Université de Montréal

Accélérer l'entraînement d'un modèle non-paramétrique de densité non normalisée par échantillonnage aléatoire

par
Jean-Sébastien Senécal
Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Mémoire présenté à la Faculté des études supérieures
en vue de l'obtention du grade de
Maîtrise es Sciences
en Informatique

Avril 2003

Université de Montréal

Faculté des études supérieures

Ce mémoire intitulé:

Accélérer l'entraînement d'un modèle non-paramétrique de densité non normalisée par échantillonnage aléatoire

présenté par:

Jean–Sébastien Senécal

a été évalué par un jury composé des personnes suivantes:

Pierre L'Écuyer
(président-rapporteur)

Yoshua Bengio
(directeur de recherche)

Balázs Kégl
(membre du jury)

Mémoire accepté le:

_____

Résumé

Des travaux récents en modélisation statistique de la langue ont montré qu'il était possible d'entraîner un réseau de neurones à prédire la probabilité de séquences de mots. Cela permet de réduire considérablement l'erreur si l'on compare à des modèles traditionnels. Cependant, l'entraînement du modèle demande de rétro-propager le gradient autant de fois qu'il y a de mots dans le vocabulaire et ce, pour chaque exemple de l'ensemble d'entraînement, ce qui est très coûteux en temps de calcul.

Dans ce mémoire, nous proposons d'estimer le gradient avec des méthodes d'échantillonnage afin d'accélérer l'entraînement. Nous discutons de différentes méthodes d'échantillonnage et nous les comparons sur des problèmes standard. Nous montrons que des accélérations très significatives peuvent être obtenues en utilisant l'échantillonnage pondéré et une distribution instrumentale adaptative.

Mots-clés : Apprentissage-machine, modèles non-paramétriques de densité non normalisée, méthodes de Monte Carlo, produits d'experts, réseaux de neurones artificiels, modélisation de la langue naturelle.

Summary

Previous work on statistical language modeling has shown that it is possible to train a feed-forward neural network to approximate probabilities over sequences of words, resulting in significant error reduction when compared to standard baseline models. However, in order to train the model on the maximum likelihood criterion, one has to make, for each example, as many network passes as there are words in the vocabulary ; that is very computationaly expansive.

In this thesis, we propose to use sampling approximations to the gradient in order to accelerate training. We discuss various sampling methods and test them on standard problems. We show that very significant speed-ups can be obtained by making use of importance sampling with an adaptive proposal (sampling) distribution.

Keywords : Machine learning, nonparametric unnormalized probabilistic models, Monte Carlo methods, products of experts, artificial neural networks, natural language modeling.

Table des matières

Liste des tableaux

Liste des figures

Liste des algorithmes

À Paul

Remerciements

Dans l'ordre habituel, j'aimerais remercier les personnes suivantes.

Yoshua Bengio, mon directeur de recherche. Travailler avec lui est agréable et enrichissant. Je tiens à le remercier pour son soutien, son dévouement et pour le caractère contagieux de sa grande passion pour la recherche.

Patrice D'Amours, pour une oreille attentive, un précieux soutien dans des moments difficiles, des discussions enflammées et, surtout, une amitié sincère.

Mireille Gaubiac qui a toujours été là pour me changer les idées dans les moments plus difficiles mais, surtout, un modèle de ténacité et d'acharnement qui ne cesse de m'inspirer. Merci d'exister, même si c'est pas ta faute.

Édithe Gaudet, ma mère, pour s'être inquiétée de mon bien-être et m'avoir forcé à sortir un peu de ma bulle, pour respirer l'air frais de Morin-Heights.

Christian Jauvin, pour m'avoir réellement intéressé au domaine du langage mais, surtout, pour ces « café-mémoires » hebdomadaires qui ont été le terreau fertile d'une amitié authentique.

David Laroche, avec qui j'ai habité pendant toute ma Maîtrise, pour un quotidien délirant et une amitié du tonerre.

Paul Senécal, mon père, à qui je dois mon goût pour les mathématiques. Ton courage et ta force de vivre ont été pour moi une source inépuisable d'inspiration et d'énergie.

Merci enfin aux organismes et instituts qui m'ont subventionné pendant la durée de mes études de Maîtrise, soit le CRSNG, le FCAR et l'IDIAP.

CHAPITRE 1

Introduction

La modélisation du langage reste aujourd'hui l'un des problèmes les plus stimulants en intelligence artificielle. La haute dimensionalité intrinsèque à la langue rend très difficile son analyse statistique. Par exemple, si on souhaite modéliser la probabilité jointe de dix mots consécutifs issus d'un vocabulaire de 100 000 mots, il y a potentiellement $100\ 000^{10} - 1 = 10^{50} - 1$ paramètres libres.

Dans (BENGIO, DUCHARME, VINCENT et JAUVIN 2003), les auteurs proposent d'utiliser un réseau de neurones artificiel pour apprendre la probabilité jointe d'une fenêtre de mots consécutifs. Leurs expériences montrent qu'il est possible d'atteindre un taux d'erreur significativement inférieur à celui de modèles plus traditionnels (e.g.,n-gramme) dans des expériences standard [1]. Malheureusement, comme nous le verrons, l'entraînement de ce réseau de neurones est très coûteux en temps de calcul. Ce coût est en effet proportionnel au nombre de classes, i.e., le nombre de mots de vocabulaire dans ce cas particulier.

[1] Dans ce mémoire, nous supposons que le lecteur est familier avec les notions fondamentales en traitement de la langue naturelle. Pour une révision rapide de ces notions, prière de se référer à l'annexe B.

Dans le présent chapitre, nous révisons d'abord quelques concepts de base en apprentissage-machine. Nous introduisons les réseaux de neurones d'un point de vue général. Nous expliquons ensuite la méthode d'entraînement par descente de gradient et la rétro-propagation. Nous terminons par une introduction au problème qui nous intéresse : accélérer l'entraînement d'un réseau de neurones dans un problème avec beaucoup de classes.

1.1 Quelques concepts de base en théorie de l'apprentissage

Le domaine de l'apprentissage-machine tente d'aborder plusieurs problèmes : classification, régression, prise de décision et estimation de densité. Nous présentons ici le cas qui nous intéresse : l'estimation d'une densité conditionnelle avec plusieurs classes (qui s'apparente de très près à un problème de classification) dans un contexte d'apprentissage supervisé.

Nous souhaitons estimer la densité conditionnelle inconnue $p(y|\mathbf{x})$ où y est une classe tirée d'un ensemble $\mathcal{C}$ fini et $\mathbf{x} \in \mathcal{X}$ est une variable conditionnelle (généralement un vecteur). Sans perte de généralités, posons $\mathcal{C} = \{1, \ldots, K\}$. Pour estimer cette densité, nous disposons seulement d'un ensemble de points $D_N = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$ tirés de la distribution $p(\mathbf{x}, y)$. Les classes $y_t \in \mathcal{C}$ associées à chaque $\mathbf{x}_t \in \mathcal{X}$ étant spécifiées dans l'ensemble de donnée, ce problème en est un d'*apprentissage supervisé.*

Soit $\mathcal{F}$ une classe de fonctions candidates pour estimer la densité cible. L'objectif de l'apprentissage est de trouver, parmi ces fonctions, celle ayant la distance minimale avec $p(y|\mathbf{x})$ (ou, à défaut, une distance aussi petite que possible). Cette notion de distance est formalisée par la notion de *risque espéré* (ou *erreur de généralisation*).

Soit $L(f, \mathbf{x}, y)$ la *perte* de la fonction $f \in \mathcal{F}$ sur le point $(\mathbf{x}, y)$. La perte est une fonction qui assigne une pénalité à la fonction f sur le point de donnée

$\mathbf{x}$ étant donné une sortie espérée y. Le risque espéré est défini par

$$R(f) = \int L(f, \mathbf{x}, y) p(\mathbf{x}, y) d(\mathbf{x}, y). \tag{1.1}$$

Bien entendu, il est impossible de calculer cette valeur sans connaître $p(\mathbf{x}, y)$. Par conséquent, on estime généralement le risque espéré par la perte moyenne sur les points d'un ensemble de données $D_N = \{(\mathbf{x}_t, y_t)\}_{t=1}^N$; on appele cet estimé le *risque empirique* :

$$\hat{R}(f, D_N) = \frac{1}{N} \sum_{t=1}^{N} L(f, \mathbf{x}_t, y_t). \tag{1.2}$$

L'objectif de l'apprentissage sera donc de trouver la fonction $f \in \mathcal{F}$ qui minimise le risque empirique sur un ensemble donné.

