y-axis than on the x-axis: the representation learnt by SDA$_{sh}$3 is more beneficial for transfer than in-domain and is thus truly tailored to domain adaptation.

8.5 Conclusion

This paper has demonstrated that a Deep Learning system based on Stacked Denoising Auto-Encoders with sparse rectifier units can perform an unsupervised feature extraction which is highly beneficial for the domain adaptation of sentiment classifiers. Indeed, our experiments have shown that linear classifiers trained with this higher-level learnt feature representation of reviews outperform the current

state-of-the-art. Furthermore, we have been able to successfully perform domain adaptation on an industrial-scale dataset of 22 domains, where we significantly improve generalization over the baseline and over a similarly structured but purely supervised alternative.

Acknowledgments

This work was supported by DARPA DL Program, CRSNG, MITACS, RQCHP and SHARCNET.

Table 8.1 – Amazon data statistics. This table depicts the number of training, testing and unlabeled examples for each domain, as well as the portion of negative training examples for both versions of the data set.

Domain	Train size	Test size	Unlab. size	% Neg. ex
Complete (large-scale) data set				
Toys	6318	2527	3791	19.63%
Software	1032	413	620	37.77%
Apparel	4470	1788	2682	14.49%
Video	8694	3478	5217	13.63%
Automotive	362	145	218	20.69%
Books	10625	10857	32845	12.08%
Jewelry	982	393	589	15.01%
Grocery	1238	495	743	13.54%
Camera	2652	1061	1591	16.31%
Baby	2046	818	1227	21.39%
Magazines	1195	478	717	22.59%
Cell	464	186	279	37.10%
Electronics	10196	4079	6118	21.94%
DVDs	10625	9218	26245	14.16%
Outdoor	729	292	437	20.55%
Health	3254	1301	1952	21.21%
Music	10625	24872	88865	8.33%
Videogame	720	288	432	17.01%
Kitchen	9233	3693	5540	20.96%
Beauty	1314	526	788	15.78%
Sports	2679	1072	1607	18.75%
Food	691	277	415	13.36%
(Smaller-scale) benchmark				
Books	1600	400	4465	50%
Kitchen	1600	400	5945	50%
Electronics	1600	400	5681	50%
DVDs	1600	400	3586	50%

9 Introduction de l'article 4

Antoine Bordes, Xavier Glorot, Jason Weston et Yoshua Bengio. A Semantic Matching Energy Function for Learning with Multi-relational Data. In *Machine Learning. Springer, DOI: 10.1007/s10994-013-5363-6*, Mai 2013.

9.1 Contexte

Les données multi-relationnelles sont des graphes dont les nœuds représentent des entités et les arcs représentent des relations reliant ces entités. Dans de nombreux domaines, les données peuvent être représentées de cette façon, comme pour les sytèmes de recommandation, le web sémantique, la bio-informatique, les bases de connaissances ou pour le Traitement Automatique des Langues Naturelles. Comme nous l'avons déjà mentionné, les modèles à base d'"embeddings" sont utilisés avec succès dans ce dernier domaine et permettent un apprentissage à grande échelle. L'objectif de ce travail était d'expérimenter l'utilisation de tels modèles pour l'apprentissage de données multi-relationnelles afin d'évaluer leurs performances pour la prédiction de lien, le classement des entités et la désambiguation de sens.

9.2 Contribution

Antoine Bordes et moi avons contribué de manière égale à la réalisation de cet article. L'idée originale venant d'Antoine et de son travail préliminaire (Bordes et al., 2011). Nous avons étroitement collaboré pour la conception du modèle et des expériences à réaliser. Antoine Bordes a effectué le pré-traitement des données et moi les expériences. Nous avons rédigé l'article conjointement. Yoshua Bengio et

Jason Weston nous ont donné de précieux conseils. J'ai développé le code permettant de reproduire les résultats de l'article et d'utiliser facilement les modèles que nous présentons.

9.3 Impact

Ce travail a permis de montrer qu'un modèle à base d'"embeddings", que nous avons appelé SME, pouvait être utilisé de manière efficace sur les données multi-relationnelles. Ce dernier obtient des performances équivalentes à l'état de l'art pour modéliser les données multi-relationnelles à petite échelle, et supérieures à l'état de l'art dans le contexte grande échelle. Finalement, nous avons montré comment ce type de modèle pouvait être utilisé en désambiguation de sens.

Suite à ce travail, la recherche a été active dans ce domaine, l'état de l'art en apprentissage de données multi-relationnelles a été amélioré en utilisant ce même genre de modèle (Jenatton et al., 2012; Bordes et al., 2014; Socher et al., 2013; Bordes et al., 2013) mais avec différentes paramétrisations et fonctions de coûts. Le dernier travail présente une propriété intéressante, les relations y sont modélisées comme des translations opérants dans l'espace des "embeddings". Malgré l'apparente simplicité du modèle, les performances obtenues sont bonnes.

L'application de ce type de modèle pour la désambiguation de sens reste, elle, originale. Son utilisation pour améliorer la représentation sémantique est toujours à explorer. Des modèles à base d'"embeddings" dans le contexte des réseaux de neurones recursifs ont été employés récemment à cette fin et donnent des résultats prometteurs (Socher et al., 2014).

10 A Semantic Matching Energy Function for Learning with Multi-relational Data

Abstract

Large-scale relational learning becomes crucial for handling the huge amounts of structured data generated daily in many application domains ranging from computational biology or information retrieval, to natural language processing. In this paper, we present a new neural network architecture designed to embed multi-relational graphs into a flexible continuous vector space in which the original data is kept and enhanced. The network is trained to encode the semantics of these graphs in order to assign high probabilities to plausible components. We empirically show that it reaches competitive performance in link prediction on standard datasets from the literature as well as on data from a real-world knowledge base (WordNet). In addition, we present how our method can be applied to perform word-sense disambiguation in a context of open-text semantic parsing, where the goal is to learn to assign a structured meaning representation to almost any sentence of free text, demonstrating that it can scale up to tens of thousands of nodes and thousands of types of relation.

10.1 Introduction

Multi-relational data, which refers to graphs whose nodes represent entities and edges correspond to relations that link these entities, plays a pivotal role in many areas such as recommender systems, the Semantic Web, or computational biology. Relations are modeled as triplets of the form (subject, relation, object), where a relation either models the relationship between two entities or between an entity and an attribute value ; relations are thus of several types. Such data sources are

equivalently termed multi-relational graphs. They can also be represented by 3-dimensional tensors, for which each slice represents an adjacency matrix for one relation. Multi-relational graphs are popular tools for encoding data via knowledge bases, semantic networks or any kind of database following the Resource Description Framework (RDF) format. Hence, they are widely used in the Semantic Web (Freebase, DBpedia, etc.) [i] but also for knowledge management in bioinformatics (GeneOntology, UMLS semantic network, etc.) [ii] or natural language processing (WordNet), [iii] to name a few. Social networks can also be represented using RDF.

In spite of their appealing ability for representing complex data, multi-relational graphs remain complicated to manipulate for several reasons. First, interactions are of multiple types and heterogeneous (various frequencies, concerning different subsets of entities, etc.). In addition, most databases have been built either collaboratively or (partly) automatically. As a consequence, data is noisy and incomplete: relations can be missing or be invalid, there can be redundancy among entities because several nodes actually refer to the same concept, etc. Finally, most multi-relational graphs are of very large dimensions in terms of numbers of entities and of relation types: Freebase contains more than 20 millions entities, DBpedia is composed of 1 billion triplets linking around 4 millions entities, GeneOntology contains more than 350k verified biological entities, etc. Conveniently represent, summarize or de-noise this kind of data is now a central challenge in statistical relational learning (Getoor and Taskar, 2007).

In this paper, we propose a new model to learn multi-relational semantics, that is, to encode multi-relational graphs into representations that capture the inherent complexity in the data, while seamlessly defining similarities among entities and relations and providing predictive power. Our work is based on an original energy function, which is trained to assign low energies (i.e. high probabilities) to plausible triplets of a multi-relational graph. This energy function, termed *semantic matching energy*, relies on a compact distributed representation: all elements (entity and relation type) are represented into the same relatively low (e.g. 50) dimensional embedding vector space. The embeddings are learnt by a neural network whose particular architecture and training process force them to encompass the original

i. Respect. available from freebase.com and dbpedia.org.
ii. Respect. available from geneontology.org and semanticnetwork.nlm.nih.gov.
iii. Available from wordnet.princeton.edu.

data structure. Unlike in previous work, in this model, relation types are modeled similarly as entities. In this way, entities can also play the role of relation type, as in natural language for instance, and this requires less parameters when the number of relation types grows. We show empirically that this model achieves competitive results on benchmark tasks of link prediction, i.e., generalizing outside of the set of given valid triplets.

We also demonstrate the flexibility and scalability of the semantic matching energy by applying it for word-sense disambiguation (WSD). The model can successfully be trained on various heterogeneous data sources (knowledge bases, free text, etc.), containing several thousands of entities *and* of relation types, to jointly learn representations for words and for senses (defined as entities of a lexical knowledge base, WordNet). On two different evaluation test sets, the proposed approach outperforms both previous work for learning with multi-relational data and standard methods for unsupervised WSD.

To summarize, the main contributions of this paper are threefold:

- an original model for encoding multi-relational data, which represents relation types and entities in the same way, and is potentially able to scale up to larger numbers of relation types than previous work ;
- a training algorithm based on a ranking objective, which allows to learn on large numbers of training samples and achieves competitive results on benchmarks of various dimensions ;
- an adaptation of the model for word-sense disambiguation, which consists of the first successful direct application of relational embeddings of a knowledge base (WordNet) for natural language processing.

Note that this paper extends a shorter version (Bordes et al., 2012), which first introduced the model. However, the previous paper was only focused on the application to word-sense disambiguation, whereas the present paper has a wider scope and considers more problems involving multi-relational data. New elements are provided: a fresh (and cleaner) form of the bilinear formulation, new experiments comparing to the state-of-the-art in link prediction, entity ranking and WSD, a more comprehensive literature review, and more details on the model formulation and the training procedure. We also provide a link to an open-source implementation of the code and to the data used in this paper: `http://goo.gl/bHWsK`.

The paper is organized as follows. Section 10.2 presents a review of previous

work on learning with multi-relational data. Section 10.3 introduces the semantic matching energy function and Section 10.4 its training procedure. Extensive experimental results are given in Section 10.5. Finally, the application to WSD is described in Section 10.6 and Section 10.7 concludes and sketches future work.

10.2 Previous Work

Several methods have been explored to represent and encode multi-relational data, such as clustering approaches. Hence, Kemp et al. (2006) introduced the Infinite Relational Model, IRM, a nonparametric Bayesian model whose latent variables are used to discover meaningful partitions among entities and relations. This model provides a great interpretability of the data but suffers from a poor predictive power. Miller et al. (2009) refined this to allow entities to have a mixed cluster membership. Sutskever et al. (2009) proposed another refinement with the Bayesian Tensor Clustered Factorization model, BCTF, in which the nonparametric Bayesian framework is coupled with the learning, via collective matrix factorization, of distributed representations for the entities and relation types. Other proposals have consisted in improving the original model by adding first-order formulae with Markov Logic. Hence, MRC, for Multiple Relation Clustering (Kok and Domingos, 2007), performs clustering of entities through several relation types simultaneously. Singla and Domingos (2006) presented another model based on Markov logic for the task of entity resolution (i.e. deciding whether two entities should be merged).

All these methods share the ability of providing an interpretation of the data but are slow and do not scale to very large databases due to the high cost of inference. Models based on tensor factorization can be faster and scale to larger data because of their continuous and usually convex optimization. Standard methods like CANDECOMP/PARAFAC (CP) (Harshman and Lundy, 1994) or those from (Tucker, 1966) have been applied on multi-relational graphs. Franz et al. (2009) used CP for ranking data from RDF knowledge bases. Other directions have also been proposed derived from probabilistic matrix factorization for multi-dimensional data (Chu and Ghahramani, 2009) or by adapting dimensionality reduction techniques such as SVD (Speer et al., 2008; Cambria et al., 2009). Recently, Nickel et al. (2011) presented RESCAL, an upgrade over previous tensor factorization methods,

which achieves strong predictive accuracies on various problems. RESCAL represents entities by low dimensional vectors and relation types by low rank matrices, which are learnt using a collective learning process similar to that of CP, but with some relaxed constraints. It has achieved the best accuracies on many benchmarks of tensor factorization. RESCAL has been applied to the knowledge base YAGO (Nickel et al., 2012) and hence showed to scale well on data with large numbers of entities (few millions). However, RESCAL has never been tested on data with large numbers of relation types (YAGO has 87 such types).

Some approaches described above (e.g. BCTF, RESCAL) end up with a distributed representation of the entities and relation types obtained via factorizing or clustering the original data. A slightly different line of work consists in focusing on learning such representations, termed *embeddings*. This idea of learning embeddings has been successful in natural language processing via the framework of language models (Bengio et al., 2003) where an embedding per word is learnt in an unsupervised fashion: it has been shown that such representations can store key information about language (mostly syntactic similarities) that helps to improve performance on standard NLP tasks (Bengio, 2008; Collobert et al., 2011). Bordes et al. (2010) adapted a related model to a small hand-crafted knowledge base and text for language understanding. For multi-relational data, Linear Relational Embeddings (Paccanaro, 2000; Paccanaro and Hinton, 2001) learn a mapping from the entities into a feature-space by imposing the constraint that relations in this feature-space are modeled by linear operations. In other words, entities are modeled by real-valued vectors and relations by matrices and parameters of both are learnt. This idea has been further improved in the Structured Embeddings (SE) framework of Bordes et al. (2011).

Our work lies in the same research area, since we also aim at learning distributed representations of multi-relational data, and we introduce several novel elements. First, unlike previous work (including BCTF, RESCAL or SE) we do not represent a relation type differently than any entity (by a matrix for instance). In our model, a relation type is represented by a vector (just like other entities) and shares the status and number of parameters of other entities. This is convenient when the number of such types is large or when they can also play the role of entities as we illustrate in Section 10.6 on a problem with more than 10k relation types. Second, we do not use a training process based on tensor reconstruction (as RESCAL) or on

clustering (as BCTF), but on a predictive approach instead. The learning objective is essentially asking the model to perform link or entity prediction (i.e. filling an empty spot in a triple). This leads to an algorithm based on a ranking objective and using stochastic gradient descent and backpropagation for updating the parameters, which has a low computational complexity per epoch (independent on the number of entities and relation types). Third, we show that our model is flexible and can be successfully trained via multi-tasking on several heterogeneous sources. Finally, even if it is not presented in this paper, our approach could potentially be adapted to learn non-linear representations of multi-relational data by adding non-linear transfer functions such as *tanh* or *sigmoid* to the neural network. We empirically compare our model with IRM, BCTF, MRC, CP, RESCAL and SE on various tasks with very different properties in our experiments of Section 10.5 and 10.6. Even if the best performing methods from earlier papers differ from one task to another, our proposed approach is competitive for all of them. The closest counterparts are SE and RESCAL, but as we show experimentally, they are not able to handle large-scale multi-relational data as we propose, and either break or are outperformed when data dimensions grows (number of entities and/or of relation types).

