EXPLORATION DES RÉSEAUX DE NEURONES À BASE D'AUTOENCODEUR DANS LE CADRE DE LA MODÉLISATION DES DONNÉES TEXTUELLES

par

Stanislas Lauly

Thèse présentée au Département d'informatique
en vue de l'obtention du grade de philosophiæ doctor (Ph.D.)

FACULTÉ DES SCIENCES

UNIVERSITÉ DE SHERBROOKE

Sherbrooke, Québec, Canada, 6 août 2016

Le 6 août 2016

Le jury a accepté le mémoire de Monsieur Stanislas Lauly dans sa version finale.

Membres du jury

Professeur Hugo Larochelle
Directeur de recherche
Département d'informatique, Université de Sherbrooke

Professeur Mario Marchand
Évaluateur externe
Département d'informatique, Université Laval

Professeur Shengrui Wang
Évaluateur interne
Département d'informatique, Université de Sherbrooke

Professeur Pierre-Marc Jodoin
Prśident rapporteur
Département d'informatique, Université de Sherbrooke

Sommaire

Depuis le milieu des années 2000, une nouvelle approche en apprentissage automatique, l'apprentissage de réseaux profonds (*deep learning* [3]), gagne en popularité. En effet, cette approche a démontré son efficacité pour résoudre divers problèmes en améliorant les résultats obtenus par d'autres techniques qui étaient considérées alors comme étant l'état de l'art. C'est le cas pour le domaine de la reconnaissance d'objets [30] ainsi que pour la reconnaissance de la parole [23]. Sachant cela, l'utilisation des réseaux profonds dans le domaine du Traitement Automatique du Langage Naturel (TALN, *Natural Language Processing*) est donc une étape logique à suivre.

Cette thèse explore différentes structures de réseaux de neurones dans le but de modéliser le texte écrit, se concentrant sur des modèles simples, puissants et rapides à entraîner.

Mots-clés: deep learning; réseaux profonds; réseau de neurones; traitement automatique du langage naturel; TALN; natural language processing; NLP.

Remerciements

Je tiens à remercier tout particulièrement Hugo Larochelle pour m'avoir si bien guidé tout au long de ce doctorat et permis d'acquérir la discipline nécessaire pour mener à bien ce projet de longue haleine.

Je veux aussi remercier mes parents, Jean-Paul et Christiane, ainsi que toute ma famille pour m'avoir soutenu pendant tout ce temps.

Finalement, je voudrais remercier les personnes avec qui j'ai travaillé et qui ont rendu cette aventure si passionnante.

Table des matières

Table des matières

Liste des figures

Liste des tableaux

Introduction

On considère généralement que le traitement automatique du langage naturel (TALN), mieux connu sous l'acronyme NLP (*Natural Language Processing*), a fait ses débuts dans les années 50. Alan Turing propose à cette époque le test de Turing, un critère d'évaluation pour l'intelligence artificielle. Pour ce test, un humain doit dialoguer par écrit en temps réel avec un programme informatique. Si la personne n'est pas en mesure de savoir si elle interagit avec une machine ou avec un autre humain, le test est réussi. Au milieu des années 50, les premiers modèles en traduction automatique font leur apparition. Certains scientifiques pensent à l'époque que le problème de la traduction serait complètement résolu en moins de cinq ans. Les systèmes experts, basés sur l'accumulation de règles écrites à la main, ont commencé à être utilisés avant les années 80 et ont obtenu des résultats intéressants pour différentes tâches, comme les algorithmes de diagnostics médicaux. Par la suite, l'apprentissage automatique révolutionne le monde du traitement de la langue vers la fin des années 80. Les approches statistiques ont permis de créer des systèmes plus performants sans avoir besoin d'utiliser toute une hiérarchie complexe de règles. L'augmentation de la capacité de calcul des ordinateurs est en partie responsable de l'éclosion des modèles statistiques.

Depuis le milieu des années 2000, une nouvelle approche en apprentissage automatique, l'apprentissage de réseaux profonds (*deep learning* [3]), gagne en popularité. En effet, cette approche a démontré son efficacité pour résoudre divers problèmes en améliorant les résultats obtenus par d'autres techniques qui étaient considérées alors comme étant l'état de l'art. Depuis quelques années, l'apprentissage de réseaux profonds commence à faire son entrée en TALN et a déjà fait ses preuves en reconnaissance de la parole.

Introduction

Tout au long de ce doctorat, nous avons développé de nouveaux modèles d'apprentissage dans le domaine des réseaux profonds. Ces modèles ont été adaptés à différentes tâches du TALN et nous ont permis d'obtenir des prédictions performantes. Dans ce document, nous allons commencer par présenter les concepts de base de l'apprentissage automatique ainsi que ceux des réseaux de neurones. Nous allons ensuite faire une mise en contexte pour présenter les différentes tâches qui nous intéressent en TALN. Ces tâches sont suivies d'une discussion sur les difficultés auxquelles nous devons faire face pour les résoudre. La thèse se poursuit en présentant les trois articles composés durant ce doctorat.

Le premier article présente le modèle DocNADE, un réseau de neurones non-supervisé qui sert de modèle de sujets pour représenter les documents. DocNADE est un modèle performant pour créer des représentations vectorielles de documents ainsi que pour servir de modèle génératif.

Le deuxième article porte sur le sujet des représentations de mots bilingues appris par des réseaux de neurones. Nous démontrons que notre modèle de représentation bilingue, simple et rapide à entraîner, peut obtenir des performances aussi bonnes que l'état de l'art.

Finalement, le troisième article présente une famille de modèles inspirés de l'approche NADE [33]. Certains de ces modèles surpassent l'état de l'art en tant que modèles génératifs de documents et sont très performants pour les représenter. D'autres enfin augmentent la qualité de la modélisation de la langue.

Chapitre 1

Apprentissage automatique, un domaine de l'intelligence artificielle

L'intelligence artificielle (I.A.) est une discipline scientifique qui étudie la façon de créer des programmes dits intelligents. Par programmes intelligents, on veut généralement parler de programmes capables de résoudre des problèmes traditionnellement considérés comme étant propres aux capacités humaines.

L'approche classique en intelligence artificielle, soit les systèmes experts, est une des premières branches de l'I.A. à avoir un véritable succès dans l'industrie. Cette approche consiste à accumuler, pour une tâche dans un domaine spécifique, toutes les connaissances humaines possibles sous forme de règles pour ensuite être utilisées dans le but de faire certaines prédictions. Le premier problème avec cette approche est d'optimiser le choix des règles et leurs potentiels paramètres. Cette tâche est souvent faite à la main par des experts et peut nécessiter un travail colossal, très coûteux en temps et en effort et qui, de surcroît, n'est pas toujours couronné de succès. Le deuxième problème se trouve au niveau des règles. En effet, certains problèmes complexes ne peuvent être entièrement expliqués par de simples règles et nécessitent souvent l'ajout de nombreuses exceptions. De plus, les règles sont déterminés par des humains et peuvent être subjectifs. Ceci a pour effet de rendre les modèles lourds, difficiles à utiliser et donne souvent des résultats peu satisfaisants pour des problèmes complexes. Par contre, un point fort de cette technique est de permettre à un être

humain de comprendre la logique utilisée par le modèle.

L'approche appelée "apprentissage automatique" (*machine learning*) est une autre branche de l'intelligence artificielle. Elle n'a pas pour but de trouver des règles interprétables, mais plutôt de trouver la logique d'un problème sous forme de fonction. Elle est donc plus appropriée pour créer un bon modèle génératif qui doit apprendre des phénomènes complexes. On laisse alors le modèle apprendre tout seul la fonction qui représente le mieux les données fournies en exemple, ce qui épargne beaucoup de travail à la main, travail qui serait nécessaire dans une approche comme celle des systèmes experts.

1.1 Concept de l'apprentissage automatique

L'apprentissage automatique (*machine learning*) cherche à permettre à l'ordinateur d'imiter la capacité humaine d'apprendre à partir d'exemples, lui donnant la possibilité d'agir sans être explicitement programmé. En général, ce domaine se concentre sur les algorithmes qui apprennent à partir d'exemples pour ensuite permettre de généraliser sur de nouveaux exemples non observés auparavant. L'apprentissage automatique est maintenant une composante importante de plusieurs domaines tels que le traitement automatique du langage naturel, la reconnaissance d'objets, la reconnaissance de la parole, la bioinformatique et bien d'autres encore. En 1997, le professeur Tom Mitchell de l'université Carnegie Mellon, définit formellement l'apprentissage automatique comme suit:

> *On dit qu'un programme informatique apprend de l'expérience E par rapport à une tâche T et une certaine mesure de performance P, si sa performance sur T, telle que mesurée par P, s'améliore avec l'expérience E.*

Par exemple, dans le cadre de la classification de documents, un ensemble de données composé de documents et des sujets qui les décrivent correspond à l'expérience E et la tâche T équivaut à assigner un sujet à un document. Le nombre de sujets correctement assignés sert de mesure de performance P. On pourra dire qu'un algorithme en apprentissage automatique a appris à partir de données si le nombre de sujets correctement assignés augmente après avoir observé l'ensemble de données.

L'apprentissage automatique se divise en plusieurs types se distinguant par la nature des tâches devant être apprises. Nous allons maintenant voir les principaux types d'apprentissages.

1.1.1 L'apprentissage supervisé

Généralement le but pour un modèle en apprentissage automatique est de générer une fonction de prédiction $f(\mathbf{x})$ à partir d'un ensemble de données D fourni, qu'on appelle ensemble d'entraînement. Dans le cadre de l'apprentissage supervisé, D est composé de paires d'exemples $(\mathbf{x}, \mathbf{y})$, où $\mathbf{x}$ est un vecteur servant d'entrée au modèle et $\mathbf{y}$ un vecteur cible qui représente ce que l'on veut prédire. Notons que les entrées et les cibles sont présentées sous forme de vecteurs, mais pourraient être remplacées par des scalaires. Le fait d'avoir une cible pour chaque exemple, et de s'en servir, est la caractéristique principale de l'apprentissage supervisé. Supposons que la fonction $f(\mathbf{x})$, prenant en entrée le vecteur $\mathbf{x}$ de taille J, ait la forme suivante:

$$f(\mathbf{x}) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ ... \ + \theta_J x_J \tag{1.1}$$

Où θ_0 à θ_J sont les paramètres du modèle d'apprentissage. Le but de ce modèle est donc de trouver les valeurs des paramètres afin d'obtenir les meilleures prédictions possible, c'est-à-dire le plus proche possible des cibles.

Dans beaucoup de cas, les données étiquetées (données avec cibles) sont générées par des personnes qui assignent manuellement une cible à chaque exemple, comme le fait d'associer un sujet à un document. Cependant, le fait de créer des données étiquetées ne nécessite pas toujours une intervention humaine, comme c'est le cas pour les prédictions météorologiques. Quelle que soit la façon dont les données ont été obtenues, on les passe ensuite à un algorithme d'apprentissage qui cherche à modéliser la relation entre les entrées et leurs cibles.

Il existe une autre catégorisation importante en apprentissage automatique, liée à la nature même des cibles, qui est le type de problèmes à résoudre. Nous allons présenter les deux types les plus utilisés dans la littérature. Il s'agit des cibles composées de valeurs discrètes ou continues.

Problème de classification

Habituellement, pour un problème de classification, l'ensemble de données utilisé possède un nombre fini de classes et chaque exemple est associé à l'une d'elles. La cible de chaque exemple aura une valeur discrète représentant une classe en particulier. Avec de telles données, un modèle en apprentissage automatique apprendra à assigner les classes aux entrées.

Par exemple, dans le cadre de la classification de documents, on peut imaginer un ensemble de données composé de trois classes. Chaque classe correspond à un sujet: "économie", "politique" et "autre". Le modèle doit apprendre à déterminer si un document traite d'économie, de politique ou encore d'un autre sujet et devra par la suite associer au document la valeur représentant la bonne classe.

Problème de régression

Dans le cadre d'un problème de régression, la cible est composée d'un ou de plusieurs éléments de valeurs continues. Un modèle en apprentissage automatique apprendra à prédire une ou des valeurs réelles.

En météorologie, la prédiction de la température est un bon exemple de problème de régression. En effet, la valeur à prédire ici est continue. On peut aussi ajouter d'autres éléments à la cible comme la pression atmosphérique et le taux d'humidité, créant ainsi un vecteur de valeurs continues.

1.1.2 L'apprentissage non-supervisé

Contrairement à l'approche supervisée, l'ensemble de données D utilisé en apprentissage non-supervisé n'est pas composé de paires d'exemples $(\mathbf{x}, \mathbf{y})$, mais seulement de $\mathbf{x}$, c'est-à-dire qu'il n'y a plus de cible associée à chaque exemple. Dans un tel cadre, un modèle en apprentissage automatique modélisera l'information fournie en entrée seulement.

Dans l'approche non-supervisée, la fonction $f(\mathbf{x})$ retournée par un algorithme d'apprentissage n'est pas dictée par la nature des données. En effet, contrairement à l'approche supervisée, il n'y a pas de but explicitement exprimé par le biais de cibles.

Le problème devant être résolu par la fonction est donc défini par l'usager. Cependant, peu importe le problème choisi, un modèle non supervisé apprendra toujours des caractéristiques en lien avec la distribution de probabilité générant les données d'entraînement.

Les problèmes en apprentissage non-supervisé sont nombreux. En voici une liste des plus courants:

- **Extraction de caractéristiques:** La fonction $f(\mathbf{x})$ apprise fournit une nouvelle représentation pour l'entrée $\mathbf{x}$. En général, cette représentation sert à accomplir une autre tâche pour laquelle elle est plus utile que l'entrée d'origine.
- **Estimation de densité:** Ici $f(\mathbf{x})$ doit estimer la distribution de probabilité des exemples de l'ensemble d'entraînement.
- **Regroupement (*clustering*):** Dans l'espace où se trouve les données de l'ensemble d'entraînement, l'algorithme d'apprentissage doit identifier des regroupements d'exemples distincts. On obtient une fonction $f(\mathbf{x})$ qui fournit l'indice du regroupement associé à l'exemple $\mathbf{x}$.
- **Réduction de dimensionnalité:** Comme pour l'extraction de caractéristiques, $f(\mathbf{x})$ doit fournir une nouvelle représentation pour l'entrée $\mathbf{x}$. Cependant, le but cette fois est d'obtenir une représentation de dimensionnalité plus petite que celle de l'entrée, tout en conservant l'information importante.

1.1.3 L'apprentissage semi-supervisé

L'apprentissage semi-supervisé est en fait un mélange des deux approches que l'on vient de présenter, soit l'apprentissage supervisé et non-supervisé. Pourquoi vouloir se servir des deux types d'apprentissages ensemble ? La réponse se situe au niveau des données. En effet, il est important de réaliser qu'il n'est pas toujours facile d'obtenir des données étiquetées, c'est-à-dire un ensemble d'entraînement où chaque exemple est relié à une cible. Souvent, la taille de l'ensemble d'entraînement n'est pas assez grande pour bien représenter la distribution des données et ainsi permettre de généraliser adéquatement sur de nouveaux exemples. Le manque de données étiquetées pour certaines tâches n'est pas rare. Cette pénurie s'explique par le fait qu'il faut parfois

avoir recours à des personnes spécialisées devant associer les cibles à la main, ce qui peut s'avérer trop long ou trop coûteux.

Pour pallier au manque de données étiquetées, on peut se servir également d'ensembles non-étiquetés qui sont habituellement plus faciles à générer et donc beaucoup plus nombreux. Habituellement, un algorithme en apprentissage semi-supervisé commencera à se servir des données non-étiquetées pour faire de l'estimation de densité. Cette première étape est utilisée dans le but d'initialiser les paramètres du modèle et ainsi capturer une certaine information sur la distribution des données. On continue ensuite avec ce même modèle en faisant de l'apprentissage supervisé sur l'ensemble d'entraînement étiqueté, comme il a déjà été expliqué dans la section 1.1.1.

Au final, un algorithme d'apprentissage fournira une fonction $f(\mathbf{x})$ qui prédit les cibles des données étiquetées. Cette approche s'avère particulièrement efficace lorsqu'on a un petit nombre d'exemples étiquetés et un grand nombre d'exemples non-étiquetés.

1.1.4 L'apprentissage par renforcement

Le domaine de l'apprentissage par renforcement cherche à apprendre à un agent à se comporter de la bonne façon à l'intérieur d'un environnement spécifique, c'est-à-dire de façon à atteindre un but choisi préalablement par l'utilisateur. Le problème que l'on désire résoudre est divisé en une séquence d'étapes. À chaque étape, un agent doit choisir parmi un ensemble d'actions, lui donnant la possibilité d'interagir avec son environnement. Contrairement à l'apprentissage supervisé, il n'y a pas de cible qui donne la possibilité d'apprendre un comportement. À la place, l'agent reçoit un signal (déterminé par l'utilisateur) qui lui permet de savoir s'il a agi correctement. Pour chaque étape de la séquence, l'agent reçoit de l'information sur son environnement qui l'aidera à choisir l'action appropriée. Durant l'apprentissage, l'agent cherchera à maximiser le nombre de signaux positifs afin d'améliorer son comportement.

1.2 Descente de gradient, une approche pour apprendre

La mesure de performance utilisée par un modèle en apprentissage automatique s'appelle une "fonction objectif". Plus sa valeur diminue, plus le modèle performe bien.

La descente de gradient est une technique utilisée pour minimiser une fonction objectif $J(\boldsymbol{\theta})$ en modifiant la valeur des paramètres $\boldsymbol{\theta}$ d'un modèle, à laquelle on passe un ensemble d'entraînement D, lui permettant ainsi d'apprendre à partir d'exemples. Le gradient $\frac{\partial J(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}$ de la fonction objectif par rapport aux paramètres est utilisé pour appliquer une modification des paramètres $\boldsymbol{\theta}$. La figure 1.1 est une représentation simplifiée en une dimension de $J(\boldsymbol{\theta})$ qui varie en fonction de la valeur du paramètre $\boldsymbol{\theta}$. La flèche orange dans la même figure représente la direction du gradient pour un point donné (valeur de $\boldsymbol{\theta}$ spécifique). Puisque le gradient est un vecteur indiquant la direction de la pente la plus abrupte, il suffit de le soustraire aux paramètres $\boldsymbol{\theta}$ pour se déplacer vers le minimum local le plus proche. Le taux d'apprentissage η, que l'on multiplie avec le gradient, détermine la taille du pas fait en direction inverse du gradient. On répète le procédé pour se rapprocher à chaque fois du minimum.

Il existe trois variantes de descente de gradient qui diffèrent par la quantité de données utilisées pour estimer le gradient de la fonction objectif. La première est la descente de gradient stochastique et utilise un seul exemple à la fois. La deuxième variante est la descente de gradient par "Mini-Batch", qui utilise un petit groupe d'exemples pour calculer le gradient. Finalement, la descente de gradient par "Batch" est la variante qui se sert de l'ensemble D complet pour faire son calcul.

Nous allons maintenant nous concentrer sur une des branches les plus actives en apprentissage automatique, celle des réseaux de neurones.

1.3 Introduction aux réseaux de neurones et réseaux profonds

Un réseau de neurones est un modèle statistique qui simule, en version simplifiée, l'activité d'un système neuronal biologique. Comme pour tous les modèles en appren-

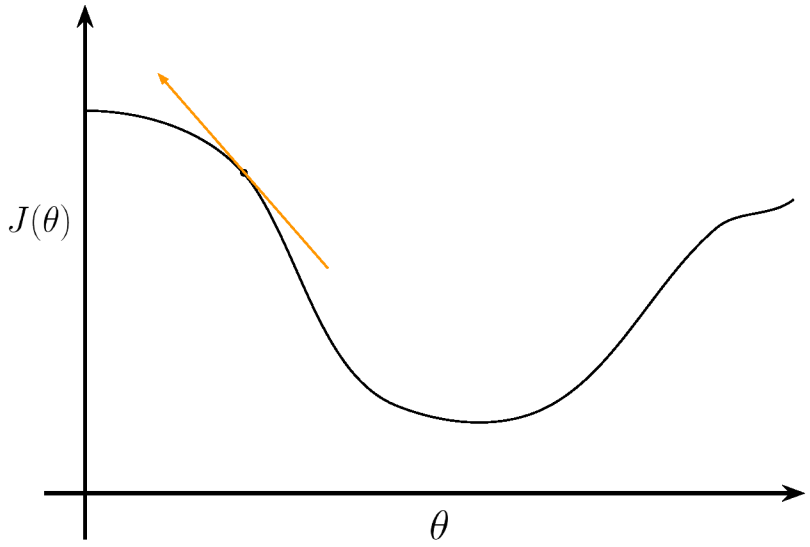


figure 1.1 – Direction du gradient $\frac{\partial J(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}$. L'axe des x représente le paramètre. L'axe des y représente la fonction objectif.

tissage automatique, un réseau de neurones est construit à partir d'un ensemble de données pour ensuite être utilisé dans le but de faire des prédictions sur de nouvelles données (qui n'ont pas été observées lors de la construction du modèle). On fait de la classification dans le cas où les valeurs à prédire sont discrètes et de la régression, dans le cas où elles sont continues.

La nomenclature suivante est utilisée pour présenter les réseaux de neurones. L'ensemble d'entraînement D, utilisé par les modèles, est composé de N paires $(\mathbf{x}, \mathbf{y})$, où $\mathbf{x}$ est un vecteur (qui sert d'entrée pour les modèles) représentant un exemple et $\mathbf{y}$ un vecteur cible qui représente ce que l'on veut prédire. Un vecteur d'entrée $\mathbf{x}$ possède J composantes x_j, où $1 \leq j \leq J$. Une cible $\mathbf{y}$ contient K composantes y_i, où $1 \leq i \leq K$. Ces vecteurs peuvent être composés de valeurs discrètes ou continues.

On peut voir un réseau de neurones comme étant un graphe orienté de noeuds, où chaque noeud représente un neurone. Ces neurones sont souvent disposés en couches. Nous allons commencer par présenter le concept de réseau de neurones sous une de

ses formes les plus simples: le réseau de neurones linéaire.

1.3.1 Réseau de neurones linéaire

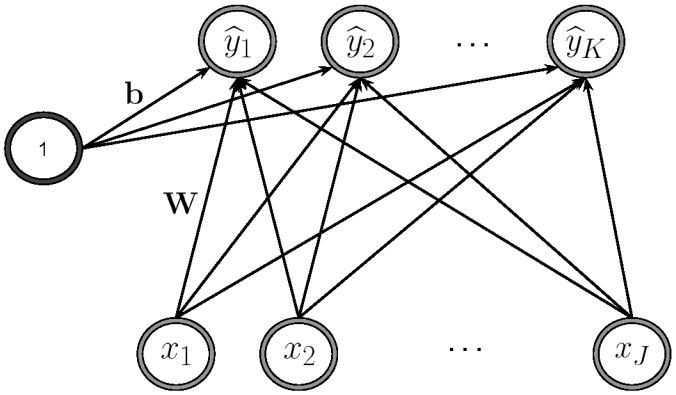


figure 1.2 – Réseau de neurones linéaire.

Un réseau de neurones linéaire est composé de variables en entrée, formant une couche d'entrée, et de variables en sortie, formant une couche de sortie. L'entrée peut être vue comme de l'information provenant directement de l'environnement que l'on cherche à modéliser, ou encore comme venant de la sortie d'un autre réseau de neurones. Le vecteur d'entrée $\mathbf{x}$ du modèle est composé des variables aléatoires x_j pouvant prendre des valeurs discrètes ou continues. Les variables de sortie $\widehat{y}_i$ forment le vecteur de valeurs continues $\widehat{\mathbf{y}}$ et sont les prédictions des cibles y_i correspondantes. Un poids $w_{ij} \in \mathbb{R}$, élément de la matrice de connexion $\mathbf{W}$, est associé à chaque paire $(x_j, \widehat{y}_i)$. La prédiction $\widehat{y}_i$ représente la somme des éléments de l'entrée multipliée par leur poids assigné (voir figure 1.2).

$$\widehat{y}_i = \sum_{j=1}^{J} W_{ij} x_j + b_i \tag{1.2}$$

$$\widehat{\mathbf{y}} = \mathbf{W}\mathbf{x} + \mathbf{b} \tag{1.3}$$

où b_i, associé à la sortie $\widehat{y}_i$, est un élément du vecteur $\mathbf{b}$ et représente le biais qui rend le modèle plus général.

Dans le cas de la régression, l'équation 1.3 est suffisante. Cependant, il faudra appliquer une fonction d'activation, qui calcule une probabilité, à la couche de sortie du modèle si l'on cherche à classifier un nombre discret de classes (appelé aussi étiquettes), où pour chaque exemple une seule classe lui est associée à la fois. Dans un tel cas, chacun des élément y_i de la cible représente une des étiquettes et sa valeur est égale à "1" si celle-ci correspond à la bonne réponse, sinon elle aura la valeur "0". On utilise la fonction d'activation pour calculer la probabilité des étiquettes. Dans le cas d'une classification binaire, nous n'avons besoin que d'une seule variable aléatoire et $\widehat{\mathbf{y}}$ devient le scalaire $\widehat{y}$. La matrice de connections $\mathbf{W}$ change aussi pour devenir un vecteur et $\mathbf{b}$ devient un scalaire. On calcule alors la probabilité conditionnelle $\widehat{y} = P(y = 1|\mathbf{x})$ en utilisant la fonction sigmoïde:

$$a = \mathbf{W}\mathbf{x} + b \tag{1.4}$$

$$\widehat{y} = \text{sigmoid}(a) = \frac{1}{1 + e^{-a}} \ . \tag{1.5}$$

Dans le cas où le nombre d'étiquettes est plus grand que deux, on assigne simplement une étiquette par neurone de sortie (aligné avec la cible $\mathbf{y}$) et on calcule la probabilité conditionnelle $\widehat{y}_i = P(y_i = 1|\mathbf{x})$ en utilisant la fonction softmax:

$$\widehat{y}_i = \text{softmax}(a_i) = \frac{\exp(a_i)}{\sum_{k=1}^{K} \exp(a_k)} \ . \tag{1.6}$$

Chaque valeur des neurones de sortie $\widehat{y}_i$ indique à quel point les données en entrée peuvent faire partie de la classe que y_i représente. Pour classifier les données en entrée, on choisit la classe correspondant au $\widehat{y}_i$ qui a la valeur la plus élevée:

$$classe = \underset{i}{\text{argmax}}(\widehat{y}_i), \ 1 \leq i \leq K \ . \tag{1.7}$$

Noter qu'un réseau de neurones linéaire avec la sigmoïde ou la softmax en sortie se nomme aussi modèle de régression logistique. Le fait de propager une entrée à

travers le modèle pour en obtenir les valeurs de sortie s'appelle la propagation avant (*forward propagation*), mais pour obtenir de bonnes prédictions, il faut trouver les valeurs appropriées des paramètres $\mathbf{W}$ et $\mathbf{b}$.

Apprendre la valeur des paramètres

Le but du modèle est d'estimer une fonction qui prend en entrée l'exemple $\mathbf{x}$ et prédit la cible $\mathbf{y}$ qui lui correspond. On veut donc que la prédiction $\widehat{\mathbf{y}}$ soit le plus près possible de la cible $\mathbf{y}$. Dans le but d'évaluer les performances du modèle, on utilise pour chaque $\widehat{\mathbf{y}}$ calculé une fonction objectif (aussi appelée fonction de coût ou d'erreur) dont la valeur diminue quand la qualité de la prédiction augmente. Différentes fonctions de coût peuvent être utilisées, la plus fréquente étant la log-vraisemblance négative (NLL, *Negative Log Likelihood*) de la classe cible,

$$NLL = -\sum_{i=1}^{K} 1_{(y_i=1)} \cdot \log\, \widehat{y}_i = -\log\, P(y_i = 1|\mathbf{x}) \ , \tag{1.8}$$

où $y_i = 1$ pour la classe cible de $\mathbf{x}$. Avec la NLL, on cherche à maximiser la probabilité de la bonne classe. Pour faire de la régression, on utilise une fonction de coût adaptée à cette tâche comme la différence au carré (SE, *squared error*) qui est une mesure de distance entre le vecteur de prédiction et le vecteur cible:

$$SE = \sum_{k=1}^{K} (y_k - \widehat{y}_k)^2 = \|\mathbf{y} - \widehat{\mathbf{y}}\|^2 \ . \tag{1.9}$$

Une fois qu'on a une fonction de coût, on emploie la technique de descente de gradient stochastique pour entraîner notre réseau de neurones. Entraîner un modèle consiste à modifier la valeur des connections (poids et biais) dans le but d'obtenir des prédictions qui se rapprochent des cibles. Pour savoir quelle est la valeur de la modification d'un poids, il faut calculer le gradient de la fonction de coût du modèle par rapport à ce poids. Le gradient représente la pente de la fonction de coût pour le poids. Il faut donc soustraire la valeur du gradient au poids pour diminuer l'erreur. Le gradient du biais $\mathbf{b}$ et de la matrice de connexions $\mathbf{W}$ qui fait le lien entre la couche d'entrée et la couche de sortie se calcule en faisant la dérivée de la fonction de coût

par rapport à chaque poids. Ce qui, dans le contexte de classification, donne:

$$\frac{\partial Coût}{\partial \mathbf{W}} = \frac{\partial NLL}{\partial \mathbf{W}} = \hat{\mathbf{y}}\mathbf{x}^T - \mathbf{y}\mathbf{x}^T \tag{1.10}$$

$$\frac{\partial Coût}{\partial \mathbf{b}} = \frac{\partial NLL}{\partial \mathbf{b}} = \hat{\mathbf{y}} - \mathbf{y} \tag{1.11}$$

où $\hat{\mathbf{y}}$ est le vecteur de sortie du réseau de neurones et $\mathbf{y}$ la cible de forme $\mathbf{y} = [0\ 1\ 0\ ...\ 0]^T$. Pour ce qui est de la régression, le gradient de la matrice de connections $\mathbf{W}$ est différent, car la fonction de coût change:

$$\frac{\partial Coût}{\partial \mathbf{W}} = \frac{\partial SE}{\partial \mathbf{W}} = -2(\mathbf{y} - \hat{\mathbf{y}})\mathbf{x}^T \tag{1.12}$$

$$\frac{\partial Coût}{\partial \mathbf{b}} = \frac{\partial SE}{\partial \mathbf{b}} = -2(\mathbf{y} - \hat{\mathbf{y}})\ . \tag{1.13}$$

La mise à jour des poids se calcule en multipliant le gradient qui leur correspond par le taux d'apprentissage η, ce qui modifie le pas du gradient. On soustrait ensuite ce résultat aux paramètres courants:

$$\mathbf{W} \leftarrow \mathbf{W} - \eta(\frac{\partial Coût}{\partial \mathbf{W}}) \tag{1.14}$$

$$\mathbf{b} \leftarrow \mathbf{b} - \eta(\frac{\partial Coût}{\partial \mathbf{b}})\ . \tag{1.15}$$

Le réseau que l'on vient de voir permet de répondre aux besoins de certaines applications, mais il n'est pas toujours assez puissant. Le fait de n'avoir qu'une seule couche de poids a comme inconvénient de n'estimer que des fonctions linéaires. Lorsqu'on veut estimer une fonction non linéaire, on le fait à l'aide d'une couche cachée. Il s'agit d'une couche de neurones supplémentaire que l'on place entre la couche d'entrée et celle de sortie. Le modèle que l'on vient de voir est une sorte de réseau de neurones qui n'a pas de couche cachée. Nous allons donc passer à l'étape suivante, qui est celle des réseaux de neurones avec couche cachée.

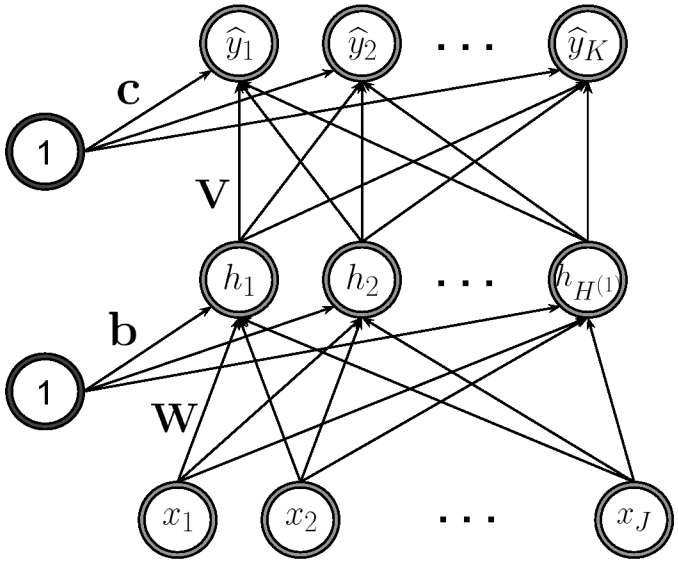


figure 1.3 – Réseau de neurones à une couche cachée.