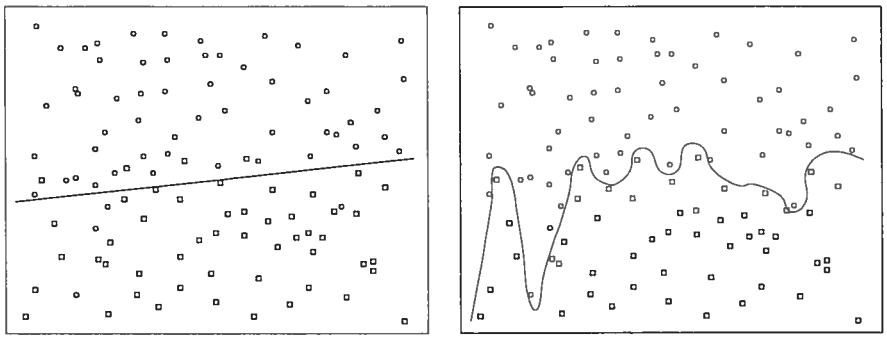
En estimation de densité, nous utilisons la *log-vraisemblance négative* comme fonction de perte, i.e., $L(f, \mathbf{x}, y) = -log f(y|\mathbf{x})$. Il est aisé de montrer que dans ce cas, minimiser le risque empirique est équivalent à maximiser la vraisemblance.

L'un des problèmes bien connus en apprentissage-machine est celui du *surentraînement* (« overfitting »). Au cours de l'entraînement, on tentera de minimiser le risque empirique $\hat{R}(f, D_N)$ – où D_N est appelé *l'ensemble d'entraînement* – de façon itérative, i.e., en choisissant itérativement des fonctions $f \in \mathcal{F}$ ayant un risque empirique $\hat{R}(f, D_N)$ de plus en plus petit. À mesure qu'on entraîne le modèle, ce risque diminue de façon monotone jusqu'à atteindre un plateau. Si la classe de fonctions $\mathcal{F}$ de laquelle est tirée f est assez complexe, on peut théoriquement atteindre un plateau aussi bas qu'on le souhaite. Or, si on évalue $\hat{R}(f, D_M)$ sur un ensemble disjoint D_M de points tirés de la même distribution que D_N, on se rend compte qu'à un certain moment, $\hat{R}(f, D_M)$ commence à augmenter bien que $\hat{R}(f, D_N)$ diminue. Nous appelons ce point de l'entraînement le *point de surentraînement.* Ce qui se passe, c'est que la fonction f devient de moins en moins lisse et perd ses capacités de généralisation. Elle approxime presque parfaitement l'ensemble d'entraînement D_N, mais elle ne généralise pas à d'autres points de donnée.

C'est pourquoi, traditionnelement, on divise les données en trois ensembles disjoints.

1. L'*ensemble d'entraînement* est utilisé pour entraîner le modèle, i.e., choisir la fonction f optimale.
2. L'*ensemble de validation* est utilisé pour trouver le point de surentraînement. Généralement, on arrêtera l'entraînement à ce point et ce, même si le risque sur l'ensemble d'entraînement continue de diminuer.
3. L'*ensemble de test* est utilisé pour évaluer le modèle. On peut en fait voir le risque empirique sur l'ensemble de test comme un estimé *non biaisé* de l'erreur de généralisation.

Les figures 1(a) et 1(b) illustrent le problème du surentraînement dans une tâche de classification simple (deux classes). La figure 1(a) présente une simple fonction linéaire pour séparer l'espace des points ; cette fonction commet quelques erreurs. Par contre, la figure 1(b) montre une fonction polynomiale ; celle-ci ne commet aucune erreur mais ne généralise pas bien, elle est *surentraînée.*



(a) Fonction linéaire

(b) Fonction polynomiale (surentraînement)

Figure 1 – Problème simple de classification binaire

1.2 Réseaux de neurones artificiels

Les réseaux de neurones artificiels sont une classe de fonctions largement utilisée en apprentissage-machine. Ils ont l'avantage d'être des approximateurs universels (ils peuvent approximer n'importe quelle fonction continue) ; de plus, il existe des méthodes connues et éprouvées pour les optimiser.

L'histoire des réseaux de neurones artificiels débute en 1957. Rosenblatt publie un article dans lequel il décrit un modèle, baptisé *perceptron*, permettant de classifier des points de donnée (ROSENBLATT 1957). Ce modèle est en fait un simple séparateur linéaire. Soit $\mathbf{x} = (x_i) \in \mathbf{R}^n$ un point de donnée et $y \in \{-1, +1\}$ une classe binaire associée. Le perceptron est une fonction de la forme

$$f(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + b \tag{1.3}$$

où $\mathbf{w} = (w_i) \in \mathbf{R}^n$ est un vecteur de paramètres (poids) et $b \in \mathbf{R}$ est un paramètre appelé le *biais*. Pour classifier un point $\mathbf{x}$, on lui attribue la classe -1 si $f(\mathbf{x}) \leq 0$ et $+1$ si $f(\mathbf{x}) > 0$. Le modèle peut être entraîné par *descente de gradient* (voir section 1.3) sur les paramètres.

Mais cette fonction n'arrive pas à classifier des points qui ne sont pas linéairement séparables (MINSKY et PAPERT 1969). Comment résoudre ce problème ?

Une solution consiste à superposer plusieurs « couches » de perceptrons. Chaque couche consiste en un certain nombre de perceptrons activés de façon parallèle et indépendante, et dont les sorties sont soumises à une fonction non-linéaire, par exemple une tangente hyperbolique ($\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$). Les couches inférieures sont donc utilisées pour projeter les points de donnée $\mathbf{x}$ dans un autre espace, de façon non-linéaire. Cette classe de modèles, appelée *réseau de neurones* (ou *perceptron multi-couches*), est décrite ci-dessous de façon formelle dans le cas où il n'y a que deux couches.

Nous présentons d'abord le cas où il n'y a que deux classes. Soit $\mathbf{x} = (x_i) \in \mathbf{R}^n$ et $y \in \{-1, +1\}$. Un réseau de neurones artificiel à une couche

avec n_h neurones cachées est une classe de fonctions de la forme

$$f(\mathbf{x}) = \mathbf{w} \cdot \mathbf{a} + b \tag{1.4}$$

où $\mathbf{w} = (w_i) \in \mathbf{R}^n$ sont les « poids de sortie » et $b \in \mathbf{R}$ est le « biais de sortie » . Le vecteur $\mathbf{a} = (a_i) \in \mathbf{R}^{n_h}$ est défini par

$$\mathbf{a} = h(\mathbf{W} \cdot \mathbf{x} + \mathbf{d}) \tag{1.5}$$

où $\mathbf{W} = (w_{ij}) \in \mathbf{R}^{(n_h \times n)}$ sont les « poids cachée », $\mathbf{d} \in \mathbf{R}^{n_h}$ sont les « biais cachés » et $h(\cdot)$ est une fonction non-linéaire (e.g., tangente hyperbolique) [2].

Il est démontré (BISHOP 1995, section 4.3.2) qu'un réseau de neurones avec suffisament de neurones cachées peut approximer n'importe quelle fonction continue. Nous verrons plus tard (section 1.4) comment nous pouvons généraliser pour faire de la classification avec plus de deux classes. Mais pour l'instant, voyons comment il est possible d'entraîner un réseau de neurones à effectuer une classification binaire.