10.3 Semantic Matching Energy Function

This section introduces our model designed to embed multi-relational data into fully distributed representations via a custom energy function.

10.3.1 Notations

This work considers multi-relational databases as graph models. The data structure is defined by a set of nodes and a set of links. To each individual node of the graph corresponds an element of the database, which we term an *entity*, and each link defines a *relation* between entities. Relations are directed and there are typically several different kinds of relations. Let $\mathcal{C}$ denote the dictionary which includes all entities and relation types, and let $\mathcal{R} \subset \mathcal{C}$ be the subset of entities which are relation types. In the remainder of the paper, a relation is denoted by a triplet (lhs,

rel, rhs), where lhs is the *left* entity, rhs the *right* one and rel the *type* of relation between them.

10.3.2 Main ideas

The main ideas behind our semantic matching energy function are the following.

- Named symbolic entities (entities *and* relation types) are associated with a d-dimensional vector space, termed the "embedding space", following previous work in neural language models (Bengio, 2008). The i^{th} entity is assigned a vector $\mathbf{e}_i \in \mathbb{R}^d$. Note that more general mappings from an entity to its embedding are possible.
- The semantic matching energy value associated with a particular triplet (lhs, rel, rhs) is computed by a parametrized function $\mathcal{E}$ that starts by mapping all symbols to their embeddings and then combines them in a structured fashion. Our model is termed "semantic matching" because $\mathcal{E}$ relies on a matching criterion computed between both sides of the triplet.
- The energy function $\mathcal{E}$ is optimized to be lower for training examples than for other possible configurations of symbols. Hence the semantic energy function can distinguish plausible combinations of entities from implausible ones, and can be used, for instance, to answer questions, e.g. corresponding to a triplet $(lhs, rel, ?)$ with a missing rhs, by choosing among the possible entities a rhs with a relatively lower energy. See (Lecun et al., 2006) for a review of energy-based learning.

Our approach is inspired by the framework introduced by Bordes et al. (2011) as well as by recent work of Bottou (2011). Our main motivation is to conceive a model where entities and relation types would share the same kind of representation. Embedding all symbols defining a multi-relational graph into the same space amounts to deleting the usual conceptual difference between entities (lhs and rhs) and relation types. This modeling is more natural when entities can act as rel as well as lhs or rhs. In Section 10.6, we apply our method to natural language data, where relation types typically correspond to verbs. Since verbs may also occur in lhs or rhs, it is reasonable to share the same representation, and this can ease learning representations of verbs appearing rarely as relation type. Our choice to encode symbols by vectors helps too by causing the overall number of tunable

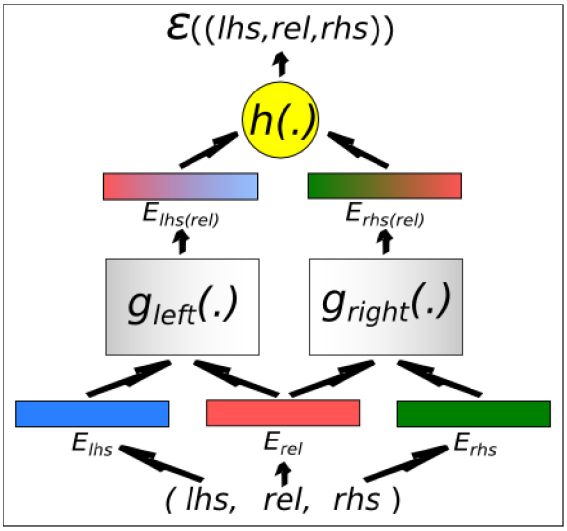

Figure 10.1 – Semantic matching energy function. A triple of entities (lhs, rel, rhs) is first mapped to its embeddings $\mathbf{e}_{lhs}$, $\mathbf{e}_{rel}$ and $\mathbf{e}_{rhs}$. Then $\mathbf{e}_{lhs}$ and $\mathbf{e}_{rel}$ are combined using $g_{left}(.)$ to output $\mathbf{e}_{lhs(rel)}$ (similarly $\mathbf{e}_{rhs(rel)} = g_{right}(\mathbf{e}_{rhs}, \mathbf{e}_{rel})$). Finally the energy $\mathcal{E}((lhs, rel, rhs))$ is obtained by matching $\mathbf{e}_{lhs(rel)}$ and $\mathbf{e}_{rhs(rel)}$ with the $h(.)$ function.

parameters to learn to remain low, especially when the cardinality of $\mathcal{R}$ grows.

10.3.3 Neural network parametrization

The energy function $\mathcal{E}$ (denoted SME) is encoded using a neural network, whose parallel architecture is based on the intuition that the relation type should first be used to extract relevant components from each argument's embedding, and put them in a space where they can then be compared (see Figure 10.1). Hence, pairs $(lhs,\ rel)$ and $(rel,\ rhs)$ are first combined separately and then, these semantic combinations are *matched*.

(1) Each symbol of the input triplet $(lhs,\ rel,\ rhs)$ is mapped to its embedding $\mathbf{e}_{lhs}$, $\mathbf{e}_{rel}$ and $\mathbf{e}_{rhs} \in \mathbb{R}^d$.

(2) The embeddings $\mathbf{e}_{lhs}$ and $\mathbf{e}_{rel}$ respectively associated with the lhs and rel arguments are used to construct a new relation-dependent embedding $\mathbf{e}_{lhs(rel)}$ for the lhs in the context of the relation type represented by $\mathbf{e}_{rel}$, and similarly for the rhs: $\mathbf{e}_{lhs(rel)} = g_{left}(\mathbf{e}_{lhs}, \mathbf{e}_{rel})$ and $\mathbf{e}_{rhs(rel)} = g_{right}(\mathbf{e}_{rhs}, \mathbf{e}_{rel})$, where g_{left} and g_{right} are parametrized functions whose parameters are tuned during training. Even if it remains low-dimensional, nothing forces the dimension of

$\mathbf{e}_{lhs(rel)}$ and $\mathbf{e}_{rhs(rel)}$, which we denote p, to be equal to d, the one of the entity embedding space.

(3) The energy is computed by "matching" the transformed embeddings of the left-hand side and right-hand side: $\mathcal{E}((lhs, rel, rhs)) = h(\mathbf{e}_{lhs(rel)}, \mathbf{e}_{rhs(rel)})$, where h can be a simple operator such as a dot product or a more complex function whose parameters are learnt.

With this formulation, $\mathcal{E}$ is not able to handle variable-size arguments for lhs or rhs (like tuples of entities). However, it can be adapted to it by adding a pooling stage between steps (1) and (2) as we show in Section 10.6.1.

Different types of parametrizations can be used for the g and h functions. We chose to use a dot product for the output h function because it is simple and has shown to work well in related work (e.g. in (Weston et al., 2010)). For the g functions, we studied two options, one linear and the other bilinear, which lead to two versions of SME detailed below:

– *Linear form* (denoted SME(linear) in the following), in this case g functions are simply linear layers:

$$\begin{aligned} \mathbf{e}_{lhs(rel)} &= g_{left}(\mathbf{e}_{lhs}, \mathbf{e}_{rel}) &&= \mathbf{e}_{lhs} W_{l1} + \mathbf{e}_{rel} W_{l2} + \mathbf{b}_l. \\ \mathbf{e}_{rhs(rel)} &= g_{right}(\mathbf{e}_{rhs}, \mathbf{e}_{rel}) &&= \mathbf{e}_{rhs} W_{r1} + \mathbf{e}_{rel} W_{r2} + \mathbf{b}_r. \end{aligned}$$

with W_{l1}, W_{l2}, W_{r1}, $W_{r2} \in \mathbb{R}^{d \times p}$ (weights), $\mathbf{b}_l$, $\mathbf{b}_r \in \mathbb{R}^p$ (biases). This leads to the following form for the energy:

$$\mathcal{E}((lhs, rel, rhs)) = -\left(\mathbf{e}_{lhs} W_{l1} + \mathbf{e}_{rel} W_{l2} + \mathbf{b}_l\right) \left(\mathbf{e}_{rhs} W_{r1} + \mathbf{e}_{rel} W_{r2} + \mathbf{b}_r\right)^\intercal . \quad (10.1)$$

– *Bilinear form* (denoted SME(bilinear) in the following), in this case g functions are using 3-modes tensors as core weights:

$$\begin{aligned} \mathbf{e}_{lhs(rel)} &= g_{left}(\mathbf{e}_{lhs}, \mathbf{e}_{rel}) &&= \mathbf{e}_{lhs} \left(\mathbf{e}_{rel} \bar{\times}_1 W_l\right) + \mathbf{b}_l. \\ \mathbf{e}_{rhs(rel)} &= g_{right}(\mathbf{e}_{rhs}, \mathbf{e}_{rel}) &&= \mathbf{e}_{rhs} \left(\mathbf{e}_{rel} \bar{\times}_1 W_r\right) + \mathbf{b}_r. \end{aligned}$$

with W_l, $W_r \in \mathbb{R}^{d \times d \times p}$ (weights) and $\mathbf{b}_l$, $\mathbf{b}_r \in \mathbb{R}^p$ (biases). $\bar{\times}_1$ denotes the mode-1 vector-tensor product, which, for $\mathbf{u}, \mathbf{v} \in \mathbb{R}^d$ and $W \in \mathbb{R}^{d \times d \times p}$, is

defined as:

$$\forall i \in 1, \dots p, \ \left(\mathbf{u}(\mathbf{v}\bar{\times}_1 W)\right)_i = \sum_{i=1}^{d}\sum_{j=1}^{d} u_j v_i w_{ijk}.$$

This leads to the following form for the energy:

$$\mathcal{E}((lhs, rel, rhs)) = -\left(\mathbf{e}_{lhs}\left(\mathbf{e}_{rel}\bar{\times}_1 W_l\right) + \mathbf{b}_l\right)\left(\mathbf{e}_{rhs}\left(\mathbf{e}_{rel}\bar{\times}_1 W_r\right) + \mathbf{b}_r\right)^\intercal. \quad (10.2)$$

10.3.4 Discussion

We can notice that Eq. (10.1), defining the energy for SME(linear), can be re-written as (bias terms are removed for clarity):

$$\mathcal{E}((lhs, rel, rhs)) = -\mathbf{e}_{lhs}\tilde{W}_1\mathbf{e}_{rhs}^\intercal - \mathbf{e}_{lhs}\tilde{W}_2\mathbf{e}_{rel}^\intercal - \mathbf{e}_{rel}\tilde{W}_3\mathbf{e}_{rhs}^\intercal - \mathbf{e}_{rel}\tilde{W}_4\mathbf{e}_{rel}^\intercal,$$

with $\tilde{W}_1 = W_{l1}W_{r1}^\intercal$, $\tilde{W}_2 = W_{l1}W_{r2}^\intercal$, $\tilde{W}_3 = W_{l2}W_{r1}^\intercal$ and $\tilde{W}_4 = W_{l2}W_{r2}^\intercal \in \mathbb{R}^{d\times d}$. Hence, the energy can be decomposed into three terms coding for pairs (lhs, rhs), (lhs, rel) and (rel, rhs), and an additional quadratic term for rel. This shows that SME(linear) actually represents a triplet as a combination of pairwise interactions (similar to what is captured by bigrams).

Similarly, Eq. (10.2), defining the energy for SME(bilinear) can be re-written as:

$$\mathcal{E}((lhs, rel, rhs)) = -\mathbf{e}_{lhs}\tilde{W}(rel)\mathbf{e}_{rhs}^\intercal,$$

with $\tilde{W}(rel) = \left(\mathbf{e}_{rel}\bar{\times}_1 W_l\right)\left(\mathbf{e}_{rel}\bar{\times}_1 W_r\right)^\intercal \in \mathbb{R}^{d\times d}$. In this case, the energy is composed of a single term, which depends on all three entities, with a central role for rel. Hence, SME(bilinear) represents a triplet through 3-way interactions (similar to what is captured by a trigram). The choice between a linear or a bilinear form for g leads to a very different formulation overall.

The trigram formulation can model ternary interactions but requires more parameters. SME(bilinear) has $\mathcal{O}(n_e d + n_r d + p d^2)$ parameters while SME(linear) has only $\mathcal{O}(n_e d + n_r d + pd)$, with n_e the number of entities (never being a relation type), n_r the number of relation types, d the low-level embedding dimension and p the dimension of the higher-level relation-dependent representation (with both p and d much smaller than n_e and n_r). Still, both formulations can scale up to

large numbers of relation types ($n_r >> 1$) without requiring too many parameters, contrary to previous methods such as RESCAL and SE, which entail $\mathcal{O}(n_e d + n_r d^2)$ parameters. This property comes from our original choice of modeling rel in the same way as lhs and rhs, using vectors. Interestingly, we can remark that, in Equation (10.2), $\mathbf{e}_{rel} \bar{\times}_1 W_l$ and $\mathbf{e}_{rel} \bar{\times}_1 W_r$ act as a pair of matrices coding for rel: this can be seen as a distributed or factorized version of what RESCAL or SE proposed.

10.4 Training

This section details the training procedure for the semantic matching energy function, SME.

10.4.1 Training criterion

We are given a training set $\mathcal{D}$ containing m triplets $x = (x_{lhs}, x_{rel}, x_{rhs})$, where $x_{lhs} \in \mathcal{C}$, $x_{rel} \in \mathcal{R}$, and $x_{rhs} \in \mathcal{C}$. We recall that the energy of a triplet is denoted $\mathcal{E}(x) = \mathcal{E}(x_{lhs}, x_{rel}, x_{rhs})$. Ideally, we would like to perform maximum likelihood over $P(x) \propto e^{-\mathcal{E}(x)}$ but this is intractable. The approach we follow here has already been used successfully in ranking settings (Collobert et al., 2011; Weston et al., 2010) and corresponds to performing two approximations. First, like in pseudo-likelihood we only consider one input at a time given the others, e.g. lhs given rel and rhs, which makes normalization tractable. Second, instead of sampling a negative example from the model posterior,[i] we use a ranking criterion (based on uniformly sampling a negative example, in a way that is reminiscent of Noise Contrastive Estimation (Gutmann and Hyvärinen, 2010)).