1.3.2 Réseau de neurones avec couche cachée

Pour simplifier l'explication du modèle, nous allons continuer la présentation des réseaux de neurones dans le cadre de la classification seulement. Notons qu'il suffit d'utiliser l'équation 1.3 pour le calcul de la sortie ainsi que la fonction de coût de l'équation 1.9 pour transformer le modèle et faire de la régression.

Dans cette section, nous allons définir formellement un réseau de neurones avec une seule couche cachée (figure 1.3) et expliquer au fur et à mesure la façon d'étendre le modèle pour en avoir plusieurs. On part du réseau que l'on vient de présenter (section 1.3.1) et on y ajoute une couche cachée, comportant $H^{(1)}$ neurones, entre la couche d'entrée (J neurones) et la couche de sortie (K neurones). (1) représente l'index de la première couche cachée. C'est cette couche qui permet au modèle d'estimer une fonction non linéaire. Dans le but d'expliquer la structure de ce réseau de neurones, nous allons suivre le flux de calculs en partant de l'entrée $\mathbf{x}$ pour nous diriger vers l'estimation de la cible $\mathbf{y}$.

On commence par propager l'entrée $\mathbf{x}$ à travers la matrice de poids $\mathbf{W}^{(1)}$, qui est de dimension $H^{(1)} \times J$ et qui fait le lien entre la couche d'entrée et la couche cachée, et on y ajoute un vecteur de biais $\mathbf{b}^{(1)}$ de dimension $H^{(1)}$. Le résultat que l'on obtient s'appelle le vecteur de pré-activation $\mathbf{a}^{(1)}$ et est aussi de dimension $H^{(1)}$.

$$\mathbf{a}^{(1)} = \mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)} \tag{1.16}$$

On doit ensuite appliquer une fonction d'activation sur le vecteur $\mathbf{a}^{(1)}$ afin d'obtenir la couche cachée $\mathbf{h}^{(1)}$. Le but de cette fonction est d'incorporer une non-linéarité à la prédiction calculée par le modèle. Différentes fonctions peuvent être utilisées pour l'activation. Pour cet exemple, nous nous contenterons d'appliquer la sigmoïde sur chaque élément a_j du vecteur $\mathbf{a}^{(1)}$.

$$h_i^{(1)} = \text{sigmoid}(a_i) = \frac{1}{1 + e^{-a_i}} \tag{1.17}$$

$$\mathbf{h}^{(1)} = \text{sigmoid}(\mathbf{a}^{(1)}) \tag{1.18}$$

Il faut maintenant propager ce nouveau vecteur $\mathbf{h}^{(1)}$ à travers une autre matrice de poids $\mathbf{V}$ (de dimension $K \times H^{(1)}$) qui fait la connexion entre la couche cachée et la couche de sortie, ainsi qu'ajouter le vecteur de biais $\mathbf{c}$ de dimension K. Ce calcul nous donne le vecteur de pré-activation de la couche de sortie $\mathbf{a}^{(s)}$, auquel on applique la fonction d'activation softmax pour calculer la probabilité des classes:

$$\mathbf{a}^{(s)} = \mathbf{V}\mathbf{h}^{(1)} + \mathbf{c} \tag{1.19}$$

$$\hat{\mathbf{y}} = \text{softmax}(\mathbf{a}^{(s)}) \tag{1.20}$$

On peut voir le calcul de la sortie du réseau de neurones comme étant un modèle de régression logistique (section 1.3.1) qui prend en entrée la couche cachée.

Maintenant qu'un réseau de neurones avec une seule couche a été défini, on peut facilement étendre le concept à un réseau composé de M couches cachées (figure 1.4). En effet, on peut simplement ajouter le nombre de couches cachées désiré entre $\mathbf{h}^{(1)}$ et la sortie. Pour ce faire, chaque couche cachée prend en entrée le vecteur final

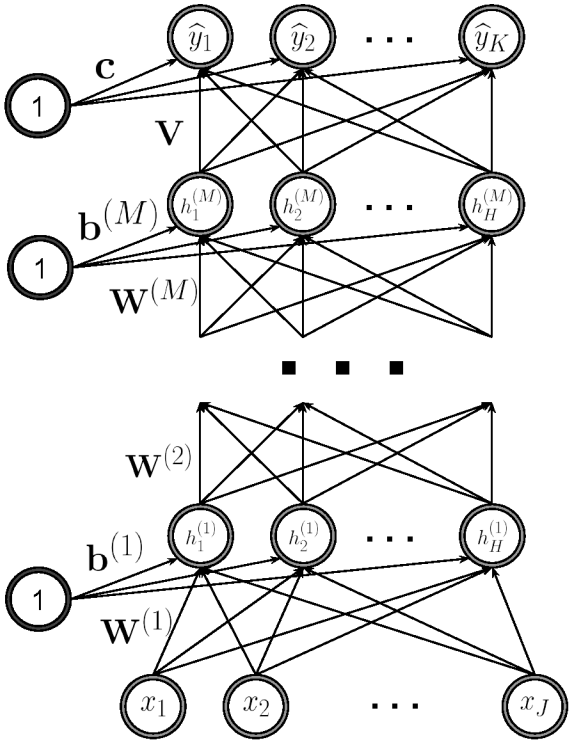


figure 1.4 – Réseau de neurones à plusieurs couche cachée.

de la couche précédente $\mathbf{h}^{(i-1)}$. Il faut propager ce vecteur à travers la matrice de connections $\mathbf{W}^{(i)}$, ajouter le vecteur de biais $\mathbf{b}^{(i)}$ et appliquer la sigmoïde comme fonction d'activation. Le dernier vecteur $\mathbf{h}^{(M)}$ de cette suite de couches cachées servira d'entrée à la couche de sortie finale présentée précédemment.

$$\widehat{\mathbf{y}} = \text{softmax}(\mathbf{V}\ \text{sig}(\mathbf{W}^{(M)}\ \text{sig}(...\ \text{sig}(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)})...) + \mathbf{b}^{(M)}) + \mathbf{c}) \tag{1.21}$$

Où sig() représente la fonction sigmoïde. La suite de calculs qui vient d'être présentée, c'est-à-dire propager l'entrée à travers le réseau de neurones et obtenir la prédiction en sortie, s'appelle la propagation avant (*forward propagation*). Le chemin inverse doit maintenant être parcouru, avec la technique de rétropropagation, pour

faire l'entraînement du modèle.

1.3.3 Rétropropagation (*Backpropagation*)

La technique de descente de gradient stochastique est utilisée pour entraîner notre réseau de neurones. Comme il a déjà été expliqué dans la section 1.3.1, il faut calculer le gradient du coût en fonction de chacun des poids du modèle. Ce calcul est utilisé pour modifier la valeur des poids de façon à obtenir des prédictions qui se rapprochent des cibles. Calculer indépendamment la dérivée pour chaque paramètre peut s'avérer long et fastidieux, c'est pourquoi on utilise la technique de rétropropagation. En effet, cette technique nous permet de calculer de façon efficace les gradients de tous les poids d'un réseau de neurones. La rétropropagation porte bien son nom, car le flux de calculs fait le chemin inverse de la propagation avant. Elle commence donc par la sortie et se dirige vers l'entrée.

L'idée est de calculer la dérivée de la fonction objectif par rapport à la couche de sortie pour ensuite propager cette information à travers le réseau jusqu'à l'entrée du modèle. Le gradient de chaque couche cachée est exprimé en réutilisant les dérivées des couches qui les suivent.

On commence par évaluer la dérivée de la fonction de coût relativement au vecteur de pré-activation $\mathbf{a}^{(s)}$ de la couche de sortie:

$$\frac{\partial NLL}{\partial a_i^{(s)}} = \widehat{y}_i - y_i \tag{1.22}$$

$$\frac{\partial NLL}{\partial \mathbf{a}^{(s)}} = \widehat{\mathbf{y}} - \mathbf{y} \tag{1.23}$$

La formule de dérivée en chaîne est utilisée pour trouver les autres dérivées du modèle. Cette approche est récursive, c'est-à-dire que chacune des dérivées de la fonction de coût est exprimée en utilisant les dérivées - provenant d'autres sections du réseau de neurones - calculées préalablement. On utilise maintenant cette technique pour évaluer les dérivées des connections V_{ij} de la couche de sortie:

$$\frac{\partial NLL}{\partial V_{ij}} = \frac{\partial NLL}{\partial a_i^{(s)}} \frac{\partial a_i^{(s)}}{\partial V_{ij}} \tag{1.24}$$

$$\frac{\partial NLL}{\partial V_{ij}} = (\widehat{y}_i - y_i)x_j \tag{1.25}$$

Ce qui sous forme matricielle donne le gradient de la matrice de sortie, se traduisant par l'équation suivante:

$$\frac{\partial NLL}{\partial \mathbf{V}} = (\widehat{\mathbf{y}} - \mathbf{y})\mathbf{x}^T \tag{1.26}$$

On utilise le même procédé pour les biais c_i de la couche de sortie.

$$\frac{\partial NLL}{\partial c_i} = \frac{\partial NLL}{\partial a_i^{(s)}} \frac{\partial a_i^{(s)}}{\partial c_i} \tag{1.27}$$

$$\frac{\partial NLL}{\partial c_i} = (\widehat{y}_i - y_i)1 = \widehat{y}_i - y_i \tag{1.28}$$

$$\frac{\partial NLL}{\partial \mathbf{c}} = \widehat{\mathbf{y}} - \mathbf{y} \tag{1.29}$$

Notons que ces dérivées sont équivalentes à celles des équations 1.10 et 1.11 de la section sur les réseaux de neurones linéaires. Pour faire circuler à travers le réseau l'information sur la dérivée de la fonction de coût, nous devons calculer le gradient pour le vecteur de sortie de la couche cachée $\mathbf{h}^{(M)}$.

$$\frac{\partial NLL}{\partial \mathbf{h}^{(M)}} = \frac{\partial NLL}{\partial \mathbf{a}^{(s)}} \frac{\partial \mathbf{a}^{(s)}}{\partial \mathbf{h}^{(M)}} \tag{1.30}$$

$$\frac{\partial NLL}{\partial \mathbf{h}^{(M)}} = \mathbf{V}^T(\widehat{\mathbf{y}} - \mathbf{y}) \tag{1.31}$$

Grâce à la règle de dérivée en chaîne, on peut utiliser le gradient de $\mathbf{h}^{(M)}$ pour exprimer celui du vecteur de pré-activation de la couche cachée $\mathbf{a}^{(M)}$.

$$\frac{\partial NLL}{\partial \mathbf{a}^{(M)}} = \frac{\partial NLL}{\partial \mathbf{h}^{(M)}} \frac{\partial \mathbf{h}^{(M)}}{\partial \mathbf{a}^{(M)}} \tag{1.32}$$

$$\frac{\partial NLL}{\partial \mathbf{a}^{(M)}} = (\mathbf{V}^T(\widehat{\mathbf{y}} - \mathbf{y})) \odot (\mathbf{h}^{(M)} \odot (1 - \mathbf{h}^{(M)})) \tag{1.33}$$

où $\odot$ représente une multiplication terme à terme (produit matriciel de Hadamard). Ici nous avons utilisé la dérivée de la fonction d'activation sigmoïde pour chaque élément du vecteur de pré-activation:

$$\frac{\partial \text{sig}(a_j^{(M)})}{\partial a_j^{(M)}} = \text{sig}(a_j^{(M)})(1 - \text{sig}(a_j^{(M)})) = h_j^{(M)}(1 - h_j^{(M)}) \tag{1.34}$$

Maintenant, avec le gradient du vecteur de pré-activation, on peut facilement calculer la dérivée des connections $\mathbf{W}^{(M)}$ de la couche cachée:

$$\frac{\partial NLL}{\partial \mathbf{W}^{(M)}} = \frac{\partial NLL}{\partial \mathbf{a}^{(M)}} \frac{\partial \mathbf{a}^{(M)}}{\partial \mathbf{W}^{(M)}} \tag{1.35}$$

$$\frac{\partial NLL}{\partial \mathbf{W}^{(M)}} = \left(\frac{\partial NLL}{\partial \mathbf{a}^{(M)}}\right) (\mathbf{h}^{(M-1)})^T \tag{1.36}$$

Encore une fois, on utilise le même procédé pour le gradient du vecteur de biais $\mathbf{b}^{(M)}$ de cette couche cachée.

$$\frac{\partial NLL}{\partial \mathbf{b}^{(M)}} = \frac{\partial NLL}{\partial \mathbf{a}^{(M)}} \frac{\partial \mathbf{a}^{(M)}}{\partial \mathbf{b}^{(M)}} \tag{1.37}$$

$$\frac{\partial NLL}{\partial \mathbf{b}^{(M)}} = \frac{\partial NLL}{\partial \mathbf{a}^{(M)}} \tag{1.38}$$

Ce qui donne le même résultat que le gradient de la pré-activation puisque $\frac{\partial \mathbf{a}^{(M)}}{\partial \mathbf{b}^{(M)}} = [1\ 1\ ...\ 1]$. Dans le cas ou $M = 1$, un réseau de neurones avec une seule couche cachée, on peut arrêter de propager l'information sur la dérivée du coût, car le gradient de tous les paramètres a été calculé. On doit juste remplacer le vecteur $\mathbf{h}^{(M-1)}$ de l'équation 1.36 par l'entrée $\mathbf{x}$ du modèle (car c'est elle qui a été utilisée pour le calcul), ce qui donne:

$$\frac{\partial NLL}{\partial \mathbf{W}^{(1)}} = \left(\frac{\partial NLL}{\partial \mathbf{a}^{(1)}}\right) (\mathbf{x})^T \tag{1.39}$$

Dans le cas ou $M > 1$, on doit cependant continuer à propager l'information du coût. Le procédé que l'on vient d'appliquer à la couche M est répété pour chacune des couches suivantes, c'est-à-dire que l'on calcule le gradient de $\mathbf{h}^{(i-1)}$ et $\mathbf{a}^{(i-1)}$ en

utilisant celui de $\mathbf{a}^{(i)}$ de la couche précédente pour ensuite trouver les dérivées des connections $\mathbf{W}^{(i-1)}$ et des biais $\mathbf{b}^{(i-1)}$.

$$\frac{\partial NLL}{\partial \mathbf{a}^{(i-1)}} = \frac{\partial NLL}{\partial \mathbf{a}^{(i)}} \frac{\partial \mathbf{a}^{(i)}}{\partial \mathbf{h}^{(i-1)}} \frac{\partial \mathbf{h}^{(i-1)}}{\partial \mathbf{a}^{(i-1)}} \tag{1.40}$$

$$\frac{\partial NLL}{\partial \mathbf{W}^{(i-1)}} = \frac{\partial NLL}{\partial \mathbf{a}^{(i-1)}} \frac{\partial \mathbf{a}^{(i-1)}}{\partial \mathbf{W}^{(i-1)}} \tag{1.41}$$

$$\frac{\partial NLL}{\partial \mathbf{b}^{(i-1)}} = \frac{\partial NLL}{\partial \mathbf{a}^{(i-1)}} \frac{\partial \mathbf{a}^{(i-1)}}{\partial \mathbf{b}^{(i-1)}} \tag{1.42}$$

Une fois que tous les paramètres du réseau de neurones ont été calculés, on peut procéder à leur mise à jour, comme il a déjà été expliqué avec les équations 1.14 et 1.15 de la section sur les réseaux de neurones linéaires.

1.4 Autoencodeur

Un autoencodeur [44] est une sorte de réseau de neurones utilisé en apprentissage non-supervisé (voir section 1.1). Contrairement à l'apprentissage supervisé, l'ensemble de données utilisé pour l'entraînement du modèle est composé d'exemples qui n'ont pas de cible. Pour un autoencodeur, l'entrée $\mathbf{x}$ devient aussi sa cible. Ce modèle apprend donc à reconstruire son entrée, ce qui a pour but d'apprendre une représentation des données compacte mais riche en information. Cette représentation sera utile pour accomplir certaines tâches ou encore pour faire de la réduction de dimensionnalité.

En général, un autoencodeur prend en entrée un vecteur $\mathbf{x}$ et calcule une couche cachée avec une fonction d'activation. Ensuite, il construit le vecteur de sortie $\hat{\mathbf{x}}$, prédiction de l'entrée $\mathbf{x}$ qui devient aussi la cible du modèle (voir figure 1.5). Ici $\mathbf{x}$ remplace le vecteur cible $\mathbf{y}$ présenté dans les sections précédentes. Notons qu'un autoencodeur peut prendre n'importe laquelle des formes présentées précédemment (section 1.3.1 et 1.3.2) et d'autres formes encore. L'important est que l'entrée soit aussi la cible.

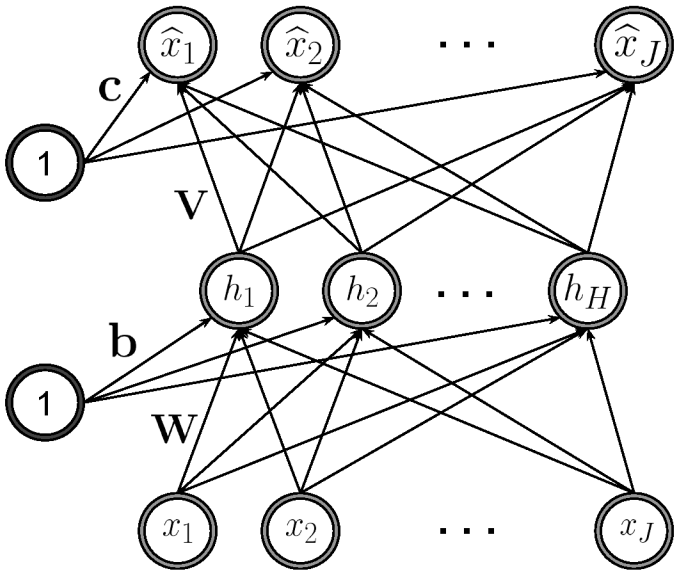


figure 1.5 – Autoencodeur typique.

1.5 Extraction de caractéristiques

Les réseaux de neurones peuvent aussi être utilisés dans le but de faire de l'extraction de caractéristiques, c'est-à-dire utiliser le modèle pour obtenir un nouveau vecteur de représentation pour chaque exemple. On peut vouloir se servir de cette approche pour différentes raisons, par exemple pour faire de la réduction de dimensionnalité et ainsi obtenir un vecteur de taille réduite. Celui-ci contient l'information nécessaire pour effectuer une tâche quelconque, mais permet de l'effectuer plus rapidement. Un autre exemple est de vouloir passer d'une représentation discrète "*one-hot*", qui ne permet pas d'exprimer directement certaines informations - comme la similitude entre deux entrées - à une représentation vectorielle continue qui permet de se servir des mesures de distance de façon utile.

On commence par entraîner normalement un réseau de neurones comme il a été montré précédemment. Bien qu'il soit fréquent de faire de l'extraction de caractéristiques à partir d'un autoencodeur à une seule couche cachée, on peut très bien se

servir d'un modèle qui prédit une cible $\mathbf{y}$ et/ou qui utilise plusieurs couches cachées. Habituellement, une fois le modèle entraîné, il suffit de propager l'entrée $\mathbf{x}$ à travers le modèle jusqu'à la dernière couche cachée du modèle (celle qui sert au calcul de la couche de sortie). C'est cette dernière couche qui servira de nouvelle représentation pour $\mathbf{x}$ (voir figure 1.6). Bien entendu, il existe d'autres façons d'extraire une nouvelle représentation dont le choix dépend de la tâche.

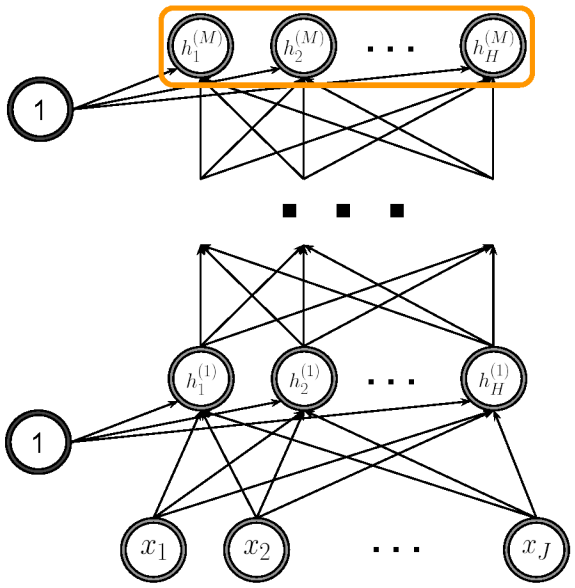


figure 1.6 – Dernière couche cachée d'un réseau de neurones qui sert de nouvelle représentation pour l'exemple en entrée.

Chapitre 2

Mise en contexte

Dans le cadre de ce doctorat, nous avons développé de nouveaux réseaux de neurones appliqués au domaine du Traitement Automatique du Langage Naturel (TALN), mieux connu sous l'acronyme NLP (*Natural Language Processing*). Cette section présente une partie du vaste domaine du TALN, en fait, celle liée à la recherche effectuée. Nous allons ensuite parler de l'état de l'art en 2012, au commencement du doctorat, pour les tâches abordées dans cette thèse. Cette section se termine par une discussion sur les défis qui ont dû être relevés.

Les approches à base de règles ont longtemps été utilisées en traitement automatique du langage, depuis les années 60 jusqu'au milieu des années 80, et ont obtenu des résultats intéressants, comme simuler une conversation écrite avec un humain. Les approches statistiques ont ensuite révolutionné le monde du traitement de la langue vers la fin des années 80. Ces approches statistiques ont permis de créer des systèmes plus performants sans avoir besoin d'utiliser une hiérarchie complexe de règles. Ce document n'abordera que la partie statistique du domaine appliquée à du texte écrit. Cette section débute par un sujet essentiel et fondamental, c'est-à-dire comment obtenir une représentation utile du texte pouvant servir à un ordinateur dans le but de résoudre des problèmes.

2.1 Représentation du texte en TALN

Dans le domaine du traitement automatique du langage, un corpus est un ensemble de différents textes, comme un ensemble de phrases ou de documents. Avant d'être utilisé par des modèles statistiques, un corpus doit passer par une étape de pré-traitement.

2.1.1 Pré-traitement pour le texte

Le pré-traitement sert à prendre du texte en format brut et à le transformer en un format plus facile à manipuler. Nous présentons ici une liste des pré-traitements les plus couramment utilisés, qui peuvent être employés seuls ou encore combinés entre eux.

Pour commencer, on doit découper les corpus dont on se sert en séquences de mots.

Segmentation (*Tokenization*)

Pour expliquer la segmentation dans le cadre du traitement automatique du langage, on redéfinit le concept de "mot" pour y inclure l'idée d'unités lexicales élémentaires. Donc à partir de maintenant un mot peut représenter aussi des caractères comme ".", ",", "l'", etc.

Le texte dans son format brut peut être vu comme une simple séquence de caractères. La segmentation consiste à identifier les mots de cette séquence pour la convertir ensuite en une liste de mots. Par exemple, la segmentation de la phrase suivante:

"Le chat boit de l'eau."

Donne la liste de mots:

"Le", "chat", "boit", "de", "l'", "eau", "."

Une fois la liste de mots obtenue, la modification de ceux-ci peut être une étape utile.

Racinisation ou lemmatisation (*Stemming or lemmetization*)

Cette étape n'est pas toujours nécessaire et dépend grandement des tâches que l'on désire accomplir. Le but de la lemmatisation est de minimiser les variations des mots qui ont une même signification. Par exemple, on peut vouloir remplacer les mots "lenteur", "ralentir" ou "lentement" par leur radical "lent". La même idée peut-être appliquée aux différents mois de l'année (remplacer "juin" ou "juillet" par "mois") ou encore aux chiffres (remplacer "4" ou "7" par "num"). On se sert donc seulement des approches qui aident la tâche choisie, en enlevant l'information qui ne lui est pas utile. Similairement, un autre pré-traitement couramment utilisé est d'enlever la majuscule du début de phrase et de garder une majuscule aux noms propres.

Enlever les mots vides (*stop words*)

Comme la lemmatisation, ce pré-traitement est optionnel et son utilisation dépend grandement des tâches que l'on désire accomplir. Les mots vides sont les mots qui n'ont pas de sens significatif et qui souvent sont les plus communs dans des textes, par exemple "le", "la" ou "des". On peut également décider de créer une liste de mots vides pour un domaine en particulier. Pour certaines tâches, par exemple la classification de documents, les mots vides peuvent être nuisibles et les enlever peut s'avérer utile.

Créer un vocabulaire (dictionnaire)

Une étape importante en traitement automatique des langues est la création d'un vocabulaire (aussi appelé dictionnaire), c'est-à-dire une liste de tous les mots que l'on désire garder pour représenter un texte. Habituellement, un vocabulaire est créé à partir d'un corpus qui peut comporter beaucoup de mots différents. Un dictionnaire peut être de très grande taille et il n'est pas rare d'en avoir qui comptent plusieurs centaines de milliers de mots. Il n'est donc pas toujours souhaitable de tous les garder. Le choix de ces mots peut être fait selon différents critères, comme leur fréquence ou leur pertinence.

Il est commun de rajouter des mots spéciaux au vocabulaire, servant à représenter des concepts utiles comme le début ou la fin d'une phrase. Aussi, les mots ne faisant pas partie du vocabulaire sont rassemblés sous la bannière d'un seul mot spécial, qui

est ajouté au dictionnaire, prenant une forme populaire comme "UNK" ou "OOV".

Au final, le vocabulaire sert à donner un identifiant (ID) unique à chacun de ses mots, correspondant généralement à l'index dans la liste. On obtient en fin de compte une liste de chiffres pour représenter chaque exemple d'un corpus, facilitant la manipulation des données pour un ordinateur.

Une fois le pré-traitement terminé, on peut passer à une autre étape et trouver de nouvelles représentations de mots, plus utiles et plus informatives.

2.1.2 Représentation des mots

Le fait d'avoir pour chaque mot un identifiant unique parmi un ensemble d'identifiants s'apparente à un concept de représentation vectorielle appelé *one-hot*. En effet, une représentation *one-hot* consiste en un vecteur où tous les éléments ont une valeur de 0 à l'exception d'un seul qui est égal à 1. Dans notre cas, chaque élément du vecteur correspond à un mot du vocabulaire et l'identifiant (qui est à la fois l'index pour le vecteur et pour le vocabulaire) signale quel élément (mot) a la valeur 1. Chaque identifiant possède donc une représentation vectorielle *one-hot*.

Chacun des mots du vocabulaire est maintenant associé à un encodage *one-hot* et peut être utilisé comme représentation de base qui servira d'entrée à un modèle d'apprentissage. Notons que le nombre d'éléments (dimension) du vecteur grandit linéairement avec la taille du dictionnaire. Dans le domaine du traitement automatique du langage, les dictionnaires ont tendance à être volumineux, ce qui peut donner des vecteurs de très haute dimension. Le problème avec cette approche est que le vecteur *one-hot* ne donne pas beaucoup d'information sur le mot qu'il représente. Par exemple, il ne permet pas de faire des comparaisons pour savoir si deux mots sont sémantiquement similaires. Nous allons maintenant voir comment obtenir de meilleures représentations.

Word embeddings [6]

Comme il a déjà été mentionné, dans le contexte du traitement automatique du langage, un vecteur *one hot* aura tendance à être de grande taille, ce qui peut poser un problème. En effet, plus le nombre de dimensions augmente, plus l'espace dans

lequel se trouve l'entrée grandit et plus on a besoin de données pour couvrir tout cet espace. De plus, l'augmentation du nombre de paramètres peut rendre le calcul plus long pour les modèles qui utilisent cette représentation *one-hot*.

On peut s'attaquer à ces différents problèmes en apprenant pour chaque mot du vocabulaire une représentation vectorielle composée de valeurs réelles ($\mathbb{R}$), communément appelée *word embeddings*. Pour des fins pratiques, on utilise souvent une matrice de représentation des mots où chaque colonne de la matrice est un vecteur qui représente un mot (voir figure 2.1).

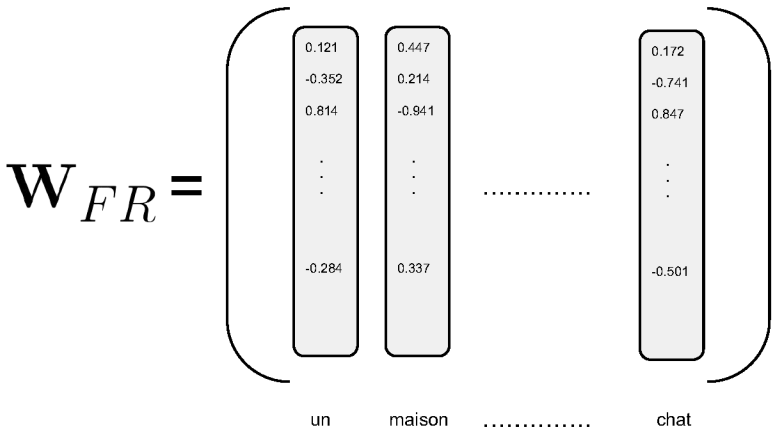


figure 2.1 – Matrice de représentation des mots pour un vocabulaire en français.

Le *word embeddings* permet d'utiliser pour les mots un vecteur plus petit et plus riche en information. En effet, le vocabulaire est alors représenté dans un espace continu où les mots sémantiquement similaires sont des points placés à courte distance les uns des autres. Cette approche est utile, car elle permet d'obtenir de l'information sur les relations qui peuvent exister entre les mots.

Pour obtenir une matrice de représentation des mots, on considère chacun de ses éléments comme faisant partie des paramètres du modèle choisi. Ensuite, il suffit d'apprendre la valeur de ces paramètres en utilisant la technique de descente de gradient, minimisant la fonction objectif du modèle. On apprend ainsi des représentations de

mots contenants de l'information utile au modèle pour accomplir la tâche qui lui est assignée.

Maintenant que l'on sait comment représenter des mots, il s'agit de faire la même chose pour du texte en général, comme des phrases, ou encore pour des documents.

2.1.3 Représentation du texte

Une première approche simple pour représenter du texte est de le faire sous forme de sac de mots (*bag-of-words*), qui est un prolongement de l'idée des vecteurs *one-hot*. En effet, cette représentation simplifiée du texte consiste en un vecteur où chaque élément est associé à un mot du vocabulaire, comme pour l'approche *one-hot*. Cependant, ce vecteur ne représente pas un seul mot, mais plutôt un groupe de mots (un sac de mots), correspondant à du texte (comme une ou des phrases ou encore un document). Avec cette approche, on perd l'information sur l'ordre des mots. Pour un texte donné, on calcule la fréquence des mots du vocabulaire et chaque élément du vecteur de représentation est égal à la fréquence du mot auquel il est associé. Par exemple, supposons que l'on veut représenter la phrase (qui a déjà subi la phase de pré-traitement):

"le chat est sur le comptoir ."

Pour ce faire, on utilise le vocabulaire suivant, constitué de dix mots:

— ID 0: "assis"
— ID 1: "le"
— ID 2: "chien"
— ID 3: "sur"
— ID 4: "est"
— ID 5: "mange"
— ID 6: "chat"
— ID 7: "."
— ID 8: "comptoir"
— ID 9: "OOV"

Ou "OOV" représente tous les mots qui ne font pas partie du vocabulaire, tel qu'expliqué précédament (section 2.1.1). On se sert donc d'un vecteur de taille 10 ainsi que des identifiants du vocabulaire pour créer le sac de mots, ce qui donne le vecteur:

$$[\ 0,\ 2,\ 0,\ 1,\ 1,\ 0,\ 1,\ 0,\ 1,\ 0\ ]$$

La technique du sac de mots donne une représentation utile pour analyser du texte et ce malgré la perte d'information sur l'ordonnancement. D'autres approches similaires sont utilisées en traitement automatique du langage, comme la très populaire représentation tf-idf (*term frequency–inverse document frequency*), dont la valeur associée à chaque mot est affectée par sa fréquence dans le document en relation avec celle du corpus. Plus récemment, une nouvelle approche, qui donne de très bons résultats, permet d'obtenir une nouvelle représentation vectorielle à base de sujets.