1.3 Descente de gradient

L'objectif de l'apprentissage, rappelons-le, est de minimiser le risque empirique (1.2) sur de nouveaux points de donnée (par exemple, sur un ensemble de test disjoint de l'ensemble d'entraînement). Pour ce faire, on minimise ce risque sur un ensemble d'entraînement et on s'arrête au point de surentraînement. Mais comment choisir la fonction $f \in \mathcal{F}$ qui minimise le risque ? Cela dépend du problème auquel on s'attaque et du modèle utilisé, i.e., de la classe $\mathcal{F}$. Cette section décrit l'algorithme le plus couramment utilisé pour entraîner un réseau de neurones : la descente de gradient.

Soit $D_N = \{(\mathbf{x}_t, y_t)\}_{t=1}^n$ où $\mathbf{x}_t \in \mathbf{R}^n$ et $y_t \in \{-1, +1\}$ l'ensemble d'en-

[2] Nous nous permettons un léger abus de notation ici, car une fonction n'est généralement pas définie sur un vecteur. Pour clarifier, nous entendons par $h(\mathbf{v})$ (où $\mathbf{v} = (v_1, \ldots, v_n)^t$) le vecteur $(h(v_1), \ldots, h(v_n))^t$.

traînement et soit $L(f, \mathbf{x}, y)$ la perte de f sur un point de donnée $(\mathbf{x}, y)$. Soit un réseau de neurones à sortie $f(\mathbf{x})$ et à n_h neurones cachées tel que défini dans les équations (1.4) et (1.5). Soit $\theta = (\mathbf{w}, b, \mathbf{W}, \mathbf{d})$ les paramètres du modèle et soit m le nombre de paramètres. La fonction f se présente en fait comme une fonction paramétrée $f = f_\theta$. L'ensemble $\mathcal{F} = \{f_\theta | \theta \in \mathbf{R}^m\}$ représente l'ensemble des fonctions pouvant être représentées par le réseau de neurones. Choisir une fonction $f_\theta \in \mathcal{F}$ est équivalent à spécifier un vecteur de paramètres θ. Le problème de minimisation consiste donc à trouver

$$\theta^* = \underset{\theta \in \mathbf{R}^m}{\operatorname{argmin}} \hat{R}(f_\theta, D_N) \tag{1.6}$$

où D_N est un ensemble d'entraînement et $\hat{R}$ est le risque empirique (1.2). [3] [4].

Malheureusement, il n'existe généralement pas de solution analytique à ce problème de minimisation. Par conséquent, on a généralement recours à une approximation par *descente de gradient*. Sous certaines conditions, cette approximation est garantie de converger vers un minimum local, mais pas vers le minimum global (Luo 1991).

La descente de gradient est basée sur une optimisation locale. Supposons que les paramètres θ soient fixés à une certaine valeur θ_a à un certain temps a de l'optimisation. Comment choisir θ_{a+1} à partir de θ_a tels que $\hat{R}(f_{\theta_{a+1}}, D_N) \leq \hat{R}(f_{\theta_a}, D_N)$? En opérant une légère modification sur les éléments de θ_t dans la direction où $\hat{R}(f_{\theta_a}, D_N)$ décroît le plus. Cette direction est donnée par le gradient de $\hat{R}(f_{\theta_a}, D_N)$ par rapport aux paramètres θ_a, ce

[3] Nous faisons ici l'assomption qu'il n'y a qu'un seul minimum global, ce qui n'est pas nécessairement vrai. Mais comme nous le verrons, nous n'avons pas la prétention de trouver un minimum global, donc nous pouvons supposer qu'il n'en existe qu'un sans perte de généralité par rapport à ce qui suit.

[4] Les fonction argmin et argmax seront utilisées tout au long de ce mémoire, par conséquent nous les définissons ici. Soit f une fonction définie sur l'ensemble $\mathcal{X}$. Soit $\mathcal{Y} = \{f(x) | x \in \mathcal{X}\}$. S'il n'existe qu'un seul point x_{min} tel que $f(x_{min}) \leq y$, $\forall y \in \mathcal{Y}$, alors $\operatorname{argmin}_{x \in \mathcal{X}} f(x) = x_{min}$. S'il existe plus d'un point, la fonction n'est pas définie. De façon similaire, la fonction argmax est définie s'il n'existe qu'un seul point x_{max} tel que $f(x_{max}) \geq y$, $\forall y \in \mathcal{Y}$; alors $\operatorname{argmax}_{x \in \mathcal{X}} f(x) = x_{max}$.

qui nous donne une règle de mise-à-jour pour θ_{a+1} :

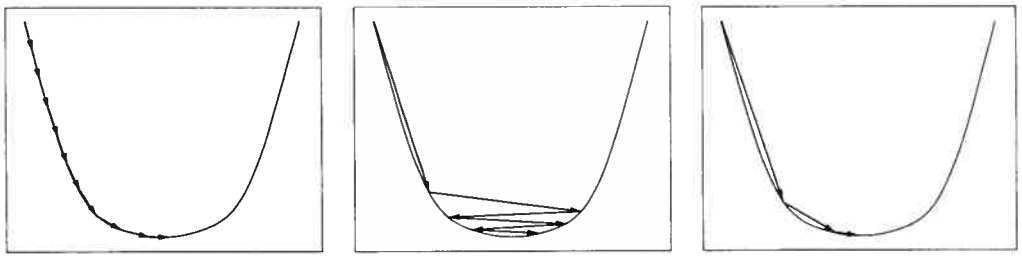
$$\theta_{a+1} = \theta_a - \eta_a \frac{\partial \hat{R}(f_{\theta_a}, D_N)}{\partial \theta_a} \tag{1.7}$$

où $\eta_a > 0$ est appelé le *taux d'apprentissage* (BISHOP 1995, section 7.5). Pour garantir la convergence vers un minimum local, ce taux doit être tel que

1. $\sum_{a=0}^{\infty} \eta_a = \infty$
2. $\sum_{a=0}^{\infty} \eta_a^2 < \infty$.

Une fonction répondant à ces conditions, largement utilisée en descente de gradient avec des réseaux de neurones, est $\eta_a = \frac{\eta_0}{1+\epsilon a}$ où $\eta_0 > 0$ est le *taux d'apprentissage initial* et $\epsilon > 0$ est appelé la *constante de décroissement.*

Les figures 2(a), 2(b) et 2(c) illustrent l'importance du choix d'un bon taux d'apprentissage. Si ce taux est trop petit (figure 2(a)) on atteindra un minimum mais la convergence sera très longue. S'il est trop gros (figure 2(b)) on risque de ne pas converger vers un minimum. Un choix plus raisonnable consiste à choisir un taux de plus en plus petit (figure 2(c)).



(a) Taux d'apprentissage trop petit

(b) Taux d'apprentissage trop gros

(c) Taux d'apprentissage raisonnable

Figure 2 – Descente de gradient sur une fonction de coût quadratique

Dans les faits, si on utilise cette méthode telle quelle, les paramètres θ_a ont tendance à devenir très « lourds », i.e., leur norme augmente en même temps que t. Ceci rend la fonction moins lisse, ce qui a pour effet d'augmenter l'erreur de généralisation. Afin d'éviter ce problème, on recourt généralement à une astuce toute simple qui consiste à diminuer la norme des poids à chaque

mise-à-jour. La règle de mise-à-jour devient

$$\theta_{a+1} = \theta_a - \eta_a \frac{\partial \hat{R}(f_{\theta_a}, D_N)}{\partial \theta_a} - \lambda \theta_a \tag{1.8}$$

où $\lambda > 0$ est appelé la *pénalité sur les poids* (BISHOP 1995, section 9.2) (« weight decay »).

Afin d'accélérer la descente de gradient, on utilisera généralement une approximation stochastique à $\frac{\partial \hat{R}(f_{\theta_a}, D_N)}{\partial \theta_a}$. Soit un point $(\mathbf{x}, y)$ tiré iid de l'ensemble D_N. Alors le gradient sur la perte $L(f_\theta, \mathbf{x}, y)$ est un estimateur non biaisé de $\frac{\partial \hat{R}(f_\theta, D_N)}{\partial \theta}$, i.e.,

$$\mathrm{E}\left[\frac{\partial L(f_\theta, \mathbf{x}, y)}{\partial \theta}\right] = \frac{\partial \hat{R}(f_\theta, D_N)}{\partial \theta}. \tag{1.9}$$

Cette approximation est à la base d'une variante bien connue de la descente de gradient, appelée *descente de gradient stochastique*. À chaque cycle de l'algorithme – appelé *époque* – on effectue un descente de gradient sur chaque exemple de l'ensemble d'entraînement. À la fin de chaque époque, on vérifie l'évolution du risque empirique par rapport à la dernière époque. La procédure est arrêtée quand la différence entre le risque actuel et celui calculé lors de la dernière époque est inférieur à un seuil $\epsilon > 0$. L'algorithme 1 résume la procédure.