Intuitively, if one of the elements of a given triplet were missing, then we would like the model to be able to predict the correct entity. The objective of training

i. In an energy-based model such as the Boltzmann machine, the gradient of the negative log-likelihood is equal to the gradient of the energy of a positive example (observed and valid) minus the expected value of the gradient of a negative example (sampled from the model). In the case of pseudo-likelihood training one would consider conditional likelihoods $P(x_i|x_1, \cdots, x_{i-1}, x_{i+1}, \cdots, x_d)$, and only the x_i part of the positive example needs to be re-sampled for constructing the negative example, using this same posterior.

is to learn the semantic energy function $\mathcal{E}$ such that it can successfully rank the training samples x below all other possible triplets:

$$\mathcal{E}(x) < \mathcal{E}((i, x_{rel}, x_{rhs})) \quad \forall i \in \mathcal{C} : (i, x_{rel}, x_{rhs}) \notin \mathcal{D} \tag{10.3}$$

$$\mathcal{E}(x) < \mathcal{E}((x_{lhs}, j, x_{rhs})) \quad \forall j \in \mathcal{R} : (x_{lhs}, k, x_{rhs}) \notin \mathcal{D} \tag{10.4}$$

$$\mathcal{E}(x) < \mathcal{E}((x_{lhs}, x_{rel}, k)) \quad \forall k \in \mathcal{C} : (x_{lhs}, x_{rel}, j) \notin \mathcal{D} \tag{10.5}$$

Towards achieving this, the following stochastic criterion is minimized:

$$\sum_{x \in \mathcal{D}} \sum_{\tilde{x} \sim Q(\tilde{x}|x)} \max\left(\mathcal{E}(x) - \mathcal{E}(\tilde{x}) + 1, 0\right) \tag{10.6}$$

where $Q(\tilde{x}|x)$ is a corruption process that transforms a training example x into a corrupted *negative example*. Note that $\max\left(\mathcal{E}(x) - \mathcal{E}(\tilde{x}) + 1, 0\right)$ is similar in shape to the negative log-likelihood $-\log \frac{e^{-\mathcal{E}(x)}}{e^{-\mathcal{E}(x)} + e^{-\mathcal{E}(\tilde{x})}} = -\log \text{sigmoid}(\mathcal{E}(\tilde{x}) - \mathcal{E}(x))$, which corresponds to the probability of sampling x given that only x and $\tilde{x}$ are considered. In the experiments, Q only changes one of the three members of the triplet (as in a pseudo-likelihood setup), replacing it by an entity uniformly sampled either from $\mathcal{R}$ if the replaced entity is a relation type, or from $\mathcal{C}/\mathcal{R}$ otherwise. We do not actually check if the negative example is in $\mathcal{D}$. Note that this is not necessary because if we have the symmetry $Q((\tilde{a}, b, c)|(a, b, c)) = Q((a, b, c)|(\tilde{a}, b, c))$ etc. for all elements of the triplet, and it is true here, then the expected contribution to the total expected gradient due to cases where $\tilde{x} \in \mathcal{D}$ is 0. This is because if we consider only the pairs $x, \tilde{x} \in \mathcal{D}$, the average over $\mathcal{D}$ of the gradients $\frac{\partial \mathcal{E}(x)}{\partial \theta}$ equals the average over $\mathcal{D}$ of the gradients $\frac{\partial \mathcal{E}(\tilde{x})}{\partial \theta}$, by our symmetry assumption.

10.4.2 Ranking algorithm

To train the parameters of the energy function $\mathcal{E}$ we loop over all of the training data resources and use stochastic gradient descent (Robbins and Monro, 1951). That is, we iterate the following steps:

1. Select a positive training triplet $x_i = (lhs_i, rel_i, rhs_i)$ at random from $\mathcal{D}$.
2. Select at random resp. constraint (10.3), (10.4) or (10.5).
3. Create a negative triplet $\tilde{x}$ by sampling one entity either from $\mathcal{R}$ to replace rel_i or from $\mathcal{C}/\mathcal{R}$ to replace lhs_i or rhs_i.

4. If $\mathcal{E}(x_i) > \mathcal{E}(\tilde{x}) - 1$, make a stochastic gradient step to minimize (10.6).
5. Enforce the constraint that each embedding is normalized, $||\mathbf{e}_i||_2 = 1$, $\forall i$.

The gradient step requires a learning rate λ. The constant 1 in step 4 is the *margin* as is commonly used in many margin-based models such as SVMs (Boser et al., 1992). The normalization in step 5 helps remove scaling freedoms from the model, makes the impact of the margin actually effective and regularizes the optimization objective, preventing weights to collapse or diverge.

Each update of the model parameters is carried out by backpropagation. Its computational complexity is dominated by the cost of the n-mode vector-tensor product for SME(bilinear): $\mathcal{O}(pd^2)$, and by the cost of the matrix product for SME(linear): $\mathcal{O}(pd)$. Therefore, to perform one epoch over $\mathcal{D}$, the bilinear form requires in the order of $\mathcal{O}(mpd^2)$ operations and the linear form in the order of $\mathcal{O}(mpd)$. *Note that this is independent of the number of entities or relation types.*

Matrix E contains the representations of the entities and is learnt via a *multi-task learning* procedure (Caruana, 1995; Collobert and Weston, 2008) because the same embedding matrix is used for all relation types (each corresponding to a different distribution of entities, i.e., a different task). As a result, the embedding of an entity contains factorized information coming from all the relations in which the entity is involved as *lhs*, *rhs* or even *rel*. For each entity, the model is forced to learn how an entity interacts with other entities in many different ways. One can think of the elements $e_{i,j}$ of each embedding vector $\mathbf{e}_i$ as *learned attributes* for entity i or relation type i. Different tasks may demand different attributes, so that entities that have a common behavior[i] in some way will get the same values for some of their attributes. If the same attributes can be useful for several tasks, then statistical power is gained through parameter sharing, and transfer of information between tasks can happen, making the data of some task informative for generalizing properly on another task.

10.4.3 Implementation details

All the code for the experiments has been implemented in Python and using the Theano library (Bergstra et al., 2010). Training is carried out using mini-

i. e.g., appear in semantically similar contexts, i.e., in instances containing the same entities or ones with close-by values of their embedding.

Table 10.1 – Statistics of datasets used in our experiments. The top two are fully observed, very sparse i.e. only a small minority of relations are valid and hence are used in a cross-validation scheme. Only a fraction of relations are observed in Nations and WordNet.

Dataset	**Nb. of relation types**	**Nb. of entities**	**Nb. of observed relations**	**% valid relations in obs. ones**
UMLS	49	135	893,025	0.76
Kinships	26	104	281,216	3.84
Nations	56	14	11,191	22.9
WordNet	18	40,943	151,442	100

batches (we create 200 mini-batches for each dataset, independent of its size). All hyperparameter values are set using a validation set. The dimension of the embeddings (d) and the dimension of the output space of g functions (p) are selected among $\{10, 25, 100\}$. There is a different learning rate for the embedding matrix E and for the parameters of g. It is chosen among $\{0.03, 0.01, 0.003, 0.001, 0.0003\}$ for E and among $\{3., 1., 0.3, 0.1, 0.03\}$ for g. Training stops using early stopping on the validation set error (or after a maximum of 2,000 epochs).

10.5 Empirical Evaluation

This section proposes an experimental comparison of SME with current state-of-the-art methods for learning representations of multi-relational data.

10.5.1 Datasets

In order to evaluate against existing methods, we performed experiments on benchmarks from the literature. Kinships and UMLS are fully observed, i.e. for each relation type and each potential pair of entities it has been observed whether the given triplet is valid or not. They are also sparse, i.e. only a small fraction of triplets are valid. We also illustrate the properties of our model on Nations and WordNet, which are partially observed: we only observe some valid triplets in the case of WordNet and some valid or invalid triplets for Nations. The rest is unknown, that is, missing triplets can be valid or not. And of course, in that case only a tiny

Table 10.2 – Relation types of WordNet used in our experiments.

WordNet
_hypernym, *_hyponym*, *_instance_hyponym*, *_instance_hypernym*, *_related_form*, *_has_part*, *_part_of*, *_member_has_part*, *_member_part_of*, *_also_see*, *_attribute*, *_synset_domain_region*, *_synset_domain_usage*, *_synset_domain_topic*, *_verb_group*, *_member_of_domain_region*, *_member_of_domain_usage*, *_member_of_domain_topic*

fraction of potential triplets are observed. We describe all datasets below, with some statistics displayed in Table 10.1.

UMLS This dataset contains data from the Unified Medical Language System semantic work gathered by McCray (2003). This consists in a graph with 135 entities and 49 relation types. The entities are high-level concepts like 'Disease or Syndrome', 'Diagnostic Procedure', or 'Mammal'. The relations represent verbs depicting causal influence between concepts like 'affect' or 'cause'.

Nations This dataset groups 14 countries (Brazil, China, Egypt, etc.) with 56 binary relation types representing interactions among them like 'economic_aid', 'treaties' or 'rel_diplomacy', and 111 features describing each country. See (Rummel, 1999) for details.

Kinships Australian tribes are renowned among anthropologists for the complex relational structure of their kinship systems. This dataset, created by Denham (1973), focuses on the Alyawarra, a tribe from Central Australia. 104 tribe members were asked to provide kinship terms for each other. This results in a graph of 104 entities and 26 relation types, each of them depicting a different kinship term, such as Adiadya or Umbaidya. See (Denham, 1973) or (Kemp et al., 2006) for more details.

WordNet This knowledge base is designed to produce intuitively usable dictionary and thesaurus, and supports automatic text analysis. It encompasses comprehensive knowledge within its graph structure, whose entities (termed *synsets*) correspond to senses, and relation types define lexical relations between those senses. We considered all the entities that were connected with the relation types given

Table 10.3 – Link prediction. Comparisons of area under the precision-recall curve (AUC) computed in a 10-fold cross-validation setting between two versions of SME (this paper) and previously published algorithms (SE, RESCAL, CP, BCTF, MRC, IRM) on UMLS, Nations and Kinships. Emb. is an unstructured version of SME. Best performing methods, with a significant difference with the rest, are indicated in bold.

Method	**UMLS**	**Nations**	**Kinships**
SME(linear)	**0.979** ± 0.003	0.777 ± 0.025	0.149 ± 0.003
SME(bilinear)	**0.985** ± 0.003	**0.865** ± 0.015	0.894 ± 0.011
Emb.	0.035 ± 0.002	0.345 ± 0.025	0.038 ± 0.001
SE	**0.983** ± 0.004	**0.869** ± 0.016	0.913 ± 0.006
RESCAL	**0.98**	0.84	**0.95**
CP	0.95	0.83	**0.94**
BCTF	**0.98**	n/a	0.90
MRC	**0.98**	0.75	0.85
IRM	0.70	0.75	0.66

in Table 10.2, although we did remove some entities for which we have too little information: we filtered out the synsets appearing in less that 15 triplets. We obtain a graph with 40,943 synsets and 18 relations types. Examples of triplets are (*_score_NN_1*, *_hypernym*, *_evaluation_NN_1*) or (*_score_NN_2*, *_has_part*, *_musical_notation_NN_1*). As WordNet is composed of words with different meanings, we describe its entities by the concatenation of the word, its part-of-speech tag ('NN' for noun, 'VB' for verb, 'JJ' for adjective and 'RB' for adverb) and a digit indicating which sense it refers to i.e. *_score_NN_1* is the entity encoding the first meaning of the noun "score". This version of WordNet is different from that used in (Bordes et al., 2011) because the original data has been preprocessed differently: this version contains less entities but more relation types.

10.5.2 Link prediction

The link prediction task consists in predicting whether two entities should be connected by a given relation type. This is useful for completing missing values of a graph, forecasting the behavior of a network, etc. but also to assess the quality of a representation. We evaluate our model on UMLS, Nations and Kinships, following the setting introduced in (Kemp et al., 2006): data tensors are split in ten folds of valid configurations using (lhs, rel, rhs) triplets as statistical units,

and experiments are performed by cross-validation. The standard evaluation metric is *area under the precision-recall curve* (AUC). For our own models, we used one of the nine training folds for validation. Table 10.3 presents results of SME along with those of RESCAL, BCTF, MRC, IRM and CP, which have been extracted from (Kemp et al., 2006; Kok and Domingos, 2007; Sutskever et al., 2009; Nickel et al., 2011) and that of SE, which we computed ourselves. Table 10.3 also displays performance of an *unstructured version* of SME, termed Emb.: in this case, the score of a triplet (lhs, rel, rhs) is simply determined by the dot product between lhs and rhs embeddings, without any influence of the relation type.

The linear formulation of SME is outperformed by SME(bilinear) on all three tasks. The largest differences for Nations and Kinships indicate that, for these problems, a joint interaction between both lhs, rel and rhs is crucial to represent the data well: relations cannot be simply decomposed as a sum of bigrams. This is particularly true for the complex kinship systems of the Alyawarra. On the contrary, interactions within the UMLS network can be represented by simply considering the various (entity, entity) and (entity, relation type) bigrams. Compared to other methods, SME(bilinear) performs similarly to SE, RESCAL, BCTF and MRC on UMLS and similarly to SE on Nations. It is worth noting than, on Nations, SE and SME(bilinear) perform better by a vast margin. On Kinships, it is outperformed by RESCAL and CP: on this dataset with complex ternary interactions, the training process of these tensor factorization methods, based on reconstruction, seems to be beneficial compared to our predictive approach. Simply representing a relation by a vector might also be detrimental w.r.t. using matrices, but SME reaches similar performance as SE (using matrices and a predictive training process). Still, compared to MRC, which is not using a matrix-based encoding, SME(bilinear) remains highly competitive. As expected, Emb. performs poorly, outlining the crucial influence of rel for correctly modeling such data.

To summarize, on these 3 benchmarks with moderate sizes, our method is either state-of-the-art (represented mainly by SE and RESCAL) or very close to it, and can be considered as the best performing method on average. However, SME is primarily designed for large-scale conditions and we show in the following that it outperforms RESCAL when the number of entities increases (in Section 10.5.3) and SE when the number of relation types does (in Section 10.6.4).

Table 10.4 – Entity ranking on WordNet. Comparisons between two versions of SME (this paper) and SE, RESCAL and Emb., an unstructured version of SME. Mean/median predicted rank and precision@10 (p@10, in %) are computed on the test set. Best performances are indicated in bold, worst in italic.

Method	**Rank** (median/mean)	**p@10**
SME(linear)	5 / 559	6.51
SME(bilinear)	8 / 526	5.47
Emb.	*26* / **317**	*3.51*
SE	**3** / *1011*	**6.85**
RESCAL	12 / 893	4.76

10.5.3 Entity ranking

Performing an evaluation based on link prediction for WordNet is problematic because only positive triplets are observed. Hence, in this case, there is no negative triplet but only unknown ones for which it is impossible to state whether they are valid or not. For this setting, for which we only have access to positive training and test examples, AUC is not a satisfying metric anymore. Hence, we evaluate our model on this data using the ranking setting proposed in (Bordes et al., 2011) and described below, which allows an analysis on positive samples only.

We measure the mean and median predicted ranks and the prediction@10, computed with the following procedure. For each test triplet, the left entity is removed and replaced by each of the entities of the dictionary in turn. Energies of those corrupted triplets are computed by the model and sorted by ascending order and the rank of the correct synset is stored. This whole procedure is also repeated when removing the right-hand argument instead. We report the mean and median of those predicted ranks and the precision@10 (or p@10), which is the proportion of ranks within 1 and 10, divided by 10. WordNet data was split in training, validation and test sets with 141,442 observed triplets for training, 5,000 for validation and 5,000 for testing.