Représentation par sujets (*topic representation*)

Un modèle de sujets [47] (*topic model*) est une famille de modèles statistiques en apprentissage automatique employé dans le cadre du traitement automatique du langage. Ce modèle, qui utilise généralement des documents comme données d'apprentissage, permet de découvrir des regroupements de mots ayant une signification reliée. Chaque regroupement de mots correspond à un sujet (*topic*) ou thème particulier. Un *topic model* donne lui aussi une représentation mathématique à un document et ce sous la forme d'un vecteur (de valeurs continues). Cette représentation peut être vue comme une composition de différentes intensités des sujets. Par exemple, la figure 2.2 illustre un vecteur (généré par un modèle de sujets) représentant un document. Ce vecteur est composé de quatre valeurs continues, où chaque valeur est associée à un sujet. Cette représentation nous informe que le document est fortement relié à l'économie et à la politique, mais pas à l'informatique et aux mathématiques.

Cette approche procure une représentation riche en information et donne habituellement des vecteurs de taille beaucoup plus réduite, comparativement aux autres approches présentées. Ceci permet d'améliorer et d'accélérer considérablement le fait de parcourir, chercher ou résumer des documents faisant partie d'un gros ensemble de textes. Il n'est pas nécessaire de passer à travers tous les mots d'un document à

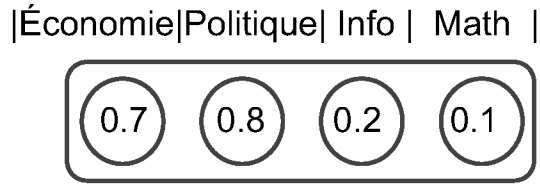


figure 2.2 – Vecteur, généré par un modèle de sujets, représentant un document.

chaque fois qu'on effectue une recherche. Il suffit de les garder en mémoire et d'utiliser la représentation vectorielle des documents.

La représentation vectorielle des documents obtenue par un *topic model* est utilisée par d'autres algorithmes pour effectuer certaines tâches spécifiques.

2.2 Les tâches en TALN

Le traitement automatique du langage est un domaine très vaste et nous concentrerons nos efforts sur les sujets présentés dans cette section.

2.2.1 Classification

La classification en apprentissage automatique appliquée au traitement automatique du langage consiste à prendre un exemple représentant du texte et à l'associer à une étiquette appartenant à un ensemble de classes.

La classification est une tâche importante largement utilisée en TALN. En effet, on se sert de la classification pour trouver des documents, regrouper des livres selon leur style, organiser des courriels, détecter des pourriels (*spam*) et pour faire bien d'autres choses encore.

2.2.2 Recherche d'information (*information retrieval*)

Dans le cadre du traitement automatique du langage, la recherche d'information (*information retrieval*) est une tâche qui consiste à trouver un ou des documents faisant partie d'un large corpus (ensemble de documents). La pertinence des documents retournés doit être en lien avec une source (comme un autre document ou tout simplement du texte). La représentation de documents obtenue par un *topic model* est souvent utilisée pour faire de l'*information retrieval* et évaluer une certaine similitude entre les documents. On désire obtenir une représentation permettant d'améliorer la recherche d'information.

2.2.3 Modélisation de la langue

Le but de la modélisation de la langue est d'assigner une probabilité à n'importe quelle séquence de mots.

On veut apprendre un modèle qui donne de faibles probabilités aux séquences de mots qui sont mal formées et qui n'ont pas de sens. Inversement, on désire que ce modèle donne une forte probabilité aux séquences de mots bien construites, c'est-à-dire celles que les gens ont l'habitude d'échanger et qui sont grammaticalement correctes.

Pour modéliser la langue, plusieurs modèles sont basés sur une approche Markovienne d'ordre n. La probabilité d'une séquence de mots est calculée en utilisant la règle des probabilités en chaîne. La propriété de Markov d'ordre n appliquée au TALN consiste à supposer que la probabilité conditionnelle d'un mot futur, utilisée pour calculer la distribution d'une séquence de mots v_i, ne dépend que des $n-1$ mots appartenant au passé.

$$p(v_1, v_2, ..., v_T) = \prod_{i=1}^{T} p(v_i | v_{i-1}, v_{i-2}, ..., v_{i-(n-1)}) \tag{2.1}$$

Avec l'hypothèse de Markov, on ne peut faire qu'une approximation de la véritable distribution de probabilité des mots. Un plus grand n permet plus de précision. Il s'agira maintenant d'évaluer la performance d'un modèle.

Perplexité

La perplexité est une mesure permettant d'évaluer la qualité des prédictions d'un modèle génératif de textes. Avec cette mesure, le meilleur modèle est celui qui donne la plus grande probabilité à un ensemble de phrases (ou de textes) test (qui n'a jamais été vu lors de l'entraînement).

La perplexité est l'inverse de la probabilité d'une phrase $\mathbf{v}$, composée des mots $v_1, v_2, ..., v_T$ et normalisée par le nombre de mots T (équation 2.2).

$$\begin{aligned} PP(\mathbf{v}) &= p(v_1 v_2 ... v_T)^{-\frac{1}{T}} \\ &= \sqrt[T]{\frac{1}{p(v_1 v_2 ... v_T)}} \end{aligned} \tag{2.2}$$

Plus la perplexité est basse, plus la vraisemblance augmente. On cherche donc à minimiser la perplexité.

Pour un ensemble $\mathcal{D}$ de documents $\mathbf{v}^t$, on calcule souvent la perplexité normalisée par le nombre N de documents.

$$PP(\mathcal{D}) = \sqrt[N]{\prod_{t=1}^{N} \sqrt[|\mathbf{v}^t|]{\frac{1}{p(\mathbf{v}^t)}}} \tag{2.3}$$

2.3 Revue de littérature

Nous allons maintenant présenter certains modèles utilisés pour faire différentes tâches en traitement automatique du langage et considérés comme faisant partie de l'état de l'art au moment où cette recherche de doctorat a commencé.

2.3.1 Latent Dirichlet Allocation (LDA)

Nous présentons le modèle LDA [9], car il est l'un des plus connus et des plus importants dans le monde du *topic modeling*.

Le LDA est un modèle génératif que l'on peut décomposer en trois parties. La figure 2.3 est utilisée pour expliquer le modèle et comment (hypothétiquement) les

documents sont générés par celui-ci.

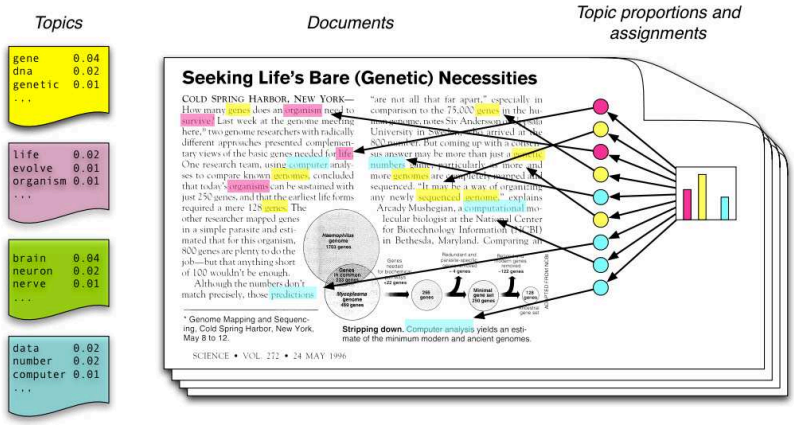


figure 2.3 – Génération d'un document avec le modèle LDA [9].

On commence avec la distribution de Dirichlet qui est une distribution de distributions multinomiales. Cela signifit que pour un document, la distribution de Dirichlet génère une distribution de sujets (le nombre de sujets est préétabli) correspondant à l'histogramme à droite de la figure 2.3 et qui est définie par l'équation 2.4:

$$p(\boldsymbol{\theta}|\boldsymbol{\alpha}) = \frac{\Gamma\left(\sum_i \alpha_i\right)}{\prod_i \Gamma(\alpha_i)} \prod_{i=1}^{S} \theta_i^{\alpha_i - 1} \tag{2.4}$$

où Γ est la fonction gamma et S le nombre de sujets. Chaque dimension du vecteur $\boldsymbol{\theta}$ représente un sujet différent. Les éléments $\theta_i \geq 0$ et leur somme doivent faire un total de 1. Les paramètres de la distribution de Dirichlet α_i doivent avoir une valeur plus grande que 0. Notons qu'on obtient une distribution parcimonieuse quand les α_i ont une valeur entre 0 et 1. La figure 2.4 montre des exemples de distributions possibles sur trois sujets ($\boldsymbol{\theta} = \{\theta_1, \theta_2, \theta_3\}$).

La deuxième partie du modèle LDA est d'associer aléatoirement (à partir de la distribution de sujets obtenue) un sujet z à chaque mot du document et est illustrée dans la figure 2.3 par les petits cercles de couleur. Chaque cercle représente un mot du document et sa couleur correspond à un sujet qui lui est associé.

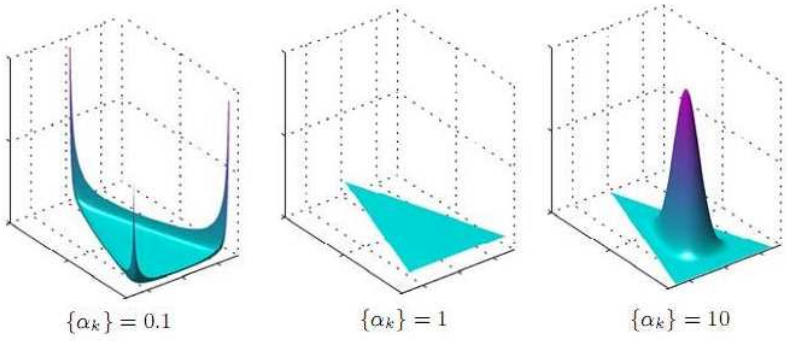


figure 2.4 – Exemple de distribution de Dirichlet à trois dimensions [1].

Finalement, chaque sujet z (*topic*) a une distribution de probabilité différente, qui lui est associée, pour tous les mots du vocabulaire utilisés par le modèle. La liste de carrés de couleur à gauche de la figure 2.3 correspond à une liste de sujets (ex.: génétique, informatique, etc.). La troisième et dernière partie du modèle LDA consiste donc à générer un mot, provenant du vocabulaire, pour chaque cercle à partir de la distribution de mots se rapportant au sujet qui lui est associé.

On utilise le postérieur sachant un document v ($p(\boldsymbol{\theta}|v)$), c'est-à-dire la distribution multinomiale des sujets (qui provient de la Dirichlet) d'un document donné, pour obtenir une représentation sous forme de vecteur. Cette représentation est ensuite utilisée pour effectuer différentes tâches comme calculer la similarité entre les documents ou faire de la recherche d'information.

2.3.2 Classificateur multilangue avec traduction automatique

Dans le cas où l'on désire classifier automatiquement un ensemble test de documents dans une langue, mais que le ou les ensembles d'entraînement étiquetés n'existent que dans une autre langue, comment peut-on entraîner un modèle de classification sur une langue en particulier et l'utiliser sur une autre ?

Une approche naïve, mais qui peut s'avérer très efficace, est d'utiliser un modèle de traduction automatique. En effet, on peut simplement traduire l'ensemble test (dans

la langue de l'entraînement) et appliquer le classifieur sur la traduction. On peut aussi faire l'inverse, c'est-à-dire traduire l'ensemble d'entraînement (dans la langue du test) et entraîner le modèle de classification sur cette traduction pour ensuite pouvoir l'utiliser directement sur l'ensemble test.

La performance de la classification des documents de l'ensemble test dépend grandement de la qualité des traductions. Il est donc important d'utiliser un bon modèle de traduction automatique.

Modèle de traduction automatique

Nous nous basons ici sur le système Moses [29] pour décrire le fonctionnement d'un modèle de traduction automatique. Pour alléger les explications de cette section, on élargit la signification du mot "phrase" pour y inclure le concept de segment de phrase (partie de phrase).

On a d'abord besoin de données bilingues pour entraîner le modèle de traduction. L'ensemble de données doit être composé de paires de phrases bilingues, où chaque paire possède une phrase $\mathbf{v^s}$ dans la langue source ainsi que sa traduction $\mathbf{v^c}$ dans la langue cible. Le nombre de mots de la source n'est pas nécessairement le même que celui de la cible.

Un certain pré-traitement des données est effectué pour obtenir un format qui puisse être utilisé par le modèle. Généralement les étapes de pré-traitement les plus importantes sont la "tokenisation" et le "truecasing". La "tokenisation" permet de découper les phrases par mots pour obtenir une liste ordonnée des mots et des signes de ponctuation. Le "truecasing" sert à enlever les majuscules présentes dans les mots qui ne sont pas des noms propres (ex.: mot de début de phrase). Il est aussi courant de se débarrasser des phrases qui sont considérées comme trop longues.

Les modèles en traduction automatique qui obtiennent les meilleures performances utilisent habituellement l'approche *phrase-based*, qui consiste à traduire de phrase à phrase (au lieu de mot à mot). Le système de traduction tente de trouver la phrase traduite $\mathbf{v^c}$ qui maximise $p(\mathbf{v^c}|\mathbf{v^s})$. On peut reformuler le problème en utilisant la loi de Bayes.

$$\text{argmax}_{\mathbf{v^c}}\ p(\mathbf{v^c}|\mathbf{v^s}) = \text{argmax}_{\mathbf{v^c}}\ \frac{p(\mathbf{v^s}|\mathbf{v^c})p(\mathbf{v^c})}{p(\mathbf{v^s})} \tag{2.5}$$

Étant donné que l'on maximise en fonction de la phrase cible $\mathbf{v^c}$, on peut simplifier l'équation 2.5 en enlevant le dénominateur $p(\mathbf{v^s})$ comme suit:

$$\text{argmax}_{\mathbf{v^c}}\ p(\mathbf{v^c}|\mathbf{v^s}) = \text{argmax}_{\mathbf{v^c}}\ p(\mathbf{v^s}|\mathbf{v^c})p(\mathbf{v^c}) \tag{2.6}$$

Notre système de traduction automatique se sépare en deux, un modèle de traduction *phrase-based* $p(\mathbf{v^s}|\mathbf{v^c})$ et un modèle de langue $p(\mathbf{v^c})$. Le modèle de langue sert à s'assurer que la traduction a une forme cohérente pour la langue cible.

Un modèle de langage pour la langue cible permet de prédire à une phrase $\mathbf{v^c}$ une probabilité $p(\mathbf{v^c}) = p(v_1^c, v_2^c, ..., v_D^c)$. Un bon modèle donne une probabilité élevée à une phrase syntaxiquement et sémantiquement correcte, mais une probabilité faible pour une phrase dénuée de sens ou de structure. Il existe plusieurs approches pour calculer la probabilité d'une phrase. Une façon de faire est d'utiliser une combinaison de n-gram, calculant la probabilité d'un mot à partir des $n-1$ mots précédents (séquence total de n mots). Habituellement, les plus petits n-gram sont utilisés pour combler le fait que certaines séquences de mots n'existent pas dans les plus longs n-gram (pas observés dans l'ensemble d'entraînement). Les modèles de langage sont un domaine en soit et nous n'irons pas plus en détail sur le sujet dans cette section.

Il faut maintenant un modèle de traduction *phrase-based* qui modélise la probabilité de la source sachant la cible ($p(\mathbf{v^s}|\mathbf{v^c})$). On suppose que $\mathbf{v^s}$ et $\mathbf{v^c}$ sont des phrases syntaxiquement complètes qui sont découpées en petits segments $\mathbf{v^s} = \mathbf{v}_1^s, \mathbf{v}_2^s, ..., \mathbf{v}_D^s$ et $\mathbf{v^c} = \mathbf{v}_1^c, \mathbf{v}_2^c, ..., \mathbf{v}_D^c$. Une table de traduction (*phrase table*) est ce qui donne une probabilité à une paire de petits segments. Elle associe des probabilités à un segment dans une langue sachant un segment dans une autre langue ($p(\mathbf{v}_i^s|\mathbf{v}_i^c)$). Cette probabilité peut se calculer simplement avec la fréquence relative, c'est-à-dire le nombre de fois (dans un ensemble d'entrainement) que $\mathbf{v}_i^s$ et $\mathbf{v}_i^c$ ont été alignés ensemble (i.e. $\mathbf{v}_i^s$ est considéré comme étant la traduction de $\mathbf{v}_i^c$), divisé par le nombre de fois que le segment $\mathbf{v}_i^c$ est aligné avec n'importe quel segment. Notons que la table de traduction peut être remplacée par n'importe quel modèle qui peut donner une probabilité à une paire de phrases.

Lorsqu'on entraîne la table de traduction, on a besoin d'un autre modèle qui s'occupe d'aligner les segments entre deux phrases pour relier les traductions entre elles. L'alignement de segments est une partie très importante du système de traduction et est un domaine de recherche encore très actif.

Finalement, avec la modélisation de la distribution de $p(\mathbf{v^s}|\mathbf{v^c})$, on a tout ce qu'il faut pour générer des traductions (ce qui correspond à l'étape *decoder* dans Moses). La table de traduction nous permet de générer de nombreuse traductions possibles, mais cet espace de recherche est exponentiel (trop de choix) et on ne peut pas toujours parcourir toutes les possibilités. Une technique inspirée du *beam-search* est utilisée quand le nombre de choix devient trop grand et que l'on a besoin de réduire le temps de calcul. Le *beam-search* est une procédure de recherche qui est approximative. On considère un ensemble d'hypothèses (choix) et quand il est trop grand on en filtre un certain nombre. La technique de filtration est complexe et ne sera pas abordée dans cette section.

2.3.3 Réseau de neurones pour la modélisation de la langue

Le réseau de neurones décrit dans cette section a longtemps été considéré comme étant l'état de l'art en modélisation de la langue. Le but est d'entraîner le modèle à assigner des probabilités à des phrases. Comme pour beaucoup d'autres modèles de langue, les réseaux de neurones utilisent l'hypothèse de Markov d'ordre n, où le prochain mot d'une séquence est prédit sachant les $n-1$ mots précédents (figure 2.5).

Pour ce modèle, on se sert de la représentation vectorielle des mots (*word embeddings*), expliquée dans la section 2.1.2, comme entrée pour le réseau de neurones. Plus précisément, c'est le vecteur $\mathbf{W}_{:,v_i}$, une colonne de la matrice $\mathbf{W}$, qui représente le mot ayant l'index v_i et qui sert d'entrée au modèle.

Chaque représentation vectorielle des $n-1$ derniers mots est respectivement associée à une des matrices ordonnées $\mathbf{U}_1,\ \mathbf{U}_2, ...,\ \mathbf{U}_{n-1}$ pour être transformée linéairement. On combine ensuite les transformations linéaires entre elles, ce qui permet d'obtenir une couche cachée représentant les $n-1$ derniers mots.

$$\mathbf{a}(\mathbf{v}_{<i}) = \mathbf{b} + \sum_{k=1}^{n-1} \mathbf{U}_k \cdot \mathbf{W}_{:,v_{i-k}} \tag{2.7}$$

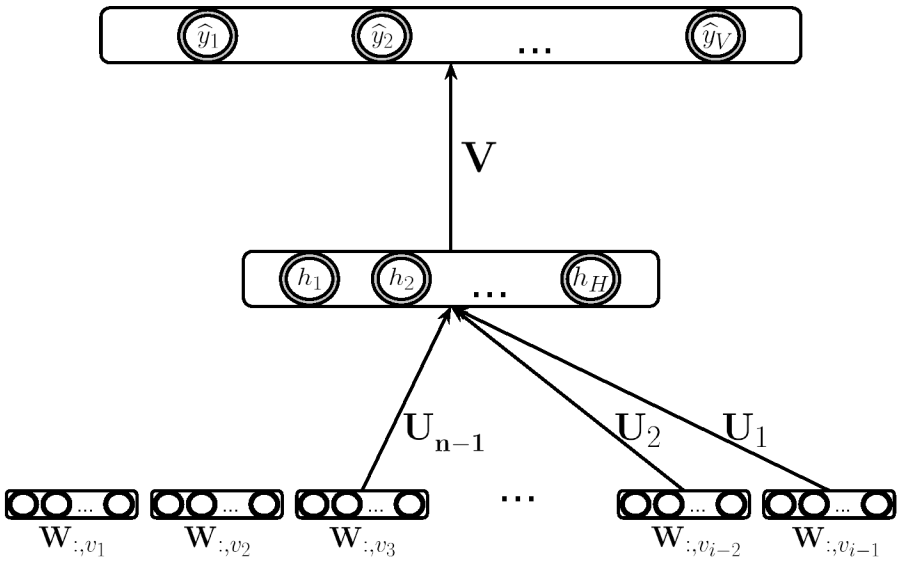


figure 2.5 – Réseau de neurones pour la modélisation de langue qui calcule la probabilité des mots du vocabulaire pour la $i^{\text{ème}}$ position d'une séquence.

$$\mathbf{h}(\mathbf{v}_{<i}) = \text{sigmoid}(\mathbf{a}(\mathbf{v}_{<i})) \tag{2.8}$$

où $\mathbf{v}_{<i}$ est un vecteur représentant tous les mots précédents v_i et où $\mathbf{b}$ est le vecteur de biais pour la couche cachée. Différentes fonctions d'activation peuvent être utilisées. Pour cet exemple, nous nous contenterons d'appliquer la sigmoïde.

Le vecteur $\mathbf{h}(\mathbf{v}_{<i})$ est ensuite propagé à travers la matrice de poids $\mathbf{V}$ (de dimension $H \times V$, ou V est la taille du vocabulaire) qui fait la connexion entre la couche cachée et la couche de sortie. Ce calcul donne le vecteur de pré-activation de la couche de sortie $\mathbf{a}^{(s)}(\mathbf{v}_{<i})$, auquel on applique la fonction d'activation softmax pour calculer la probabilité du prochain mot.

$$\mathbf{a}^{(s)}(\mathbf{v}_{<i}) = \mathbf{V} \cdot \mathbf{h}(\mathbf{v}_{<i}) + \mathbf{c} \tag{2.9}$$

$$\widehat{\mathbf{y}} = \text{softmax}(\mathbf{a}^{(s)}(\mathbf{v}_{<i})) \tag{2.10}$$

où $\widehat{\mathbf{y}}$ est donc l'estimation de la probabilité de tous les mots du vocabulaire pour la $i^{\text{ème}}$ position de la séquence. Le vecteur $\mathbf{c}$ est le biais pour la couche de sortie.

La matrice de représentation des mots $\mathbf{W}$ est incluse dans les paramètres du modèle qui est entraîné par descente de gradient. Un réseau de neurones pour la modélisation de la langue est généralement composé d'une seule couche cachée, il est cependant tout à fait envisageable d'en rajouter.

2.4 Problématique

De nombreux défis doivent être relevés pour tenter de résoudre les tâches en traitement automatique de la langue. Nous présentons ici une liste des problèmes les plus évidents ainsi qu'une approche pour les affronter.

On utilise la notation $\mathcal{V}$ pour représenter un vocabulaire de taille V (ensemble de mots). Chaque mot est associé à un indice v où $v \in \{1, ..., V\}$.

2.4.1 Haute dimensionnalité

Un vocabulaire doit avoir une représentation de base pouvant être donnée en entrée à un modèle d'apprentissage. On associe un mot écrit à un encodage *one hot* sous forme de vecteur. Le nombre d'éléments (dimension) du vecteur grandit linéairement avec la taille du vocabulaire (chaque dimension représente un mot). En traitement du langage naturel, les vocabulaires ont tendance à être volumineux et on obtient une représentation vectorielle de très haute dimension. En effet, le nombre de mots utilisé dans un vocabulaire varie généralement entre dix mille et deux cent mille. Le nombre de dimensions pose un problème computationnel et il devient difficile pour un modèle génératif de réussir à calculer en un temps raisonnable des prédictions ou encore les gradients pour l'apprentissage. D'habitude, le réseau est composé d'une matrice $\mathbf{W}$ de sortie qui sert à calculer la probabilité, sachant une certaine couche cachée h, de chaque mot v du vocabulaire. Pour ce faire la fonction *softmax* (2.11) est utilisée:

$$p(v = i|h) = \frac{e^{\mathbf{W}_{v=i,:}\mathbf{h}}}{\sum_{v=1}^{V} e^{\mathbf{W}_{v,:}\mathbf{h}}} \tag{2.11}$$

On peut voir que la complexité de calcul avec la *softmax* grandit linéairement en fonction de V, ce qui peut être très coûteux dans notre cas puisque la taille des vocabulaires en TALN est souvent très grande.

Les réseaux de neurones (en particulier, les réseaux profonds) ont démontré leur puissance en tant que modèle de données de très haute dimension, soit les images et le son (speech). Les documents textes sont un autre exemple. Cependent, au début de la thèse, l'état de l'art en modélisation de documents était largement considéré comme étant LDA, qui n'est pas un réseau de neurones. Donc on a voulu transféré les succès des réseaux de neurones vers ce problème.

2.4.2 Dépendance temporelle: une information difficile à modéliser

Les documents, les phrases ou encore les segments de phrase sont souvent représentés sous forme de sac de mots (*bag-of-words*) pour servir d'entrée aux différents modèles en TALN. Cette façon de faire permet d'utiliser des modèles plus simples pour modéliser le texte, car on ne s'occupe que de l'occurrence des mots et pas de leur ordre. Une telle approche reste performante pour certaines tâches comme la classification de documents. En effet, savoir que les mots "moteur" et "roue" sont présents dans une phrase est beaucoup plus important que de savoir leur ordre quand il s'agit d'associer ce texte à une étiquette comme "véhicule motorisé". Malgré tout, on perd beaucoup d'information en ne prenant pas en considération l'ordre des mots, information qui peut s'avérer cruciale pour certaines tâches comme la traduction automatique. La difficulté est donc d'incorporer de façon efficace l'information de l'ordre des mots dans un modèle, ce qui est beaucoup plus complexe. Plus la dépendance que l'on veut apprendre entre les mots est éloignée dans le temps, plus elle est difficile à modéliser.

Un autre problème est qu'habituellement les modèles prennent en entrée des exemples de taille fixe, et que les documents ou phrases que l'on désire utiliser sont de tailles variables, ce qui rajoute de la difficulté à la modélisation.

Une façon d'approcher le problème est d'inclure de l'information contextuelle qui dépasse les n derniers mots qui sont normalement modélisés par un modèle de langue.

2.4.3 Portabilité d'un classifieur entre deux langues

Une tâche récurrente en TALN est de vouloir classifier automatiquement du texte (comme des documents). Cette classification s'apprend pour une langue en particulier et nécessite un ensemble de données étiquetées. Plus on a de données, mieux ce sera pour créer un bon classifieur. La problématique est qu'il n'y a pas toujours un ensemble de données accessibles pour une langue et que sa création peut s'avérer très coûteuse. De plus, pour certaine tâche de classification largement utilisées, il y a de fortes chances que les données recherchées existent déjà dans une autre langue. L'anglais étant de plus en plus répandu, il n'est pas rare d'avoir pour certaines tâches des ensembles étiquetés existants seulement dans cette langue.

Une approche serait d'entraîner un modèle de classification sur des données étiquetées dans une langue et de réussir ensuite à l'appliquer sur les exemples d'une autre langue.

2.4.4 Discussion sur l'apprentissage de réseaux profonds (deep learning) et motivation

En apprentissage automatique, les réseaux de neurones s'inspirent du fonctionnement des neurones biologiques. Depuis 2006, le domaine a fait de grandes avancées avec l'apprentissage de réseaux profonds (*deep learning*). Cette approche permet, à partir de données d'entrée, d'en extraire une représentation plus riche. Les réseaux profonds sont composés de plusieurs couches de neurones. Chaque couche est une étape qui représente les données de façon un peu plus complexe (abstraite) en se basant sur ce qui a été appris dans la couche précédente. Cette approche est à l'image de l'apprentissage humain qui commence par apprendre des concepts simples, comme l'addition et la soustraction en mathématiques, pour ensuite se baser sur ces concepts afin d'en apprendre des plus complexes, comme la multiplication et le principe de fonction.

2.4. Problématique

Les algorithmes en *Deep learning* ont démontré leur efficacité dans divers problèmes en améliorant les résultats obtenus par d'autres techniques en apprentissage automatique considérées comme étant l'état de l'art. C'est le cas pour le domaine de la reconnaissance d'objets [30] ainsi que pour la reconnaissance de la parole [23]. L'utilisation des réseaux profonds dans le domaine langagier (données textuelles) est intéressante et très prometteuse, mais était relativement peu explorée en 2012. Depuis, le domaine à réussi, et continue, à faire avancer la performance des systèmes en TALN, comme cela a été fait dans les autres domaines.

Une autre raison qui fait penser que les réseaux de neurones sont une bonne approche pour le traitement de la langue est que ces modèles utilisent à l'interne une représentation distribuée des données. Cela veut dire qu'un concept est représenté par un patron d'activation pour un groupe d'unités (généralement de valeur continue) et qu'une partie de ce patron peut être réutilisée pour représenter un autre concept (plus il utilise une partie du même patron d'activation et plus on peut dire qu'il y a une similitude). Un exemple faisant partie d'un ensemble d'entraînement peut être représenté par plusieurs concepts et un autre peut partager une partie de ces concepts, ce qui aidera à la généralisation.

Par exemple, le mot "chat" peut être caractérisé par des concepts comme "animal", "domestiqué" ou encore "marche à quatre pattes" qui seront partagés avec le mot "chien". Si on modélise la probabilité de segments de phrases et que l'on observe pour la première fois le segment "le chat mange", mais que le segment "le chien mange" a été vu en entraînement, alors on peut voir comment le modèle pourrait réussir à généraliser et donner une bonne prédiction pour un exemple jamais observé grâce à une représentation distribuée.

Le contraire d'une représentation distribuée est une représentation locale où un concept est représenté par une ou plusieurs unités qui ne seront jamais réutilisées pour d'autres concepts. Cet atout des réseaux de neurones, entraînés sur un ensemble d'exemples spécifiques, permet de généraliser la prédiction à de nouveaux exemples qui non seulement n'ont jamais été vus, mais qui sont aussi relativement éloignés (ou différents) de l'ensemble d'entraînement. Certaines variations dans une représentation peuvent être plus utiles que d'autres. Quand ce genre de variations est présent dans un exemple complètement différent de tout ce que l'on a déjà vu, alors on a

la possibilité d'obtenir de l'information pertinente permettant de bien généraliser, ce qui permet aussi de mieux contrer la malédiction de la dimensionnalité. Une représentation locale, comme celle d'un bigramme, ne permet pas de généraliser pour des exemples trop différents de ceux vus en entraînement. Ainsi, il ne sert à rien de comparer deux représentations locales entre elles, car on n'obtient aucune information sur une certaine similitude possible. Une représentation distribuée a aussi l'avantage d'être robuste aux bruits et aux erreurs dans les données, mais présente le désavantage de rendre le modèle difficilement interprétable et relativement lent à entraîner.

Dans les prochains chapitres, nous présentons notre recherche à travers nos publications montrant ainsi différentes façons d'utiliser les réseaux de neurones pour apprendre à représenter du texte et à accomplir un certain nombre de tâches dans le domaine du traitement automatique du langage.

Chapitre 3

Réseau de neurones auto-régressif comme modèle de sujets

Résumé

Nous présentons un nouveau modèle qui permet d'apprendre, à partir d'un ensemble de documents non étiquetés, des représentations utiles de documents textuels. Ce modèle est inspiré par le *Replicated Softmax*, un modèle graphique non dirigé qui modélise le dénombrement de mots pour un document. Il a été démontré que le *Replicated Softmax* apprend de meilleurs modèles génératifs ainsi que des représentations de documents plus utiles. Plus précisément, nous nous inspirons de l'équation récursive du champ moyen (*mean-field*) conditionnel afin de définir une architecture de réseaux de neurones qui estiment la probabilité d'observer un nouveau mot pour un document, compte tenu des mots observés précédemment. Ce paradigme permet également de remplacer l'équation softmax, qui calcule de façon coûteuse la distribution des mots, par une distribution hiérarchique des chemins d'un arbre de mots binaire. Le résultat final est un modèle dont la quantité de calculs durant l'entraînement grandit logarithmiquement avec la taille du vocabulaire au lieu de linéairement comme pour le *Replicated Softmax*. Nos expériences montrent que notre modèle est compétitif à la fois comme mo-

dèle génératif de documents et comme algorithme d'apprentissage pour la représentation de documents.