Le gradient sur les paramètres est calculé par *rétro-propagation*, une simple application de la règle de chaîne aux réseaux de neurones (RUMELHART, HINTON et WILLIAMS 1986). Posons $\gamma = \frac{\partial L(f, \mathbf{x}, y)}{\partial f(\mathbf{x})}$. Les gradients sur la couche de sortie sont :

$$\begin{aligned} \frac{\partial L(f, \mathbf{x}, y)}{\partial v_i} &= \gamma a_i \\ \frac{\partial L(f, \mathbf{x}, y)}{\partial b} &= \gamma. \end{aligned} \tag{1.10}$$

Algorithme 1 Entraînement d'un réseau de neurones par descente de gradient stochastique

(1) **fonction** *descenteDeGradientStochastique*($D_N = \{(\mathbf{x}_t, y_t)\}_{t=1}^N$)
(2) $\theta_0 \leftarrow$ *init()* {Initialisation des paramètres}
(3) $r_0 \leftarrow \hat{R}(f_{\theta_0}, D_N)$ {Risque empirique}
(4) $a \leftarrow 0$ {Âge du réseau, i.e., nombre total d'exemples vus}
(5) $e \leftarrow 0$ {Nombre d'époques}
(6) **répéter**
(7) **pour** $t \leftarrow 1$ **à** N **faire**
(8) $\theta_{a+1} \leftarrow \theta_a - \eta_a \frac{\partial L(f_{\theta_a}, \mathbf{x}_t, y_t)}{\partial \theta_a}$
(9) $a \leftarrow a + 1$
(10) $e \leftarrow e + 1$
(11) $r_e \leftarrow \hat{R}(f_{\theta_a}, D_N)$
(12) **jusqu'à** $|r_e - r_{e-1}| < \epsilon$

Posons $\mathbf{s} = (s_i) = \mathbf{W} \cdot \mathbf{x} + \mathbf{d}$ (i.e., $\mathbf{a} = h(\mathbf{s})$). Posons

$$\gamma_i = \frac{\partial L(f, \mathbf{x}, y)}{\partial s_i} = \frac{\partial L(f, \mathbf{x}, y)}{\partial f(\mathbf{x})} \frac{\partial f(\mathbf{x})}{\partial a_i} \frac{\partial a_i}{\partial s_i} = \gamma v_i h'(s_i).$$

Alors les gradients de la couche cachée sont :

$$\begin{aligned} \frac{\partial L(f, \mathbf{x}, y)}{\partial w_{ij}} &= \gamma_i x_j \\ \frac{\partial L(f, \mathbf{x}, y)}{\partial d_i} &= \gamma_i. \end{aligned} \tag{1.11}$$

On appele cette méthode *rétro-propagation* parce que le gradient est « propagé » de la couche de sortie aux couches inférieures.

1.4 Réseaux de neurones artificiels avec K classes

Nous avons présenté les différents algorithmes permettant d'entraîner un réseau de neurones artificiel dans une tâche de classification binaire. Com-

ment généraliser au cas qui nous intéresse : la classification avec K classes $(K > 2)$?

Soit $D_N = \{(\mathbf{x}_t, y_t)\}_{t=1}^N$ un ensemble d'entraînement avec $\mathbf{x}_t \in \mathbf{R}^n$ et $y_t \in \{1, \ldots, K\}$. Dans la définition du réseau de neurones, remplaçons la couche de sortie à une valeur de l'équation (1.4) par K couches de sorties, chacune avec sa propre valeur d'activation :

$$f(y, \mathbf{x}) = \mathbf{V}_y \cdot \mathbf{a} + b_y \tag{1.12}$$

où $\mathbf{V}_y \in \mathbf{R}^{n_h}$ sont les poids de la sortie y et $b_y \in \mathbf{R}$ est le biais de la sortie y. Le vecteur $\mathbf{a} \in \mathbf{R}^{n_h}$ est défini dans (1.5) (il est partagé par toutes les classes).

Les sorties du réseau $f(y, \mathbf{x})$ ne sont pas nécessairement positives et elles ne représentent pas une probabilité. Afin d'obtenir une fonction de probabilité, on prend généralement l'exponentielle des sorties et on les normalise :

$$p(y|\mathbf{x}) = \frac{e^{f(y,\mathbf{x})}}{\sum_{k=1}^K e^{f(k,\mathbf{x})}}. \tag{1.13}$$

Mais quand le nombre de classes K est important (par exemple, plusieurs milliers de classes), le calcul de la fonction de normalisation $\sum_{k=1}^K e^{f(k,\mathbf{x})}$ peut devenir très coûteux en temps de calcul. De plus, le calcul du gradient sur la log-vraisemblance inverse est également très coûteux. En effet, ce gradient a la forme suivante :

$$\frac{\partial f(y, \mathbf{x})}{\partial \theta} - \frac{1}{Z} \sum_{k=1}^K e^{f(k,\mathbf{x})} \frac{\partial f(k, \mathbf{x})}{\partial \theta}$$

où $Z = \sum_{k=1}^K e^{f(k,\mathbf{x})}$ est la constante de normalisation et θ sont les paramètres du modèle.

Des développements récents dans le domaine des produits d'experts (PoE) (HINTON 1999; HINTON 2000) nous ont donné l'idée d'utiliser des méthodes d'échantillonnage (Monte Carlo) pour réduire le temps d'entraînement. Nous sommes parvenus ultimement à une accélération de l'ordre de deux magnitudes (i.e. 100 fois plus rapide) sur des entraînements standards avec un

réseau de neurones probabiliste modélisant le langage (appelé *NNLM*).

Le **chapitre 2** présente le NNLM d'un point de vue général. Il expose ses ressemblances directes avec un PoE et propose des applications autres que la modélisation du langage. Le **chapitre 3** décrit plus précisément les différentes approches qui ont été considérées pour accélérer l'entraînement avec des méthodes d'échantillonnage aléatoire. Le **chapitre 4** décrit en détail les différents algorithmes que nous avons développés. Le **chapitre 5** présente les expérimentations et les résultats. Le **chapitre 6** conclut en discutant des résultats obtenus et en proposant des travaux futurs pour poursuivre cette recherche.

L'**annexe A** contient un glossaire de termes fréquemment utilisés en apprentissage-machine, pour référence. L'**annexe B** introduit le domaine du traitement de la langue naturelle. L'**annexe C** contient des développements mathématiques et des démonstrations.

CHAPITRE 2

Modèles non-paramétriques de densité non normalisée

La distribution de Boltzmann décrit la probabilité qu'un système fermé se trouve en différents états en fonction de l'énergie. S'il y a N états distincts, la probabilité que le système soit dans l'état i d'énergie $\mathcal{E}(i)$ est égale à

$$\frac{e^{-\mathcal{E}(i)}}{Z} \tag{2.1}$$

où $Z = \sum_{j=1}^{N} e^{-\mathcal{E}(i)}$ est la *fonction de partition* [1] (aussi appelée *constante de normalisation*). Cette fonction est à l'origine de plusieurs théorèmes fondamentaux en thermodynamique.

Plusieurs chercheurs en apprentissage machine on utilisé cette forme générale pour définir des modèles de distribution probabiliste [2] : les modèles probabilistes à base énergétique (« energy-based models »), modèles log-linéaires

[1] La fonction d'énergie utilisée dans la formule de Boltzmann est définie en fonction de l'énergie potentielle $U(i)$, de la température T et d'une constante k appelée constante de Boltzmann, de telle façon que $\mathcal{E}(i) = \frac{U(i)}{kT}$.