Table 10.4 presents comparative results on the test set, together with the performance of Emb., SE and RESCAL, which we all computed. No method is able to perform best for all metrics. SE obtains a low median rank and the best p@10, but has the worst mean rank. This indicates an instability: SE works very well for most examples but can be terrible in some cases. In the original paper introducing

SE, Bordes et al. (2011) proposed to stack a Kernel Density Estimator (KDE) on top of the structured embeddings to improve stability. However, throughout this paper, when we refer to SE, we mean *without* KDE, because this makes a fairer comparison. We could also stack KDE on top of SME but this involves a very expensive extra-cost, that forbids any large-scale ambition. Emb. performs quite well and reaches the best mean rank indicating that, on WordNet, the influence of the relation type is not crucial to get a fair rough estimate of the likely lhs given rhs, and vice-versa. Still, Emb. does not solve the task of handling multi-relational data, since it simply ignores the relation types. This seems to be fine on this task, but it was terrible on those of the previous section.

SME is the only method able to perform well for all metrics (while never reaching on top). In particular, SME(linear) is very close to SE in median rank and p@10 while being much better in mean rank: it does not seem to suffer from instability issues. It is hard on WordNet to be accurate on average (low mean rank) and still have a large proportion of examples very well ranked (low median rank) and SME appears to be the best for this compromise. RESCAL performs consistently worse than SME. We tried very hard to make the code provided by the authors work as well as possible: to behave properly, the model requires large latent dimensions d but this slows it down a lot. Results of Table 10.4 have been obtained with $d = 2000$ and a training time of almost 2 days (compared to around 4h for SME).

10.5.4 Entity embeddings

The matrix E factorizes information from all relations in which the entity appears. We propose here to illustrate the kind of semantics captured by the representations.

We selected 115 entities from WordNet corresponding to countries from all over the world and to U.S. states. We selected this subset because we know that there exist an underlying structure among them. Then, we projected the corresponding embeddings learnt by the linear and bilinear versions of SME and created 2D plots using t-SNE (van der Maaten and Hinton, 2008). They are given in Figure 10.2: a different color is used for each continent ; suffixes depicting POS tag and sense indices have been removed for clarity.

The representations learnt by the linear model seem to nicely reflect the geo-

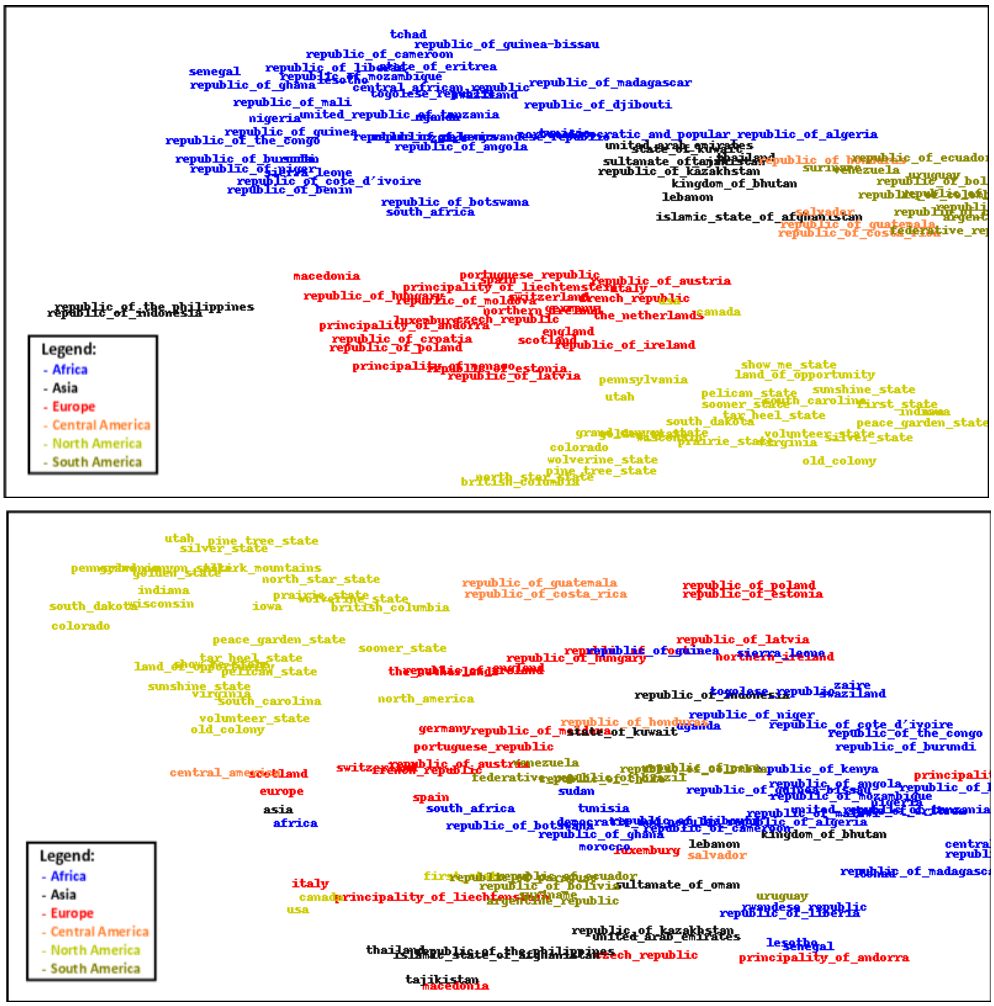

Figure 10.2 – Entity embeddings. Plots of representations (matrix E), learnt by SME(linear) (top) and SME(bilinear) (bottom), for 115 countries selected from WordNet and projected in 2D by t-SNE. SME(linear) encoded geographical similarities within the embeddings.

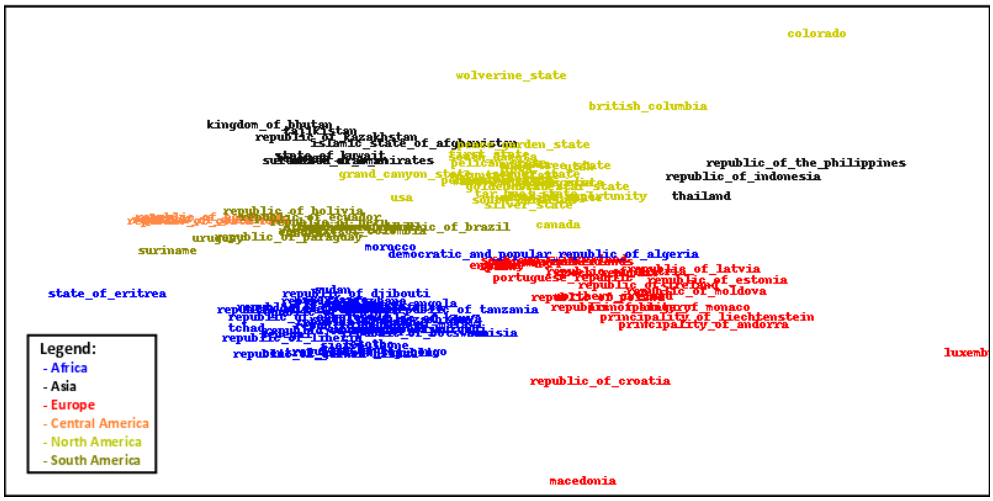

Figure 10.3 – Pairwise similarities. A pairwise similarity matrix of 115 countries from WordNet is computed with the Lesk measure of Wordnet::Similarity and projected in 2D by t-SNE.

graphical semantics, hence encoding the "part-of" information contained in WordNet: nice clusters are formed for each continent. To assess more objectively the quality of this plot, Figure 10.3 proposes the one obtained for the same entities with the Lesk similarity measure of the WordNet::Similarity package (Banerjee and Pedersen, 2002). [i] We tried several measures and chose Lesk because it gave the best result. Comparing both plots tends to indicate that embeddings learnt by SME(linear) could be used to form a very decent similarity measure on WordNet. But, the comparison is not fair because the Lesk measure does not only rely on the WordNet graph but is also improved using *glosses* (i.e. definitions). Performing the same experiment with WordNet::Similarity measures based only on the graph gives much worse results. SME seems able to nicely capture the multi-relational semantics of the WordNet graph, without any other source of information.

The picture changes with the representations learnt by the bilinear models: the plot (bottom of Figure 10.2) is much harder to interpret and suggests that the interactions occurring in the SME(bilinear) neural network are more complex, with a more invasive role for the relation type. This intuition is confirmed by the plots of Figure 10.4. They still display t-SNE projections of representations of the

i. Freely available from wn-similarity.sourceforge.net.

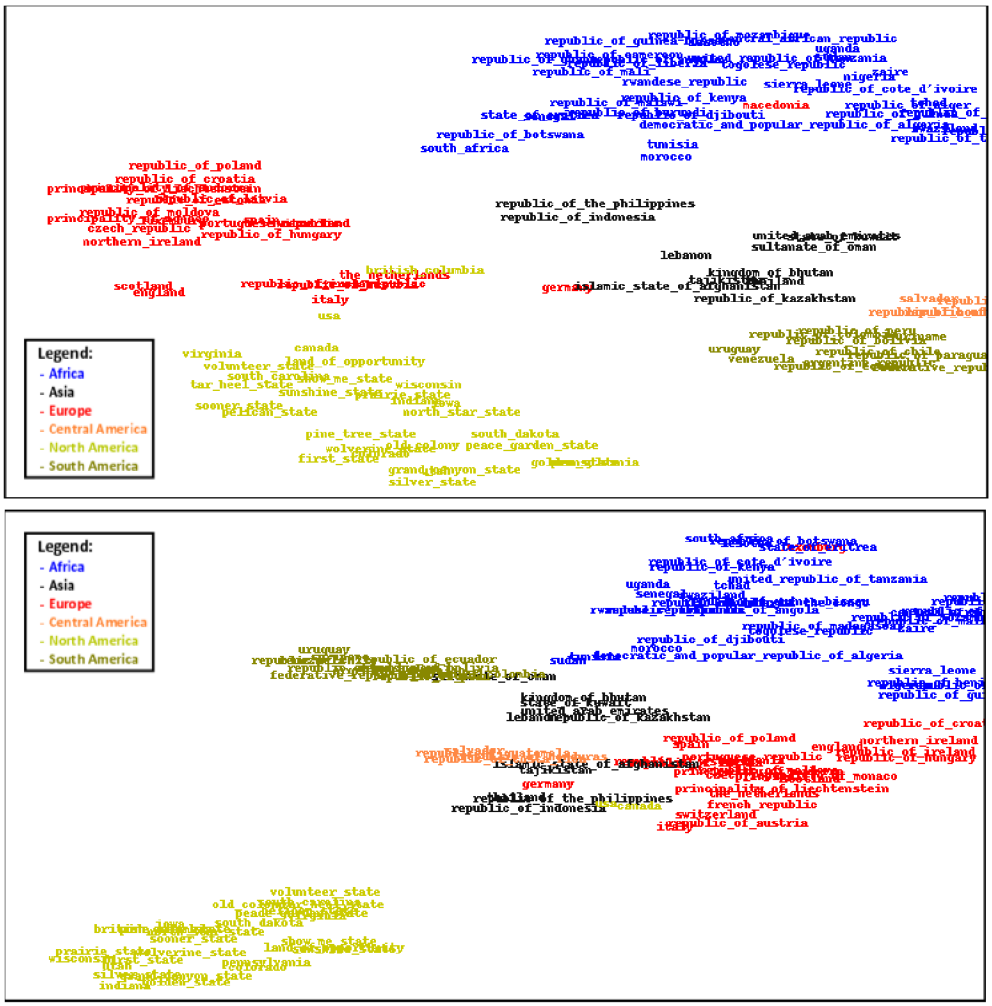

Figure 10.4 – Entity-relation embeddings. Plots of $\mathbf{e}_{lhs(rel)}$ representations, learnt by SME(linear) (top) and SME(bilinear) (bottom), with 115 countries selected from WordNet as *lhs* and *_part_of* as *rel*, projected in 2D by t-SNE. SME(linear) and SME(bilinear) encoded geographical similarities at this stage.

same models for the same entities but not taken at the same level in the network. In this case, we projected the representations obtained by the embeddings when combined with the embedding of the relation type *_part_of* by the g_{left} function. In other words, these are plots of $\mathbf{e}_{lhs(rel)}$. The top plot corresponds to the linear model and resemble to the one of Figure 10.2: as expected, the linear g_{left} does not have a dramatic effect on the embedding landscape. The bottom plot, depicting SME(bilinear), is much more interesting because it shows that what was messy at the root level is much more organized: clusters are now formed for continents with the one corresponding to U.S. states further apart from the countries. Embeddings of SME(bilinear) are more interpretable *given a relation type*. The bilinear g functions drastically modify the distances within the embedding space depending on the relation type, as we expect that it should.

This last remark indicates that, by encoding data with SME(bilinear), one can expect that similarities between two entities existing given one relation (i.e. short distances between their transformed embeddings) would not automatically translate into a similarity between them for any relation type. This kind of behavior seems much harder to reproduce for SME(linear), where a similarity between entities seems to exist independent of the relation. This could explain the bad performance of this model on Kinships, where correct associations highly depend on the relation types. Still, drastic relations like antonymy remain unlikely to be well encoded by SME(bilinear), this might require to add non-linearities to the model or to use larger sparse representations (to code for orthogonality among entities).

10.6 Application for Word-sense Disambiguation

We have introduced a new neural network architecture for learning multi-relational semantics. Its stochastic learning process and its distributed representation of entities *and* relations allow it to scale to large graphs in terms of nodes and link types. In this section, we illustrate these appealing properties by applying our model for learning to carry out all-words word-sense disambiguation on knowl-

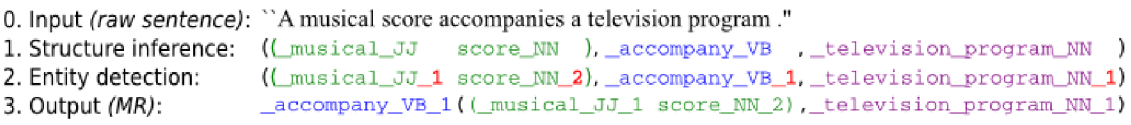

Figure 10.5 – Open-text semantic parsing on simple sentences. To parse an input sentence (0.), a preprocessing (lemmatization, POS, chunking, SRL) is first performed (1.) to clean data and uncover the MR structure. Then, to each lemma is assigned a corresponding WordNet synset (2.), hence defining a complete meaning representation (3.).

edge extracted from free text with semantic role labeling (SRL), with a view to performing open-text semantic parsing.