Commentaires

L'article a été publié dans la conférence NIPS (*Neural Information Processing Systems*) en 2012. Mes contributions pour cet article sont les suivantes:

— Participer au développement du modèle DocNADE (*Document Neural Autoregressive Distribution Estimation*) ainsi qu'à plusieurs variantes. DocNADE est un modèle de sujets basé sur l'approche des réseaux de neurones qui permet de calculer efficacement une distribution valide des mots pour les documents.

— Exploration des hyperparamètres et évaluation des performances de DocNADE comme modèle génératif de documents. L'évaluation a été faite en calculant la perplexité sur deux ensembles de données différents.

— Évaluation de la qualité des représentations de documents apprises par le modèle DocNADE. La tâche de recherche d'information (*Information retrieval*) a été utilisée sur un ensemble de données étiquetées pour faire cette évaluation.

A Neural Autoregressive Topic Model

Hugo Larochelle
Département d'informatique
Université de Sherbrooke
hugo.larochelle@usherbrooke.ca

Stanislas Lauly
Département d'informatique
Université de Sherbrooke
stanislas.lauly@usherbrooke.ca

Keywords: Topic model, Neural network, Autoregressive, NLP

Abstract

We describe a new model for learning meaningful representations of text documents from an unlabeled collection of documents. This model is inspired by the recently proposed Replicated Softmax, an undirected graphical model of word counts that was shown to learn a better generative model and more meaningful document representations. Specifically, we take inspiration from the conditional mean-field recursive equations of the Replicated Softmax in order to define a neural network architecture that estimates the probability of observing a new word in a given document given the previously observed words. This paradigm also allows us to replace the expensive softmax distribution over words with a hierarchical distribution over paths in a binary tree of words. The end result is a model whose training complexity scales logarithmically with the vocabulary size instead of linearly as in the Replicated Softmax. Our experiments show that our model is competitive both as a generative model of documents and as a document representation learning algorithm.

3.1 Introduction

In order to leverage the large amount of available unlabeled text, a lot of research has been devoted to developing good probabilistic models of documents. Such models are usually embedded with latent variables or topics, whose role is to capture salient statistical patterns in the co-occurrence of words within documents.

The most popular model is latent Dirichlet allocation (LDA) [8], a directed graphical model in which each word is a sample from a mixture of global word distributions (shared across documents) and where the mixture weights vary between documents. In this context, the word multinomial distributions (mixture components) correspond to the topics and a document is represented as the parameters (mixture weights) of its associated distribution over topics. Once trained, these topics have been found to extract meaningful groups of semantically related words and the (approximately) inferred topic mixture weights have been shown to form a useful representation for documents.

More recently, Salakhutdinov and Hinton [43] proposed an alternative undirected model, the Replicated Softmax which, instead of representing documents as distributions over topics, relies on a binary distributed representation of the documents. The latent variables can then be understood as topic features: they do not correspond to normalized distributions over words, but to unnormalized factors over words. A combination of topic features generates a word distribution by multiplying these factors and renormalizing. They show that the Replicated Softmax allows for very efficient inference of a document's topic feature representation and outperforms LDA both as a generative model of documents and as a method for representing documents in an information retrieval setting.

While inference of a document representation is efficient in the Replicated Softmax, one of its disadvantages is that the complexity of its learning update scales linearly with the vocabulary size V, i.e. the number of different words that are observed in a document. The factor responsible for this complexity is the conditional distribution of the words given the latent variables, which corresponds to a V-way multinomial logistic regression. In a realistic application scenario, V will usually be in the 100 000's.

The Replicated Softmax is in fact a generalization of the restricted Boltzmann machine (RBM). The RBM is an undirected graphical model with binary observed and latent variables organized in a bi-partite graph. The Replicated Softmax instead has multinomial (softmax) observed variables and shares (replicates) across all observed variables the parameters between an observed variable and all latent variables.

A good alternative to the RBM is the neural autoregressive distribution estimator (NADE) [33]. It is similar to an autoencoder neural network, in that it takes as input a vector of observations and outputs a vector of the same size. However, the connectivity of NADE has been specifically chosen so as to make it a proper generative model for vectors of binary observations. More specifically, NADE outputs the conditional probabilities of each observation given the other observations to its left in the vector. Taking the product of all these conditional probabilities thus yields a proper joint probability over the whole input vector of observations. One advantage of NADE is that computing the parameter gradient of the data negative log-likelihood requires no approximation (unlike in an RBM). Also, unlike in the RBM, NADE does not require a symmetric connectivity, i.e. the weights going in and out of its hidden units can be different.

In this work, we describe DocNADE, a neural network topic model that is similarly inspired by the Replicated Softmax. From the Replicated Softmax, we derive an efficient approach for computing the hidden units of the network. As for the computation of the distribution of words given the hidden units, our feed-forward neural network approach leaves us free to use other conditionals than the V-way multinomial logistic regression implied by the Replicated Softmax. In particular, we instead opt for a hierarchy of binary logistic regressions, organized in a binary tree where each leaf corresponds to a word of the vocabulary. This allows us to obtain a complexity of computing the probability of an observed word scaling sublinearly with V. Our experiments show that DocNADE is competitive both as a generative model of documents and as a learning algorithm for extracting meaningful representations of documents.

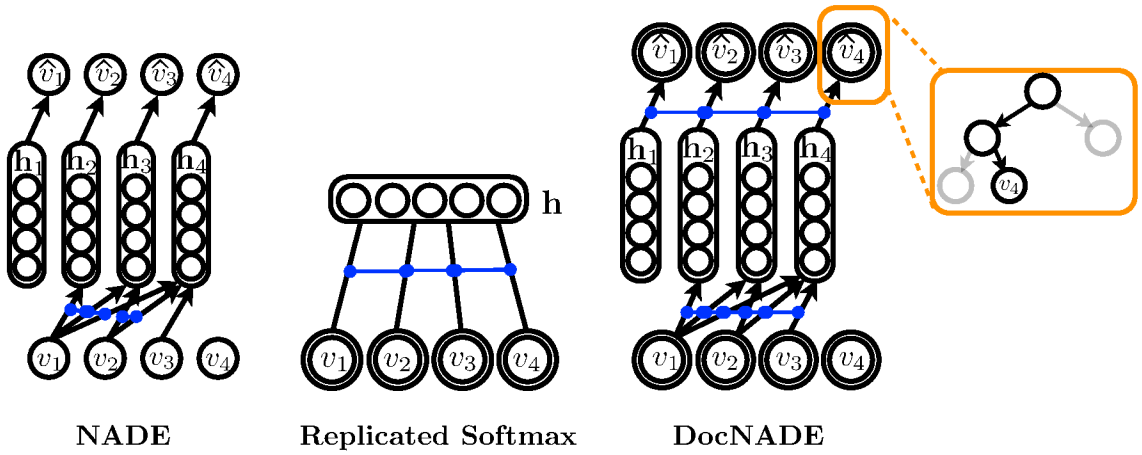


Figure 3.1 – **(Left)** Illustration of NADE. Colored lines identify the connections that share parameters and $\widehat{v}_i$ is a shorthand for the autoregressive conditional $p(v_i|\mathbf{v}_{<i})$. The observations v_i are binary. **(Center)** Replicated Softmax model. Each multinomial observation v_i is a word. Connections between each multinomial observation v_i and hidden units are shared. **(Right)** DocNADE, our proposed model. Connections between each multinomial observation v_i and hidden units are also shared, and each conditional $p(v_i|\mathbf{v}_{<i})$ is decomposed into a tree of binary logistic regressions.

3.2 Neural Autoregressive Distribution Estimation

We start with the description of the original NADE. NADE is a generative model over vectors of binary observations $\mathbf{v} \in \{0,1\}^D$. Through the probability chain rule, it decomposes $p(\mathbf{v}) = \prod_{i=1}^{D} p(v_i|\mathbf{v}_{<i})$ and computes all $p(v_i|\mathbf{v}_{<i})$ using the feed-forward architecture

$$\mathbf{h}_i(\mathbf{v}_{<i}) = \text{sigm}\left(\mathbf{c} + \mathbf{W}_{:,<i}\mathbf{v}_{<i}\right), \qquad p(v_i = 1|\mathbf{v}_{<i}) = \text{sigm}\left(b_i + \mathbf{V}_{i,:}\mathbf{h}_i(\mathbf{v}_{<i})\right) \tag{3.1}$$

for $i \in \{1, \ldots, D\}$, where $\text{sigm}(x) = 1/(1+\exp(-x))$, $\mathbf{W} \in \mathbb{R}^{H \times D}$ and $\mathbf{V} \in \mathbb{R}^{D \times H}$ are connection parameter matrices, $\mathbf{b} \in \mathbb{R}^D$ and $\mathbf{c} \in \mathbb{R}^H$ are bias parameter vectors, $\mathbf{v}_{<i}$ is the subvector $[v_1, \ldots, v_{i-1}]^\top$ and $\mathbf{W}_{:,<i}$ is a matrix made of the $i-1$ first columns of $\mathbf{W}$.

This architecture corresponds to a neural network with several parallel $\mathbf{h}_i(\mathbf{v}_{<i})$ hidden layers and tied weighted connections between v_i and each hidden unit $h_{ij}(\mathbf{v}_{<i})$. Figure 3.1 gives an illustration. Though each $p(v_i = 1|\mathbf{v}_{<i})$ requires the computation of its own hidden layer $\mathbf{h}_i(\mathbf{v}_{<i})$, the tied weights allows to compute them all in $O(DH)$,

where H is the size of each hidden layer $\mathbf{h}_i(\mathbf{v}_{<i})$.

Equation 3.1 provides all the necessary conditionals to compute $p(\ \mathbf{v}\ ) = \prod_i p(v_i|\mathbf{v}_{<i})$. The parameters $\{\mathbf{b}, \mathbf{c}, \mathbf{W}, \mathbf{V}\}$ can then be learned by minimizing the negative log-likelihood with stochastic gradient descent.

The connectivity behind NADE (i.e. the presence of a separate hidden layer $\mathbf{h}_i(\mathbf{v}_{<i})$ for each $p(v_i = 1|\mathbf{v}_{<i})$ with weight sharing) were directly inspired from the RBM. An RBM is an undirected graphical model in which latent binary variables $\mathbf{h}$ interact with the observations $\mathbf{v}$ through an energy function $E(\mathbf{v}, \mathbf{h})$, converted into a distribution over $\mathbf{v}$ as follows:

$$E(\mathbf{v}, \mathbf{h}) = -\mathbf{h}^\top \mathbf{W}\mathbf{v} - \mathbf{b}^\top \mathbf{v} - \mathbf{c}^\top \mathbf{h}, \qquad p(\mathbf{v}) = \sum_{\mathbf{h}} \exp(-E(\mathbf{v}, \mathbf{h}))/Z\,, \tag{3.2}$$

where Z is known as the partition function and ensures that $p(\mathbf{v})$ is a valid distribution and sums to 1. Computing the conditional $p(v_i = 1|\mathbf{v}_{<i})$ in an RBM is generally intractable but can be approximated through mean-field inference. Mean-field inference approximates the full conditional $p(v_i, \mathbf{v}_{>i}, \mathbf{h}|\mathbf{v}_{<i})$ as a product of independent Bernoulli distributions $q(v_k = 1|\mathbf{v}_{<i}) = \mu_k(i)$ and $q(h_j = 1|\mathbf{v}_{<i}) = \tau_j(i)$. To find the values of the variational parameters $\mu_k(i), \tau_j(i)$ that minimize the KL-divergence with $p(v_i, \mathbf{v}_{>i}, \mathbf{h}|\mathbf{v}_{<i})$, the following message passing equations are applied until convergence, for $k \in \{i, \ldots, D\}$ and $j \in \{1, \ldots, H\}$ (see Larochelle and Murray [33] for the derivation):

$$\tau_j(i) \leftarrow \text{sigm}\left(c_j + \sum_{k \geq i} W_{jk}\mu_k(i) + \sum_{k<i} W_{jk}v_k\right), \quad \mu_k(i) \leftarrow \text{sigm}\left(b_k + \sum_j W_{jk}\tau_j(i)\right). \tag{3.3}$$

The variational parameter $q(v_i = 1|\mathbf{v}_{<i}) = \mu_i(i)$ can then be used to approximate $p(v_i = 1|\mathbf{v}_{<i})$.

NADE is derived from the application of each message passing equation only once (with $\mu_j(i)$ initialized to 0), but compensates by untying the weights between each equation and training the truncation directly to fit the available data. The end result is thus the feed-forward architecture of Equation 3.1.

The relationship between the RBM and NADE is important, as it specifies an

effective way of sharing the hidden layer parameters across the conditionals $p(v_i = 1|\mathbf{v}_{<i})$. In fact, other choices not inspired by the RBM have proven less successful (see Bengio and Bengio [4] and Larochelle and Murray [33] for a discussion).

3.3 Replicated Softmax

Documents can't be easily modeled by the RBM for two reasons: words are not binary but multinomial observations and documents may contain a varying number of words. An observation vector $\mathbf{v}$ is now a sequence of words indices v_i taking values in $\{1, \ldots, V\}$, while the size D of $\mathbf{v}$ can vary.

To address these issues, Salakhutdinov and Hinton [43] proposed the Replicated Softmax model, which uses the following energy function

$$E(\mathbf{v}, \mathbf{h}) = -D\ \mathbf{c}^\top \mathbf{h} + \sum_{i=1}^{D} -\mathbf{h}^\top \mathbf{W}_{:,v_i} - b_{v_i} = -D\ \mathbf{c}^\top \mathbf{h} - \mathbf{h}^\top \mathbf{W}\mathbf{n}(\mathbf{v}) - \mathbf{b}^\top \mathbf{n}(\mathbf{v}), \quad (3.4)$$

where $\mathbf{W}_{:,v_i}$ is the v_i^{th} column vector of matrix $\mathbf{W}$ and $\mathbf{n}(\mathbf{v})$ is a vector of size V containing the word count of each word in the vocabulary. Notice that this energy shares its connection parameters across different positions i in $\mathbf{v}$. Figure 3.1 provides an illustration. Notice also that the larger $\mathbf{v}$ is, the more important the terms summed over i in the energy will be. Hence, the hidden bias term $\mathbf{c}^\top \mathbf{h}$ is multiplied by D to maintain a certain balance between all terms.

In this model, the conditional across layers $p(\mathbf{v}|\mathbf{h}) = \prod_{i=1}^{D} p(v_i|\mathbf{h})$ and $p(\mathbf{h}|\mathbf{v}) = \prod_j p(h_j|\mathbf{v})$ factorize and are such that:

$$p(h_j = 1|\mathbf{v}) = \text{sigm}(Dc_j + \sum_i W_{jv_i}) \quad p(v_i = w|\mathbf{h}) = \frac{\exp(b_w + \mathbf{h}^\top \mathbf{W}_{:,w})}{\sum_{w'} \exp(b_{w'} + \mathbf{h}^\top \mathbf{W}_{:,w'})} \quad (3.5)$$

The normalized exponential in $p(v_i = w|\mathbf{h})$ is known as the softmax nonlinearity. We see that, given a value of the topic features $\mathbf{h}$, the distribution each word v_i in the document can be understood as the normalized product of multinomial topic factors $\exp(h_j \mathbf{W}_{j,:}^\top)$ and $\exp(\mathbf{b})$, as opposed to a mixture of multinomial topic distributions.

The gradient of the negative log-likelihood of a single training document $\mathbf{v}^t$ with

respect to any parameter θ has the simple form

$$\frac{\partial - \log p(\mathbf{v}^t)}{\partial \theta} = \mathbb{E}_{\mathbf{h}|\mathbf{v}^t}\left[\frac{\partial}{\partial \theta} E(\mathbf{v}^t, \mathbf{h})\right] - \mathbb{E}_{\mathbf{v},\mathbf{h}}\left[\frac{\partial}{\partial \theta} E(\mathbf{v}, \mathbf{h})\right]. \tag{3.6}$$

Computing the last expectation exactly is too expensive, hence the contrastive divergence [24] approximation is used: the expectation over $\mathbf{v}$ is replaced by a point estimate at a so-called "negative" sample, obtained from K steps of blocked Gibbs sampling based on Equation 3.5 initialized at $\mathbf{v}^t$. Once a negative sample is obtained Equation 3.6 can be estimated and used with stochastic gradient descent training.

Unfortunately, computing $p(v_i = w|\mathbf{h})$ to sample the words during Gibbs sampling is linear in V and H, where V tends to be quite large. Fortunately, given $\mathbf{h}$, it needs to be computed only once before sampling all D words in $\mathbf{v}$. However, when $\mathbf{h}$ is re-sampled, $p(v_i = w|\mathbf{h})$ must be recomputed. Hence, the computation of $p(v_i = w|\mathbf{h})$ is usually the most expensive component of the learning update: sampling the hidden layer given $\mathbf{v}$ is only in $O(DH)$, and repeatably sampling from the softmax multinomial distribution can be in $O(V)$. This makes for a total complexity in $O(KVH+DH)$ of the learning update.

3.4 Document NADE

More importantly for the context of this paper, it can be shown that mean-field inference of $p(v_i = w|\mathbf{v}_{<i})$ in the Replicated Softmax corresponds to the following message passing equations, for $k \in \{i, \ldots, D\}$, $j \in \{1, \ldots, H\}$ and $w \in \{1, \ldots, V\}$:

$$\tau_j(i) \leftarrow \text{sigm}\left(D\ c_j + \sum_{k \geq i} \sum_{w'=1}^{V} W_{jw'} \mu_{kw'}(i) + \sum_{k<i} W_{jv_k}\right), \tag{3.7}$$

$$\mu_{kw}(i) \leftarrow \frac{\exp(b_w + \sum_j W_{jw} \tau_j(i))}{\sum_{w'} \exp(b_{w'} + \sum_j W_{jw'} \tau_j(i))}. \tag{3.8}$$

Following the derivation of NADE, we can truncate the application of these equations to obtain a feed-forward architecture providing an estimate of $p(v_i = w|\mathbf{v}_{<i})$ through $\mu_{iw}(i)$ for all i. Specifically, if we consider a single iteration of message passing with $\mu_{kw'}(i)$ initialized to 0, we untie the parameter weight matrix between each equation

into two separate matrices $\mathbf{W}$ and $\mathbf{V}$ and remove the multiplication by D of the hidden bias, we obtain the following feed-forward architecture:

$$\mathbf{h}_i(\mathbf{v}_{<i}) = \text{sigm}\left(\mathbf{c} + \sum_{k<i} \mathbf{W}_{:,v_k}\right), \quad p(v_i = w|\mathbf{v}_{<i}) = \frac{\exp(b_w + \mathbf{V}_{w,:}\mathbf{h}_i(\mathbf{v}_{<i}))}{\sum_{w'} \exp(b_{w'} + \mathbf{V}_{w',:}\mathbf{h}_i(\mathbf{v}_{<i}))} \tag{3.9}$$

for $i \in \{1, \ldots, D\}$. In words, the probability of the i^{th} word v_i is based on a position dependent hidden layer $\mathbf{h}_i(\mathbf{v}_{<i})$ which extracts a representation out of all previous words $\mathbf{v}_{<i}$. This latent representation is efficient to compute, as it consists simply in a linear transformation followed by an element-wise sigmoidal nonlinearity. Unlike in the Replicated Softmax, we have found that multiplying the hidden bias by D was not necessary and, in fact, slightly hampered the performance of the model, so we opted for its removal.

To obtain the probability of the next word v_{i+1}, one must first compute the hidden layer

$$\mathbf{h}_{i+1}(\mathbf{v}_{<i+1}) = \text{sigm}(\mathbf{c} + \sum_{k<i+1} W_{:,v_k}) = \text{sigm}(\mathbf{W}_{:,v_i} + \mathbf{c} + \sum_{k<i} \mathbf{W}_{:,v_k}) \tag{3.10}$$

which is efficiently computed by reusing the previous linear transformation $\mathbf{c} + \sum_{k<i} \mathbf{W}_{:,v_k}$ and adding $\mathbf{W}_{:,v_i}$. With this procedure, we see that computing all hidden layers $\mathbf{h}_i(\mathbf{v}_{<i})$ is in $O(DH)$.

Computing the softmax nonlinearity of each $p(v_i = w|\mathbf{v}_{<i})$ in Equation 3.9 requires time linear in V, which we would like to avoid. Fortunately, unlike in the Replicated Softmax, we are not tied to the use of a large softmax nonlinearity to model probabilities over words. In the literature on neural probabilistic language models, the large softmax over words is often replaced by a probabilistic tree model in which each path from the root to a leaf corresponds to a word [40, 39]. The probabilities of each left/right transitions in the tree are modeled by a set of binary logistic regressors and the probability of a given word is then obtained by multiplying the probabilities of each left/right choice of the associated tree path.

Specifically, let $\mathbf{l}(v_i)$ be the sequence of tree nodes on the path from the root to the word v_i and let $\boldsymbol{\pi}(v_i)$ be the sequence of binary left/right choices for each of

those nodes (e.g. $l(v_i)_1$ will always be the root of the tree and $\pi(v_i)_1$ will be 0 if the word leaf node is in its left subtree or 1 otherwise). Let matrix $\mathbf{V}$ now be the matrix containing the logistic regression weights $\mathbf{V}_{l(v_i)_m,:}$ of each tree node $n(v_i)_m$ as its rows and $b_{l(v_i)_m}$ be its bias. The probability $p(v_i = w|\mathbf{v}_{<i})$ is now computed from hidden layer $\mathbf{h}_i(\mathbf{v}_{<i})$ as follows:

$$p(v_i = w|\mathbf{v}_{<i}) = \prod_{m=1}^{|\boldsymbol{\pi}(v_i)|} p(\pi(v_i)_m = \pi(w)_m|\mathbf{v}_{<i}) \tag{3.11}$$

$$p(\pi(v_i)_m = 1|\mathbf{v}_{<i}) = \text{sigm}(b_{l(v_i)_m} + \mathbf{V}_{l(v_i)_m,:}\mathbf{h}_i(\mathbf{v}_{<i})) \tag{3.12}$$

The conditionals of Equation 3.11 let us compute $p(\mathbf{v}) = \prod_i p(v_i = 1|\mathbf{v}_{<i})$ for any document and the parameters $\{\mathbf{b}, \mathbf{c}, \mathbf{W}, \mathbf{V}\}$ can be learned by minimizing the negative data log-likelihood with stochastic gradient descent. Once the model is trained, it can be used to extract a representation from a new document $\mathbf{v}^*$ by computing the value of its hidden layer after observing all of its words, which we note $\mathbf{h}(\mathbf{v}^*) = \text{sigm}(\mathbf{c} + \sum_i W_{:,v_i^*})$.

For a full binary tree of all V words, computing Equation 3.11 will involve $O(\log(V))$ binary logistic regressions. In our experiments, we used a randomly generated full binary tree with V leaves, each assigned to a unique word of the vocabulary. An even better option would be to derive the tree using Hoffman coding, which would reduce even more the average path lengths.

Since the computation of each logistic regression is in $O(H)$ and there are D words in a document, the complexity of computing all $p(v_i = w|\mathbf{v}_{<i})$ given the hidden layers is in $O(\log(V)DH)$. The total complexity of computing $p(\mathbf{v})$ and updating the parameters under the model is therefore $O(\log(V)DH + DH)$. When compared to the complexity $O(KVH + DH)$ of Replicated Softmax, this is quite competitive[1]. Indeed, Salakhutdinov and Hinton [43] suggest gradually increasing K from 1 to 25, which is larger than $\log(V)$ for a very large vocabulary of one million words. Also, the

1. In our experiments, a single training pass of DocNADE on the 20 Newgroups and RCV1-v2 data sets (see Section 3.6.1 for details) took on average 13 seconds and 726 seconds respectively. On the other hand, for $K = 1$ Gibbs sampling steps, our implementation of Replicated Softmax requires 28 seconds and 4945 seconds respectively. For $K = 5$, running time increases even more, to 60 seconds and 11000 seconds.

number of words in a document D will usually be much smaller than the vocabulary size V.

The final model, which we refer to as Document NADE (DocNADE), is illustrated in Figure 3.1. A pseudocode for computing $p(\mathbf{v})$ and the parameter learning gradients for a given document is provided in the supplementary material and our code is available here: http://www.dmi.usherb.ca/~larocheh/code/DocNADE.zip.

3.4.1 Training from bags of word counts

So far, we have assumed that the ordering of the words in the document was known. However, document data sets often take the form of set of word counts vectors in which the original word order, which is required by DocNADE to specify the sequence of conditionals $p(v_i|\mathbf{v}< i)$, has been lost.

One solution is to assume the following generative story: first, a *seed* document $\widetilde{\mathbf{v}}$ is sampled from DocNADE and, finally, a random permutation of its words is taken to produce the observed document $\mathbf{v}$. This translates into the following probability distribution:

$$p(\mathbf{v}) = \sum_{\widetilde{\mathbf{v}} \in \mathcal{V}(\mathbf{v})} p(\mathbf{v}|\widetilde{\mathbf{v}}) p(\widetilde{\mathbf{v}}) = \frac{1}{|\mathcal{V}(\mathbf{v})|} \sum_{\widetilde{\mathbf{v}} \in \mathcal{V}(\mathbf{v})} p(\widetilde{\mathbf{v}}) \tag{3.13}$$

where $p(\widetilde{\mathbf{v}})$ is modeled by DocNADE and $\mathcal{V}(\mathbf{v})$ is the set of all documents $\widetilde{\mathbf{v}}$ with the same word count vector $\mathbf{n}(\mathbf{v}) = \mathbf{n}(\widetilde{\mathbf{v}})$. This distribution is a mixture over all possible permutations that could have generated the original document $\mathbf{v}$. Now, we can use the fact that sampling uniformly from $\mathcal{V}(\mathbf{v})$ can be done solely on the basis of the word counts of $\mathbf{v}$, by randomly sampling words without replacement from those word counts. Therefore, we can train DocNADE on those generated word sequences, as if they were the original documents from which the word counts were extracted. While this is only an approximation of true maximum likelihood learning on the original documents, we've found it to work well in practice.

This approach of training DocNADE can be understood as learning a model that is good at predicting which new words should be inserted in a document at any position, while maintaining its general semantics. The model is therefore learning not

to insert "intruder" words, i.e. words that do not belong with the others. After training, a document's learned representation $\mathbf{h}(\mathbf{v})$ should contain valuable information to identify intruder words for this document. It's interesting to note that the detection of such intruder words has been used previously as a task in user studies to evaluate the quality of the topics learned by LDA, though at the level of single topics and not whole documents [11].

3.5 Related Work

We mentioned that the Replicated Softmax models the distribution over words as a product of topic-dependent factors. The Sparse Additive Generative Model (SAGE) [17] is also based on topic-dependent factors, as well as a background factor. The distribution of a word is the renormalized product of its topic factor and the background factor. Unfortunately, much like the Replicated Softmax, training in SAGE scales linearly with the vocabulary size, instead of logarithmically as in DocNADE. Recent work has also been able to improve the complexity of RBM training on word observations. However, for the specific case of the Replicated Softmax, the proposed method does not allow to remove the linear dependence on V of the complexity [13].

There has been fairly little work on using neural networks to learn generative topic models of documents. Glorot et al. [20], Dauphin et al. [15] have trained neural network autoencoders on documents in their binary bag of words representation, but such neural networks are not generative models of documents. One potential advantage of having a proper generative model under which $p(\mathbf{v})$ can be computed exactly is it becomes possible to do Bayesian learning of the parameters, even on a large scale, using recent online Bayesian inference approaches [53, 2].

3.6 Experiments

We present two quantitative comparison of DocNADE with the Replicated Softmax. The first compares the performance of DocNADE as a generative model, while the later evaluates whether DocNADE hidden layer can be used as a meaningful representation for documents. Following Salakhutdinov and Hinton [43], we use a

hidden layer size of $H = 50$ in all experiments. A validation set is always set aside to perform model selection of other hyper-parameters, such as the learning rate and the number of learning passes over the training set (based on early stopping). We also tested the use of a hidden layer hyperbolic tangent nonlinearity $\tanh(x) = (\exp(x) - \exp(-x))/(\exp(x) + \exp(-x))$ instead of the sigmoid and always used the best option based on the validation set performance. We end this section with a qualitative inspection of the implicit word representation and topic-features learned by DocNADE.

3.6.1 Generative Model Evaluation

We first evaluated DocNADE's performance as a generative model of documents. We performed our evaluation on the 20 Newsgroups and the Reuters Corpus Volume I (RCV1-v2) data sets and we followed the same evaluation as in Salakhutdinov and Hinton [43]: word counts were replaced by $\log(1+n_i)$ rounded to the closest integer and a subset of 50 test documents (2193 words for 20 Newsgroups, 4716 words for RCV1-v2) were used to estimate the test perplexity per word $\exp(-\frac{1}{N}\sum_t \frac{1}{|\mathbf{v}^t|}\log p(\mathbf{v}^t))$. The vocabulary size for 20 Newsgroups was 2000 and 10 000 for RCV1-v2.

We used the version of DocNADE that trains from document word counts. To approximate the corresponding distribution $p(\mathbf{v})$ of Equation 3.13, we sample a single permuted word sequence $\widetilde{\mathbf{v}}$ from the word counts. This might seem like a crude approximation, but, as we'll see, the value of $p(\widetilde{\mathbf{v}})$ tends not to vary a lot across different random permutations of the words.

Instead of minimizing the average document negative log-likelihood $-\frac{1}{N}\sum_t \log p(\mathbf{v}^t)$, we also considered minimizing a version normalized by each document's size $-\frac{1}{N}\sum_t \frac{1}{|\mathbf{v}^t|}\log p(\mathbf{v}^t)$, though the difference in performance between both ended up not being large. For 20 newsgroups, the model with the best perplexity on the validation set used a learning rate of 0.001, sigmoid hidden activation and optimized the average document negative log-likelihood (non-normalized). For RCV1-v2, a learning rate of 0.1, with sigmoid hidden activation and optimization of the objective normalized by each document's size performed best.

The results are reported in Table 3.1. A comparison is made with LDA using 50

Data Set	LDA (50)	LDA (200)	Replicated Softmax (50)	DocNADE (50)	DocNADE St. Dev
20 Newsgroups	1091	1058	953	**896**	6.9
RCV1-v2	1437	1142	988	**742**	4.5

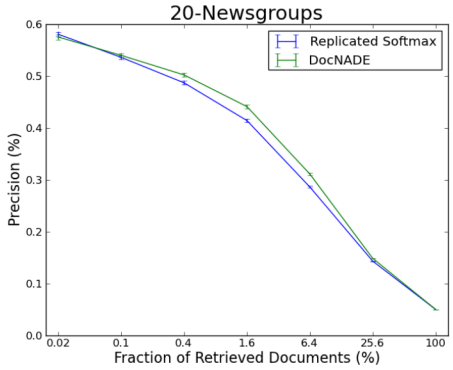
Table 3.1 – Test perplexity per word for LDA with 50 and 200 latent topics, Replicated Softmax with 50 topics and DocNADE with 50 topics. The results for LDA and Replicated Softmax were taken from Salakhutdinov and Hinton [43].

or 200 topics and the Replicated Softmax with 50 topics. The results for LDA and Replicated Softmax were taken from Salakhutdinov and Hinton [43]. We see that DocNADE achieves lower perplexity than both models. On RCV1-v2, DocNADE reaches a perplexity that is almost half that of LDA with 50 topics. We also provide the standard deviation of the perplexity obtained by repeating 100 times the calculation of the perplexity on the test set using different permuted word sequences $\widetilde{\mathbf{v}}$. We see that it is fairly small, which confirms that the value of $p(\widetilde{\mathbf{v}})$ does not vary a lot across different permutations. This is consistent with the observation made by Larochelle and Murray [33] that results are stable with respect to the choice of ordering for the conditionals $p(v_i|\mathbf{v}_{<i})$.

3.6.2 Document Retrieval Evaluation

We also evaluated the quality of the document representation $\mathbf{h}(\mathbf{v})$ learned by DocNADE in an information retrieval task using the 20 Newsgroups data set and its label information. In this context, all test documents were each used as queries and compared to a fraction of the closest documents in the original training set. Similarity between documents is computed using the cosine angle between document representations. We then compute the average number of retrieved training documents sharing the same label as the query (precision), and so for different fractions of retrieved documents.