[2] Dans la littérature, elle est également connue sous le nom de « softmax ».

ou encore *modèles de densité non normalisée*. Le NNLM, comme nous le verrons, en est un cas particulier appliqué au traitement de la langue naturelle. C'est un modèle non-paramétrique au sens où on ne fait pas d'assomption quant à la loi qui gouverne la distribution des données (e.g., loi normale) : on utilise plutôt une famille de fonctions universelles (réseaux de neurones) pour modéliser la distribution.

Dans le présent chapitre, nous commençons par faire quelques rappels sur les représentations distribuées et sur les produits d'experts, dans le contexte de la modélisation de la langue naturelle. Ensuite, nous décrivons en détails le NNLM et les problèmes entraînés par la descente de gradient.

2.1 Représentation distribuée

Selon une certaine perspective dualiste, la langue est formée d'un ensemble de *symboles* (les mots) qui sont complètement indépendants d'un ensemble de *sens*. Les mots et les sens sont des entités discrètes et séparées, reliées entre elles par des relations *d'interprétabilité* (RIEGER 1991). Selon cette théorie, déterminer la probabilité d'une phrase consiste à faire une représentation structurée des interactions entre les mots et les sens. Par exemple, un graphe dans lequel chaque noeud représente un mot ou un sens et chaque arc, une relation d'interprétation. C'est ce qu'on appelle une *représentation symbolique* ou *représentation locale*, en ce sens que chaque élément – mot, sens ou relation – est contenu intégralement dans une seule unité (e.g., un des noeuds/arcs de l'arbre sémantique).

Ce type de représentation cause deux problèmes majeurs. D'abord, il ne peut en aucun cas modéliser le *processus* par lequel de telles structures émergent (RIEGER 1991) ; plusieurs théoriciens du langage ont pourtant démontré le rôle fondamental du processus et de l'usage dans le langage (WITTGESTEIN 1961). Ensuite, la langue se manifestant comme un phénomène en très haute dimensionalité, un problème de stockage évident se pose avec une telle représentation. Ce problème inhérent aux représentations lo-

calistes a été depuis longtemps pointé du doigt par les connectionistes des années 80 (HINTON 1986).

Selon une conception *connectioniste*, le sens peut être représenté par un ensemble d'interactions entre différentes unités représentant chacune une micro-caractéristique. C'est de l'activation conjointe de ces unités, selon une certaine configuration, qu'*émerge* le sens. C'est ce qu'on appelle une *représentation distribuée*. Par exemple, dans un NNLM, les contextes de mots sont représentés par un ensemble d'unités (les neurones) dont les valeurs dépendent d'un ensemble de paramètres réels (les poids et les biais). Les mots eux-mêmes sont représentés par des vecteurs dans un espace réel.

Cette représentation est très compacte : en effet, au lieu d'avoir à stocker chaque sens dans une case séparée, les sens sont distribués à travers un petit nombre d'unités disjointes. De plus, elle permet d'extraire de la « soupe » dimensionnelle les dépendances les plus importantes (BENGIO et BENGIO 2000; BENGIO et SENÉCAL 2003).

Le désavantage est que l'estimation des paramètres est généralement beaucoup plus difficile. La fonction à optimiser n'est généralement pas convexe et peut en fait être extrêmement complexe, nécessitant des ressources computationnelles substantielles.

2.2 Sommes vs produits

Une façon très efficace de modéliser des données en haute dimension consiste à multiplier plusieurs modèles probabilistes et à renormaliser (HINTON 1999). De cette façon, chaque modèle peut s'affairer à donner une haute probabilité aux points de l'espace des données qui satisfont à une contrainte particulière. Parmi ces points, ceux qui ne satisfont pas aux contraintes des autres modèles verront leur probabilité largement amoindrie par ces derniers.

Ce type de modèle est appelé un *produit d'experts* (PoE) (HINTON 1999).

L'équation générale est donnée par

$$P(\mathbf{x}|\theta_1,\ldots,\theta_n) = \frac{\prod_m p_m(\mathbf{x}|\theta_m)}{\sum_{\mathbf{x}'\in\mathcal{X}} \prod_m p_m(\mathbf{x}'|\theta_m)} \tag{2.2}$$

où $\mathbf{x}$ est un vecteur dans un espace discret $\mathcal{X}$, θ_m est le vecteur de paramètres de l'expert m et $p_m(\mathbf{x}|\theta_m)$ est la fonction d'activation (sortie) de l'expert m sur l'entrée $\mathbf{x}$. Il n'est pas nécessaire que cette fonction représente une probabilité, mais il faut qu'elle réponde aux deux conditions suivantes :

1. Elle doit être positive, i.e., $p_m(\mathbf{x}|\theta_m) > 0\ \forall \mathbf{x}$.
2. Il existe une constante c telle que pour tout expert m,

$$p_m(\mathbf{x}|\theta_m) = cP_m(\mathbf{x}|\theta)$$

où $P_m(\mathbf{x}|\theta)$ est une probabilité.

On a remarqué que de tels modèles fonctionnaient mieux que des modèles plus traditionnels, comme les modèles n-grammes « back-off » ou les chaînes de Markov, dans des applications de traitement de la langue naturelle (Foster 2002; Charniak 1999), connus dans la littérature sous le nom de *mixtures d'experts* :

$$P(\mathbf{x}|\theta_1,\ldots,\theta_m) = \sum_m P(m)P_m(\mathbf{x}|\theta_m) \tag{2.3}$$

où $P(m)$ est la probabilité qu'un exemple soit généré par la loi distributive $P_m(\mathbf{x}|\theta_m)$ (de l'expert m).

Une mixture d'experts peut être vue comme une forme de *disjonction*, i.e., la probabilité du modèle sera moyenne si l'un *ou* l'autre des experts donne une probabilité moyenne. À ce titre, un produit d'experts peut être vu comme une *conjonction* i.e. la probabilité sera moyenne si l'un *et* l'autre des experts donne une probabilité moyenne. Cette particularité rend les produits d'experts plus difficiles à entraîner ; par contre, elle les rend moins prompts à gaspiller de la masse de probabilité, ce que les mixtures d'experts ont largement tendance à faire.

2.3 Un modèle de langue neuronal

Dans (BENGIO, DUCHARME, VINCENT et JAUVIN 2003) les auteurs proposent plusieurs architectures neuronales pour modéliser la langue. Nous formalisons ici une architecture particulière qui se prête bien aux méthodes d'approximation par échantillonnage que nous proposons d'utiliser dans l'entraînement.

La sortie du réseau de neurones dépend du prochain mot v_t et d'un *contexte* des n derniers mots $h_t = (v_{t-n}, \ldots, v_{t-1})$. Chaque mot v est issu du vocabulaire $\mathcal{V}$. Il n'y a, *a priori*, aucune notion de similarité entre les mots à cette étape. La première caractéristique du NNLM est d'introduire une notion de similarité sémantique entre les mots en projetant chaque mot v_{t-i} de la fenêtre $(v_{t-n}, ..., v_t)$ dans un espace continu $\mathbf{R}^m$. À chaque mot v_{t-i} est ainsi associé un vecteur réel $\mathbf{z}_i \in \mathbf{R}^m$:

$$\mathbf{z}_i = \mathbf{C}_{v_{t-i}},\ i = 1, ..., n \tag{2.4}$$

$$\mathbf{z}_0 = \mathbf{D}_{v_t} \tag{2.5}$$

$$\mathbf{z} = (\mathbf{z}_0, ..., \mathbf{z}_n) \tag{2.6}$$

où $\mathbf{C}_j$ est la j-ième colonne d'une matrice de *vecteurs de caractéristiques*. Un vecteur de caractéristiques est un vecteur dans un espace réel de basse dimension qui représente un mot. La matrice $\mathbf{D}$ est une matrice de vecteurs de caractéristiques spécialisés pour le mot cible v_t dont $\mathbf{D}_j$ est la j-ième colonne. Le vecteur $\mathbf{z}$ est simplement la concaténation des vecteurs $\mathbf{z}_i$.