Semantic parsing (Mooney, 2004) aims at building systems able to read text and express its meaning in a formal representation i.e. able to interpret statements expressed in natural language. The purpose of a semantic parser is to analyze the structure of sentence meaning and, formally, this consists of mapping a natural language sentence into a logical *meaning representation* (MR). Open-text semantic parsing consists of learning to associate a MR to any kind of natural language.

SME could make an interesting piece of a SRL system, especially for producing MRs of the following form: $relation(subject, object)$, i.e. relations with subject and object arguments, where each component of the resulting triplet refers to a disambiguated entity, via the following two-stages process: (1) SRL step predicts the semantic structure, and (2) a disambiguation step assigns a corresponding entity to each relevant word, so as to minimize an energy given to the whole input. Even if such a SRL system using SME would not be able to cover the whole range of sentences and solve the highly complex problem of open-text semantic parsing, it could make a useful tool. Its process is illustrated in Figure 10.5 and detailed in the next section. Our focus is on the application of SME for Step (2), which is an all-words WSD step on extracted knowledge.

10.6.1 Methodology

This section details how SME could be inserted into a simple open-text semantic parsing system. In this framework, MRs are simple logical expressions $REL(A_0, \ldots, A_n)$, where REL is the relation symbol, and A_0, ..., A_n are its arguments. Note that several such forms can be recursively constructed to build

more complex structures. The goal is to parse open-domain raw text so a large set of relation types and arguments should be considered. Hence, WordNet is used for defining REL and A_i arguments as proposed in (Shi and Mihalcea, 2004), using the version introduced in Section 10.5.1. This results in a dictionary of 70,116 words that can be mapped to 40,943 possible entities. The simplified semantic parsing process consists of two stages.

Step (1): MR structure inference The first stage consists in preprocessing the text and inferring the structure of the MR. For this stage we use standard approaches, the major novelty of our work lies in applying SME for step (2).

We use the SENNA software[i] (Collobert et al., 2011) to perform part-of-speech (POS) tagging, chunking, lemmatization[i] and SRL. In the following, we call a *lemma* the concatenation of a lemmatized word and a POS tag (such as *_score_NN* or *_accompany_VB*). Note the absence of an integer suffix, which distinguishes a lemma from a WordNet synset: a lemma is allowed to be semantically ambiguous. The SRL step consists in assigning a semantic role label to each grammatical argument associated with a verb for each proposition.

As a simplification, only sentences that match the template (*subject*, *verb*, *direct object*) are considered here. Each of the three elements of the template is associated with a tuple of lemmatized words (i.e. with a multi-word phrase) and SRL is used to structure the sentence into the (lhs = subject, rel = verb, rhs = object) template. The order is not necessarily subject / verb / direct object in the raw text (e.g. in passive sentences). Clearly, the subject-verb-object composition causes the resulting MRs to have a straightforward structure (with a single relation), but this pattern is common and a good choice to test our ideas at scale. Learning to infer more elaborate grammatical patterns and MR structures is left as future work: we chose here to focus on handling the large scale of the set of entities.

To summarize, this step starts from a sentence and either rejects it or outputs a triplet of lemma tuples, one tuple for the subject, one for the relation or verb, and one for the direct object. To complete our semantic parse (or MR), lemmas must be converted into WordNet synsets, that is, we still have to perform disambiguation, which takes place in step (2).

i. Freely available from ml.nec-labs.com/senna/.

i. Lemmatization is not carried out with SENNA but with the NLTK toolkit (nltk.org) and transforms a word into its canonical or base form.

Step (2): Detection of MR entities This second step aims at identifying each semantic entity expressed in a sentence. Given a relation triplet (lhs^{lem}, rel^{lem}, rhs^{lem}) where each element of the triplet is associated with a tuple of lemmas, a corresponding triplet (lhs^{syn}, rel^{syn}, rhs^{syn}) is produced, where the lemmas are replaced by synsets. This step is a form of all-words WSD in a particular setup, i.e., w.r.t. the logical form of the semantic parse from Step (1). This can be either straightforward (some lemmas such as *_television_program_NN* or *_world_war_ii_NN* correspond to a single synset) or very challenging (*_run_VB* can be mapped to 33 different synsets and *_run_NN* to 10). Hence, in this proposed framework, MRs correspond to triplets of synsets (lhs^{syn}, rel^{syn}, rhs^{syn}), which can be reorganized to the form rel^{syn} (lhs^{syn}, rhs^{syn}), as shown in Figure 10.5.

Since the model is structured around triplets, MRs and WordNet relations are cast into the same scheme. For example, the WordNet relation (*_score_NN_2* , *_has_part*, *_musical_notation_NN_1*) fits the same pattern as our MRs, with the relation type *_has_part* playing the role of the verb, and the same entities being present in WordNet relations and MRs. The semantic matching energy function is trained to assign energies to triplets of lemmas and synsets. It is important to notice that such triplets involve a large number of relation types because any verb (under a lemma or a synset form) can act like it. Hence, our data, detailed in Section 10.6.2, contains more than 16,000 relation types.

The architecture introduced in Section 10.3.3 cannot be applied directly. Indeed, here $\mathcal{E}$ must be able to handle variable-size arguments, since for example there could be multiple lemmas in the subject part of the sentence. Hence, we add a pooling stage between steps (1) and (2) (of Section 10.3.3). The embeddings associated with all the symbols (synsets or lemmas) within the same tuple are aggregated by a pooling function π (we used the mean but other plausible candidates include the sum, the max, and combinations of several such element-wise statistics, such as in (Hamel et al., 2011)). This re-defines $\mathbf{e}_{lhs}$, $\mathbf{e}_{rel}$ and $\mathbf{e}_{rhs}$ as follows:

$$\begin{aligned}
\mathbf{e}_{lhs} &= \pi(\mathbf{e}_{lhs_1}, \mathbf{e}_{lhs_2}, \ldots), \\
\mathbf{e}_{rel} &= \pi(\mathbf{e}_{rel_1}, \mathbf{e}_{rel_2}, \ldots), \\
\mathbf{e}_{rhs} &= \pi(\mathbf{e}_{rhs_1}, \mathbf{e}_{rhs_2}, \ldots),
\end{aligned}$$

where lhs_j denotes the j-th individual element of the left-hand side tuple, etc.

We use this slightly modified semantic matching energy function to solve the WSD step. We label a triplet of lemmas $((lhs_1^{lem}, lhs_2^{lem}, \ldots), (rel_1^{lem}, \ldots), (rhs_1^{lem}, \ldots))$ with synsets in a greedy fashion, one lemma at a time. For labeling lhs_2^{lem} for instance, we fix all the remaining elements of the triplet to their lemma and select the synset leading to the lowest energy:

$$lhs_2^{syn} = \text{argmin}_{S \in \mathcal{C}(syn|lem)} \mathcal{E}((lhs_1^{lem}, S, \ldots), (rel_1^{lem}, \ldots), (rhs_1^{lem}, \ldots))$$

with $\mathcal{C}(syn|len)$ the set of allowed synsets to which lhs_2^{lem} can be mapped. We repeat that for all lemmas. We always use lemmas as context, and never the already assigned synsets. Future work should investigate more advanced inference schemes, which would probably be iterative and would gradually refine the estimated set of synsets taking into account their mutual agreement. Nevertheless, this is an efficient process as it only requires the computation of a small number of energies, equal to the number of senses for a lemma, for each position of a sentence. However, it requires good representations (i.e. good embedding vectors $\mathbf{e}_i$) for synsets and lemmas because they are used jointly to perform disambiguation.

10.6.2 Multi-task training

This section describes how we adapted the training scheme presented in Section 10.4 for learning embeddings for synsets and lemmas using various data sources.

Multiple data resources

In order to endow the model with as much common-sense knowledge as possible, the following heterogeneous data sources are combined. Their statistics are summarized in Table 10.5. On overall, this large-scale complex data groups 111,135 entities (70,116 lemmas and 40,943 synsets) and 16,765 relation types (7,714 encoded by verb synsets, 8,862 by verb lemmas, 18 for WordNet and 17 for ConceptNet) into 3,328,703 observed training triplets (and 20,000 for testing). Note that the actual number of triplets used for training our system is much larger than that, because we generate negative triplets (by perturbing oberved ones) during the learning phase (see Section 10.6.2).

Table 10.5 – Multiple data sources used for learning representations of 70,116 lemmas and 40,943 synsets. "Labeled" indicates when triplets consist of text lemmas for which the corresponding synsets are known. The total number of observed training triplets is 3,328,703 and the total number of relation types appearing in those triplets is more than 10,000.

Dataset	**Train. size**	**Test size**	**Labeled**	**Symbols**
WordNet	146,442	5,000	No	synsets
ConceptNet	11,332	0	No	lemmas
Wikipedia	2,146,131	10,000	No	lemmas
Extended WordNet	42,957	5,000	Yes	lemmas+synsets
Unambig. Wikipedia	981,841	0	Yes	lemmas+synsets

WordNet (WN). Already described in Section 10.5, this is the main resource, defining the dictionary of 40,943 entities. WordNet contains only relations between synsets. However, the disambiguation process needs embeddings for synsets and for lemmas. Following Havasi et al. (2010), we created two other versions of this dataset to leverage WN in order to also learn lemma embeddings: "Ambiguated" WN and "Bridge" WN. In "Ambiguated" WN synset entities of each triplet are replaced by one of their corresponding lemmas. "Bridge" WN is designed to teach the model about the connection between synset and lemma embeddings, thus in its relations the *lhs* or *rhs* synset is replaced by a corresponding lemma. Sampling training examples from WN involves actually sampling from one of its three versions, resulting in a triplet involving synsets, lemmas or both.

ConceptNet. This common-sense knowledge base (Liu and Singh, 2004) groups lemmas or groups of lemmas, which are linked together with rich semantic relations such as (*_kitchen_table_NN, _used_for, _eat_VB _breakfast_NN*). It is based on *lemmas* and not synsets, and it does not make distinctions between different senses of a word. Only triplets containing lemmas from the WN dictionary are kept, to finally obtain a total of 11,332 training triplets.

Wikipedia. This resource is simply raw text meant to provide knowledge to the model in an unsupervised fashion. In this work 50,000 Wikipedia articles were considered, although many more could be used. Using the protocol of Step (1) of Section 10.6.1, we created a total of 1,484,966 triplets of lemmas. Imperfect training

triplets (containing a mix of lemmas and synsets) are produced by performing the disambiguation step on one of the lemmas. This is equivalent to MAP (Maximum A Posteriori) training, i.e., we replace an unobserved latent variable by its mode according to a posterior distribution (i.e. to the minimum of the energy function, given the observed variables). We have used the 50,000 articles to generate more than 3M examples.

EXtended WordNet (XWN). XWN (Harabagiu and Moldovan, 2002) is built from WordNet *glosses*, syntactically parsed and with content words semantically linked to WN synsets. Using the protocol of Step (1) of Section 10.6.1, we processed these sentences and collected 47,957 lemma triplets for which the synset MRs were known. We removed 5,000 of these examples to use them as an evaluation set for the word-sense disambiguation task. With the remaining 42,957 examples, we created unambiguous training triplets to help the performance of the disambiguation algorithm: for each lemma in each triplet, a new triplet is created by replacing the lemma by its true corresponding synset and by keeping the other members of the triplet in lemma form (to serve as examples of lemma-based context). This led to a total of 786,105 training triplets, from which 10k were removed for validation.

Unambiguous Wikipedia (Wku). We added to this training set some triplets extracted from the Wikipedia corpus which were modified with the following trick: if one of its lemmas corresponds unambiguously to a synset, and if this synset maps to other ambiguous lemmas, we create a new triplet by replacing the unambiguous lemma by an ambiguous one. Hence, we know the true synset in that ambiguous context. This allowed to create 981,841 additional triplets with supervision.

Training procedure

The training algorithm described in Section 10.4 was used for all the data sources except XWN and Wku. In those two cases, positive triplets are composed of lemmas (as context) and of a disambiguated lemma replaced by its synset. Unlike for Wikipedia, this is labeled data, so we are certain that this synset is the valid sense. Hence, to increase training efficiency and yield a more discriminant disambiguation, in step 3 of the ranking algorithm with probability $\frac{1}{2}$ we either sample randomly from the set of all entities or we sample randomly from the set of

remaining candidate synsets corresponding to this disambiguated lemma (i.e. the set of its other meanings).

During training, we sequentially alternate between all sources, performing an update of the model parameters with one mini-batch of examples each time. Sizes of mini-batches differ between sources because we decided to split each source into 50 mini-batches. We always loop over sources in the same order. Training is stopped after 8,000 epochs on all sources (or 7 computation days). There is one learning rate for the g functions and another for the embeddings: their values are set using a grid search, choosing among $\{3., 1., 0.3, 0.1, 0.03, 0.01\}$ and $\{0.03, 0.01, 0.003, 0.001, 0.0003, 0.0001\}$ respectively. The model selection criterion is the mean rank from the entity ranking task on the WordNet validation set. Dimensions of embeddings and of the g output space are equal for these experiments and set to 50 (i.e. $d = p = 50$).

10.6.3 Related work

The application of SME to WSD is related to work on vector-based models of word meaning (Lund and Burgess, 1996; Landauer and Dumais, 1997) and neural language models (Bengio, 2008; Collobert et al., 2011), in the sense that we learn a vector representation for each lemma to disambiguate. More precisely, it is connected to models aiming at composing such vector embeddings for obtaining phrase or context representations (Mitchell and Lapata, 2008), via tensor products (Smolensky, 1990) or matrix operations (Paccanaro and Hinton, 2001; Socher et al., 2012). However, besides that many of these methods would not scale to the problems introduced here, there exist a major difference with our work: we aim at learning jointly representations for words (lemmas) and senses (synsets), considering structures within language (via SRL triplets) and within a knowledge base (via WordNet triplets) together. To the best of our knowledge, this is the first attempt of mixing relational embeddings of a knowledge base and word embeddings. Note that the original architecture of SME could be adapted for recursively learning phrase representations as proposed by Socher et al. (2012) but this is beyond the scope of this paper.

Our approach is also connected to previous work targeting to improve WSD by using extra-knowledge by either automatically acquiring examples (Martinez et al.,

Table 10.6 – Relation type ranking on Wikipedia. Comparisons between two versions of SME (this paper) and SE. Mean/median predicted rank and precision@10 (p@10, in %) are computed on the test set. Best performances are indicated in bold.

Method	**Rank** (median/mean)	**p@10**
SME(linear)	**92** / **267**	**1.951**
SME(bilinear)	97 / 286	1.862
SE	118 / 283	0.882

2008) or by connecting different knowledge bases (Havasi et al., 2010), but uses a totally different method.

Finally, our ambition towards open-text semantic parsing is related to previous work by Shi and Mihalcea (2004), who proposed a rule-based system for open-text semantic parsing using WordNet and FrameNet (Baker et al., 1998) and by Giuglea and Moschitti (2006), who proposed a model to connect WordNet, VerbNet and PropBank (Kingsbury and Palmer, 2002) for semantic parsing using tree kernels. Poon and Domingos (2009, 2010) introduced a method based on Markov-Logic Networks for unsupervised semantic parsing that can be also used for information acquisition. However, instead of connecting MRs to an existing ontology as done here, it constructs a new ontology and does not leverage pre-existing knowledge.