For learning, we set aside 1000 documents for validation. For model selection, we used the validation set as the query set and used the average precision at 0.02% retrieved documents as the performance measure. We used only the training objective normalized by the document size and set the maximum number of training passes to 973 (approximately 10 million parameter updates). The best learning rate was



Hidden unit topics			
jesus	shuttle	season	encryption
atheism	orbit	players	escrow
christianity	lunar	nhl	pgp
christ	spacecraft	league	crypto
athos	nasa	braves	nsa
atheists	space	playoffs	rutgers
bible	launch	rangers	clipper
christians	saturn	hockey	secure
sin	billion	pitching	encrypted
atheist	satellite	team	keys

Figure 3.2 – **(Left)** Information retrieval task results, on 20 Newsgroups data set. The error bars correspond to the standard errors. **(Right)** Illustration of some topics learned by DocNADE. A topic i is visualized by picking the 10 words w with strongest connection W_{iw}.

0.01, with tanh hidden activation. Notice that the labels are not used during training. Since Salakhutdinov and Hinton [43] showed that it strictly outperforms LDA on this problem, we only compare to the Replicated Softmax. We performed stochastic gradient descent based on the contrastive divergence approximation during 973 training passes, and so for different learning rates. As recommended in Salakhutdinov and Hinton [43], we gradually increased the number of Gibbs sampling steps K from 1 to 25, but also tried increasing it only to 5 or maintaining it to $K = 1$. Optionally, we also used mean-field inference for the first few training passes. The best combination of these choices was selected based on validation performance.

The final results are presented in Figure 3.2. We see that DocNADE compares favorably with the Replicated Softmax. DocNADE is never outperformed by the Replicated Softmax and outperforms it for the intermediate retrieval fractions.

3.6.3 Qualitative Inspection of Learned Representations

Since topic models are often used for the exploratory analysis of unlabeled text, we looked at whether meaningful semantics were captured by DocNADE. First, to inspect the nature of topics modeled by the hidden units, we looked at the words with strongest positive connections to that hidden unit, i.e. the words w that have the largest values of $W_{i,w}$ for the i^{th} hidden unit. Figure 3.2 shows four topics extracted

Table 3.2 – The five nearest neighbors in the word representation space learned by DocNADE.

weapons	**medical**	**companies**	**define**	**israel**	**book**	**windows**
weapon	treatment	demand	defined	israeli	reading	dos
shooting	medecine	commercial	definition	israelis	read	microsoft
firearms	patients	agency	refer	arab	books	version
assault	process	company	make	palestinian	relevent	ms
armed	studies	credit	examples	arabs	collection	pc

this way and that could be understood as topics about religion, space, sports and security, which are label (sub)categories in 20 Newsgroups. We can also extract word representations, by using the columns $W_{:,w}$ as the vector representation of each word w. Table 3.2 shows the five nearest neighbors of some selected words in this space, confirming that the word representations are meaningful. In the supplementary material, we also provide 2D visualizations of these representations based on t-SNE [51], for 20 Newsgroups and RCV1-v2.

3.7 Conclusion

We have proposed DocNADE, an unsupervised neural network topic model of documents and have shown that it is a competitive model both as a generative model and as a document representation learning algorithm. Its training has the advantageous property of scaling sublinearly with the vocabulary size. Since the early work on topic modeling, research on the subject has progressed by developing Bayesian algorithms for topic modeling, by exploiting labeled data and by incorporating more structure within the latent topic representation. We feel like this is a plausible and most natural course to follow for future research.

Acknowledgment

We thank Ruslan Salakhutdinov for providing us with the data sets used in the experiments. This work was supported by NSERC and Google.

Chapitre 4

Une approche à base d'autoencodeur pour apprendre une représentation bilingue des mots

Résumé

L'apprentissage multilangue permet d'utiliser les données d'entraînement d'une langue en particulier pour construire des modèles appliqués à une autre langue. De nombreuses approches en apprentissage bilingue exigent que les phrases de corpus parallèles soient alignées au niveau des mots. Dans ce travail, nous explorons l'utilisation de méthodes basées sur les autoencodeurs pour apprendre des représentations vectorielles bilingues de mots. Ces représentations doivent rester cohérentes entre les deux langues, sans nécessiter d'alignement au niveau des mots. Nous montrons qu'en apprenant simplement à reconstruire les représentations sac-de-mots des phrases alignées, nous pouvons apprendre des représentations de haute qualité sans s'occuper de l'alignement des mots. Nous avons étudié les résultats de notre approche de façon empirique sur le problème de classification de textes multilangues, où un classificateur formé sur une langue donnée (*e.g.*, anglais)

doit apprendre à généraliser pour une autre langue (*e.g.*, allemand). À partir d'expériences sur trois paires de langues différentes, nous montrons que notre approche permet d'atteindre l'état de l'art au niveau des performances, dépassant celles d'une méthode qui exploite l'alignement des mots et d'une approche puissante en traduction automatique.

Commentaires

L'article a été publié dans la conférence NIPS (*Neural Information Processing Systems*) en 2014 et fait aussi partie du mémoire du coauteur Sarath Chandar [10]. Mes contributions pour cet article sont les suivantes:

— Développer l'autoencodeur bilingue avec arbre binaire en sortie à l'aide de la librairie Theano. L'apprentissage de ce modèle se fait en un temps compétitif et permet d'obtenir des représentations de documents à partir des représentations bilingues des mots.

— Évaluation de la qualité des représentations bilingues des mots apprises par le modèle. La classification multilangue de documents est la tâche qui a été utilisée pour faire cette évaluation sur trois paires de langues différentes.

An Autoencoder Approach to Learning Bilingual Word Representations

Sarath Chandar A P[1] *, **Stanislas Lauly**[2] *, **Hugo Larochelle**[2],
Mitesh M Khapra[3], **Balaraman Ravindran**[1], **Vikas Raykar**[3], **Amrita Saha**[3]
[1]Indian Institute of Technology Madras, [2]Université de Sherbrooke,
[3]IBM Research India
apsarathchandar@gmail.com,
{stanislas.lauly,hugo.larochelle}@usherbrooke.ca,
{mikhapra,viraykar,amrsaha4}@in.ibm.com, ravi@cse.iitm.ac.in
* Both authors contributed equally

Keywords: Autoencoder, Neural Networks, NLP, Document Classification, Cross Language Classification

Abstract

Cross-language learning allows one to use training data from one language to build models for a different language. Many approaches to bilingual learning require that we have word-level alignment of sentences from parallel corpora. In this work we explore the use of autoencoder-based methods for cross-language learning of vectorial word representations that are coherent between two languages, while not relying on word-level alignments. We show that by simply learning to reconstruct the bag-of-words representations of aligned sentences, within and between languages, we can in fact learn high-quality representations and do without word alignments. We empirically investigate the success of our approach on the problem of cross-language text classification, where a classifier trained on a given language (*e.g.*, English) must learn to generalize to a different language (*e.g.*, German). In experiments on 3 language pairs, we show that our approach achieves state-of-the-art performance, outperforming a method exploiting word alignments and a strong machine translation baseline.

4.1 Introduction

The accuracy of Natural Language Processing (NLP) tools for a given language depend heavily on the availability of annotated resources in that language. For example, high quality POS taggers [48], parsers [45], sentiment analyzers [34] are readily available for English. However, this is not the case for many other languages such as Hindi, Marathi, Bodo, Farsi, and Urdu, for which annotated data is scarce. This situation was acceptable in the past when only a few languages dominated the digital content available online and elsewhere. However, the ever increasing number of languages on the web today has made it important to accurately process natural language data in such resource-deprived languages also. An obvious solution to this problem is to improve the annotated inventory of these languages, but the cost, time and effort required act as a natural deterrent to this.

Another option is to exploit the unlabeled data available in a language. In this context, vectorial text representations have proven useful for multiple NLP tasks [49, 12]. It has been shown that meaningful representations, capturing syntactic and semantic similarity, can be learned from unlabeled data. While the majority of previous work on vectorial text representations has concentrated on the monolingual case, there has also been considerable interest in learning word and document representations that are aligned across languages [16, 42, 55, 27, 57, 36, 18]. Such aligned representations allow the use of resources from a resource-fortunate language to develop NLP capabilities in a resource-deprived language.

One approach to cross-lingual exploitation of resources is to project parameters learned from the annotated data of one language to another language [54, 14, 35, 52, 41]. These approaches rely on a bilingual resource such as a Machine Translation (MT) system. Recent attempts at learning common bilingual representations [27, 57, 36] aim to eliminate the need of such an MT system. A common property of these approaches is that a *word-level alignment* of translated sentences is leveraged to derive a regularization term relating word embeddings across languages. Such methods not only eliminate the need for an MT system but also outperform MT based projection approaches.

In this paper, we experiment with methods that learn bilingual word represen-

tations *without word-to-word alignments* of bilingual corpora during training. Unlike previous approaches, we only require aligned sentences and do not rely on word-level alignments (e.g., extracted using GIZA++, as is usual), simplifying the learning procedure. To do so, we propose and investigate bilingual autoencoder models, that learn hidden encoder representations of paired bag-of-words sentences that are not only informative of the original bag-of-words but also predictive of the other language. Word representations can then easily be extracted from the encoder and used in the context of a supervised NLP task. Specifically, we demonstrate the quality of these representations for the task of cross-language document classification, where a labeled data set can be available in one language, but not in another one. As we'll see, our approach is able to reach state-of-the-art performance, outperforming a method exploiting word alignments and a strong machine translation baseline.

4.2 Autoencoder for Bags-of-Words

Let $\mathbf{x}$ be the bag-of-words representation of a sentence. Specifically, each x_i is a word index from a fixed vocabulary of V words. As this is a bag-of-words, the order of the words within $\mathbf{x}$ does not correspond to the word order in the original sentence. We wish to learn a D-dimensional vectorial representation of our words from a training set of sentence bags-of-words $\{\mathbf{x}^{(t)}\}_{t=1}^{T}$.

We propose to achieve this by using an autoencoder model that encodes an input bag-of-words $\mathbf{x}$ with a sum of the representations (embeddings) of the words present in $\mathbf{x}$, followed by a non-linearity. Specifically, let matrix $\mathbf{W}$ be the $D \times V$ matrix whose columns are the vector representations for each word. The encoder's computation will involve summing over the columns of $\mathbf{W}$ for each word in the bag-of-word. We will denote this encoder function $\boldsymbol{\phi}(\mathbf{x})$. Then, using a decoder, the autoencoder will be trained to optimize a loss function that measures how predictive of the original bag-of-words is the encoder representation $\boldsymbol{\phi}(\mathbf{x})$.

There are different variations we can consider in the design of the encoder/decoder and the choice of loss function. One must be careful however, as certain choices can be inappropriate for training on word observations, which are intrinsically sparse and high-dimensional. In this paper, we explore and compare two different approaches,

described in the next two sub-sections.

4.2.1 Binary bag-of-words reconstruction training with merged bags-of-words

In the first approach, we start from the conventional autoencoder architecture, which minimizes a cross-entropy loss that compares a binary vector observation with a decoder reconstruction. We thus convert the bag-of-words $\mathbf{x}$ into a fixed-size but sparse binary vector $\mathbf{v}(\mathbf{x})$, which is such that $v(\mathbf{x})_{x_i}$ is 1 if word x_i is present in $\mathbf{x}$ and otherwise 0.

From this representation, we obtain an encoder representation by multiplying $\mathbf{v}(\mathbf{x})$ with the word representation matrix $\mathbf{W}$

$$\mathbf{a}(\mathbf{x}) = \mathbf{c} + \mathbf{W}\mathbf{v}(\mathbf{x}), \quad \boldsymbol{\phi}(\mathbf{x}) = \mathbf{h}(\mathbf{a}(\mathbf{x})) \tag{4.1}$$

where $\mathbf{h}(\cdot)$ is an element-wise non-linearity such as the sigmoid or hyperbolic tangent, and $\mathbf{c}$ is a D-dimensional bias vector. Encoding thus involves summing the word representations of the words present at least once in the bag-of-words.

To produce a reconstruction, we parametrize the decoder using the following non-linear form:

$$\widehat{\mathbf{v}}(\mathbf{x}) = \text{sigm}(\mathbf{V}\boldsymbol{\phi}(\mathbf{x}) + \mathbf{b}) \tag{4.2}$$

where $\mathbf{V} = \mathbf{W}^T$, $\mathbf{b}$ is the bias vector of the reconstruction layer and $\text{sigm}(a) = 1/(1 + \exp(-a))$ is the sigmoid non-linearity.

Then, the reconstruction is compared to the original binary bag-of-words as follows:

$$\ell(\mathbf{v}(\mathbf{x})) = -\sum_{i=1}^{V} v(\mathbf{x})_i \log(\widehat{v}(\mathbf{x})_i) + (1 - v(\mathbf{x})_i) \log(1 - \widehat{v}(\mathbf{x})_i) \ . \tag{4.3}$$

Training proceeds by optimizing the sum of reconstruction cross-entropies across the training set, *e.g.*, using stochastic or mini-batch gradient descent.

Note that, since the binary bags-of-words are very high-dimensional (the dimensionality corresponds to the size of the vocabulary, which is typically large), the above training procedure which aims at reconstructing the complete binary bag-of-word,

will be slow. Since we will later be training on millions of sentences, training on each individual sentence bag-of-words will be expensive.

Thus, we propose a simple trick, which exploits the bag-of-words structure of the input. Assuming we are performing mini-batch training (where a mini-batch contains a list of the bags-of-words of adjacent sentences), we simply propose to merge the bags-of-words of the mini-batch into a single bag-of-words and perform an update based on that merged bag-of-words. The resulting effect is that each update is as efficient as in stochastic gradient descent, but the number of updates per training epoch is divided by the mini-batch size . As we'll see in the experimental section, this trick produces good word representations, while sufficiently reducing training time. We note that, additionally, we could have used the stochastic approach proposed by Dauphin et al. [15] for reconstructing binary bag-of-words representations of documents, to further improve the efficiency of training. They use importance sampling to avoid reconstructing the whole V-dimensional input vector.

4.2.2 Tree-based decoder training

The previous autoencoder architecture worked with a binary vectorial representation of the input bag-of-words. In the second autoencoder architecture we investigate, we consider an architecture that instead works with the bag (unordered list) representation more directly.

First, the encoder representation will now involve a sum of the representation of all words, reflecting the relative frequency of each word:

$$\mathbf{a}(\mathbf{x}) = \mathbf{c} + \sum_{i=1}^{|\mathbf{x}|} \mathbf{W}_{\cdot,x_i}, \quad \boldsymbol{\phi}(\mathbf{x}) = \mathbf{h}\left(\mathbf{a}(\mathbf{x})\right) \ . \tag{4.4}$$

Moreover, decoder training will assume that, from the decoder's output, we can obtain a probability distribution $p(\widehat{x}|\boldsymbol{\phi}(\mathbf{x}))$ over any word $\widehat{x}$ observed at the reconstruction output layer. Then, we can treat the input bag-of-words as a $|\mathbf{x}|$-trials multinomial sample from that distribution and use as the reconstruction loss its negative log-likelihood:

$$\ell(\mathbf{x}) = \sum_{i=1}^{V} -\log p(\widehat{x} = x_i|\boldsymbol{\phi}(\mathbf{x})) \ . \tag{4.5}$$

We now must ensure that the decoder can compute $p(\widehat{x} = x_i|\boldsymbol{\phi}(\mathbf{x}))$ efficiently from $\boldsymbol{\phi}(\mathbf{x})$. Specifically, we'd like to avoid a procedure scaling linearly with the vocabulary size V, since V will be very large in practice. This precludes any procedure that would compute the numerator of $p(\widehat{x} = w|\boldsymbol{\phi}(\mathbf{x}))$ for each possible word w separately and normalize it so it sums to one.

We instead opt for an approach borrowed from the work on neural network language models [40, 39]. Specifically, we use a probabilistic tree decomposition of $p(\widehat{x} = x_i|\boldsymbol{\phi}(\mathbf{x}))$. Let's assume each word has been placed at the leaf of a binary tree. We can then treat the sampling of a word as a stochastic path from the root of the tree to one of the leaves.

We denote as $\mathbf{l}(x)$ the sequence of internal nodes in the path from the root to a given word x, with $l(x)_1$ always corresponding to the root. We will denote as $\boldsymbol{\pi}(x)$ the vector of associated left/right branching choices on that path, where $\pi(x)_k = 0$ means the path branches left at internal node $l(x)_k$ and otherwise branches right if $\pi(x)_k = 1$. Then, the probability $p(\widehat{x} = x|\boldsymbol{\phi}(\mathbf{x}))$ of reconstructing a certain word x observed in the bag-of-words is computed as

$$p(\widehat{x}|\boldsymbol{\phi}(\mathbf{x})) = \prod_{k=1}^{|\boldsymbol{\pi}(\widehat{x})|} p(\pi(\widehat{x})_k|\boldsymbol{\phi}(\mathbf{x})) \tag{4.6}$$

where $p(\pi(\widehat{x})_k|\boldsymbol{\phi}(\mathbf{x}))$ is output by the decoder. By using a full binary tree of words, the number of different decoder outputs required to compute $p(\widehat{x}|\boldsymbol{\phi}(\mathbf{x}))$ will be logarithmic in the vocabulary size V. Since there are $|\mathbf{x}|$ words in the bag-of-words, at most $O(|\mathbf{x}| \log V)$ outputs are required from the decoder. This is of course a worst case scenario, since words will share internal nodes between their paths, for which the decoder output can be computed just once. As for organizing words into a tree, as in Larochelle and Lauly [32] we used a random assignment of words to the leaves of the full binary tree, which we have found to work well in practice.

Finally, we need to choose a parametrized form for the decoder. We choose the following form:

$$p(\pi(\widehat{x})_k = 1|\boldsymbol{\phi}(\mathbf{x})) = \text{sigm}(b_{l(\hat{x}_i)_k} + \mathbf{V}_{l(\hat{x}_i)_k,\cdot}\boldsymbol{\phi}(\mathbf{x})) \tag{4.7}$$

where $\mathbf{b}$ is a (V-1)-dimensional bias vector and $\mathbf{V}$ is a $(V-1) \times D$ matrix. Each left-/right branching probability is thus modeled with a logistic regression model applied on the encoder representation of the input bag-of-words $\boldsymbol{\phi}(\mathbf{x})$.

4.3 Bilingual autoencoders

Let's now assume that for each sentence bag-of-words $\mathbf{x}$ in some source language $\mathcal{X}$, we have an associated bag-of-words $\mathbf{y}$ for this sentence translated in some target language $\mathcal{Y}$ by a human expert.

Assuming we have a training set of such $(\mathbf{x}, \mathbf{y})$ pairs, we'd like to use it to learn representations in both languages that are aligned, such that pairs of translated words have similar representations.

To achieve this, we propose to augment the regular autoencoder proposed in Section 4.2 so that, from the sentence representation in a given language, a reconstruction can be attempted of the original sentence in the other language. Specifically, we now define language specific word representation matrices $\mathbf{W}^x$ and $\mathbf{W}^y$, corresponding to the languages of the words in $\mathbf{x}$ and $\mathbf{y}$ respectively. Let $V^{\mathcal{X}}$ and $V^{\mathcal{Y}}$ also be the number of words in the vocabulary of both languages, which can be different. The word representations however are of the same size D in both languages. For the binary reconstruction autoencoder, the bag-of-words representations extracted by the encoder become

$$\boldsymbol{\phi}(\mathbf{x}) = \mathbf{h}\left(\mathbf{c} + \mathbf{W}^{\mathcal{X}}\mathbf{v}(\mathbf{x})\right) \quad , \quad \boldsymbol{\phi}(\mathbf{y}) = \mathbf{h}\left(\mathbf{c} + \mathbf{W}^{\mathcal{Y}}\mathbf{v}(\mathbf{y})\right)$$

and are similarly extended for the tree-based autoencoder. Notice that we share the bias $\mathbf{c}$ before the non-linearity across encoders, to encourage the encoders in both languages to produce representations on the same scale.

From the sentence in either languages, we want to be able to perform a reconstruction of the original sentence in both the languages. In particular, given a representation in any language, we'd like a decoder that can perform a reconstruction in language $\mathcal{X}$ and another decoder that can reconstruct in language $\mathcal{Y}$. Again, we use decoders of the form proposed in either Section 4.2.1 or 4.2.2 (see Figure 4.1), but let

the decoders of each language have their own parameters $(\mathbf{b}^{\mathcal{X}}, \mathbf{V}^{\mathcal{X}})$ and $(\mathbf{b}^{\mathcal{Y}}, \mathbf{V}^{\mathcal{Y}})$.

This encoder/decoder decomposition structure allows us to learn a mapping within each language and across the languages. Specifically, for a given pair $(\mathbf{x}, \mathbf{y})$, we can train the model to (1) construct $\mathbf{y}$ from $\mathbf{x}$ (loss $\ell(\mathbf{x}, \mathbf{y})$), (2) construct $\mathbf{x}$ from $\mathbf{y}$ (loss $\ell(\mathbf{y}, \mathbf{x})$), (3) reconstruct $\mathbf{x}$ from itself (loss $\ell(\mathbf{x})$) and (4) reconstruct $\mathbf{y}$ from itself (loss $\ell(\mathbf{y})$). We follow this approach in our experiments and optimize the sum of the corresponding 4 losses during training.

4.3.1 Joint reconstruction and cross-lingual correlation

We also considered incorporating two additional terms to the loss function, in an attempt to favour even more meaningful bilingual representations:

$$\ell(\mathbf{x}, \mathbf{y}) + \ell(\mathbf{y}, \mathbf{x}) + \ell(\mathbf{x}) + \ell(\mathbf{y}) + \beta\ell([\mathbf{x}, \mathbf{y}], [\mathbf{x}, \mathbf{y}]) - \lambda \cdot cor(\mathbf{a}(\mathbf{x}), \mathbf{a}(\mathbf{y})) \tag{4.8}$$

The term $\ell([\mathbf{x}, \mathbf{y}], [\mathbf{x}, \mathbf{y}])$ is simply a joint reconstruction term, where both languages are simultanouesly presented as input and reconstructed. The second term $cor(\mathbf{a}(\mathbf{x}), \mathbf{a}(\mathbf{y}))$ encourages correlation between the representation of each language. It is the sum of the scalar correlations between each pair $a(\mathbf{x})_k, a(\mathbf{y})_k$, across all dimensions k of the vectors $\mathbf{a}(\mathbf{x}), \mathbf{a}(\mathbf{y})$ [1]. To obtain a stochastic estimate of the correlation, during training, small mini-batches are used.

4.3.2 Document representations

Once we learn the language specific word representation matrices $\mathbf{W}^x$ and $\mathbf{W}^y$ as described above, we can use them to construct document representations, by using their columns as word vector representations. Given a document $\mathbf{d}$ written in language $\mathcal{Z} \in \{\mathcal{X}, \mathcal{Y}\}$ and containing m words, $z_1, z_2, \ldots, z_m$, we represent it as the tf-idf weighted sum of its words' representations $\psi(\mathbf{d}) = \sum_{i=1}^{m} tf\text{-}idf(z_i) \cdot \mathbf{W}^{\mathcal{Z}}_{\cdot,z_i}$. We use the document representations thus obtained to train our document classifiers, in the cross-lingual document classification task described in Section 4.5.

1. While we could have applied the correlation term on $\boldsymbol{\phi}(\mathbf{x}), \boldsymbol{\phi}(\mathbf{y})$ directly, applying it to the pre-activation function vectors was found to be more numerically stable.

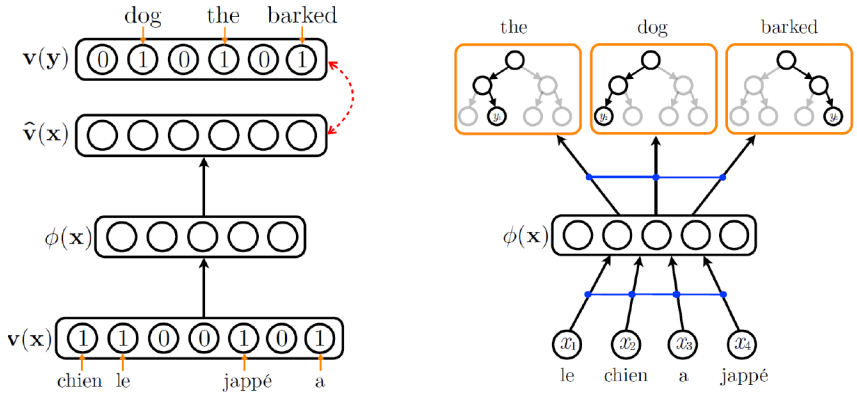


Figure 4.1 – **Left:** Bilingual autoencoder based on the binary reconstruction error. **Right:** Tree-based bilingual autoencoder. In this example, they both reconstruct the bag-of-words for the English sentence "*the dog barked*" from its French translation "*le chien a jappé*".

4.4 Related Work

Recent work that has considered the problem of learning bilingual representations of words usually has relied on word-level alignments. Klementiev et al. [27] propose to train simultaneously two neural network languages models, along with a regularization term that encourages pairs of frequently aligned words to have similar word embeddings. Thus, the use of this regularization term requires to first obtain word-level alignments from parallel corpora. Zou et al. [57] use a similar approach, with a different form for the regularizer and neural network language models as in [12]. In our work, we specifically investigate whether a method that does not rely on word-level alignments can learn comparably useful multilingual embeddings in the context of document classification.

Looking more generally at neural networks that learn multilingual representations of words or phrases, we mention the work of Gao et al. [19] which showed that a useful linear mapping between *separately trained* monolingual skip-gram language models could be learned. They too however rely on the specification of pairs of words in the two languages to align. Mikolov et al. [36] also propose a method for training a neural network to learn useful representations of phrases, in the context of a phrase-based translation model. In this case, phrase-level alignments (usually extracted from word-

level alignments) are required. Recently, Hermann and Blunsom [22], [21] proposed neural network architectures and a margin-based training objective that, as in this work, does not rely on word alignments. We will briefly discuss this work in the experiments section.

4.5 Experiments

The techniques proposed in this paper enable us to learn bilingual embeddings which capture cross-language similarity between words. We propose to evaluate the quality of these embeddings by using them for the task of cross-language document classification. We followed closely the setup used by Klementiev et al. [27] and compare with their method, for which word representations are publicly available [2]. The set up is as follows. A labeled data set of documents in some language $\mathcal{X}$ is available to train a classifier, however we are interested in classifying documents in a different language $\mathcal{Y}$ at test time. To achieve this, we leverage some bilingual corpora, which is not labeled with any document-level categories. This bilingual corpora is used to learn document representations that are coherent between languages $\mathcal{X}$ and $\mathcal{Y}$. The hope is thus that we can successfully apply the classifier trained on document representations for language $\mathcal{X}$ directly to the document representations for language $\mathcal{Y}$. Following this setup, we performed experiments on 3 data sets of language pairs: English/German (EN/DE), English/French (EN/FR) and English/Spanish (EN/ES).

4.5.1 Data

For learning the bilingual embeddings, we used sections of the Europarl corpus [28] which contains roughly 2 million parallel sentences. We considered 3 language pairs. We used the same pre-processing as used by Klementiev et al. [27]. We tokenized the sentences using NLTK [7], removed punctuations and lowercased all words. We did not remove stopwords.

As for the labeled document classification data sets, they were extracted from sections of the Reuters RCV1/RCV2 corpora, again for the 3 pairs considered in

2. http://people.mmci.uni-saarland.de/~aklement/data/distrib/

our experiments. Following Klementiev et al. [27], we consider only documents which were assigned exactly one of the 4 top level categories in the topic hierarchy (CCAT, ECAT, GCAT and MCAT). These documents are also pre-processed using a similar procedure as that used for the Europarl corpus. We used the same vocabularies as those used by Klementiev et al. [27] (varying in size between $35,000$ and $50,000$).

For each pair of languages, our overall procedure for cross-language classification can be summarized as follows:

Train representation: Train bilingual word representations $\mathbf{W}^x$ and $\mathbf{W}^y$ on sentence pairs extracted from Europarl for languages $\mathcal{X}$ and $\mathcal{Y}$. Optionally, we also use the monolingual documents from RCV1/RCV2 to reinforce the monolingual embeddings (this choice is cross-validated). These non-parallel documents can be used through the losses $\ell(\mathbf{x})$ and $\ell(\mathbf{y})$ (*i.e.* by reconstructing $\mathbf{x}$ from $\mathbf{x}$ or $\mathbf{y}$ from $\mathbf{y}$). Note that Klementiev et al. [27] also used this data when training word representations.

Train classifier: Train document classifier on the Reuters training set for language $\mathcal{X}$, where documents are represented using the word representations $\mathbf{W}^x$ (see Section 4.3.2). As in Klementiev et al. [27] we used an averaged perceptron trained for 10 epochs, for all the experiments.

Test-time classification: Use the classifier trained in the previous step on the Reuters test set for language $\mathcal{Y}$, using the word representations $\mathbf{W}^y$ to represent the documents.

We trained the following autoencoders [3]: **BAE-cr** which uses reconstruction error based decoder training (see Section 4.2.1) and **BAE-tr** which uses tree-based decoder training (see Section 4.2.2).

Models were trained for up to 20 epochs using the same data as described earlier. BAE-cr used mini-batch (of size 20) stochastic gradient descent, while BAE-tr used regular stochastic gradient. All results are for word embeddings of size $D = 40$, as in Klementiev et al. [27]. Further, to speed up the training for BAE-cr we merged each 5 adjacent sentence pairs into a single training instance, as described in Section 4.2.1. For all language pairs, the joint reconstruction β was fixed to 1 and the cross-lingual correlation factor λ to 4 for BAE-cr. For BAE-tr, none of these additional terms

3. Our word representations and code are available at http://www.sarathchandar.in/crl.html

were found to be particularly beneficial, so we set their weights to 0 for all tasks. The other hyperparameters were tuned to each task using a training/validation set split of 80% and 20% and using the performance on the validation set of an averaged perceptron trained on the smaller training set portion (notice that this corresponds to a monolingual classification experiment, since the general assumption is that no labeled data is available in the test set language).

4.5.2 Comparison of the performance of different models

We now present the cross language classification results obtained by using the embeddings produced by our two autoencoders. We also compare our models with the following approaches:

Klementiev et al.: This model uses word embeddings learned by a multitask neural network language model with a regularization term that encourages pairs of frequently aligned words to have similar word embeddings. From these embeddings, document representations are computed as described in Section 4.3.2.

MT: Here, test documents are translated to the language of the training documents using a standard phrase-based MT system, MOSES [4] which was trained using default parameters and a 5-gram language model on the Europarl corpus (same as the one used for inducing our bilingual embeddings).

Majority Class: Test documents are simply assigned the most frequent class in the training set.

For the EN/DE language pairs, we directly report the results from Klementiev et al. [27]. For the other pairs (not reported in Klementiev et al. [27]), we used the embeddings available online and performed the classification experiment ourselves. Similarly, we generated the MT baseline ourselves.

Table 4.1 summarizes the results. They were obtained using 1000 RCV training examples. We report results in both directions, *i.e.* language $\mathcal{X}$ to $\mathcal{Y}$ and vice versa. The best performing method is always either BAE-cr or BAE-tr, with BAE-cr having the best performance overall. In particular, BAE-cr often outperforms the approach of Klementiev et al. [27] by a large margin.

4. http://www.statmt.org/moses/

Table 4.1 – Cross-lingual classification accuracy for 3 language pairs, with 1000 labeled examples.

	EN → DE	DE → EN	EN → FR	FR → EN	EN → ES	ES → EN
BAE-tr	81.8	60.1	70.4	61.8	**59.4**	60.4
BAE-cr	**91.8**	**74.2**	**84.6**	**74.2**	49.0	**64.4**
Klementiev et al.	77.6	71.1	74.5	61.9	31.3	63.0
MT	68.1	67.4	76.3	71.1	52.0	58.4
Majority Class	46.8	46.8	22.5	25.0	15.3	22.2

We also mention the recent work of Hermann and Blunsom [22], who proposed two neural network architectures for learning word and document representations using sentence-aligned data only. Instead of an autoencoder paradigm, they propose a margin-based objective that aims to make the representation of aligned sentences closer than non-aligned sentences. While their trained embeddings are not publicly available, they report results for the EN/DE classification experiments, with representations of the same size as here ($D = 40$) and trained on 500K EN/DE sentence pairs. Their best model in that setting reaches accuracies of 83.7% and 71.4% respectively for the EN → DE and DE → EN tasks. One clear advantage of our model is that unlike their model, it can use additional monolingual data. Indeed, when we train BAE-cr with 500k EN/DE sentence pairs, plus monolingual RCV documents (which come at no additional cost), we get accuracies of 87.9% (EN → DE) and 76.7% (DE → EN), still improving on their best model. If we do not use the monolingual data, BAE-cr's performance is worse but still competitive at 86.1% for EN → DE and 68.8% for DE → EN. Finally, without constraining D to 40 (they use 128) and by using additional French data, the best results of Hermann and Blunsom [22] are 88.1% (EN → DE) and 79.1% (DE → EN), the later being, to our knowledge, the current state-of-the-art.