Ici, la notion de similarité est entendue au sens de la possibilité de remplacement d'un mot par un autre : on voudrait que deux mots qui peuvent être inter-changés dans une séquence de mots aient des vecteurs de caractéristiques proches (au sens euclidien). Ceci permet une plus grande généralisation. Par exemple, augmenter la probabilité de la phrase :

« Le chat mange le rat. »

devrait également augmenter celle des phrases suivantes :

« Un chien mange un chat.

Un chat dévore un rat.
Le chien mange un rat.
Le chat mange un chien.
Un chien dévore le chien.
... »

La projection effectuée permet d'atteindre cet objectif. En effet, si la distance euclidienne $\|\mathbf{C}_i - \mathbf{C}_j\|$ (ou $\|\mathbf{D}_i - \mathbf{D}_j\|$) entre les mots i et j est petite, la sortie du réseau (une fonction lisse des vecteurs de caractéristiques) sera peu affectée par la substitution du mot i par le mot j. L'espace des caractéristiques, duquel sont issus les vecteurs $\mathbf{z}_i$, constitue ainsi une forme de représentation distribuée des mots (voir section 2.1). Les vecteurs de caractéristiques sont appris par le modèle en même temps que ses paramètres.

Le vecteur résultant $\mathbf{z}$ est placé en entrée à la couche suivante :

$$\mathbf{a} = \tanh(\mathbf{W} \cdot \mathbf{z} + \mathbf{d}) \tag{2.7}$$

où $\mathbf{W}$ est une matrice de paramètres (les poids de la couche cachée), $\mathbf{d}$ est un vecteur de paramètres (les biais de la couche cachée) et $\mathbf{a}$ est le vecteur des activations de la couche cachée [3].

La sortie du réseau est une fonction énergétique :

$$\mathcal{E}(v_t, h_t) = \mathbf{V}_{v_t} \cdot \mathbf{a} + b_{v_t} \tag{2.8}$$

où $\mathbf{V}$ est une matrice de paramètres libres avec une colonne $\mathbf{V}_i$ par mot du vocabulaire et b_{v_t} est un biais de sortie.

Les probabilités jointes sont données par le modèle énergétique suivant :

$$P(v_t, h_t) = \frac{e^{-\mathcal{E}(v_t, h_t)}}{\sum_{(v', h') \in \mathcal{V}^{n+1}} e^{-\mathcal{E}(v', h')}} \tag{2.9}$$

En réalité, avec l'approche markovienne (voir section B.3), on s'intéresse

[3] **Rappel :** La fonction tanh est la tangente hyperbolique : $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$.

plutôt à modéliser les probabilités conditionnelles :

$$P(v_t|h_t) = \frac{e^{-\mathcal{E}(v_t,h_t)}}{\sum_{v' \in \mathcal{V}} e^{-\mathcal{E}(v',h_t)}}. \tag{2.10}$$

Le calcul de la fonction de partition de la densité jointe (2.9) est extrêmement coûteux ; celui de la densité conditionnelle (2.10) l'est moins car il croit linéairement avec le nombre de mots dans le vocabulaire $\mathcal{V}$, mais il demeure difficile.

La distribution (2.10) peut être vue comme un produit de deux experts (voir section 2.2), le premier étant de la forme :

$$e^{V_{i,v_t} a_i(v_t,h_t)} \tag{2.11}$$

où V_{i,v_t} est le i-ième élément de la v_t-ième colonne de la matrice V et $a_i(v_t, h_t)$ est le i-ième élément du vecteur d'activation de la couche cachée donné par l'équation 2.7. Le second expert est l'expert « unigrammes », i.e. dont l'énergie ne dépend que du mot cible v_t :

$$e^{b_{v_t}}. \tag{2.12}$$

Le réseau de neurones est entraîné par descente de gradient stochastique sur un critère de log-vraisemblance. Soit $\theta = (\mathbf{C}, \mathbf{D}, \mathbf{V}, \mathbf{b}, \mathbf{W}, \mathbf{d})$ les paramètres du modèle ($\mathbf{b} = (b_1 \dots b_{|\mathcal{V}|})^t, \mathbf{V} = (\mathbf{V}_1 \dots \mathbf{V}_{|\mathcal{V}|})$). On définit le point θ^* optimal comme le vecteur de paramètres qui maximisent la log-vraisemblance sur le corpus d'entraînement $D_N = (v_1, \dots, v_N)$ [4] :

$$\theta^* = \underset{\theta}{\operatorname{argmax}} \sum_{t=1}^{N} \log P(v_t|h_t). \tag{2.13}$$

On ne peut trouver de solution analytique à (2.13), mais on peut trouver un minimum local satisfaisant par descente de gradient stochastique. Cette

[4] Afin d'alléger la notation, nous posons $P_\theta(v|h) = P(v|h)$ et $\mathcal{E}_\theta(v,h) = \mathcal{E}(v,h)$ dans le reste de ce texte.

méthode nécessite de calculer, pour chaque $t = 1, \ldots, N$, le gradient sur la log-probabilité conditionnelle inverse :

$$\frac{\partial(-\log P(v_t|h_t))}{\partial\theta} = \underbrace{\frac{\partial\mathcal{E}(v_t, h_t)}{\partial\theta}}_{\text{renforcement positif}} \underbrace{-\sum_{v'\in\mathcal{V}} P(v'|h_t)\frac{\partial\mathcal{E}(v', h_t)}{\partial\theta}}_{\text{renforcement négatif}}. \tag{2.14}$$

Le gradient peut ainsi être décomposé en deux parties : un *renforcement positif* pour la valeur observée v_t et un *renforcement négatif* pour chaque mot v', avec un poids $P(v'|h_t)$. Ce dernier est clairement la partie du gradient difficile à évaluer, car elle nécessite de calculer le gradient par rétro-propagation pour chaque mot du vocabulaire.

Le chapitre suivant donne une approche basée sur l'échantillonnage aléatoire pour estimer la partie négative du gradient (terme de droite) afin d'accélérer l'entraînement d'un NNLM.

CHAPITRE 3

Méthodes de Monte Carlo

Comme il a été précisé dans la précédente section, l'entraînement d'un NNLM nécessite de calculer le gradient de la fonction énergétique (2.8) pour chaque mot du vocabulaire (2.14), ce qui est extrêmement coûteux. Cette limitation semble être le goulot d'étranglement majeur du NNLM. L'article original (BENGIO, DUCHARME, VINCENT et JAUVIN 2003) propose d'accélérer le calcul de la fonction énergétique et du gradient en parallélisant sur plusieurs machines au niveau de la couche de sortie. L'idée consiste à calculer le gradient négatif seulement pour les mots les plus fréquents dans un contexte donné.

Cette méthode se rapproche d'une méthode originale, la *divergence contrastive* (HINTON 2000). L'idée de base de la divergence contrastive consiste à remarquer que la partie négative du gradient d'un produit d'experts correspond directement à l'espérance du gradient selon la distribution conditionnelle $P(\cdot|h_t)$. Le gradient (2.14) peut donc être vu sous la forme :

$$\frac{\partial(-\log P(v_t|h_t))}{\partial\theta} = \frac{\partial\mathcal{E}(v_t,h_t)}{\partial\theta} - \mathrm{E}_P\left[\frac{\partial\mathcal{E}(V|h_t)}{\partial\theta}\right] \tag{3.1}$$

où la variable aléatoire V issue de la fonction de probabilité $P(\cdot|h_t)$, la dis-

tribution du NNLM.

Or, il existe des méthodes connues et éprouvées pour estimer l'espérance : les méthodes aléatoires dites de Monte Carlo. Dans (HINTON 2000), l'auteur utilise une méthode appelée *échantillonnage de Gibbs* pour entraîner des produits de gaussiennes.

Dans le présent chapitre, nous présentons les méthodes de Monte Carlo d'un point de vue général. Nous donnons d'abord une brève introduction aux méthodes de Monte Carlo. Nous présentons ensuite deux méthodes de Monte Carlo bien connues : l'échantillonnage pondéré et l'algorithme de Metropolis-Hastings (chaînes de Markov de Monte Carlo).