10.6.4 Experiments

To assess the performance w.r.t. the multi-task training and the diverse data sources, we evaluated models trained with several combinations of data sources. WN denotes SME models trained on WordNet, "Ambiguated" WordNet and "Bridge" WordNet, while ALL denotes models are trained on all sources.

Entity ranking

We first evaluated SME(linear), SME(bilinear), SE and RESCAL in ranking on Wikipedia (Wk) because this dataset offers a test-bed with many types of relation (5,448). The goal of the task was to rank the correct rel given a pair of (lhs, rhs), because this is much easier than ranking lhs or rhs that contain multiple lemmas. All methods has only been trained on the Wk training set of 2,146,131 observed lemma triplets and evaluated on the corresponding test set of 10k triplets. Hence,

Table 10.7 – Word-sense Disambiguation results. F1-scores (in %) of different versions of SME (linear/bilinear, with different data sources, combined with the most frequent sense (MFS) information or not) are compared on XWN and a subset of SensEval-3, with previous work SE ; Emb., the unstructured version of SME ; Lesk, a standard WSD method ; Gamble, the algorithm that won SensEval-3 on the full test set ; and Random, which chooses uniformly among allowed synsets. Best performing methods, with a significant difference, are indicated in bold.

Method		XWN	SensEval3
SME(linear)	ALL+MFS	**72.3**	**65.9**
	ALL	66.0	44.7
	WN	31.6	29.3
SME(bilinear)	ALL+MFS	**72.1**	**68.3**
	ALL	67.1	49.5
	WN	29.6	28.4
SE	ALL+MFS	68.9	**68.3**
	ALL	51.9	47.1
Emb.	ALL+MFS	68.7	**69.2**
	ALL	39.2	40.4
Lesk		70.2	50.5
Gamble		n/a	**66.4**
MFS		67.2	**67.8**
Random		26.7	29.6

we measure the mean and median predicted ranks and the prediction@10, computed with the following procedure. For each test triplet, the relation type is removed and replaced by each of the relation types of the dictionary in turn. Energies of those corrupted triplets are computed by the model and sorted by ascending order and the rank of the correct type is stored. Table 10.6 reports the average and median of those predicted ranks and the precision@10 (or p@10). Results clearly indicate the advantage of using SME on data with large numbers of relation types, compared to SE. There is no result for RESCAL because we have not been able to run it on Wk. This confirms that SME is the method of choice when dealing with large-scale multi-relational data.

Word-sense disambiguation

Performance on WSD is assessed on two test sets: the XWN test set and a subset of English All-words WSD task of SensEval-3. [i] For the latter, we processed the original data using the protocol of Step (1) of Section 10.6.1 and obtained a total of 208 words to disambiguate (out of $\approx$ 2000 originally). We compare with results obtained by Emb., which uses the same embeddings as SME, but without the structure of its energy function. The performance of SE and of the most frequent sense for a given lemma (MFS) as well as of the standard Lesk algorithm are also evaluated. MFS frequencies have been obtained from WordNet, which provides such information (in `cntlist` files). We attempted to adapt RESCAL to this task, but the code took too long to converge with the whole set or even a reduced set of relation types. We implemented a version of Lesk by following the work of Banerjee and Pedersen (2002), which performs WSD based on the WordNet::Similarity package. Hence, a triplet of lemmas is labeled with the triplet of synsets which exhibit the highest cumulated Lesk similarity measure (according to WordNet::Similarity): this cumulated measure is computed as the sum of Lesk similarities of all pairs of synsets composing the triplet. Finally, we report the F1-score of Gamble (Decadt et al., 2004), winner of Senseval-3, on our subset of its data. [i]

F1 scores are presented in Table 10.7. The difference between models learnt on ALL and on WN indicates that the information from Wikipedia, XWN and Wku is crucial (+35%) and yields performance equivalent to that of MFS (a strong baseline in WSD) on the XWN test set. Performance of the model trained on WN alone are roughly equivalent to that of Random. This confirms that knowledge from WordNet and free text are difficult to combine. Still, it is interesting to see that SME is able to train on these various sources and to somewhat capture information from them all. Emb., without the structure taking the relation type into account, performs very poorly. SE is affected by the large number of relation types: on XWN, it remains 15% below SME. It can not learn proper matrix representations (d^2 parameters) for all verbs involved as relation types, especially for the rare ones. SME does not undergo this problem.

i. More details at www.senseval.org/senseval3.

i. A side effect of our preprocessing of SensEval-3 data is that our subset contains mostly frequent words. This is easier for MFS than for Gamble because Gamble is efficient on rare terms. Hence, Gamble performs worse than during the challenge and seems to be outperformed by MFS. However, performance of both systems are statistically equivalent.

Table 10.8 – Predicted triplets reported by SME(bilinear) trained on all data sources (ALL), by ReVerb and using the Lesk measure from Wordnet::Similarity.

	SME(bilinear)	ReVerb	Lesk
lhs	_army_NN_1	army	_army_NN_1
rel	_attack_VB_1	attacked	_attack_VB_1
top ranked *rhs*	_troop_NN_4	the city	_army_unit_NN_1
	_armed_service_NN_1	the village	_army_corps_NN_1
	_ship_NN_1	Israel	_invade_VB_1
	_territory_NN_1	Poland	_military_unit_NN_1
	_military_unit_NN_1	force	_armed_service_NN_1
top ranked *lhs*	_business_firm_NN_1	People	_monetary_system_NN_1
	_person_NN_1	Players	_money_supply_NN_1
	_family_NN_1	Interest	_currency_NN_1
	_payoff_NN_3	Work	_monetary_standard_NN_1
	_card_game_NN_1	Students	_monetary_resource_NN_1
rel	_earn_VB_1	earn	_earn_VB_1
rhs	_money_NN_1	money	_money_NN_1

Performance can be greatly improved by combining models trained on the ALL sources and the MFS score. To do so, we converted the frequency information into an energy by taking minus the log frequency and used it as an extra energy term. The total energy function is used for disambiguation. This yields the results denoted by ALL+MFS which achieve the best results of all the methods tried. On the XWN test set, the difference in performance between SME(linear) and SME(bilinear) is not statistically significant but their gap with Lesk is (according to a χ^2 test at the 0.05 level). For SensEval-3, differences between F1 scores of SME(linear), SME(bilinear) (with ALL+MFS), Gamble and MFS are not statistically significant (χ^2 test – 0.05 level). Emb. and SE can also be upgraded using MFS information. On SensEval-3, this makes them to equal the best performance. However, on XWN (a better indicator), they remain significantly outperformed by SME. Our method seems to be the best for encoding supportive information to that of MFS: combined with SME, MFS's F1-score increases by 5%.

WordNet enrichment

WordNet uses a limited number of relation types (18 in our version), and does not consider most verbs as relations. Thanks to our multi-task training and unified

representation for MRs and WordNet relations, our model is potentially able to generalize to such relations that do not exist in WordNet originally.

As illustration, predicted lists of synsets for relation types that do not exist in WordNet are given in Table 10.8. We also compare with lists returned by ReVerb (Fader et al., 2011) (an information extraction tool having extracted information from millions of web pages, [i] to be compared with our 50k Wikipedia articles + knowledge bases). Lists from both systems seem to reflect common sense. However, contrary to our system, ReVerb does not disambiguate different senses of a lemma, and thus it cannot connect its knowledge to an existing resource to enrich it. We also provide the lists predicted by the Lesk measure of Wordnet::Similarity, where the score of a WordNet synset is simply defined as the sum of its similarity measures with both input synsets. The predictions are semantically related but they do not reflect the triplet structure because semantic roles of lhs, rel or rhs do not mean anything for Lesk.

10.7 Conclusion

This paper presented SME, a new energy-based model for learning multi-relational semantics. This method encodes multi-relational graphs or tensors into a flexible continuous vector space in which the original data is stored and enhanced. We empirically showed that SME: (i) is highly competitive with the state-of-the-art methods for modeling multi-relational data, (ii) outperforms them in large-scale conditions, (iii) can be successfully trained on graphs with tens of thousands of entities and thousands of kinds of relation (more than 100k entities, 10k relation types and 3.5M training triplets). This is the only method able to be efficient on all the different datasets considered in this paper.

In addition, we presented how SME could be applied to perform WSD using a dictionary of more than 70,000 words based on WordNet. Our system, trained on WordNet and free text (and other sources), can capture the deep semantics of sentences in its energy function, which, combined with most frequent sense information, leads to improvement in disambiguation over standard methods. Future

i. See the online ReVerb demo at http://openie.cs.washington.edu/.

work could explore the capabilities of such systems further including more general sentence structures, other semantic tasks, and more evolved grammars, e.g. with FrameNet (Baker et al., 1998; Coppola and Moschitti, 2010).

An interesting extension of the model presented here extends its applicability to domains where the objects of interest are not all symbolic (i.e., from a finite set). In that case, one cannot associate a free parameter (its embedding vector) to each possible object. An example of such objects are image patches, which are generally described by a "raw" feature vector. We could learn a mapping from this raw feature space to the embedding space, where the symbolic objects are mapped (in a similar fashion as WSABIE (Weston et al., 2010)). Whereas for discrete object, one can view the object's embedding as the product of the embedding matrix by a one-hot vector (with a 1 at the position associated with the object symbol), for continuous objects, in the linear mapping case, the "embedding matrix" maps the raw features (richer than one-hot) to the embedding vector. In this way, relations could involve both discrete and continuous objects. Such extensions are possible because of the flexibility and scalability, that models based on embeddings like SME or WSABIE offer for dealing with multimodal inputs.

10.8 Acknowledgements

The authors would like to acknowledge Léon Bottou, Ronan Collobert, Nicolas Usunier, Nicolas Le Roux, Rodolphe Jenatton and Guillaume Obozinski for inspiring discussions. This work was supported by the French ANR (EVEREST-12-JS02-005-01), the Pascal2 European NoE, the DARPA Deep Learning Program, NSERC, CIFAR, the Canada Research Chairs, and Compute Canada.

11 Conclusion

Les travaux réalisés dans le cadre de cette thèse couvrent des thématiques de recherche importantes dans le domaine des Réseaux de Neurones Artificiels et présentent des exemples d'applications de ce type de modèles en Traitement des Langues Naturelles.

Le premier thème abordé est celui des problèmes d'optimisation des architectures profondes. Nous avons montré que des choix simples, comme la procédure d'initialisation des poids ou la fonction d'activation, jouent un rôle important sur la dynamique d'apprentissage. Nous avons proposé l'initialisation normalisée et la fonction d'activation rectificatrice. Ces deux composantes sont maintenant utilisées couramment en recherche et en industrie. Grâce à ces dernières et à des méthodes complémentaires, comme la technique du "Dropout", il est maintenant possible de se passer de la procédure du pré-entraînement non-supervisé pour entraîner des réseaux profonds, à condition d'avoir assez de données supervisées. Ces méthodes ont permis d'améliorer l'état de l'art dans divers champs d'application comme la vision par ordinateur, le traitement de la parole et le traitement des langues naturelles.

À l'inverse, lorsque la quantité de données supervisées est insuffisante, il est primordial d'utiliser un apprentissage non-supervisé afin d'obtenir une représentation des données facilitant l'apprentissage. Ceci est le deuxieme thème abordé dans cette thèse. Plus précisemment, nous nous sommes intéressés à l'adaptation de domaine pour l'analyse de sentiment, une tâche du traitement des langues naturelles. Nous avons montré que les Auto-Encodeurs Débruitants permettent d'obtenir des représentations pour lesquelles les facteurs de variations correspondants aux domaines et ceux correspondants aux sentiments sont en partie démêlés, ce qui est hautement désirable pour l'adaptation de domaine. Nous avons montré que cette méthode est l'état de l'art dans ce contexte.

Le dernier thème présenté dans cette thèse concerne les applications en traitement des langues naturelles des Réseaux de Neurones Artificiels. Nous avons d'abord

exploré des applications en analyse de sentiment et en adaptation de domaine. Puis, nous avons montré les bénéfices ce type de modèles pour les données relationnelles, cela par le biais d'un modèle à base d'énergie pouvant être utilisé pour la prédiction de liens, le classement d'entités et la disambiguation de sens dans le contexte de l'analyse de rôle sémantique.

Autour de ces trois thèmes, les perspectives de recherche futures sont riches. Voici quelques exemples de directions que je souhaite poursuivre:

- La neuroscience est une source d'inspiration intéressante, nous savons *de facto* que le cerveau est un système performant pour de nombreuses tâches. Il est utile de s'inspirer de son fonctionnement pour tenter d'améliorer nos modèles et algorithmes, sans pour autant se limiter à reproduire les données biologiques. En faisant cela, nous pouvons tirer parti des centaines de millions d'années de l'évolution du système nerveux et, aussi, mieux comprendre les principes importants de son fonctionnement.
- L'apprentissage de représentations, et notamment le démêlement des facteurs de variations, est un sujet de première importance. En effet, les représentations démêlées ont de nombreux avantages: apprentissage rapide, bonne généralisation, possibilité de transférer l'apprentissage.
- Les modèles à base d'"embeddings", qui permettent d'obtenir une représentation distribuée pour le texte, ont déjà prouvé leur utilité. Cependant, l'utilisation de telles méthodes pour représenter des phrases ou des textes entiers est encore difficile, il s'agit d'un sujet de recherche actif. Le language est une composante cruciale pour l'intelligence humaine, notamment pour l'apprentissage, améliorer la manipulation de ce type de données par la machine revêt donc une importance particulière.

Bibliographie

Amari, S. (1997). Neural learning in structured parameter spaces - natural Riemannian gradient. In *Advances in Neural Information Processing Systems*, pp. 127–133. MIT Press.

Araque, A. and M. Navarrete (2010). Glial cells in neuronal network function. *Philosophical Transactions of the Royal Society B: Biological Sciences 365*(1551), 2375–2381.

Attwell, D. and S. Laughlin (2001). An energy budget for signaling in the grey matter of the brain. *Journal of Cerebral Blood Flow and Metabolism 21*(10), 1133–1145.

Baker, C., C. Fillmore, and J. Lowe (1998). The berkeley FrameNet project. In *ACL '98*, pp. 86–90.

Banerjee, S. and T. Pedersen (2002). An adapted lesk algorithm for word sense disambiguation using wordnet. In *Proceedings of the Third International Conference on Computational Linguistics and Intelligent Text Processing*, CICLing '02, pp. 136–145. Springer-Verlag.

Ben-David, S., J. Blitzer, K. Crammer, and P. M. Sokolova (2007). Analysis of representations for domain adaptation. In *Proc. of NIPS 20*.

Bengio, Y. (2008). Neural net language models. *Scholarpedia 3*(1), 3881.