We also evaluate the effect of varying the amount of supervised training data for training the classifier. For brevity, we report only the results for the EN/DE pair, which are summarized in Figure 4.2. We observe that BAE-cr clearly outperforms the other models at almost all data sizes. More importantly, it performs remarkably well at very low data sizes (100), suggesting it learns very meaningful embeddings,

Table 4.2 – Example English words along with the closest words both in English (EN) and German (DE), using the Euclidean distance between the embeddings learned by BAE-cr.

Word	Lang	Nearest neighbors	Word	Lang	Nearest neighbors
january	EN	january, march, october	oil	EN	oil, supply, supplies, gas
	DE	januar, märz, oktober		DE	öl, boden, befindet, gerät
president	EN	president, i, mr, presidents	microsoft	EN	microsoft, cds, insider
	DE	präsident, präsidentin		DE	microsoft, cds, warner
said	EN	said, told, say, believe	market	EN	market, markets, single
	DE	gesagt, sagte, sehr, heute		DE	markt, marktes, märkte

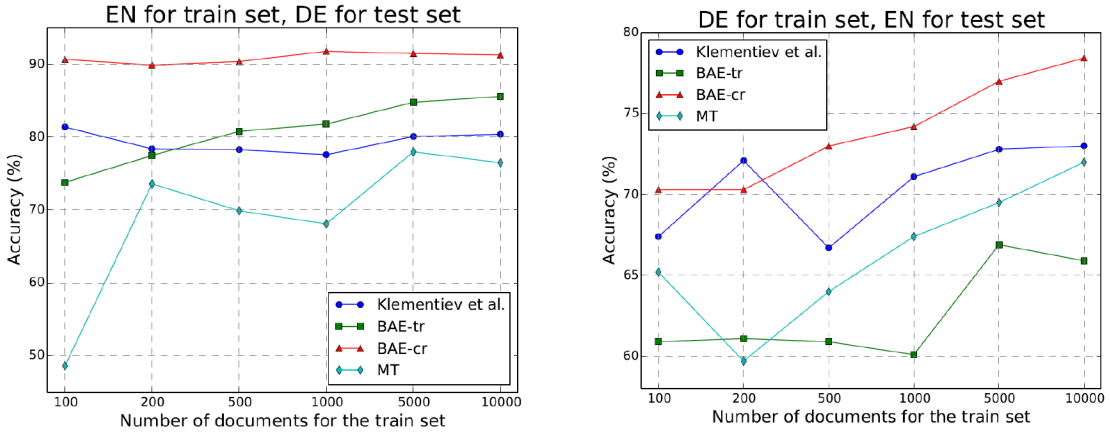


Figure 4.2 – Cross-lingual classification accuracy results, from EN → DE (**left**), and DE → EN (**right**).

though the method can still benefit from more labeled data (as in the DE → EN case).

Table 4.2 also illustrates the properties captured within and across languages, for the EN/DE pair[5]. For a few English words, the words with closest word representations (in Euclidean distance) are shown, for both English and German. We observe that words that form a translation pair are close, but also that close words within a language are syntactically/semantically similar as well.

The excellent performance of BAE-cr suggests that merging several sentences into

5. See also the supplementary material (Annex A) for a t-SNE visualization of the word representations.

single bags-of-words can still yield good word embeddings. In other words, not only we do not need to rely on word-level alignments, but exact sentence-level alignment is also not essential to reach good performances. We experimented with the merging of 5, 25 and 50 adjacent sentences (see the supplementary material in Annex A). Generally speaking, these experiments also confirm that even coarser merges can sometimes not be detrimental. However, for certain language pairs, there can be an important decrease in performance. On the other hand, when comparing the performance of BAE-tr with the use of 5-sentences merges, no substantial impact is observed.

4.6 Conclusion and Future Work

We presented evidence that meaningful bilingual word representations could be learned without relying on word-level alignments or using fairly coarse sentence-level alignments. In particular, we showed that even though our model does not use word level alignments, it is able to reach state-of-the-art performance, even compared to a method that exploits word-level alignments. In addition, it also outperforms a strong machine translation baseline. For future work, we would like to investigate extensions of our bag-of-words bilingual autoencoder to bags-of-n-grams, where the model would also have to learn representations for short phrases. Such a model should be particularly useful in the context of a machine translation system. We would also like to explore the possibility of converting our bilingual model to a multilingual model which can learn common representations for multiple languages given different amounts of parallel data between these languages.

Acknowledgement

We would like to thank Alexander Klementiev and Ivan Titov for providing the code for the classifier and data indices. This work was supported in part by Google.

Chapitre 5

Réseau de neurones auto-régressif pour l'estimation de densité des documents

Résumé

Cet article journal est une extension du travail présenté plus tôt dans ce document (chapitre 3). Nous présentons une approche basée sur les réseaux neuronaux pour apprendre la distribution de documents textuels. Cette approche est inspirée par le modèle NADE (*Neural Autoregressive Distribution Estimator*), qui est un bon estimateur de distribution pour les vecteurs de grande dimension à valeurs discrètes. Dans cet article, nous montrons comment NADE peut être adapté au cas des données textuelles, tout en conservant du modèle NADE la propriété d'échantillonner ou de calculer la probabilité des observations de façon exacte et efficace. L'approche peut également être utilisée pour apprendre des représentations profondes de documents qui concurrencent celles apprises par les approches alternatives en modélisation de sujets. Enfin, nous décrivons comment cette approche peut être combinée avec un réseau de neurones N-gram typique et ainsi améliorer significativement ses performances, lui faisant apprendre une représentation qui sera sensible au contexte général du document .

Commentaires

L'article a été soumis au journal JMLR (*Journal of Machine Learning Research*) en janvier 2016. Mes contributions pour cet article sont les suivantes:

— Participer au développement du modèle DocNADE (*Document Neural Autoregressive Distribution Estimation*). DocNADE est un modèle de sujets basé sur l'approche des réseaux de neurones, modèle qui permet de calculer efficacement une distribution valide des mots pour les documents.

— Participer au développement de DeepDocNADE (*Deep Document Neural Autoregressive Distribution Estimation*), une version profonde du modèle DocNADE.

— Développer le modèle DocNADE-LM (*Document Neural Autoregressive Distribution Estimation for Language Modeling*), une version de DocNADE qui fait de la modélisation de la langue en prenant en compte l'information sur l'ordre des mots.

— Évaluation des performances du modèle DocNADE-LM comme modèle génératif de documents en calculant la perplexité sur un ensemble de données.

— Évaluation qualitative des représentations apprises par le modèle DocNADE-LM en observant les mots proches dans l'espace des représentations.

Document Neural Autoregressive Distribution Estimation

Stanislas Lauly
Département d'informatique
Université de Sherbrooke
stanislas.lauly@usherbrooke.ca

Yin Zheng
Hulu LLC.
Beijing, China
yin.zheng@hulu.com

Alexandre Allauzen
LIMSI-CNRS
Université Paris Sud
Orsay, France
allauzen@limsi.fr

Hugo Larochelle
Département d'informatique
Université de Sherbrooke
hugo.larochelle@usherbrooke.ca

Keywords: Autoencoder, Neural Networks, NLP, Deep Learning, Topic models, Language models, Autoregressive models

Abstract

We present an approach based on feed-forward neural networks for learning the distribution of textual documents. This approach is inspired by the Neural Autoregressive Distribution Estimator (NADE) model, which has been shown to be a good estimator of the distribution of discrete-valued high-dimensional vectors. In this paper, we present how NADE can successfully be adapted to the case of textual data, retaining from NADE the property

that sampling or computing the probability of observations can be done exactly and efficiently. The approach can also be used to learn deep representations of documents that are competitive to those learned by the alternative topic modeling approaches. Finally, we describe how the approach can be combined with a regular neural network N-gram model and substantially improve its performance, by making its learned representation sensitive to the larger, document-specific context.

5.1 Introduction

One of the most common problems addressed by machine learning is estimating the distribution $p(\mathbf{v})$ of multidimensional data from a set of examples $\{\mathbf{v}^{(t)}\}_{t=1}^{T}$. Indeed, good estimates for $p(\mathbf{v})$ implicitly requires modeling the dependencies between the variables in $\mathbf{v}$, which is required to extract meaningful representations of this data or make predictions about this data.

The biggest challenge one faces in distribution estimation is the well-known curse of dimensionality. In fact, this issue is particularly important in distribution estimation, even more so than in other machine learning problems. This is because a good distribution estimator effectively requires providing an accurate value for $p(\mathbf{v})$ for any value of $\mathbf{v}$ (i.e. not only for likely values of $\mathbf{v}$), with the number of possible values taken by $\mathbf{v}$ growing exponentially as the number of the dimensions of the input vector $\mathbf{v}$ increases.

One example of a model that has been successful at tackling the curse of dimensionality is the restricted Boltzmann machine (RBM) [24]. The RBM and other models derived from it (e.g. the Replicated Softmax of [43]) are frequently trained as models of the probability distribution of high-dimensional observations and then used as feature extractors. Unfortunately, one problem with these models is that for moderately large models, calculating their estimate of $p(\mathbf{v})$ is intractable. Indeed, this calculation requires computing the so-called partition function, which normalizes the model distribution. The consequences of this property of the RBM are that approximations must be taken to train it by maximum likelihood and its estimation of $p(\mathbf{v})$ cannot be entirely trusted.

In an attempt to tackle these issues of the RBM, the Neural Autoregressive Distribution Estimator (NADE) was introduced by [33]. NADE's parametrization is inspired by the RBM, but uses feed-forward neural networks and the framework of autoregression for modeling the probability distribution of binary variables in high-dimensional vectors. Importantly, computing the probability of an observation under NADE can be done exactly and efficiently.

In this paper, we describe a variety of ways to extend NADE to model data from text documents. We start by describing Document NADE (DocNADE), a single

hidden layer feed-forward neural network model for bag-of-words observations, i.e. orderless sets of words. This requires adapting NADE to vector observations $\mathbf{v}$, where each of element v_i represents a word and where the order of the dimensions is random. Each word is represented with a lower-dimensional, real-valued embedding vector, where similar words should have similar embeddings. This is in line with much of the recent work on using feed-forward neural network models to learn word vector embeddings [5, 38, 39, 36] to counteract the curse of dimensionality. However, in DocNADE, the word representations are trained to reflect the topics (i.e. semantics) of documents only, as opposed to their syntactical properties, due to the orderless nature of bags-of-words.

Then, we describe how to train deep versions of DocNADE. First described by [56] in the context of image modeling, here we empirically evaluate them for text documents and show that they are competitive to alternative topic models, both in terms of perplexity and document retrieval performances.

Finally, we present how the topic-level modeling ability of DocNADE can be used to obtain a useful representation of context (semantic information about the past) for language modeling. We empirically demonstrate that by learning a topical representation (representation based on subjects, topics) of previous sentences, we can improve the perplexity performance of an N-gram neural language model.

5.2 Document NADE (DocNADE)

DocNADE is derived from the Neural Autoregressive Distribution Estimation (NADE) that will be first described in this section.

5.2.1 Neural Autoregressive Distribution Estimation (NADE)

NADE, introduced in [33], is a tractable distribution estimator for modeling the distribution of high-dimensional vectors of binary variables. Let us consider a binary vector of D observations, $\mathbf{v} \in \{0,1\}^D$. We use the notation v_i to denotes the i-th component of $\mathbf{v}$ and $\mathbf{v}_{<i} \in \{0,1\}^{i-1}$ to contain the first $i-1$ components of $\mathbf{v}$. $\mathbf{v}_{<i}$ is

the sub-vector $[v_1, \ldots, v_{i-1}]^\top$:

$$\mathbf{v} = [\underbrace{v_1, \ \ldots \ , v_{i-1}}_{\mathbf{v}_{<i}}, v_i, \ \ldots \ , v_D]^T. \tag{5.1}$$

The NADE model estimates the probability of the vector $\mathbf{v}$ by applying the probability chain rule as follows:

$$p(\mathbf{v}) = \prod_{i=1}^{D} p(v_i|\mathbf{v}_{<i}). \tag{5.2}$$

The peculiarity of NADE lies in the neural architecture designed to estimate the conditional probabilities involved in Equation 5.2. To predict the component i, the model first computes its hidden layer of dimension H

$$\mathbf{h}_i(\mathbf{v}_{<i}) = \mathbf{g}\left(\mathbf{c} + \mathbf{W}_{:,<i} \cdot \mathbf{v}_{<i}\right), \tag{5.3}$$

leading to the following probability model:

$$p(v_i = 1|\mathbf{v}_{<i}) = \text{sigm}\left(b_i + \mathbf{V}_{i,:} \cdot \mathbf{h}_i(\mathbf{v}_{<i})\right). \tag{5.4}$$

In these two equations, $\text{sigm}(x) = 1/(1 + \exp(-x))$ denotes the sigmoid activation function while function $\mathbf{g}(\cdot)$ could be any activation function, though [33] also used the sigmoid function. $\mathbf{W} \in \mathbb{R}^{H \times D}$ and $\mathbf{V} \in \mathbb{R}^{D \times H}$ are the parameter matrices along with the associated bias terms $\mathbf{b} \in \mathbb{R}^D$ and $\mathbf{c} \in \mathbb{R}^H$. The notation $\mathbf{W}_{:,<i}$ represent a matrix made of the $i-1$ first columns of $\mathbf{W}$. The vector $\mathbf{V}_{i,:}$ is made from the i^{th} row of $\mathbf{V}$ and is composed of weights associated to a logistic classifier, meaning that each row of $\mathbf{V}$ represent a different logistic classifier.

Instead of a single projection of the input vector (like for a typical autoencoder, left part of figure 5.1), the NADE model relies on a set of separate hidden layers $\mathbf{h}_i(\mathbf{v}_{<i})$ that each represent the previous inputs in a latent space (right part of figure 5.1). The connections between input dimension v_i and each hidden layer $\mathbf{h}_i(\mathbf{v}_{<i})$ are tied

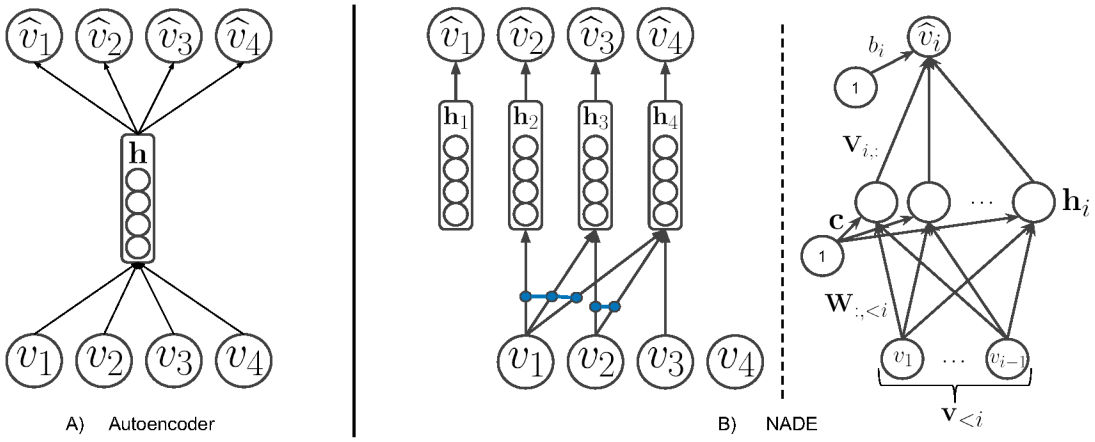


Figure 5.1 – **A)** Typical structure for an autoencoder. **B)** Illustration of NADE. Colored lines identify the connections that share parameters and $\hat{v}_i$ is a shorthand for the autoregressive conditional $p(v_i|\mathbf{v}_{<i})$. The right part show how one $p(v_i|\mathbf{v}_{<i})$ is computed. The observations v_i are binary.

as shown with the blue lines in figure 5.1, allowing the model to compute all the hidden layers for one input in $O(DH)$. The parameters $\{\mathbf{b}, \mathbf{c}, \mathbf{W}, \mathbf{V}\}$ are learned by minimizing the average negative log-likelihood using stochastic gradient descent.

5.2.2 From NADE to DocNADE

The Document NADE model (DocNADE) aims at learning meaningful representations of texts from an unlabeled collection of documents. Implemented as a feed-forward architecture, it extends NADE to provide an efficient and meaningful generative model of document bags-of-words. This model embeds, like NADE, a set of hidden layers. Their role is to capture salient statistical patterns in the co-occurrence of words within documents and can be considered as modeling hidden topics. Each neuron of the hidden layer can be associated to a hidden subject where its value would represent how much that subject is part of the text.

To represent a document, the vector $\mathbf{v}$ is now a sequence of arbitrary size D. Each element of $\mathbf{v}$ corresponds to a multinomial observation over a fixed vocabulary of size V:

$$\mathbf{v} = [v_1, v_2, \ \ldots\ , v_D]\,, \quad v_i \in \{1, 2, \ \ldots\ , V\}\,. \tag{5.5}$$

Therefore v_i represents the index in the vocabulary of the i-th word of the document. For now, we'll assume that an ordering for the words is given, but we will discuss the general case of orderless bags-of-words in Section 5.2.2.

The DocNADE model use a word representation matrix $\mathbf{W} \in \mathbb{R}^{H \times V}$, where each column $\mathbf{W}_{:,v_i}$ of the matrix is a vector representing one of the vocabulary words (see figure 5.2). This matrix makes the connection possible between the index v_i of a word and its embedding.

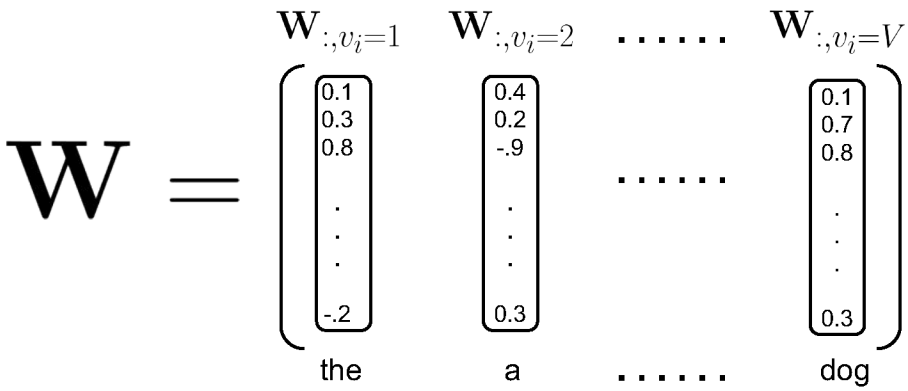


Figure 5.2 – Word representation matrix $\mathbf{W}$, where each column of the matrix is a vector representing a word of the vocabulary.

The main approach taken by DocNADE is similar to NADE, but differs significantly in the design of parameter tying. The probability of a document $\mathbf{v}$ is estimated using the probability chain rule, but the architecture is modified to cope with large vocabularies. Each word observation v_i of the document $\mathbf{v}$ leads to a hidden layer $\mathbf{h}_i$, which represents the past observations $\mathbf{v}_{<i}$ (see figure 5.3). This hidden layer is computed as follows:

$$\mathbf{h}_i(\mathbf{v}_{<i}) = \mathbf{g}\left(\mathbf{c} + \sum_{k<i} \mathbf{W}_{:,v_k}\right), \tag{5.6}$$

where each column of the matrix $\mathbf{W}$ acts as a vector of size H that represents a word. The embedding of the i^{th} word in the document is thus the column of index v_i in the matrix $\mathbf{W}$.

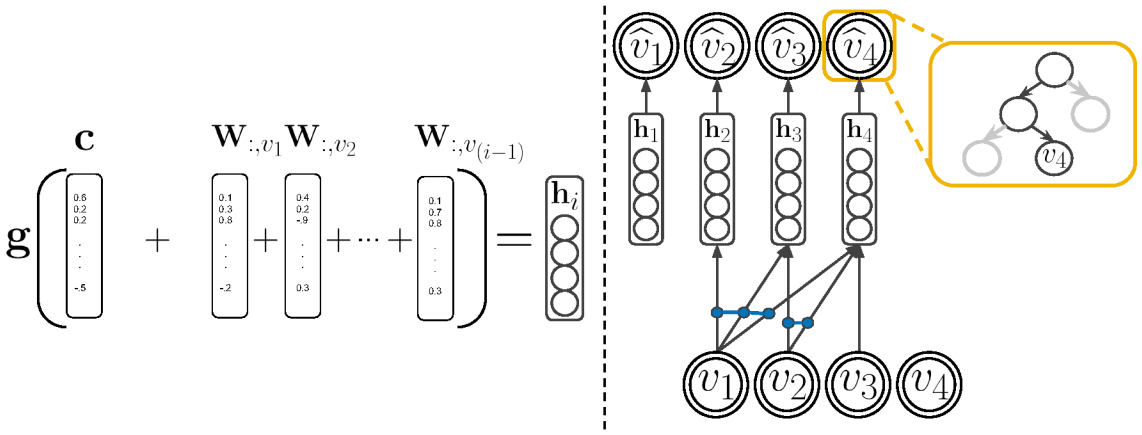


Figure 5.3 – (**Left**) Representation of the computation of a hidden layer $\mathbf{h}_i$, function $\mathbf{g}(\cdot)$ can be any activation function. (**Right**) Illustration of DocNADE. Connections between each multinomial observation v_i and hidden units are also shared, and each conditional $p(v_i|\mathbf{v}_{<i})$ is decomposed into a tree of binary logistic regressions.

Notice that by sharing the word representation matrix across positions in the document, each hidden layer $\mathbf{h}_i(\mathbf{v}_{<i})$ is in fact independent of the order of the words within $\mathbf{v}_{<i}$. We made the hypothesis that even if the word order affect semantics, most of its information depends on the set of words used. DocNADE ignores in which order the previously observed words appeared. The implications of this choice is that the learned hidden representation will not model the syntactic structure of the documents and focus on its document-level semantics, i.e. its topics. It also means that the first word at the beginning of the sequence has the same influence as the last one, its importance compared to the other words does not deteriorate over time. This phenomenon can be observed in the left part of figure 5.3, with this illustration we can see that if we swap the first word v_1 with the last one v_{i-1} we would obtain the same hidden representation $\mathbf{h}_i$.

It is also worth noticing that we can compute $\mathbf{h}_{i+1}$ recursively by keeping track of

the pre-activation of the previous hidden layer $\mathbf{h}_i$ as follows:

$$\mathbf{h}_{i+1}(\mathbf{v}_{<i+1}) = \mathbf{g}\left(\mathbf{W}_{:,v_i} + \underbrace{\mathbf{c} + \sum_{k<i} \mathbf{W}_{:,v_k}}_{\text{Precomputed for } \mathbf{h}_i(\mathbf{v}_{<i})}\right) \tag{5.7}$$

The weight sharing between hidden layers enables us to compute all hidden layers $\mathbf{h}_i(\mathbf{v}_{<i})$ for a document in $O(DH)$.

Then, to compute the probability of a full document $p(\mathbf{v})$, we need to estimate all conditional probabilities $p(v_i|\mathbf{v}_{<i})$. A straightforward solution would be to compute each $p(v_i|\mathbf{v}_{<i})$ using softmax layers with a shared weight matrix and bias, each fed with the corresponding hidden layer $\mathbf{h}_i$. However, the computational cost of this approach is prohibitive, since it scales linearly with the vocabulary size[1].To overcome this issue, we represent distribution over the vocabulary by a probabilistic binary tree, where the leaves correspond to the words. This approach is widely used in the field of neural probabilistic language models [40, 39]. Each word is represented by a path in the tree, going from the root to the leaf associated to that word. A binary logistic regression unit is associated to each node in the tree and gives the probability of the binary choice, going left or right. Therefore, a word probability can be estimated by the path's probability in this tree, resulting in a complexity in $O(\log V)$ for trees that are balanced. In our experiments, we used a randomly generated full binary tree with V leaves, each assigned to a unique word of the vocabulary.

More formally, let's denote by $\mathbf{l}(v_i)$ the sequence of nodes composing the path, from the root of the tree to the leaf corresponding to word v_i. Then, $\boldsymbol{\pi}(v_i)$ is the sequence of left/right decisions of the nodes in $\mathbf{l}(v_i)$. For example, the root of the tree is always the first element $l(v_i)_1$ and the value $\pi(v_i)_1$ will be 0 if the word is in the left sub-tree and 1 if it is in the right sub-tree (see figure 5.4). The matrix $\mathbf{V}$ stores by row the weights associated to each logistic classifier. There is one logistic classifier per node in the tree, but only a fraction of them is used for each word. Let $\mathbf{V}_{l(v_i)_m,:}$

1. For most natural language processing task, the vocabulary size exceeds $10,000$.

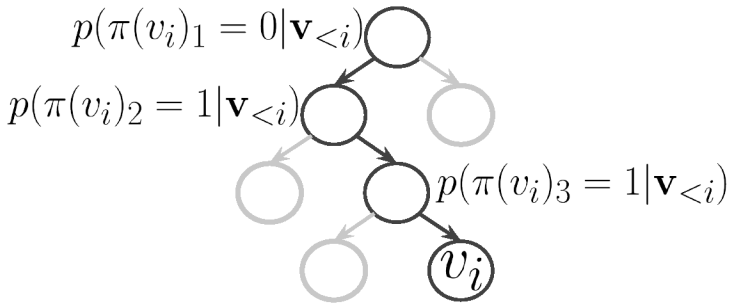


Figure 5.4 – Path of the word v_i in a binary tree. We compute the probability of the left/right choice (0/1) for each node of the path.

and $b_{l(v_i)_m}$ be the weights and bias for the logistic unit associated to the node $n(v_i)_m$. The probability $p(v_i|\mathbf{v}_{<i})$ given the tree and the hidden layer $\mathbf{h}_i(\mathbf{v}_{<i})$ is computed with the following formulas:

$$p(v_i = w|\mathbf{v}_{<i}) = \prod_{m=1}^{|\boldsymbol{\pi}(v_i)|} p(\pi(v_i)_m = \pi(w)_m|\mathbf{v}_{<i}), \text{ with} \tag{5.8}$$

$$p(\pi(v_i)_m = 1|\mathbf{v}_{<i}) = \text{sigm}\left(b_{l(v_i)_m} + \mathbf{V}_{l(v_i)_m,:} \cdot \mathbf{h}_i(\mathbf{v}_{<i})\right). \tag{5.9}$$

This hierarchical architecture allows us to efficiently compute the probability for each word in a document and therefore the probability of every documents with the probability chain rules (see Equation 5.2). As in the NADE model, the parameters of the model $\{\mathbf{b}, \mathbf{c}, \mathbf{W}, \mathbf{V}\}$ are learnt by minimizing the negative log-likelihood using stochastic gradient descent.

Since there is $\log(V)$ logistic regression units for a word (one per node), each of them has a time complexity of $O(H)$, the complexity of computing the probability for a document of D words is in $O(\log(V)DH)$.

As for using DocNADE to extract features from a complete document, we propose to use

$$\mathbf{h}(\mathbf{v}) = \mathbf{h}_{D+1}(\mathbf{v}_{<D+1}) = \mathbf{g}\left(\mathbf{c} + \sum_{k=1}^{D} \mathbf{W}_{:,v_k}\right) \tag{5.10}$$

which would be the hidden layer computed to obtain the conditional probability of a hypothetical $D+1^{\text{th}}$ word appearing in the document. This simply means that $\mathbf{h}(\mathbf{v})$ is the representation of all the words of the document. Notice that we don't need to become invariant to document length and average the hidden representation $\mathbf{h}(\mathbf{v})$ of DocNADE with the number of words. We don't average because we use DocNADE for the information retrieval task, needing only the orientation of the vector for Cosine distance.

Training from bag-of-word counts

So far, we have assumed that the ordering of the words in the document is known. However, document datasets often take the form of word-count vectors in which the original word order, required to specify the sequence of conditionals $p(v_i|\mathbf{v}_{<i})$, has been lost.

Thankfully, it is still possible to successfully train DocNADE despite the absence of this information. The idea is to assume that each observed document $\mathbf{v}$ was generated by initially sampling a *seed* document $\widetilde{\mathbf{v}}$ from DocNADE, whose words were then shuffled using a randomly generated ordering to produce $\mathbf{v}$. With this approach, we can express the probability distribution of $\mathbf{v}$ by computing the marginal over all possible seed document:

$$p(\mathbf{v}) = \sum_{\widetilde{\mathbf{v}} \in \mathcal{V}(\mathbf{v})} p(\mathbf{v}, \widetilde{\mathbf{v}}) = \sum_{\widetilde{\mathbf{v}} \in \mathcal{V}(\mathbf{v})} p(\mathbf{v}|\widetilde{\mathbf{v}}) p(\widetilde{\mathbf{v}}) \tag{5.11}$$

where $p(\widetilde{\mathbf{v}})$ is modeled by DocNADE, $\widetilde{\mathbf{v}}$ is the same as the observed document $\mathbf{v}$ but with a different (random) word sequence order and $\mathcal{V}(\mathbf{v})$ is the set of all the documents $\widetilde{\mathbf{v}}$ that have the same word count $\mathbf{n}(\mathbf{v}) = \mathbf{n}(\widetilde{\mathbf{v}})$. With the assumption of orderings being uniformly sampled, we can replace $p(\mathbf{v}|\widetilde{\mathbf{v}})$ with $\frac{1}{|\mathcal{V}(\mathbf{v})|}$ giving us:

$$p(\mathbf{v}) = \sum_{\widetilde{\mathbf{v}} \in \mathcal{V}(\mathbf{v})} \frac{1}{|\mathcal{V}(\mathbf{v})|} p(\widetilde{\mathbf{v}}) = \frac{1}{|\mathcal{V}(\mathbf{v})|} \sum_{\widetilde{\mathbf{v}} \in \mathcal{V}(\mathbf{v})} p(\widetilde{\mathbf{v}}) \ . \tag{5.12}$$

In practice, one approach to training the DocNADE model over $\widetilde{\mathbf{v}}$ is to artificially generate ordered documents by uniformly sampling words, without replacement, from

the bags-of-words in the dataset. This would be equivalent to taking each original document and shuffling the order of its words. This approach can be shown to optimize a stochastic upper bound on the actual negative log-likelihood of documents. As we'll see, experimental results show that convincing performance can still be reached.

With this training procedure, DocNADE shows its ability to learn and predict a new word in a document at a random position while preserving the overall semantic properties of the document. The model is therefore learning not to insert "intruder" words, i.e. words that do not belong with the others. After training, a document's learned representation should contain valuable information to identify intruder words for this document. It's interesting to note that the detection of such intruder words has been used previously as a task in user studies to evaluate the quality of the topics learned by LDA, though at the level of single topics and not whole documents [11].

The goal with DocNADE is to have a simple, powerful, fast to train model that can also be used with data that have lost the information about ordering. We want to learn topic-level semantics, where the order of the words is less important than the actual words used, instead of learning syntax, where the order of the words is very important. In this context, it makes no sense to apply an RNN-like model on randomly shuffled words, since there's no position varying information to model.

5.3 Deep Document NADE

The single hidden layer version of DocNADE already achieves very competitive performance for topic modeling [32]. Extending it to a deep, multiple hidden layer architecture could however allow for even better performance, as suggested by the recent and impressive success of deep neural networks. Unfortunately, deriving a deep version of DocNADE that is practical cannot be achieved solely by adding hidden layers to the definition of the conditionals $p(v_i|\mathbf{v}_{<i})$. Indeed, computing $p(\mathbf{v})$ requires computing each $p(v_i|\mathbf{v}_{<i})$ conditional (one for each word), and it is no longer possible to exploit Equation 5.7 to compute the sequence of all hidden layers in $O(DH)$ when multiple deep hidden layers are used.

In this section, we describe an alternative training procedure that enables us the introduction of multiple stacked hidden layers. This procedure was first introduced

in [56] to model images, which was itself borrowing from the training scheme introduced by [50].