3.1 Introduction aux méthodes de Monte Carlo

Soit P une fonction de probabilité sur un ensemble $\mathcal{X}$ fini [1]. Soit X une variable aléatoire définie sur l'espace de probabilité $(\mathcal{X}, P)$. On cherche à estimer

$$\mathrm{E}_P[h(X)] = \sum_{x \in \mathcal{X}} h(x)P(x). \tag{3.2}$$

Il est naturel de proposer pour ce faire de tirer un échantillon $X_1, \ldots, X_n$ de points iid [2] de P et d'estimer (3.2) par la moyenne

$$\bar{h}_n = \frac{1}{n} \sum_{i=1}^{n} h(X_i). \tag{3.3}$$

Cette méthode est appelée *méthode classique de Monte Carlo* dans la littérature (ROBERT et CASELLA 2000, section 3.2). Par la loi des grands nombres, $\bar{h}_n$ converge vers $\mathrm{E}_P[h(X)]$ quand $n \to \infty$. Cet estimateur est non biaisé et on peut réduire sa variance en augmentant n.

Un problème est qu'il est souvent coûteux d'échantillonner directement de

[1] Afin d'alléger la notation et de nous concentrer sur le problème qui nous concerne – estimer l'espérance du gradient dans un NNLM – nous nous limitons à un ensemble $\mathcal{X}$ fini.

[2] iid : Indépendamment et Identiquement Distribués.

la distribution P. Par exemple, tirer un mot de la densité d'un NNLM (voir section 2.3) requiert de calculer la fonction énergétique (2.8) pour chaque mot du vocabulaire afin d'évaluer la densité (2.10), ce qui est extrêmement coûteux. Heureusement, des techniques existent pour contourner ce problème : elles consistent à estimer (3.2) en tirant des points d'une distribution instrumentale Q pour laquelle les tirages aléatoires sont moins coûteux.

3.2 Échantillonnage pondéré

Une alternative à la méthode classique d'échantillonnage Monte-Carlo présentée dans l'équation (3.3) consiste à remarquer que

$$\begin{aligned} \mathrm{E}_P[h(X)] &= \sum_{x \in \mathcal{X}} h(x) P(x) \\ &= \sum_{x \in \mathcal{X}} h(x) \frac{P(x)}{Q(x)} Q(x) \\ &= \mathrm{E}_Q \left[h(X) \frac{P(X)}{Q(X)} \right] \end{aligned}$$

où Q est une distribution appelée *distribution instrumentale*, ayant le même support que la *distribution cible* P et pour laquelle les tirages sont moins coûteux. Si on tire m points $X_1, ..., X_m$ iid de la distribution Q et qu'on réutilise (3.3), on obtient l'approximation

$$\hat{I}_m = \frac{1}{m} \sum_{i=1}^{m} h(X_i) \frac{P(X_i)}{Q(X_i)}. \tag{3.4}$$

Encore une fois, plus m est grand, meilleure sera l'approximation. Par contre, la qualité de l'estimateur dépend également de la distribution instrumentale Q (Robert et Casella 2000, section 3.3.2). Cette méthode est connue sous le nom d'*échantillonnage pondéré.*

Malheureusement, la forme de la fonction de probabilité dans un NNLM nous empêche d'utiliser l'estimateur (3.4). En effet, il faudrait pour cela

calculer les $P(X_i)$, c'est-à-dire les $P(X_i|h_t)$ pour un échantillon de mots $X_1, \ldots, X_m$ où chaque $X_i \in \mathcal{V}$ est tiré iid d'une distribution instrumentale, ce qui nécessite encore une fois de calculer la fonction de partition (voir équation (2.10)).

Nous proposons par conséquent d'utiliser une variante biaisée de l'échantillonnage pondéré traditionnel (KONG, LIU et WONG 1994; LIU 2001, section 2.5). Dans un modèle à base énergétique, la fonction de probabilité est de la forme

$$P(x) = \frac{1}{Z} f(x)$$

où $Z = \sum_{x \in \mathcal{X}} f(x)$ est la fonction de partition et f est une fonction positive [3]. Soit $w(X) = f(X)/Q(X)$. Il suffit de remarquer que Z peut être vu comme une espérance sur la loi uniforme :

$$Z = \mathrm{E}_U[Nf(X)] = N\mathrm{E}_U[f(X)]$$

où $U(X) = \frac{1}{N}$ et $N = |\mathcal{X}|$. Ainsi il est possible d'en faire une approximation par échantillonnage pondéré de points tirés de Q. Réutilisant le même échantillon $X_1, \ldots, X_m$ que pour (3.4), nous obtenons l'estimé de Z suivant :

$$\begin{aligned} Z &= N\mathrm{E}_Q\left[f(X)\frac{1}{NQ(X)}\right] \\ &= \mathrm{E}_Q\left[\frac{f(X)}{Q(X)}\right] \\ &\approx \frac{1}{m}\sum_{i=1}^{m}\frac{f(X_i)}{Q(X_i)} \\ &= \frac{1}{m}\sum_{i=1}^{m} w(X_i). \end{aligned} \tag{3.5}$$

En remplaçant Z par son estimé (3.5) dans (3.4) et en posant $W = \sum_{i=1}^{m} w(X_i)$,

[3] Dans le cas du NNLM, $f(v) = f(v, h_t) = e^{-\mathcal{E}(v,h_t)}$, $v \in \mathcal{V}$.

on obtient l'estimateur (biaisé)

$$\tilde{I}_m = \frac{1}{W} \sum_{i=1}^{m} h(X_i) w(X_i). \tag{3.6}$$

Par la loi des grands nombres, cet estimateur converge également vers l'espérance souhaitée quand $m \to \infty$.

3.3 Chaîne de Markov Monte Carlo

Une seconde alternative à la méthode classique de Monte Carlo fait également appel à une distribution instrumentale Q de même support que P. Elle consiste à simuler une chaîne de Markov dans l'espace $\mathcal{X}$ telle que la distribution stationnaire de la chaîne soit la distribution cible P (ROBERT et CASELLA 2000, chapitre 6).

L'algorithme bien connu de *Metropolis-Hastings* (HASTINGS 1970) fait partie de cette catégorie de méthodes. Une version de cet algorithme – appelée algorithme *indépendant* de Metropolis-Hastings – est donnée ci-dessous (ROBERT et CASELLA 2000, section 6.3.1).

Algorithme 2 Algorithme indépendant de Metropolis-Hastings

(1) **fonction** *metropolisHastings*(X_t)
(2) Tirer Y de la distribution instrumentale $Q(Y)$
(3) Tirer U de l'uniforme $U(0,1)$
(4)
$$X_{t+1} = \begin{cases} Y, & \text{si } U \leq r(X_t, Y) \\ X_t & \text{sinon} \end{cases}$$
où $r(X,Y) = \min\left\{1, \frac{P(Y)Q(X)}{P(X)Q(Y)}\right\}$.
(5) **retourner** X_{t+1}

Cet algorithme correspond à une transition dans une chaîne de Markov. On peut démontrer que cette chaîne est ergodique et a une distribution stationnaire P (ROBERT et CASELLA 2000, sections 6.2 et 6.3.1). En d'autres

mots, pour t suffisamment grand, les états X_t générés par la chaîne de Markov tendent à être distribués selon la distribution cible $P(X)$. Ainsi, on peut obtenir une approximation de (3.2) en considérant la séquence des T états générés $X_1, \ldots, X_T$ comme un échantillon iid de P et en utilisant la méthode classique de Monte Carlo (3.3) :

$$\mathrm{E}_P[h(X)] \approx \frac{1}{T} \sum_{t=1}^{T} h(X_t). \tag{3.7}$$

L'état initial de la chaîne, X_0, peut être fixé ou généré aléatoirement.

CHAPITRE 4

Algorithmes

Ce chapitre présente en détail les différents algorithmes que nous avons développé. Étant donné que la plupart de ces algorithmes ont mal fonctionné (divergence de l'erreur sur l'ensemble d'entraînement), nous commençons par présenter les méthodes qui ont bien fonctionné, c'est-à-dire les méthodes utilisant l'échantillonnage pondéré biaisé sur la densité conditionnelle, puis nous présentons plus brièvement les méthodes qui ont mal fonctionné.