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

Bengio, Y. and al (2010). Deep self-taught learning for handwritten character recognition. Deep Learning and Unsupervised Feature Learning Workshop âĂŤ NIPS '10.

Bengio, Y. and O. Delalleau (2011). Shallow vs. deep sum-product networks. In *NIPS'11*.

Bengio, Y., R. Ducharme, P. Vincent, and C. Jauvin (2003). A neural probabilistic language model. *JMLR 3*, 1137–1155.

Bengio, Y., P. Lamblin, D. Popovici, and H. Larochelle (2007). Greedy layer-wise training of deep networks. In B. Schölkopf, J. Platt, and T. Hoffman (Eds.), *Advances in Neural Information Processing Systems 19 (NIPS'06)*, pp. 153–160. MIT Press.

Bengio, Y., Y. Li, G. Alain, and P. Vincent (2013). Generalized denoising auto-encoders as generative models. Technical Report arXiv:1305.6663, Universite de Montreal.

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

Bergstra, J., O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio (2010, June). Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*. Oral Presentation.

Bergstra, J., G. Desjardins, P. Lamblin, and Y. Bengio (2009, April). Quadratic polynomials learn better image features. Technical Report 1337, Département d'Informatique et de Recherche Opérationnelle, Université de Montréal.

Blitzer, J., M. Dredze, and F. Pereira (2007). Biographies, bollywood, boom-boxes and blenders: Domain adaptation for sentiment classification. In *Proc. of ACL '07*.

Blitzer, J., R. McDonald, and F. Pereira (2006). Domain adaptation with structural correspondence learning. In *Proc. of EMNLP '06*.

Bordes, A., S. Chopra, and J. Weston (2014). Question answering with subgraph embeddings. *CoRR*.

Bordes, A., X. Glorot, J. Weston, and Y. Bengio (2012). Joint learning of words and meaning representations for open-text semantic parsing. In *Proc. of the 15th Intern. Conf. on Artif. Intel. and Stat.*, Volume 22, pp. 127–135. JMLR.

Bordes, A., N. Usunier, R. Collobert, and J. Weston (2010). Towards understanding situated natural language. In *Proc. of the 13th Intern. Conf. on Artif. Intel. and Stat.*, Volume 9, pp. 65–72.

Bordes, A., N. Usunier, A. Garcia-Duran, J. Weston, and O. Yakhnenko (2013). Translating embeddings for modeling multi-relational data. In C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Weinberger (Eds.), *Advances in Neural Information Processing Systems 26*, pp. 2787–2795. Curran Associates, Inc.

Bordes, A., J. Weston, R. Collobert, and Y. Bengio (2011). Learning structured embeddings of knowledge bases. In *Proceedings of the 25th Conference on Artificial Intelligence (AAAI-11)*, San Francisco, USA.

Boser, B., I. Guyon, and V. Vapnik (1992). A training algorithm for optimal margin classifiers. *Proceedings of the fifth annual workshop on Computational learning theory*, 144–152.

Bottou, L. (2011). From machine learning to machine reasoning. Technical report, arXiv.1102.1808.

Bottou, L. and P. Gallinari (1991). A framework for the cooperation of learning algorithms. In R. Lippman, J. Moody, and D. Touretzky (Eds.), *Advances in Neural Information Processing Systems 3 (NIPS'90)*, Denver, CO, pp. 781–788.

Bourrely, J. (1989). Parallelization of a neural learning algorithm on a hypercube. In *Hypercube and distributed computers*, pp. 219–229. Elsiever Science Publishing, North Holland.

Bradley, D. (2009). *Learning in Modular Systems*. Ph. D. thesis, The Robotics Institute, Carnegie Mellon University.

Bush, P. C. and T. J. Sejnowski (1995). *The cortical neuron*. Oxford university press.

Cambria, E., A. Hussain, C. Havasi, and C. Eckl (2009). Affectivespace: Blending common sense and affective knowledge to perform emotive reasoning. In *WOMSA at CAEPIA*, pp. 32–41.

Candes, E. and T. Tao (2005). Decoding by linear programming. *IEEE Transactions on Information Theory 51*(12), 4203–4215.

Caruana, R. (1995). Learning many related tasks at the same time with backpropagation. In G. Tesauro, D. Touretzky, and T. Leen (Eds.), *Advances in Neural Information Processing Systems 7 (NIPS'94)*, Cambridge, MA, pp. 657–664. MIT Press.

Chapelle, O. and D. Erhan (2011). Improved preconditioner for hessian free optimization. *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*.

Chen, M., Z. Xu, K. Q. Weinberger, and F. Sha (2012). Marginalized stacked denoising autoencoders. Learning Workshop'2012.

Chu, W. and Z. Ghahramani (2009). Probabilistic models for incomplete multi-dimensional arrays. *Journal of Machine Learning Research - Proceedings Track 5*, 89–96.

Collobert, R. and J. Weston (2008). A unified architecture for natural language processing: Deep neural networks with multitask learning. In W. W. Cohen, A. McCallum, and S. T. Roweis (Eds.), *Proceedings of the Twenty-fifth International Conference on Machine Learning (ICML'08)*, pp. 160–167. ACM.

Collobert, R., J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa (2011). Natural language processing (almost) from scratch. *Journal of Machine Learning Research 12*, 2493–2537.

Coppola, B. and A. Moschitti (2010). A general purpose FrameNet-based shallow semantic parser. In *Proceedings of the International Conference on Language Resources and Evaluation (LREC'10)*.

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

Cristianini, N. and J. Shawe-Taylor (2000). *An Introduction to Support Vector Machines and other kernel-based learning methods*. Cambridge, UK: Cambridge University Press.

Dai, W., G.-R. Xue, Q. Yang, and Y. Yu (2007). Transferring naive Bayes classifiers for text classification. In *Proc. of AAAI '07*.

Daumé III, H. and D. Marcu (2006). Domain adaptation for statistical classifiers. *JAIR 26*, 101–126.

Dauphin, Y., R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli, and Y. Bengio (2014). Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In *NIPS'2014*.

Dayan, P. and L. Abott (2001). *Theoretical neuroscience*. MIT press.

Decadt, B., V. Hoste, W. Daeleamns, and A. van den Bosh (2004). Gamble, genetic algorithm optimization of memory-based WSD. In *Proceeding of ACL/SIGLEX Senseval-3*.

Deng, J., R. Xia, Z. Zhang, Y. Liu, and B. Schuller (2014). Introducing shared-hidden-layer autoencoders for transfer learning and their application in acoustic emotion recognition. In *IEEE International Conference on Acoustics, Speech and Signal Processing,ICASSP 2014, Florence, Italy, May 4-9, 2014*, pp. 4818–4822.

Denham, W. (1973). *The detection of patterns in Alyawarra nonverbal behavior*. Ph. D. thesis.

Desjardins, G., A. Courville, and Y. Bengio (2012). Disentangling factors of variation via generative entangling.

Doi, E., D. C. Balcan, and M. S. Lewicki (2006). A theoretical analysis of robust coding over noisy overcomplete channels. In Y. Weiss, B. Schölkopf, and J. Platt (Eds.), *Advances in Neural Information Processing Systems 18 (NIPS'05)*, pp. 307–314. Cambridge, MA: MIT Press.

Douglas, R. and al. (2003). Recurrent excitation in neocortical circuits. *Science 269*(5226), 981–985.

Dugas, C., Y. Bengio, F. Belisle, C. Nadeau, and R. Garcia (2001). Incorporating second-order functional knowledge for better option pricing. In T. K. Leen and T. Dietterich (Eds.), *Advances in Neural Information Processing Systems 13 (NIPS'00)*. MIT Press.

Erhan, D., Y. Bengio, A. Courville, P.-A. Manzagol, P. Vincent, and S. Bengio (2010, February). Why does unsupervised pre-training help deep learning ? *Journal of Machine Learning Research 11*, 625–660.

Erhan, D., Y. Bengio, A. Courville, and P. Vincent (2009). Visualizing higher-layer features of a deep network. Technical Report 1341, Université de Montréal.

Erhan, D., P.-A. Manzagol, Y. Bengio, S. Bengio, and P. Vincent (2009, April). The difficulty of training deep architectures and the effect of unsupervised pre-training. In *Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics (AISTATS 2009)*, pp. 153–160.

Fader, A., S. Soderland, and O. Etzioni (2011). Identifying relations for open information extraction. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '11, pp. 1535–1545. Association for Computational Linguistics.

Franz, T., A. Schultz, S. Sizov, and S. Staab (2009). Triplerank: Ranking semantic web data by tensor decomposition. In *Proceedings of the 8th International Semantic Web Conference*, ISWC '09, pp. 213–228.

Getoor, L. and B. Taskar (2007). *Introduction to Statistical Relational Learning (Adaptive Computation and Machine Learning)*. The MIT Press.

Ghifary, M., W. B. Kleijn, and M. Zhang (2014). Domain adaptive neural networks for object recognition. *CoRR abs/1409.6041*.

Giuglea, A. and A. Moschitti (2006). Shallow semantic parsing based on FrameNet, VerbNet and PropBank. In *Proceeding of the 17th European Conference on Artificial Intelligence (ECAI'06)*, pp. 563–567.

Glorot, X. and Y. Bengio (2010, May). Understanding the difficulty of training deep feedforward neural networks. In *JMLR W&CP: Proceedings of the Thirteenth In-*

ternational Conference on Artificial Intelligence and Statistics (AISTATS 2010), Volume 9, pp. 249–256.

Glorot, X., A. Bordes, and Y. Bengio (2011). Deep sparse rectifier neural networks. In *Proc. of AISTATS '11.*

Goodfellow, I., Q. Le, A. Saxe, and A. Ng (2009). Measuring invariances in deep networks. In Y. Bengio, D. Schuurmans, C. Williams, J. Lafferty, and A. Culotta (Eds.), *Advances in Neural Information Processing Systems 22 (NIPS'09)*, pp. 646–654.

Goodfellow, I. J., D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio (2013). Maxout networks. In *ICML'2013*.

Grother, P. (1995). Handprinted forms and character database, NIST special database 19. In *National Institute of Standards and Technology (NIST) Intelligent Systems Division (NISTIR)*.

Gutmann, M. and A. Hyvärinen (2010). Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *Proc. Int. Conf. on Artificial Intelligence and Statistics (AISTATS2010)*.

Hahnloser, R. L. T. (1998). On the piecewise analysis of networks of linear threshold neurons. *Neural Netw. 11*(4), 691–697.

Hamel, P., S. Lemieux, Y. Bengio, and D. Eck (2011). Temporal pooling and multiscale learning for automatic annotation and ranking of music audio. In *In Proceedings of the 12th International Conference on Music Information Retrieval (ISMIRÃćâĆňâĎć11)*.

Harabagiu, S. and D. Moldovan (2002). Knowledge processing on extended WordNet. In C. Fellbaum (Ed.), *WordNet: An Electronic Lexical Database and Some of its Applications*, pp. 379–405. MIT Press.

Harshman, R. A. and M. E. Lundy (1994, August). Parafac: parallel factor analysis. *Comput. Stat. Data Anal. 18*(1), 39–72.

Håstad, J. and M. Goldmann (1991). On the power of small-depth threshold circuits. *Computational Complexity 1*, 113–129.

Havasi, C., R. Speer, and J. Pustejovsky (2010). Coarse Word-Sense Disambiguation using common sense. In *AAAI Fall Symposium Series*.

Hebb, D. O. (1949). *The Organization of Behavior*. New York: Wiley.

Hinton, G. E., S. Osindero, and Y. Teh (2006). A fast learning algorithm for deep belief nets. *Neural Computation 18*, 1527–1554.

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

Hodgkin, A. L. and A. F. Huxley (1952, August). A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of physiology 117*(4), 500–544.

Hornik, K., M. Stinchcombe, and H. White (1989). Multilayer feedforward networks are universal approximators. *Neural Networks 2*, 359–366.

Jarrett, K., K. Kavukcuoglu, M. Ranzato, and Y. LeCun (2009). What is the best multi-stage architecture for object recognition? In *Proc. International Conference on Computer Vision (ICCV'09)*, pp. 2146–2153. IEEE.

Jenatton, R., N. L. Roux, A. Bordes, and G. R. Obozinski (2012). A latent factor model for highly multi-relational data. In F. Pereira, C. Burges, L. Bottou, and K. Weinberger (Eds.), *Advances in Neural Information Processing Systems 25*, pp. 3167–3175. Curran Associates, Inc.

Jiang, J. and C. Zhai (2007). Instance weighting for domain adaptation in nlp. In *Proc. of ACL '07*.

Kemp, C., J. B. Tenenbaum, T. L. Griffiths, T. Yamada, and N. Ueda (2006). Learning systems of concepts with an infinite relational model. In *Proceedings of the 21st national conference on Artificial intelligence - Volume 1*, AAAI'06, pp. 381–388. AAAI Press.

Kingsbury, P. and M. Palmer (2002). From Treebank to PropBank. In *Proc. of the 3rd International Conference on Language Resources and Evaluation*.

Koch, C., T. Poggio, and V. Torre (1983). Nonlinear interactions in a dendritic tree: localization, timing, and role in information processing. *Proceedings of the National Academy of Sciences 80*(9), 2799âĂŞ2802.

Kok, S. and P. Domingos (2007). Statistical predicate invention. In *Proceedings of the 24th international conference on Machine learning*, ICML '07, New York, NY, USA, pp. 433–440. ACM.

Krizhevsky, A. and G. Hinton (2009). Learning multiple layers of features from tiny images. Technical report, University of Toronto.

Krizhevsky, A., I. Sutskever, and G. Hinton (2012). ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25 (NIPS'2012)*.

Landauer, T. and S. Dumais (1997). A solution to plato's problem: The latent semantic analysis theory of acquisition, induction, and representation of knowledge. *Psychological Review; Psychological Review 104*(2), 211.

Larochelle, H., Y. Bengio, J. Louradour, and P. Lamblin (2009, January). Exploring strategies for training deep neural networks. *Journal of Machine Learning Research 10*, 1–40.

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

Le Roux, N., P.-A. Manzagol, and Y. Bengio (2008). Topmoumoute online natural gradient algorithm. In J. Platt, D. Koller, Y. Singer, and S. Roweis (Eds.), *Advances in Neural Information Processing Systems 20 (NIPS'07)*, pp. 849–856. Cambridge, MA: MIT Press.

LeCun, Y. and Y. Bengio (1994). Word-level training of a handritten word recognizer based on convolutional neural networks. In IEEE (Ed.), *International Conference on Pattern Recognition (ICPR'94)*, Jerusalem 1994.

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

LeCun, Y., L. Bottou, G. B. Orr, and K.-R. Müller (1998). Efficient backprop. In *Neural Networks, Tricks of the Trade*, Lecture Notes in Computer Science LNCS 1524. Springer Verlag.

Lecun, Y., S. Chopra, R. Hadsell, R. marc'aurelio, and f. Huang (2006). A tutorial on Energy-Based learning. In G. Bakir, T. Hofman, B. schölkopf, A. Smola, and B. Taskar (Eds.), *Predicting Structured Data*. MIT Press.