As mentioned in section 5.2.2, DocNADE can be trained on random permutations of the words from training documents. As noticed by [50], the use of many orderings during training can be seen as the instantiation of many different DocNADE models that share a single set of parameters. Thus, training DocNADE with random permutations also amounts to minimizing the negative log-likelihood averaged across all possible orderings, for each training example $\mathbf{v}$.

In the context of deep NADE models, a key observation is that training on all possible orderings implies that for a given context $\mathbf{v}_{<i}$, we wish the model to be equally good at predicting any of the remaining words appearing next, since for each there is an ordering such that they appear at position i. Thus, we can redesign the training algorithm such that, instead of sampling a complete ordering of all words for each update, we instead sample a single context $\mathbf{v}_{<i}$ and perform an update of the conditionals using that context. This is done as follows. For a given document, after generating vector $\mathbf{v}$ by shuffling the words from the document, a split point i is randomly drawn. From this split point results two parts of the document, $\mathbf{v}_{<i}$ and $\mathbf{v}_{\geq i}$:

$$\mathbf{v} = [\underbrace{v_1, \ \dots \ , v_{i-1}}_{\mathbf{v}_{<i}}, \underbrace{v_i, \ \dots \ , v_D}_{\mathbf{v}_{\geq i}}]^T. \tag{5.13}$$

The vector $\mathbf{v}_{<i}$ is considered as the input and the vector $\mathbf{v}_{\geq i}$ contains the targets to be predicted by the model. Since in this setting a training update relies on the computation of a single latent representation, that of $\mathbf{v}_{<i}$ for the drawn value of i, deeper hidden layers can be added at a reasonable increase in computation.

Thus, in DeepDocNADE the conditionals $p(v_i|\mathbf{v}_{<i})$ are modeled as follows. The first hidden layer $\mathbf{h}^{(1)}(\mathbf{v}_{<i})$ represents the conditioning context $\mathbf{v}_{<i}$ as in the single hidden layer DocNADE:

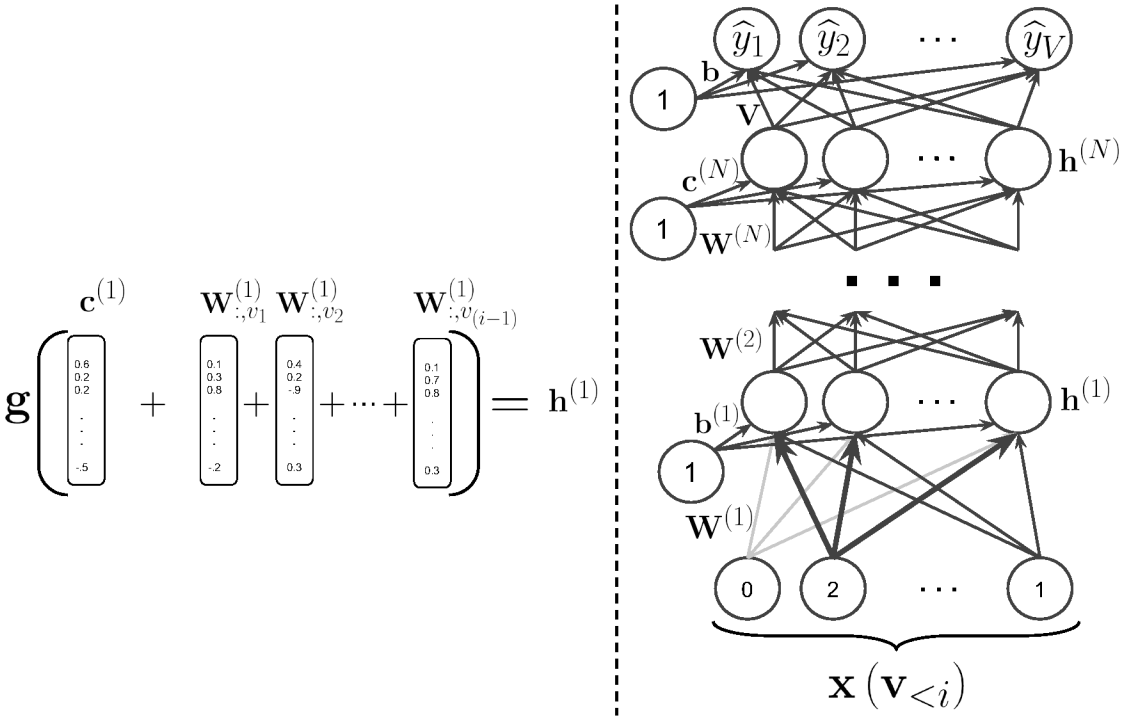


Figure 5.5 – (**Left**) Representation of the computation by summation of the first hidden layer $\mathbf{h}^{(1)}$, which is equivalent to multiplying the bag-of-word $\mathbf{x}\left(\mathbf{v}_{<i}\right)$ with the word representation matrix $\mathbf{W}^{(1)}$. (**Right**) Illustration of DeepDocNADE architecture.

$$\mathbf{h}^{(1)}\left(\mathbf{v}_{<i}\right) \quad = \mathbf{g}\left(\mathbf{c}^{(1)} + \sum_{k<i} \mathbf{W}^{(1)}_{:,v_k}\right) = \mathbf{g}\left(\mathbf{c}^{(1)} + \mathbf{W}^{(1)} \cdot \mathbf{x}\left(\mathbf{v}_{<i}\right)\right) \tag{5.14}$$

where $\mathbf{x}\left(\mathbf{v}_{<i}\right)$ is the histogram vector representation (bag-of-word) of the word sequence $\mathbf{v}_{<i}$, and the exponent is used as an index over the hidden layers and its parameters, with (1) referring to the first layer (see figure 5.5). The matrix $\mathbf{W}^{(1)}$ is the word representation matrix of DeepDocNADE. Notice that like DocNADE, the first hidden layer $\mathbf{h}^{(1)}(\mathbf{v}_{<i})$ of DeepDocNADE is independent of the order of the words within $\mathbf{v}_{<i}$. We can now easily add new hidden layers as in a regular deep feed-forward neural network:

$$\mathbf{h}^{(n)}(\mathbf{v}_{<i}) = \mathbf{g}(\mathbf{c}^{(n)} + \mathbf{W}^{(n)} \cdot \mathbf{h}^{(n-1)}(\mathbf{v}_{<i})), \tag{5.15}$$

for $n = 2, \ldots, N$, where N is the total number of hidden layers. From the last hidden layer $\mathbf{h}^N$, we can finally compute the conditional $p(v_i = w|\mathbf{v}_{<i})$, for any word w:

$$\widehat{y}_w = p(v_i = w|\mathbf{v}_{<i}) = \text{softmax}(\mathbf{V}_{w,:} \cdot \mathbf{h}^{(N)}(\mathbf{v}_{<i}) + b_w), \tag{5.16}$$

$$\widehat{\mathbf{y}} = \text{softmax}(\mathbf{V} \cdot \mathbf{h}^{(N)}(\mathbf{v}_{<i}) + \mathbf{b}), \tag{5.17}$$

where $\mathbf{V} \in \mathbb{R}^{V \times H^{(N)}}$ is the output parameter matrix and $\mathbf{b}$ the bias vector. Finally, the loss function used to update the model for the given context $\mathbf{v}_{<i}$ is:

$$L(\mathbf{v}) = \frac{D_\mathbf{v}}{D_\mathbf{v} - i + 1} \sum_{w \in \mathbf{v}_{\geq i}} -\log p(v_i = w|\mathbf{v}_{<i}), \tag{5.18}$$

where $D_\mathbf{v}$ is the number of words in $\mathbf{v}$ and the sum iterates over all words w present in $\in \mathbf{v}_{\geq i}$. Thus, as described earlier, the model predicts each remaining word after the splitting position i as if it was actually at position i. The factors in front of the sum comes from the fact that the complete log-likelihood would contain $D_\mathbf{v}$ log-conditionals and that we are averaging over $D_\mathbf{v} - i + 1$ possible choices for the word ordered at position i. For a more detailed presentation, see [56]. The average loss function of Equation 5.18 is optimized with stochastic gradient descent [2].

Note that to compute the probabilities $p(v_i = w|\mathbf{v}_{<i})$, a probabilistic tree could again be used. However, since all probabilities needed for an update are based on a single context $\mathbf{v}_{<i}$, a single softmax layer is sufficient to compute all necessary quantities. Therefore the computational burden of a conventional softmax is not as prohibitive as for DocNADE, especially with an efficient implementation on the GPU. For this reason, in our experiments with DeepDocNADE we opted for a regular softmax.

2. A document is usually represented as bag-of-words. Generating a word vector $\mathbf{v}$ from its bag-of-words, shuffling the word count vector $\mathbf{v}$, splitting it, and then regenerating the histogram $\mathbf{x}(\mathbf{v}_{<i})$ and $\mathbf{x}(\mathbf{v}_{\geq i})$ is unfortunately fairly inefficient for processing samples in a mini-batch fashion. Hence, in practice, we split the original histogram $\mathbf{x}(\mathbf{v})$ directly by uniformly sampling, for each word individually, how many are put on the left of the split (the others are put on the right of the split). This procedure, used also by [56], is only an approximation of the correct procedure mentioned in the main text, but produces a substantial speedup while also yielding good performance. Thus, we used it also in this paper.

5.4 DocNADE Language Model

While topic models such as DocNADE can be useful to learn topical representations of documents, they are actually very poor models of language. In DocNADE, this is due to the fact that, when assigning a probability to the next word in a sentence, DocNADE actually ignores in which order the previously observed words appeared. Yet, this ordering of words conveys a lot of information regarding the most likely syntactic role of the next word or the finer semantics within the sentence. In fact, most of that information is predictable from the last few words, which is why N-gram language models remain the dominating approach to language modeling.

In this section, we propose a new model that extends DocNADE to mitigate the influence of both short and long term dependencies in a single model, which we refer to as the DocNADE language model or DocNADE-LM. The solution we propose enhances the bag-of-word representation of a word's history with the explicit inclusion of n-gram dependencies for each word to predict.

The figure 5.6 depicts the overall architecture. This model can be seen as an extension of the seminal work on neural language models of [5] that includes a representation of a document's larger context. It can also be seen as a neural extension of the cache-based language model introduced in [31], where the n-gram probability is interpolated with the word distribution observed in a dynamic cache. This cache of a fixed size keeps track of previously observed words to include long term dependencies in the prediction and preserve semantic consistency beyond the scope of the n-gram. In our case, the DocNADE language model maintains an unbounded cache and defines a proper, jointly trained solution to mitigate these two kinds of dependencies.

As in DocNADE, a document $\mathbf{v}$ is modeled as a sequence of multinomial observations. The sequence size is arbitrary and this time the order is important, each element v_i consists of the i-th word of the sequence and represent an index in a vocabulary of size V. The conditional probability of a word given its history $p(v_i|\mathbf{v}_{<i})$ is now expressed as a smooth function of a hidden layer $\mathbf{h}_i(\mathbf{v}_{<i})$ used to predict word v_i. The peculiarity of the DocNADE language model lies in the definition of this hidden

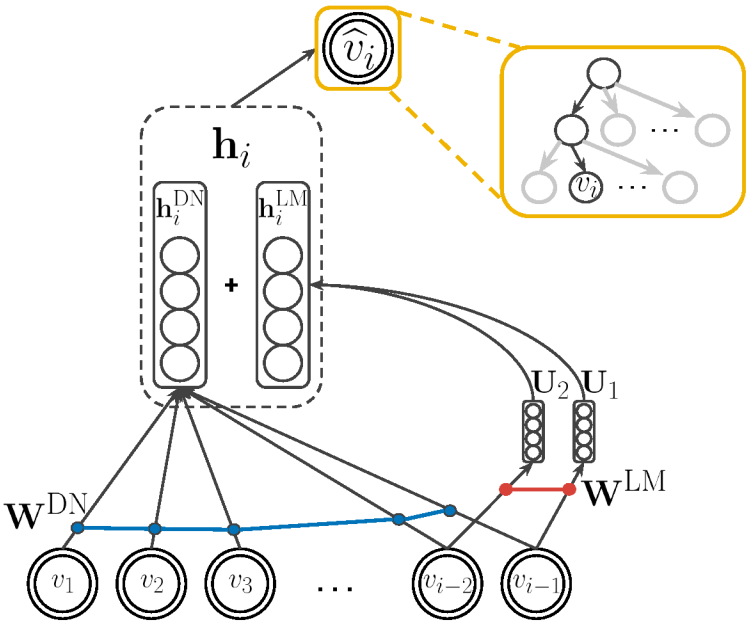


Figure 5.6 – Illustration of the conditional $p(v_i|\mathbf{v}_{<i})$ in a trigram NADE language model. Compared to DocNADE, this model incorporates the architecture of a neural language model, that first maps previous words (2 for a trigram model) to vectors using an embedding matrix $\mathbf{W}^{\text{LM}}$ before connecting them to the hidden layer using regular (untied) parameter matrices ($\mathbf{U_1}$, $\mathbf{U_2}$ for a trigram). In our experiments, each conditional $p(v_i|\mathbf{v}_{<i})$ exploits decomposed into a tree of logistic regressions, the hierarchical softmax.

layer, which now includes two terms:

$$\mathbf{h}_i(\mathbf{v}_{<i}) = \mathbf{g}(\mathbf{b} + \mathbf{h}_i^{\text{DN}}(\mathbf{v}_{<i}) + \mathbf{h}_i^{\text{LM}}(\mathbf{v}_{<i})). \tag{5.19}$$

The first term borrows from the DocNADE model by aggregating embeddings for all the previous words in the history:

$$\mathbf{h}_i^{\text{DN}}(\mathbf{v}_{<i}) = \frac{1}{i-1}\sum_{k<i}\mathbf{W}^{\text{DN}}_{:,v_k}\ , \tag{5.20}$$

where $i-1$ is the number of words used to create $\mathbf{h}_i^{\text{DN}}(\mathbf{v}_{<i})$. We average the hidden representation with the number of words to get good results for the language model task, we need to do this to become invariant to document length. The hidden representation $\mathbf{h}_i^{\text{DN}}(\mathbf{v}_{<i})$ model the semantics part of the text that have been observed

yet. Like we explained before for DocNADE, the order of the words within $\mathbf{v}_{<i}$ is not modeled. The hidden layer $\mathbf{h}_i^{\text{DN}}(\mathbf{v}_{<i})$ can be seen as a context representation of the past where the influence of a word compared to the influence of the other words does not deteriorate over time.

The second contribution derives from neural n-gram language models as follows:

$$\mathbf{h}_i^{\text{LM}}(\mathbf{v}_{<i}) = \sum_{k=1}^{n-1} \mathbf{U}_k \cdot \mathbf{W}^{\text{LM}}_{:,v_{i-k}} \ . \tag{5.21}$$

In this formula, the history for the word v_i is restricted to the $n-1$ preceding words, following the common n-gram assumption. The term $\mathbf{h}_i^{\text{LM}}$ hence represents the continuous representation of these $n-1$ words, in which word embeddings are linearly transformed by the ordered matrices $\mathbf{U}_1, \mathbf{U}_2, ..., \mathbf{U}_{n-1}$. Moreover, $\mathbf{b}$ gathers the bias terms for the hidden layer. In this model, two sets of word embeddings are defined, $\mathbf{W}^{\text{DN}}$ and $\mathbf{W}^{\text{LM}}$, which are respectively associated to the DocNADE and neural language model parts. For simplicity, we assume both are of the same size H.

Given hidden layer $\mathbf{h}_i(\mathbf{v}_{<i})$, conditional probabilities $p(v_i|\mathbf{v}_{<i})$ can be estimated, and thus $p(\mathbf{v})$. For the aforementioned reason explained in DocNADE, the output layer is structured for efficient computations. Specifically, we decided to use a variation of the probabilistic binary tree, known as a hierarchical softmax layer. In this case, instead of having binary nodes with multiple levels in the tree, we have only two levels where all words have their leaf at level two and each node is a multiclass (i.e. softmax) logistic regression with roughly $\sqrt{V}$ classes (one for each children). Computing probabilities in such a structured layer can be done using only two matrix multiplications, which can be efficiently computed on the GPU.

With a hidden layer of size H, the complexity of computing the softmax at one node is $O(H\sqrt{V})$. If we have D words in a given document, the complexity of computing all necessary probabilities from the hidden layers is thus $O(DH\sqrt{V})$. It also requires $O(DH)$ computations to compute the hidden representations for the DocNADE part and $O(nH^2)$ for the language model part. The full complexity for computing $p(\mathbf{v})$ and the updates for the parameters are thus computed in $O(DH\sqrt{V} + DH + nH^2)$.

Once again, the loss function of the model is the negative log-likelihood and we minimize it by using stochastic gradient descent over documents, to learn the values

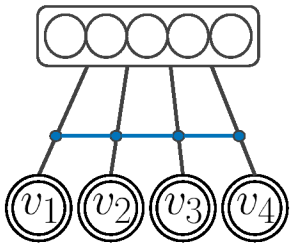


Figure 5.7 – Replicated Softmax model. Each multinomial observation v_i is a word. Connections between each multinomial observation v_i and hidden units are shared.

of the parameters $\{\mathbf{b}, \mathbf{c}, \mathbf{V}, \mathbf{W}^{\text{LM}}, \mathbf{W}^{\text{DN}}, \mathbf{U}_1, ..., \mathbf{U}_n\}$.

5.5 Related Work

Much like NADE was inspired by the RBM, DocNADE can be seen as related to the Replicated Softmax model [43], an extension of the RBM to document modeling. Here, we describe in more detail the Replicated Softmax, along with its relationship with DocNADE.

Much like the RBM, the Replicated Softmax models observations using a latent, stochastic binary layer $\mathbf{h}$. Here, the observations are the documents $\mathbf{v}$, which interact with the hidden layer $\mathbf{h}$ through an energy function similar to RBM's:

$$E(\mathbf{v}, \mathbf{h}) = -D\ \mathbf{c}^\top \mathbf{h} + \sum_{i=1}^{D} -\mathbf{h}^\top \mathbf{W}_{:,v_i} - b_{v_i} = -D\ \mathbf{c}^\top \mathbf{h} - \mathbf{h}^\top \mathbf{W}\mathbf{n}(\mathbf{v}) - \mathbf{b}^\top \mathbf{n}(\mathbf{v}), \quad (5.22)$$

where $\mathbf{n}(\mathbf{v})$ is a bag-of-word vector of size V (the size of the vocabulary) containing the word count of each word in the vocabulary for document $\mathbf{v}$. $\mathbf{h}$ is the stochastic, binary hidden layer vector and $\mathbf{W}_{:,v_i}$ is the v_i^{th} column vector of matrix $\mathbf{W}$. $\mathbf{c}$ and $\mathbf{b}$ are the bias vectors for the visible and the hidden layers. We see here that the larger $\mathbf{v}$ is, the bigger the number of terms in the sum over i is, resulting in a high energy value. For this reason, the hidden bias term $\mathbf{c}^\top \mathbf{h}$ is multiplied by D, to be commensurate with the contribution of the visible layer.. We can see also that connection parameters are shared across different positions i in $\mathbf{v}$, as illustrated by figure 5.7).

The conditional probabilities of the hidden and the visible layer factorize much

like in the RBM, in the following way:

$$p(\mathbf{h}|\mathbf{v}) = \prod_j p(h_j|\mathbf{v}) \ , \qquad p(\mathbf{v}|\mathbf{h}) = \prod_{i=1}^{D} p(v_i|\mathbf{h}) \tag{5.23}$$

where the factors $p(h_j|\mathbf{v})$ and $p(v_i|\mathbf{h})$ are such that

$$p(h_j = 1|\mathbf{v}) = \text{sigm}(Dc_j + \sum_i W_{jv_i}) \tag{5.24}$$

$$p(v_i = w|\mathbf{h}) = \frac{\exp(b_w + \mathbf{h}^\top \mathbf{W}_{:,w})}{\sum_{w'} \exp(b_{w'} + \mathbf{h}^\top \mathbf{W}_{:,w'})} \tag{5.25}$$

The normalized exponential part in $p(v_i = w|\mathbf{h})$ is simply the softmax function. To train this model, we'd like to minimize the negative log-likelihood (NLL). Its gradients for a document $\mathbf{v}^{(t)}$ with respect to the parameters $\theta = \{\mathbf{W}, \mathbf{c}, \mathbf{b}\}$ are calculated as follows:

$$\frac{\partial - \log p(\mathbf{v}^{(t)})}{\partial \theta} = \mathbb{E}_{\mathbf{h}|\mathbf{v}^{(t)}} \left[\frac{\partial}{\partial \theta} E(\mathbf{v}^{(t)}, \mathbf{h}) \right] - \mathbb{E}_{\mathbf{v},\mathbf{h}} \left[\frac{\partial}{\partial \theta} E(\mathbf{v}, \mathbf{h}) \right] . \tag{5.26}$$

As with conventional RBMs, the second expectation in Equation 5.26 is computationally too expensive. The gradient of the negative log-likelihood is therefore approximated by replacing the second expectation with an estimated value obtained by contrastive divergence [24]. This approach consists of performing K steps of blocked Gibbs sampling, starting at $\mathbf{v}^{(t)}$ and using Equations 5.24 and 5.25, to obtain point estimates of the expectation over $\mathbf{v}$. Large values of K must be used to reduce the bias of gradient estimates and obtain good estimates of the distribution. This approximation is used to perform stochastic gradient descent.

During Gibbs sampling, the Replicated Softmax model must compute and sample from $p(v_i = w|\mathbf{h})$, which requires the computation of a large softmax. Most importantly, the computation of the softmax most be repeated K times for each update, which can be prohibitive, especially for large vocabularies. Unlike DocNADE, this softmax cannot simply be replaced by a structured softmax.

It is interesting to see that DocNADE is actually related to how the Replicated Softmax approximates, through mean-field inference, the conditionals $p(v_i =$

$w|\mathbf{v}_{<i})$. Computing the conditional $p(v_i = w|\mathbf{v}_{<i})$ with the Replicated Softmax is intractable. However, we could use mean-field inference to approximate the full conditional $p(v_i, \mathbf{v}_{>i}, \mathbf{h}|\mathbf{v}_{<i})$ as the factorized

$$q(v_i, \mathbf{v}_{>i}, \mathbf{h}|\mathbf{v}_{<i}) = \prod_{k \geq i} q(v_k|\mathbf{v}_{<i}) \prod_j q(h_j|\mathbf{v}_{<i}) \ , \tag{5.27}$$

where $q(v_k = w|\mathbf{v}_{<i}) = \mu_{kw}(i)$ and $q(h_j = 1|\mathbf{v}_{<i}) = \tau_j(i)$. We would find the parameters $\mu_{kw}(i)$ and $\tau_j(i)$ that minimize the KL divergence between $q(v_i, \mathbf{v}_{>i}, \mathbf{h}|\mathbf{v}_{<i})$ and $p(v_i, \mathbf{v}_{>i}, \mathbf{h}|\mathbf{v}_{<i})$ by applying the following message passing equations until convergence:

$$\tau_j(i) \leftarrow \text{sigm}\left(D \ c_j + \sum_{k \geq i} \sum_{w'=1}^{V} W_{jw'} \mu_{kw'}(i) + \sum_{k<i} W_{jv_k}\right), \tag{5.28}$$

$$\mu_{kw}(i) \leftarrow \frac{\exp(b_w + \sum_j W_{jw} \tau_j(i))}{\sum_{w'} \exp(b_{w'} + \sum_j W_{jw'} \tau_j(i))}. \tag{5.29}$$

with $k \in \{i, \ldots, D\}$, $j \in \{1, \ldots, H\}$ and $w \in \{1, \ldots, V\}$. The conditional $p(v_i = w|\mathbf{v}_{<i})$ could then be estimated with $\mu_{kw}(i)$ for all i. We note that one iteration of mean-field (with $\mu_{kw'}(i)$ initialized to 0) in the Replicated Softmax corresponds to the conditional $p(v_i = w|\mathbf{v}_{<i})$ computed by DocNADE with a single hidden layer and a flat softmax output layer.

In our experiment, we'll see that DocNADE compares favorably to Replicated Softmax.

5.6 Topic modeling experiments

To compare the topic models, two kinds of quantitative comparisons are used. The first one evaluates the generative ability of the different models, by computing the perplexity of held-out texts. The second one compares the quality of document representations for an information retrieval task.

Two different datasets are used for the experiments of this section, 20 Newsgroups and RCV1-V2 (Reuters Corpus Volume I). The 20 Newsgroups corpus has 18,786 documents (postings) partitioned into 20 different classes (newsgroups). RCV1-V2 is

a much bigger dataset composed of 804,414 documents (newswire stories) manually categorized into 103 classes (topics). The two datasets were preprocessed by stemming the text and removing common stop-words. The 2,000 most frequent words of the 20 Newsgroups training set and the 10,000 most frequent words of the RCV1-V2 training set were used to create the dictionary for each dataset. Also, every word counts n_i, used to represent the number of times a word appears in a document was replaced by $log(1 + n_i)$ rounded to the nearest integer, following [43].

5.6.1 Generative Model Evaluation

For the generative model evaluation, we follow the experimental setup proposed by Salakhutdinov and Hinton [43] for 20 Newsgroups and RCV1-V2 datasets. We use the exact same split for the sake of comparison. The setup consists in respectively 11,284 and 402,207 training examples for 20 Newsgroups and RCV1-V2. We randomly extracted 1,000 and 10,000 documents from the training sets of 20 Newsgroups and RCV1-V2, respectively, to build a validation set. The average perplexity per word is used for comparison. This perplexity is estimated using the 50 first documents of a separate test set, as follows:

$$\exp\left(-\frac{1}{T}\sum_{t}\frac{1}{|\mathbf{v}^t|}\log p(\mathbf{v}^t)\right), \tag{5.30}$$

where T is the total number of examples in the test set and $\mathbf{v}^t$ is the t^{th} test document [3].

Table 5.1 gathers the perplexity per word results for 20 Newsgroups and RCV1-V2. Theere we compare 5 different models: the Latent Dirichlet Allocation (LDA) [8], the Replicated Softmax, the recent fast Deep AutoRegressive Networks (fDARN) [37], DocNADE and DeepDocNADE (DeepDN in the table). Each model uses 50 latent topics. For the experiments with DeepDocNADE, we provide the performance when using 1, 2, and 3 hidden layers. As shown in Table 5.1, DeepDocNADE provides the

3. Note that there is a difference between the sizes, for the training sets and test sets of 20 Newsgroups and RCV1-V2 reported in this paper and the one reported in the original data paper of Salakhutdinov and Hinton [43]. The correct values are the ones given in this section, which was confirmed after personal communication with Salakhutdinov and Hinton [43].

Dataset	LDA	Replicated Softmax	fDARN	DocNADE	DeepDN (1layer)	DeepDN (2layer)	DeepDN (3layer)
20 News	1091	953	917	896	**835**	877	923
RCV1-v2	1437	988	724	742	579	552	**539**

Table 5.1 – Test perplexity per word for models with 50 topics. The results for LDA and Replicated Softmax were taken from Salakhutdinov and Hinton [43].

best generative performances. Our best DeepDocNADE models were trained with the Adam optimizer [26] and with the tanh activation function. The hyper-parameters of Adam were selected on the validation set.

Crucially, following [50], an ensemble approach is used to compute the probability of documents, where each component of the ensembles are the same DeepDocNADE model evaluated on a different word ordering. Specifically, the perplexity with M ensembles becomes as follows:

$$\exp\left(-\frac{1}{T}\sum_{t}\frac{1}{|\mathbf{v}^t|}\log\left(\frac{1}{M}\sum_{m}p(\mathbf{v}^{(t,m)})\right)\right), \tag{5.31}$$

where T is the total number of examples, M is the number of ensembles (word orderings) and $\mathbf{v}^{(t,m)}$ denotes the m^{th} word ordering for the t^{th} documnet. We try $M = \{1, 2, 4, 16, 32, 64, 128, 256\}$, with the results in Table 5.1 using $M = 256$. For the 20 Newsgroups dataset, adding more hidden layers to DocDocNADE fails to provide further improvements. We hypothesize that the relatively small size of this dataset makes it hard to successfully train a deep model. However, the opposite is observed on the RCV1-V2 dataset, which is more than an order of magnitude larger than 20 Newsgroups. In this case, DeepDocNADE outperforms fDARN and DocNADE, with a relative perplexity reduction of 20%, 24% and 26% with respectively 1,2 and 3 hidden layers.

To illustrate the impact of M on the performance of DeepDocNADE, Figure 5.8 shows the perplexity on both datasets using the different values for M that we tried. We can observe that beyond $M = 128$, this hyper-parameter has only a minimal impact on the perplexity.

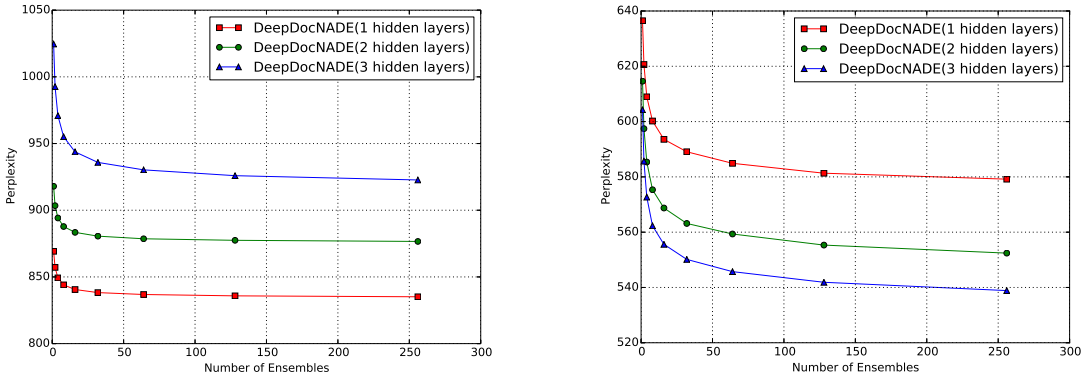


Figure 5.8 – Perplexity obtained with different numbers of word orderings m for 20 Newsgroups on the left and RCV1-V2 on the right.

5.6.2 Document Retrieval Evaluation

A document retrieval evaluation task was also used to evaluate the quality of the document representation learned by each model. As in the previous section, the datasets under consideration are 20 Newsgroups and RCV1-V2. The experimental setup is the same for the 20 Newsgroups dataset, while for the RCV1-V2 dataset, we reproduce the same setup as the one used in Srivastava et al. [46], where the training set contained 794,414 examples and 10,000 examples constituted the test set.

For DocNADE and DeepDocNADE, the representation of a document is obtained simply by computing the top-most hidden layer (Equation 5.10) when feeding all words as input.

The retrieval task follows the setup of [46]. The documents in the training and validation sets are used as the database for retrieval, while the test set is used as the query set. The similarity between a query and all examples in the database is computed using the cosine similarity between their vector representations. For each query, documents in the database are then ranked according to this similarity, and precision/recall (PR) curves are computed, by comparing the label of the query documents with those of the database documents. Since documents sometimes have multiple labels (specifically those in RCV1-V2), for each query the PR curves for each of its labels are computed individually and then averaged. Finally, we report the global average

of these (query-averaged) curves to compare models against each other. Training and model selection is otherwise performed as in the generative modeling evaluation.

As shown in Figure 5.9, DeepDocNADE always yields very competitive results, on both datasets, and outperforming the other models in most cases. Specifically, for the 20 Newsgroups dataset, DeepDocNADE with 2 and 3 hidden layers always perform better than the other methods. DeepDocNADE with 1 hidden layer also performs better than the other baselines when retrieving the top few documents (e.g. when recall is smaller than 0.2).