LeCun, Y., F.-J. Huang, and L. Bottou (2004). Learning methods for generic object recognition with invariance to pose and lighting. In *Proceedings of the Computer Vision and Pattern Recognition Conference (CVPR'04)*, Volume 2, Los Alamitos, CA, USA, pp. 97–104. IEEE Computer Society.

Lee, H., A. Battle, R. Raina, and A. Ng (2007). Efficient sparse coding algorithms. In B. Schölkopf, J. Platt, and T. Hoffman (Eds.), *Advances in Neural Information Processing Systems 19 (NIPS'06)*, pp. 801–808. MIT Press.

Lee, H., C. Ekanadham, and A. Ng (2008). Sparse deep belief net model for visual area V2. In J. Platt, D. Koller, Y. Singer, and S. Roweis (Eds.), *Advances in Neural Information Processing Systems 20 (NIPS'07)*, pp. 873–880. Cambridge, MA: MIT Press.

Lee, H., R. Grosse, R. Ranganath, and A. Y. Ng (2009). Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In L. Bottou and M. Littman (Eds.), *Proceedings of the Twenty-sixth International Conference on Machine Learning (ICML'09)*. Montreal, Canada: ACM.

Lee, H., P. Pham, Y. Largman, and A. Ng (2009). Unsupervised feature learning for audio classification using convolutional deep belief networks. In Y. Bengio, D. Schuurmans, C. Williams, J. Lafferty, and A. Culotta (Eds.), *Advances in Neural Information Processing Systems 22 (NIPS'09)*, pp. 1096–1104.

Lennie, P. (2003). The cost of cortical computation. *Current Biology 13*, 493–497.

Li, S. and C. Zong (2008). Multi-domain adaptation for sentiment classification: Using multiple classifier combining methods. In *Proc. of NLP-KE '08*.

Liu, H. and P. Singh (2004). Focusing on conceptnet's natural language knowledge representation. In *Proc. of the 8th Intl Conf. on Knowledge-Based Intelligent Information and Engineering Syst.*

Lund, K. and C. Burgess (1996). Producing high-dimensional semantic spaces from lexical co-occurrence. *Behavior Research Methods 28*(2), 203–208.

Mairal, J., F. Bach, J. Ponce, G. Sapiro, and A. Zisserman (2009). Supervised dictionary learning. In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou (Eds.), *Advances in Neural Information Processing Systems 21 (NIPS'08)*, pp. 1033–1040. NIPS Foundation.

Martens, J. (2010, June). Deep learning via Hessian-free optimization. In L. Bottou and M. Littman (Eds.), *Proceedings of the Twenty-seventh International Conference on Machine Learning (ICML-10)*, pp. 735–742. ACM.

Martinez, D., O. de Lacalle, and E. Agirre (2008, September). On the use of automatically acquired examples for all-nouns word sense disambiguation. *J. Artif. Int. Res. 33*, 79–107.

McCray, A. T. (2003). An upper level ontology for the biomedical domain. *Comparative and Functional Genomics 4*, 80–88.

Mikolov, T., S. Kombrink, L. Burget, J. Cernocky, and S. Khudanpur (2011). Extensions of recurrent neural network language model. In *Proc. 2011 IEEE international conference on acoustics, speech and signal processing (ICASSP 2011)*.

Miller, K., T. Griffiths, and M. Jordan (2009). Nonparametric latent feature models for link prediction. In *Advances in Neural Information Processing Systems 22*, pp. 1276–1284.

Minsky, M. L. and S. A. Papert (1969). *Perceptrons*. Cambridge: MIT Press.

Mitchell, J. and M. Lapata (2008). Vector-based models of semantic composition. *proceedings of ACL-08: HLT*, 236–244.

Mnih, A. and G. E. Hinton (2009). A scalable hierarchical distributed language model. In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou (Eds.), *Advances in Neural Information Processing Systems 21 (NIPS'08)*, pp. 1081–1088.

Mooney, R. (2004). Learning Semantic Parsers: An Important But Under-Studied Problem. In *Proc. of the 19th AAAI Conf. on Artif. Intel.*

Nair, V. and G. E. Hinton (2010). Rectified linear units improve restricted Boltzmann machines. In L. Bottou and M. Littman (Eds.), *Proceedings of the Twenty-seventh International Conference on Machine Learning (ICML-10)*, pp. 807–814. ACM.

Nickel, M., V. Tresp, and H.-P. Kriegel (2011). A three-way model for collective learning on multi-relational data. In L. Getoor and T. Scheffer (Eds.), *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, ICML '11, pp. 809–816. ACM.

Nickel, M., V. Tresp, and H.-P. Kriegel (2012). Factorizing yago: scalable machine learning for linked data. In *Proceedings of the 21st international conference on World Wide Web*, WWW '12, pp. 271–280.

Nocedal, J. and S. Wright (2006). *Numerical Optimization*. Springer.

Olshausen, B. A. and D. J. Field (1997, December). Sparse coding with an overcomplete basis set: a strategy employed by V1 ? *Vision Research 37*, 3311–3325.

Ozair, S. and Y. Bengio (2014). Deep directed generative autoencoders. Technical report, U. Montreal, arXiv:1410.0630.

Paccanaro, A. (2000). Learning distributed representations of concepts from relational data. *IEEE Transactions on Knowledge and Data Engineering 13*, 200–0.

Paccanaro, A. and G. Hinton (2001). Learning distributed representations of concepts using linear relational embedding. *IEEE Trans. on Knowl. and Data Eng. 13*, 232–244.

Pan, S. J., X. Ni, J.-T. Sun, Q. Yang, and Z. Chen (2010). Cross-domain sentiment classification via spectral feature alignment. In *Proc. of WWW '10*.

Pan, S. J. and Q. Yang (2010, October). A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering 22*(10), 1345–1359.

Pang, B. and L. Lee (2008). Opinion mining and sentiment analysis. *Foundations and Trends in Information Retrieval 2*(1-2), 1–135. Also published as a book. Now Publishers, 2008.

Pang, B., L. Lee, and S. Vaithyanathan (2002). Thumbs up ? Sentiment classification using machine learning techniques. In *Proc. of EMNLP' 02*.

Pascanu, R. and Y. Bengio (2013). Revisiting natural gradient for deep networks. Technical report, arXiv:1301.3584.

Poon, H. and P. Domingos (2009). Unsupervised semantic parsing. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, Singapore, pp. 1–10.

Poon, H. and P. Domingos (2010). Unsupervised ontology induction from text. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, Uppsala, Sweden, pp. 296–305.

Raina, R., A. Battle, H. Lee, B. Packer, and A. Y. Ng (2007). Self-taught learning: transfer learning from unlabeled data. In Z. Ghahramani (Ed.), *Proceedings of the Twenty-fourth International Conference on Machine Learning (ICML'07)*, pp. 759–766. ACM.

Ranzato, M., Y.-L. Boureau, and Y. LeCun (2008). Sparse feature learning for deep belief networks. In J. Platt, D. Koller, Y. Singer, and S. Roweis (Eds.), *Advances in Neural Information Processing Systems 20 (NIPS'07)*, Cambridge, MA, pp. 1185–1192. MIT Press.

Ranzato, M., C. Poultney, S. Chopra, and Y. LeCun (2007). Efficient learning of sparse representations with an energy-based model. In B. Schölkopf, J. Platt, and T. Hoffman (Eds.), *Advances in Neural Information Processing Systems 19 (NIPS'06)*, pp. 1137–1144. MIT Press.

Reed, S., K. Sohn, Y. Zhang, and H. Lee (2014). Learning to disentangle factors of variation with manifold interaction. In *Proceedings of The 31st International Conference on Machine Learning*.

Rifai, S., Y. Bengio, A. Courville, P. Vincent, and M. Mirza (2012). Disentangling factors of variation for facial expression recognition. In *Proceedings of the European Conference on Computer Vision (ECCV 6)*, pp. 808–822.

Rifai, S., P. Vincent, X. Muller, X. Glorot, and Y. Bengio (2011). Contractive auto-encoders: Explicit invariance during feature extraction. In *ICML'2011*.

Robbins, H. and S. Monro (1951). A stochastic approximation method. *Annals of Mathematical Statistics 22*, 400–407.

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review 65*, 386–408.

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

Rummel, R. J. (1999). Dimensionality of nations project: Attributes of nations and behavior of nation dyads. In *ICPSR data file*, pp. 1950–1965.

Sainath, T. N., B. Kingsbury, A.-r. Mohamed, G. E. Dahl, G. Saon, H. Soltau, T. Beran, A. Y. Aravkin, and B. Ramabhadran (2013). Improvements to deep convolutional neural networks for lvcsr. In *Automatic Speech Recognition and Understanding (ASRU), 2013 IEEE Workshop on*, pp. 315–320.

Salakhutdinov, R. and G. E. Hinton (2007). Semantic hashing. In *Proceedings of the 2007 Workshop on Information Retrieval and applications of Graphical Models (SIGIR'07)*, Amsterdam. Elsevier.

Salinas, E. and L. F. Abbott (1996). A model of multiplicative neural responses in parietal cortex. *Neurobiology 93*, 11956–11961.

Serre, T., G. Kreiman, M. Kouh, C. Cadieu, U. Knoblich, and T. Poggio (2007). A quantitative theory of immediate visual recognition. *Progress in Brain Research, Computational Neuroscience: Theoretical Insights into Brain Function 165*, 33–56.

Shi, L. and R. Mihalcea (2004). Open text semantic parsing using FrameNet and WordNet. In *HLT-NAACL 2004: Demonstration Papers*, Boston, Massachusetts, USA, pp. 19–22.

Shimodaira, H. (2000, October). Improving predictive inference under covariate shift by weighting the log-likelihood function. *Journal of Statistical Planning and Inference 90*(2), 227–244.

Sindhwani, V. and S. S. Keerthi (2006). Large scale semi-supervised linear SVMs. In *Proc. of SIGIR '06*.

Singla, P. and P. Domingos (2006). Entity resolution with markov logic. In *Proceedings of the Sixth International Conference on Data Mining*, pp. 572–582. IEEE Computer Society.

Smolensky, P. (1990). Tensor product variable binding and the representation of symbolic structures in connectionist systems. *Artificial intelligence 46*(1), 159–216.

Snyder, B. and R. Barzilay (2007). Multiple aspect ranking using the Good Grief algorithm. In *Proceedings of HLT-NAACL*, pp. 300–307.

Socher, R., D. Chen, C. D. Manning, and A. Ng (2013). Reasoning with neural tensor networks for knowledge base completion. In C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Weinberger (Eds.), *Advances in Neural Information Processing Systems 26*, pp. 926–934. Curran Associates, Inc.

Socher, R., B. Huval, C. D. Manning, and A. Y. Ng (2012). Semantic Compositionality Through Recursive Matrix-Vector Spaces. In *Proceedings of the 2012 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Socher, R., A. Karpathy, Q. V. Le, C. D. Manning, and A. Y. Ng (2014). Grounded compositional semantics for finding and describing images with sentences. *TACL 2*, 207–218.

Socher, R., J. Pennington, E. H. Huang, A. Y. Ng, and C. D. Manning (2011). Semi-supervised recursive autoencoders for predicting sentiment distributions. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Solla, S. A., E. Levin, and M. Fleisher (1988). Accelerated learning in layered neural networks. *Complex Systems 2*, 625–639.

Speer, R., C. Havasi, and H. Lieberman (2008). Analogyspace: reducing the dimensionality of common sense knowledge. In *Proceedings of the 23rd national conference on Artificial intelligence - Volume 1*, AAAI'08, pp. 548–553. AAAI Press.

Srivastava, N., G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research 15*, 1929–1958.

Sutskever, I., J. Martens, G. Dahl, and G. Hinton (2013). On the importance of initialization and momentum in deep learning. In *ICML*.

Sutskever, I., R. Salakhutdinov, and J. Tenenbaum (2009). Modelling relational data using bayesian clustered tensor factorization. In *Adv. in Neur. Inf. Proc. Syst. 22*.

Szegedy, C., W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich (2014, September). Going Deeper with Convolutions. *ArXiv e-prints*.

Taylor, G., R. Fergus, Y. LeCun, and C. Bregler (2010). Convolutional learning of spatio-temporal features. In *Proceedings of the European Conference on Computer Vision (ECCV'10)*, pp. 140–153.

Thomas, M., B. Pang, and L. Lee (2006). Get out the vote: Determining support or opposition from Congressional floor-debate transcripts. In *Proc. of EMNLP '06*.

Tucker, L. R. (1966). Some mathematical notes on three-mode factor analysis. *Psychometrika 31*, 279–311.

van der Maaten, L. and G. Hinton (2008, Nov). Visualizing high-dimensional data using t-sne. *Journal of Machine Learning Research 9*, 2579–2605.

Vincent, P. (2011). A connection between score matching and denoising autoencoders. *Neural Computation 23*(7).

Vincent, P., H. Larochelle, Y. Bengio, and P.-A. Manzagol (2008). Extracting and composing robust features with denoising autoencoders. In W. W. Cohen, A. McCallum, and S. T. Roweis (Eds.), *Proceedings of the Twenty-fifth International Conference on Machine Learning (ICML'08)*, pp. 1096–1103. ACM.

Vincent, P., H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol (2010). Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *J. Machine Learning Res. 11*.

Warde-Farley, D., I. J. Goodfellow, A. C. Courville, and Y. Bengio (2013). An empirical analysis of dropout in piecewise linear networks. *CoRR*.

Weston, J., S. Bengio, and N. Usunier (2010). Large scale image annotation: learning to rank with joint word-image embeddings. *Machine Learning 81*, 21–35.

Weston, J., F. Ratle, and R. Collobert (2008). Deep learning via semi-supervised embedding. In W. W. Cohen, A. McCallum, and S. T. Roweis (Eds.), *Proceedings of the Twenty-fifth International Conference on Machine Learning (ICML'08)*, New York, NY, USA, pp. 1168–1175. ACM.

Xie, X. and H. S. Seung (2003). Equivalence of backpropagation and contrastive Hebbian learning in a layered network. *Neural Computation*.

Zeiler, M. D. and R. Fergus (2014). Visualizing and understanding convolutional networks. In *ECCV'14*.

Zhang, X. and J. Wu (2013). Denoising deep neural networks based voice activity detection. In *IEEE International Conference on Acoustics, Speech and Signal Processing,ICASSP 2013, Vancouver, BC, Canada, May 26-31, 2013*, pp. 853–857.

Zhou, S., Q. Chen, and X. Wang (2010, August). Active deep networks for semi-supervised sentiment classification. In *Proceedings of COLING 2010*, Beijing, China, pp. 1515–1523.

Zhu, L., Y. Chen, and A. Yuille (2009). Unsupervised learning of probabilistic grammar-Markov models for object categories. *IEEE Transactions on Pattern Analysis and Machine Intelligence 31*(1), 114–128.