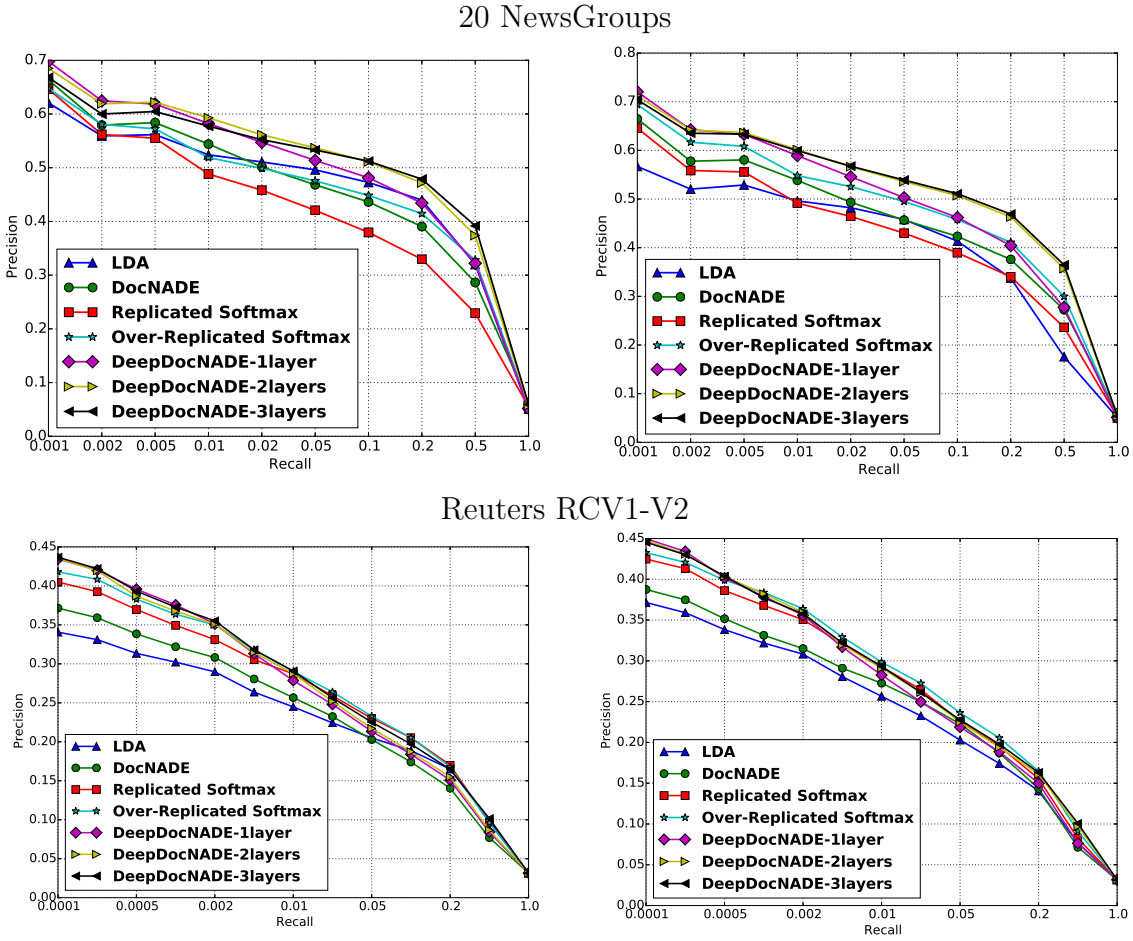


Figure 5.9 – Precision-Recall curves for document retrieval task. On the left are the results using a hidden layer size of 128, while the plots on the right are for a size of 512.

5.6.3 Qualitative Inspection of Learned Representations

In this section, we want to assess if the DocNADE approach for topic modeling can capture meaningful semantic properties of texts.

First, one way to explore the semantic properties of trained models is through their learned word embeddings. Each of the columns of the matrix W represents a word w where $W_{:,w}$ is the vector representation of w. Table 5.2 shows for some chosen words the five nearest words according to their embeddings, foro a DocNADE model. We can observe for each example the semantic consistency of the word representations. Similar results can be observed for DeepDocNADE models.

weapons	**medical**	**companies**	**define**	**israel**	**book**	**windows**
weapon	treatment	demand	defined	israeli	reading	dos
shooting	medecine	commercial	definition	israelis	read	microsoft
firearms	patients	agency	refer	arab	books	version
assault	process	company	make	palestinian	relevent	ms
armed	studies	credit	examples	arabs	collection	pc

Table 5.2 – The five nearest neighbors in the word representation space learned by DocNADE.

We've also attempted to measure whether the hidden units of the first hidden layer of DocNADE and DeepDocNADE models modeled distinct topics. Understanding the function represented by hidden units in neural networks is not a trivial affair, but we considered the following, simple approach. For a given hidden unit, its connections to words were interpreted as the importance of the word for the associated topic. Therefore, for a hidden unit, we selected the words having the strongest positive connections, i.e. for the i_{th} hidden unit we chose the words w that have the highest connection values $W_{i,w}$.

With this approach, fifty topics were obtained from a DocNADE model (with sigmoid activation function) trained on 20 Newsgroups. Four of the fifty topics are shown in Table 5.3 and can be readily interpreted as topics representing: religion, space, sports and security. Note that those four topics are actual (sub)categories in 20 Newsgroups.

That said, we've had less success understanding the topics extracted when using

the tanh activation function or when using DeepDocNADE. It thus seems that these models are then choosing to learn a latent representation that isn't aligning its dimensions with concepts that are easily interpretable, even though it is clearly capturing well the statistics of documents (since our qualitative results with DeepDocNADE are excellent).

Hidden unit topics			
jesus	shuttle	season	encryption
atheism	orbit	players	escrow
christianity	lunar	nhl	pgp
christ	spacecraft	league	crypto
athos	nasa	braves	nsa
atheists	space	playoffs	rutgers
bible	launch	rangers	clipper
christians	saturn	hockey	secure
sin	billion	pitching	encrypted
atheist	satellite	team	keys

Table 5.3 – Illustration of some topics learned by DocNADE. A topic i is visualized by picking the 10 words w with strongest connection W_{iw}.

5.7 Language Modeling Experiments

In this section, we test whether our proposed approach to incorporating a DocNADE component to a neural language model can improve the performance of a neural language model. Specifically, we considered treating a text corpus as a sequence of documents. We used the APNews dataset, as provided by Mnih and Hinton [39]. Unfortunately, information about the original segmentation into documents of the corpus wasn't available in the data as provided by Mnih and Hinton [39], thus we simulated the presence of documents by grouping one or more adjacent sentences, for training and evaluating DocNADE-LM, making sure the generated documents were non-overlapping. Thankfully, this approach still allows us to test whether DocNADE-LM is able to effectively leverage the larger context of words in making its predictions.

Since language models are generative models, the perplexity measured on some held-out texts provides an intrinsic and widely used evaluation criterion. Following

Mnih and Hinton [38] and Mnih and Hinton [39], we used the APNews dataset containing Associated Press news stories from 1995 and 1996. The dataset is again split into training, validation and test sets, with respectively 633,143, 43,702 and 44,601 sentences. The vocabulary is composed of 17,964 words. A 100 dimensional feature vectors are used for these experiments. The validation set is used for model selection and the perplexity scores of Table 5.4 are computed on the test set.

Models	Number of grouped sentences	Perplexity
KN6	-	123.5
LBL	-	117.0
HLBL	-	112.1
LM	-	119.78
DocNADE-LM	1	111.93
DocNADE-LM	2	110.9
DocNADE-LM	3	109.8
DocNADE-LM	4	109.78
DocNADE-LM	5	109.22

Table 5.4 – Test perplexity per word for models with 100 topics. The results for HLBL and LBL were taken from Mnih and Hinton [39].

The LM model in Table 5.4 corresponds to a regular neural (feed-forward) network language model. It is equivalent to only use the language model part of DocNADE-LM. These results are meant to measure whether the DocNADE part of DocNADE-LM can indeed help to improve performances.

We also compare to the log-bilinear language (LBL) model of [38]). While for the LM model we used a hierarchical softmax to compute the conditional word probabilities (see Section 5.4), the LBL model uses a full softmax output layer that uses the same word representation matrix at the input and output. This latter model is therefore slower to train. Later, Mnih and Hinton [39] also proposed adaptive approaches to learning a structured softmax layer, thus we also compare with their best approach. All aforementioned baselines are 6-gram models, taking in consideration the last 5 previous words to predict the next one. We also compare with a more traditional 6-gram model using Kneser-Ney smoothing, taken from Mnih and Hinton [38].

From Table 5.4, we see that adding context to DocNADE-LM, by increasing the

size of the multi-sentence segments, significantly improves the performance of the model (compared to LM) and also surpasses the performance of the most competitive alternative, the HLBL model.

5.7.1 Qualitative Inspection of Learned Representations

In this section we explore the semantic properties of texts learned by the DocNADE-LM model. Interestingly, we can examine the two different components (DN and LM) separately. Because the DocNADE part and the language modeling part of the model each have their own word matrix, $\mathbf{W}^{\text{DN}}$ and $\mathbf{W}^{\text{LM}}$ respectively, we can compare their contribution through these learned embeddings. As explained in the previous section, each of the columns of the matrices represents a word w where $\mathbf{W}^{\text{DN}}_{:,w}$ and $\mathbf{W}^{\text{LM}}_{:,w}$ are two different vector representations of the same word w.

We can see by observing Tables 5.5 and 5.6 that the two parts of the DocNADE-LM model have learned different semantic properties of words. An interesting example is seen in the nearest neighbors of the word *israel*, where the DocNADE focuses on the politico-cultural relation between these words, whereas the language model part seems to have learned the concept of countries in general.

weapons	**medical**	**companies**	**define**	**israel**	**book**	**windows**
security	doctor	industry	spoken	israeli	story	yards
humanitarian	health	corp	to	palestinian	novel	rubber
terrorists	medicine	firms	think_of	jerusalem	author	piled
deployment	physicians	products	of	lebanon	joke	fire_department
diplomats	treatment	company	bottom_line	palestinians	writers	shell

Table 5.5 – The five nearest neighbors in the word representation space learned by the DocNADE part of the DocNADE-LM model.

5.8 Conclusion

We have presented models inspired by NADE that can achieve state-of-the-art performances for modeling documents.

weapons	**medical**	**companies**	**define**	**israel**	**book**	**windows**
systems	special	countries	talk_about	china	film	houses
aircraft	japanese	nations	place	russia	service	room
drugs	bank	states	destroy	cuba	program	vehicle
equipment	media	americans	show	north_korea	movie	restaurant
services	political	parties	over	lebanon	information	car

Table 5.6 – The five nearest neighbors in the word representation space learned by the language model part of the DocNADE-LM model.

Indeed, for topic modeling, DocNADE had competitive results while its deep version, DeepDocNADE, outperformed the current state-of-the-art in generative document modeling, based on test set perplexity. The similarly good performances where observed when we used these models as feature extractors to represent documents for the task of information retrieval.

As for language modeling, the competitive performances of the DocNADE language models showed that combining contextual information by leveraging the DocNADE neural network architecture can significantly improve the performance of a neural probabilistic N-gram language model.

Conclusion

L'apprentissage automatique par réseaux profonds est un outil puissant qui a démontré son efficacité à plusieurs reprises en améliorant les résultats obtenus par d'autres approches considérées comme étant l'état de l'art dans des domaines comme la reconnaissance d'objets [30] ou encore la reconnaissance de la parole [23]. Au début de ce doctorat, nous avons donc émis l'hypothèse que l'utilisation des réseaux profonds dans le domaine du traitement automatique du langage naturel permettrait encore une fois d'améliorer les résultats obtenus par d'autres techniques considérées comme étant l'état de l'art.

L'utilisation des réseaux profonds sur des données textuelles était relativement peu explorée en 2012. Depuis, ce domaine a réussi, et continue encore, à faire avancer la performance des systèmes en TALN. Les contributions présentées dans cette thèse sont un bon exemple de ces avancées pour les tâches comme la représentation de documents, les modèles de classification multilangue ou la modélisation de la langue.

Nous avons commencé par présenter DocNADE, un réseau de neurones non-supervisé qui sert de modèle de sujets pour représenter les documents. Nous avons montré que DocNADE est un modèle performant pour créer des représentations vectorielles de documents ainsi que pour servir de modèle génératif. Un avantage important de ce modèle est que sa complexité durant l'entraînement augmente logarithmiquement (plutôt que linéairement) avec la taille du vocabulaire.

Nous avons ensuite montré que les réseaux de neurones pouvaient apprendre des représentations de mots bilingues utiles, sans nécessiter d'alignement au niveau des mots. Plus précisément, nous avons démontré que notre modèle de représentation bilingue peut obtenir des performances aussi bonnes que l'état de l'art, même comparées à des modèles qui eux utilisent cet alignement au niveau des mots. De plus, notre

modèle surpasse même une approche puissante basée sur la traduction automatique.

Finalement, nous avons présenté une famille de modèles inspirée par NADE. DeepDocNADE, une version profonde de DocNADE, surpasse l'état de l'art en tant que modèle génératif de documents. DeepDocNADE est aussi très performant lorsqu'il est utilisé pour représenter des documents dans le but de faire de la recherche d'information. Pour ce qui est de la modélisation de la langue, les bonnes performances de DocNADE-lm montrent que l'information contextuelle provenant de DocNADE combinée à un réseau de neurones, qui sert de modèle de langue N-gram, augmente la qualité des prédictions.

Dans cette thèse, nous avons présenté plusieurs modèles neuronaux différents, dont le but était la modélisation du texte afin d'accomplir différentes tâches. Ces réseaux de neurones ont tous un point en commun: ce sont des approches simples qui permettent d'entraîner des modèles relativement rapidement, comparativement à la majorité de leurs concurrents. Pour réussir, nous avons été obligés d'ignorer une information importante mais pas toujours nécessaire, c'est-à-dire l'ordre des mots. Cette omission s'explique par le fait qu'il est difficile de modéliser des données de taille variable ainsi que les dépendances temporelles. Une piste à suivre pour de futures recherches, dans le cadre des modèles simples et rapides, serait d'utiliser une représentation bigram de deux mots consécutifs (ou plus, comme un trigram), ce qui permettrait de modéliser en partie l'aspect temporel des phrases. L'idée n'est pas de simplement augmenter la taille du vocabulaire avec des bigrams pour les utiliser naïvement, car ceci a déjà été expérimenté auparavant. Il faut plutôt explorer différentes façons de fusionner la représentation des mots (unigram) avec la représentation des bigrams. Par exemple, on pourrait obtenir deux représentations séparées d'une même phrase, l'une provenant des unigrams et l'autre des bigrams, pour ensuite les unir à travers un réseau de neurones. Cependant, l'exploration de modèles plus complexes et plus lents à entraîner, comme les LSTM [25], devient malgré tout une option intéressante. En effet, ces modèles sont de plus en plus performants pour représenter les dépendances temporelles, informations qui s'avèrent essentielles pour certaines tâches. De plus, les processeurs ne cessant d'augmenter en rapidité, le temps d'entraînement devient alors un facteur de moins en moins important. Tout ceci montre que l'exploration des réseaux de neurones dans le domaine du TALN est loin d'être terminée.

Annexe A

Supplementary Material: An Autoencoder Approach to Learning Bilingual Word Representations

Sarath Chandar A P[1] *, **Stanislas Lauly**[2] *, **Hugo Larochelle**[2],
Mitesh M Khapra[3], **Balaraman Ravindran**[1], **Vikas Raykar**[3], **Amrita Saha**[3]
[1]Indian Institute of Technology Madras, [2]Université de Sherbrooke,
[3]IBM Research India
apsarathchandar@gmail.com,
{stanislas.lauly,hugo.larochelle}@usherbrooke.ca,
{mikhapra,viraykar,amrsaha4}@in.ibm.com, ravi@cse.iitm.ac.in
* Both authors contributed equally

Abstract

We provide here additional details relatively to our paper.

A.1 Coarser alignments

We experimented with the merging of 5, 25 and 50 adjacent sentences into a single bag-of-words. Results are shown in Table A.1. They suggest that merging several sentences into single bags-of-words does not necessarily impact the quality of the word

embeddings. Thus they confirm that exact sentence-level alignment is not essential to reach good performances as well.

Table A.1 – Cross-lingual classification accuracy for 3 different pairs of languages, when merging the bag-of-words for different numbers of sentences. These results are based on 1000 labeled examples.

	# sent.	EN → DE	DE → EN	EN → FR	FR → EN	EN → ES	ES → EN
BAE-tr	1	81.8	60.1	70.4	61.8	59.4	60.4
	5	84.0	67.7	72.08	65.7	58.325	54.48
	25	83.0	63.4	73.92	59.48	41.7	52.2
	50	75.9	68.6	73.96	62.34	46.35	47.22
BAE-cr	5	91.75	72.78	84.64	74.2	49.02	64.4
	25	88.0	64.5	78.1	70.02	68.3	54.68
	50	90.2	49.2	82.44	75.5	38.2	67.38

A.2 Visualization of the word representations

In Figures A.1 and A.2, we present a 2D visualization of the word embeddings for the language pair English/German, generated using the t-SNE dimensionality reduction algorithm [51], for the BAE-cr and BAE-tr models. We see that words with similar meanings are close to each other, for words in different languages *and* for words within the same language. This confirms that these models were able to learn a meaningful semantic representation for the words.

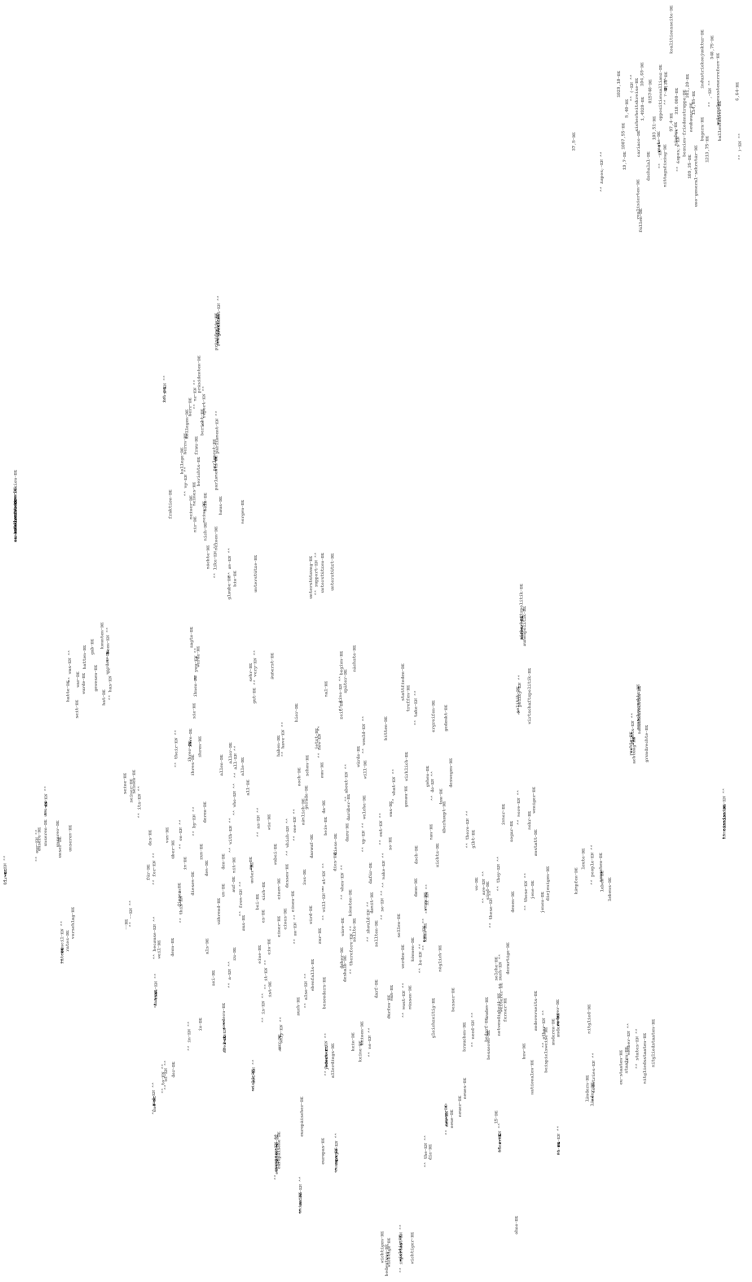


Figure A.1 – For the BAE-cr model, a t-SNE 2D visualization of the learned English/German word representations (better visualized on a computer). Words hyphenated with "EN" and "DE" are English and German words respectively.

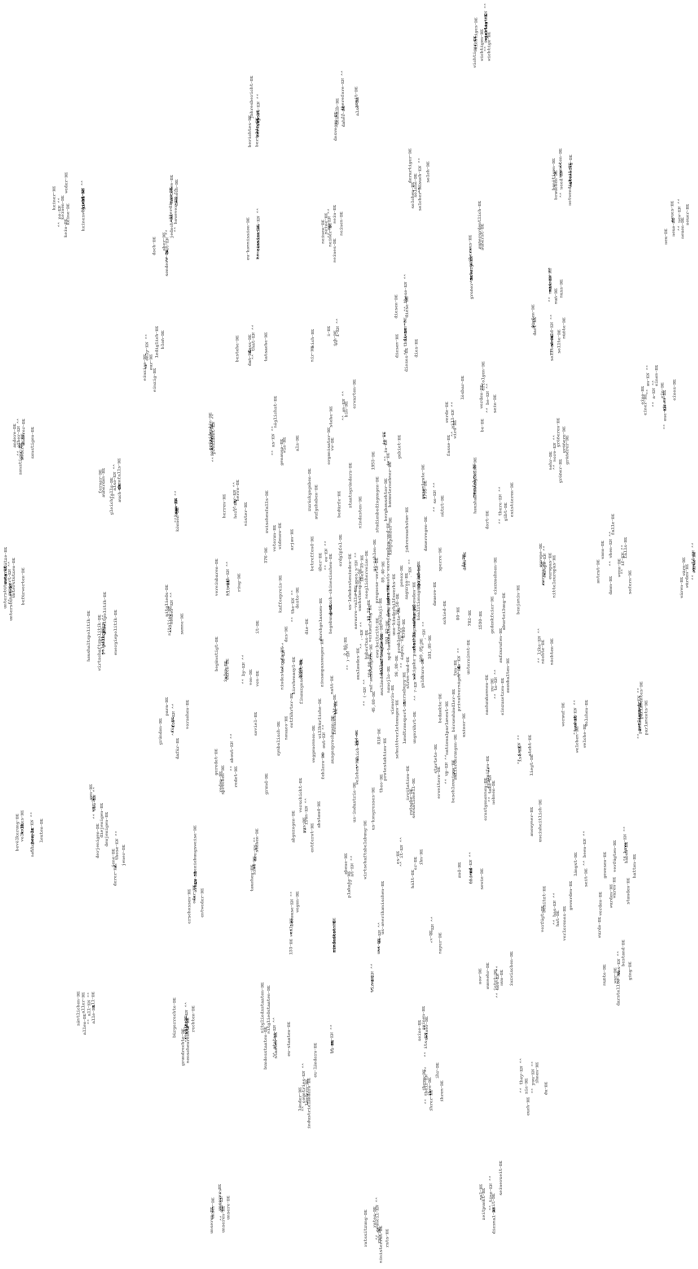


Figure A.2 – For the BAE-tr model, a t-SNE 2D visualization of the learned English/German word representations (better visualized on a computer). Words hyphenated with "EN" and "DE" are English and German words respectively.

Bibliography

[1] Image of dirichlet distribution. http://cos.name/2013/01/lda-math-gamma-function/dirichlet-distribution/. Accessed: 2016-02-03.

[2] S. Ahn, A. Korattikara, and M. Welling. Bayesian Posterior Sampling via Stochastic Gradient Fisher Scoring. In *Proceedings of the 29th International Conference on Machine Learning (ICML 2012)*, pages 1591–1598, 2012.

[3] Y. Bengio. Learning deep architectures for ai. *Foundations and trends® in Machine Learning*, 2(1):1–127, 2009.

[4] Y. Bengio and S. Bengio. Modeling High-Dimensional Discrete Data with Multi-Layer Neural Networks. In *Advances in Neural Information Processing Systems 12 (NIPS 1999)*, pages 400–406. MIT Press, 2000.

[5] Y. Bengio, R. Ducharme, P. Vincent, and C. Janvin. A neural probabilistic language model. *The Journal of Machine Learning Research*, 3:1137–1155, 2003.

[6] Y. Bengio, H. Schwenk, J.-S. Senécal, F. Morin, and J.-L. Gauvain. Neural probabilistic language models. In *Innovations in Machine Learning*, pages 137–186. Springer, 2006.

[7] E. L. Bird Steven and E. Klein. *Natural Language Processing with Python*. O'Reilly Media Inc., 2009.

[8] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent Dirichlet Allocation. *Journal of Machine Learning Research*, 3(4-5):993–1022, 2003.

BIBLIOGRAPHY

[9] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *the Journal of machine Learning research*, 3:993–1022, 2003.

[10] S. Chandar. Correlational neural networks for common representation learning. Mémoire de maîtrise, Department of Computer Science and Engineering, IIT Madras., 2015.

[11] J. Chang, J. Boyd-Graber, S. Gerrish, C. Wang, and D. Blei. Reading Tea Leaves: How Humans Interpret Topic Models. In *Advances in Neural Information Processing Systems 22 (NIPS 2009)*, pages 288–296, 2009.

[12] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa. Natural Language Processing (Almost) from Scratch. *Journal of Machine Learning Research*, 12, 2011.

[13] G. E. Dahl, R. P. Adams, and H. Larochelle. Training Restricted Boltzmann Machines on Word Observations. In *Proceedings of the 29th International Conference on Machine Learning (ICML 2012)*, pages 679–686, 2012.

[14] D. Das and S. Petrov. Unsupervised part-of-speech tagging with bilingual graph-based projections. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 600–609, Portland, Oregon, USA, June 2011.

[15] Y. Dauphin, X. Glorot, and Y. Bengio. Large-Scale Learning of Embeddings with Reconstruction Sampling. In *Proceedings of the 28th International Conference on Machine Learning (ICML 2011)*, pages 945–952. Omnipress, 2011.

[16] S. T. Dumais, T. A. Letsche, M. L. Littman, and T. K. Landauer. Automatic cross-language retrieval using latent semantic indexing. *AAAI spring symposium on cross-language text and speech retrieval*, 15:21, 1997.

[17] J. Eisenstein, A. Ahmed, and E. P. Xing. Sparse Additive Generative Models of Text. In *Proceedings of the 28th International Conference on Machine Learning (ICML 2011)*, pages 1041–1048. Omnipress, 2011.

[18] M. Faruqui and C. Dyer. Improving vector space word representations using multilingual correlation. In *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics*, pages 462–471, Gothenburg, Sweden, April 2014.

[19] J. Gao, X. He, W.-t. Yih, and L. Deng. Learning continuous phrase representations for translation modeling. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 699–709, Baltimore, Maryland, June 2014.

[20] X. Glorot, A. Bordes, and Y. Bengio. Domain Adaptation for Large-Scale Sentiment Classification: A Deep Learning Approach. In *Proceedings of the 28th International Conference on Machine Learning (ICML 2011)*, pages 513–520. Omnipress, 2011.

[21] K. M. Hermann and P. Blunsom. Multilingual Distributed Representations without Word Alignment. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2014.

[22] K. M. Hermann and P. Blunsom. Multilingual models for compositional distributed semantics. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014, June 22-27, 2014, Baltimore, MD, USA, Volume 1: Long Papers*, pages 58–68, 2014.

[23] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *Signal Processing Magazine, IEEE*, 29(6):82–97, 2012.

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

[25] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[26] D. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[27] A. Klementiev, I. Titov, and B. Bhattarai. Inducing Crosslingual Distributed Representations of Words. In *Proceedings of the International Conference on Computational Linguistics*, 2012.

[28] P. Koehn. Europarl: A parallel corpus for statistical machine translation. In *MT Summit*, 2005.

[29] P. Koehn, H. Hoang, A. Birch, C. Callison-Burch, M. Federico, N. Bertoldi, B. Cowan, W. Shen, C. Moran, R. Zens, et al. Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th annual meeting of the ACL on interactive poster and demonstration sessions*, pages 177–180. Association for Computational Linguistics, 2007.

[30] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[31] R. Kuhn and R. D. Mori. A cache-based natural language model for speech recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(6):570–583, june 1990.

[32] H. Larochelle and S. Lauly. A Neural Autoregressive Topic Model. In *Advances in Neural Information Processing Systems 25 (NIPS 25)*, pages 2708–2716, 2012.

[33] H. Larochelle and I. Murray. The Neural Autoregressive Distribution Estimator. In *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics (AISTATS 2011)*, volume 15, pages 29–37, Ft. Lauderdale, USA, 2011. JMLR W&CP.

[34] B. Liu. *Sentiment Analysis and Opinion Mining*. Synthesis Lectures on Human Language Technologies. Morgan & Claypool Publishers, 2012.

[35] R. Mihalcea, C. Banea, and J. Wiebe. Learning multilingual subjective language via cross-lingual projections. In *Proceedings of the 45th Annual Meeting*

of the Association of Computational Linguistics, pages 976–983, Prague, Czech Republic, June 2007.

[36] T. Mikolov, Q. Le, and I. Sutskever. Exploiting Similarities among Languages for Machine Translation. Rapport technique, arXiv, 2013.

[37] A. Mnih and K. Gregor. Neural variational inference and learning in belief networks. *arXiv preprint arXiv:1402.0030*, 2014.

[38] A. Mnih and G. Hinton. Three new graphical models for statistical language modelling. In *Proceedings of the 24th international conference on Machine learning*, pages 641–648. ACM, 2007.

[39] A. Mnih and G. E. Hinton. A Scalable Hierarchical Distributed Language Model. In *Advances in Neural Information Processing Systems 21 (NIPS 2008)*, pages 1081–1088, 2009.

[40] F. Morin and Y. Bengio. Hierarchical Probabilistic Neural Network Language Model. In *Proceedings of the 10th International Workshop on Artificial Intelligence and Statistics (AISTATS 2005)*, pages 246–252. Society for Artificial Intelligence and Statistics, 2005.

[41] S. Padó and M. Lapata. Cross-lingual annotation projection for semantic roles. *Journal of Artificial Intelligence Research (JAIR)*, 36:307–340, 2009.

[42] J. C. Platt, K. Toutanova, and W.-t. Yih. Translingual document representations from discriminative projections. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, EMNLP '10, pages 251–261, Stroudsburg, PA, USA, 2010.

[43] R. Salakhutdinov and G. Hinton. Replicated Softmax: an Undirected Topic Model. In *Advances in Neural Information Processing Systems 22 (NIPS 2009)*, pages 1607–1614, 2009.

[44] E. Saund. Dimensionality-reduction using connectionist networks. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 11(3):304–314, 1989.

[45] R. Socher, J. Bauer, C. D. Manning, and N. Andrew Y. Parsing with compositional vector grammars. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 455–465, Sofia, Bulgaria, August 2013.

[46] N. Srivastava, R. R. Salakhutdinov, and G. E. Hinton. Modeling documents with deep boltzmann machines. *arXiv preprint arXiv:1309.6865*, 2013.

[47] M. Steyvers and T. Griffiths. Probabilistic topic models. *Handbook of latent semantic analysis*, 427(7):424–440, 2007.

[48] K. Toutanova, D. Klein, C. D. Manning, and Y. Singer. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics*, NAACL '03, pages 173–180, 2003.

[49] J. Turian, L. Ratinov, and Y. Bengio. Word representations: A simple and general method for semi-supervised learning. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics (ACL2010)*, pages 384–394, 2010.

[50] B. Uria, I. Murray, and H. Larochelle. A deep and tractable density estimator. *JMLR: W&CP*, 32(1):467–475, 2014.

[51] L. van der Maaten and G. E. Hinton. Visualizing Data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, 2008.

[52] X. Wan. Co-training for cross-lingual sentiment classification. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 235–243, Suntec, Singapore, August 2009.

[53] M. Welling and Y. W. Teh. Bayesian Learning via Stochastic Gradient Langevin Dynamics. In *Proceedings of the 28th International Conference on Machine Learning (ICML 2011)*, pages 681–688. Omnipress, 2011.

[54] D. Yarowsky and G. Ngai. Inducing multilingual pos taggers and np bracketers via robust projection across aligned corpora. In *Proceedings of the second meeting of the North American Chapter of the Association for Computational Linguistics on Language technologies*, pages 1–8, Pennsylvania, 2001.

[55] W.-t. Yih, K. Toutanova, J. C. Platt, and C. Meek. Learning discriminative projections for text similarity measures. In *Proceedings of the Fifteenth Conference on Computational Natural Language Learning*, CoNLL '11, pages 247–256, Stroudsburg, PA, USA, 2011.

[56] Y. Zheng, Y.-J. Zhang, and H. Larochelle. A deep and autoregressive approach for topic modeling of multimodal data. *TPAMI*, 2015.

[57] W. Y. Zou, R. Socher, D. Cer, and C. D. Manning. Bilingual Word Embeddings for Phrase-Based Machine Translation. In *Empirical Methods in Natural Language Processing*, 2013.