L'algorithme 3 résume dans les grandes lignes la boucle d'entraînement d'un NNLM entraîné avec une méthode d'échantillonnage. La fonction *observer* (voir algorithme 4) met à jour les paramètres en fonction d'un couple (v_t, h_t) observé. La fonction *forward* calcule les activations (énergies) pour un couple observé (v_t, h_t) ainsi que pour les couples (v', h_t), $v' \in \mathcal{W}$ où $\mathcal{W}$ est l'ensemble des mots échantillonnés ; cet ensemble est retourné par la fonction. La fonction *backward* calcule et retourne le gradient pour le couple observé (v_t, h_t). La partie négative est estimée sur un ensemble de mots $\mathcal{W}$ (retourné par *forward*).

Dans cet algorithme, η_a est le taux d'apprentissage et λ est la pénalité sur les poids (voir section 1.3).

Enfin, il est bon de noter qu'afin d'optimiser la mise-à-jour des paramètres,

Algorithme 3 Entraînement d'un NNLM

(1) **fonction** *entraîner*$(D_N = \{(v_t, h_t)\}_{t=1}^N)$
(2) $\theta \leftarrow$ *init()* {Initialisation des paramètres}
(3) $a \leftarrow 0$ {Nombre total d'observations}
(4) **pour** $e \leftarrow 1$ à N_e **faire** {Boucle sur les époques}
(5) **pour** $t \leftarrow 1$ à N **faire** {Boucle sur l'ensemble d'entraînement}
(6) *observer*$((v_t, h_t))$
(7) $a \leftarrow a + 1$

Algorithme 4 Mise-à-jour des paramètres

(1) **fonction** *observer*$((v, h))$
(2) $\mathcal{W} \leftarrow$ *forward*$((v, h))$
(3) $\frac{\partial(-\log P(v|h))}{\partial\theta} \leftarrow$ *backward*$((v, h), \mathcal{W})$
(4) $\theta \leftarrow (1 - \lambda)\theta - \eta_a \frac{\partial P(v|h)}{\partial\theta}$

les étapes 3 et 4 (Algorithme 4) sont en réalité exécutées en une seule et même étape, c'est-à-dire que le gradient n'est pas conservé en mémoire : il est calculé à la volée et les paramètres sont immédiatement modifiés.

4.1 Entraînement par échantillonnage pondéré

De toutes les méthodes présentés dans le chapitre 3, une seule s'est avérée réellement efficace pour entraîner un NNLM : l'échantillonnage pondéré biaisé, présenté dans la section 3.2. Dans la présente section, nous présentons les détails de l'algorithme et les différentes modifications qui lui ont été apportées. Il est à noter que plusieurs de ces modifications n'ont pas bien fonctionné, comme nous le verrons dans le prochain chapitre.

En appliquant la méthode de l'échantillonnage pondéré biaisé représentée par l'équation (3.6) à l'estimation du gradient du NNLM (2.14), on obtient l'estimé suivant pour la partie négative du gradient :

$$\frac{1}{W} \sum_{v' \in \mathcal{W}} w(v') \frac{\partial \mathcal{E}(v', h_t)}{\partial \theta} \tag{4.1}$$

où $\mathcal{W}$ est l'ensemble des m mots tirés de la distribution instrumentale Q, $w(v') = \frac{e^{-\mathcal{E}(v',h_t)}}{Q(v',h_t)}$ et $W = \sum_{v' \in \mathcal{W}} w(v')$.

Deux questions se posent si l'on souhaite réduire le biais et la variance de l'estimateur :

1. Comment choisir la distribution instrumentale Q ?
2. Comment choisir le nombre d'échantillons m ?

4.1.1 Choix de la distribution instrumentale

Dans toute méthode d'estimation par échantillonnage à partir d'une distribution instrumentale, le choix de la distribution instrumentale est important. Nous voulons également que la distribution en question soit facilement échantillonnable, i.e., que le coût pour échantillonner un point soit négligeable par rapport au coût de calculer une valeur d'énergie.

Une heuristique raisonable pour le choix de Q est une distribution voisine de P. En effet, si on choisit $Q = P$, l'estimateur par échantillonnage pondéré biaisé (3.6) est équivalent à l'estimateur classique (3.3) (voir section C.2).

Mais en réalité, dans le cas de l'échantillonnage pondéré, la distribution instrumentale qui minimise la variance est [1] :

$$Q^*(v_t|h_t) = \frac{\| \frac{\partial e^{-\mathcal{E}(v_t,h_t)}}{\partial \theta} \| e^{-\mathcal{E}(v_t,h_t)}}{\sum_{v' \in \mathcal{V}} \| \frac{\partial e^{-\mathcal{E}(v',h_t)}}{\partial \theta} \| e^{-\mathcal{E}(v',h_t)}}. \tag{4.2}$$

Cette distribution peut être difficile à approximer car elle nécessite de calculer explicitement le gradient et sa norme.

Distribution instrumentale n-gramme

Un choix évident pour la distribution instrumentale Q est un modèle n-*gramme* (voir la preuve, section B.4), car il est aisé d'en échantillonner des points. Pour chaque contexte de mots de longueur $(n-1)$, un n-gramme donne la probabilité du mot suivant. Ces valeurs étant stockées dans une

[1] La dérivation de ce résultat peut être trouvé en annexe, section C.1.

structure non-distribuée, il est impossible de garder en mémoire une valeur pour chaque contexte et pour chaque mot suivant. En effet, cela nécessiterait un tableau de taille $|\mathcal{V}|^n$ ce qui ne tient tout simplement pas en mémoire même pour n assez petit. Par conséquent, un n-gramme ne conserve des statistiques que sur les séquences *observées* dans l'ensemble d'entraînement.

La conséquence est que la probabilité des séquences non-observées est nulle, ce qui n'est pas souhaitable pour une distribution instrumentale (elle doit avoir le même support que la distribution cible). La solution traditionnelle consiste à interpoler le modèle avec un modèle qui donne toujours une probabilité non-nulle à un mot (e.g., un modèle unigramme, ou zérogramme [2]).

Au début de l'entraînement d'un NNLM, nous initialisons les poids de sortie b_v (voir équation (2.12)) avec les fréquences relatives des mots afin que la distribution initiale du NNLM soit exactement celle de l'unigramme :

$$P_1(v) = \frac{|v| + \epsilon}{\sum_{v' \in \mathcal{V}} (|v'| + \epsilon)} \tag{4.3}$$

où $|v|$ est la fréquence du mot v, c'est-à-dire le nombre de fois que ce mot apparaît dans le corpus d'entraînement. La variable $\epsilon > 0$ assure que tous les mots v ont une probabilité non-nulle.

Partant de ce fait, il fait beaucoup de sens d'utiliser l'unigramme comme distribution instrumentale au début de l'entraînement. Par contre, à mesure que le réseau est entraîné, sa distribution s'éloigne nécessairement de celle de l'unigramme, donc il faut soit augmenter la taille de l'échantillon ou changer de distribution.

Distribution instrumentale n-gramme adaptée sur la perplexité

De façon générale, on remarque que plus n est grand, plus un n-gramme interpolé a une perplexité faible (sur l'ensemble d'entraînement). Partant de ce constat, nous faisons l'hypothèse suivante : plus la perplexité d'un NNLM

[2] Probabilité égale pour tous les mots.

est « proche » de la perplexité d'un modèle n-gramme, plus leurs distributions sont proches.

Cette hypothèse nous suggère une heuristique pour choisir la distribution instrumentale. Considérons un modèle trigramme interpolé P_{tri} (voir annexe B.4) :

$$P_{tri}(v_t|v_{t-2}v_{t-1}) = \sum_{i=1}^{3} \alpha_i(v_{t-1}) P_i(v_t|v_{t-2}v_{t-1}) \tag{4.4}$$

où $P_0(v_t|v_{t-2}v_{t-1}) = P_0 = \frac{1}{|\mathcal{V}|}$ est le zéro-gramme, $P_1(v_t|v_{t-2}v_{t-1}) = P_1(v_t)$ est l'unigramme, $P_2(v_t|v_{t-2}v_{t-1}) = P_2(v_t|v_{t-1})$ est le bigramme et $P_3(v_t|v_{t-2}v_{t-1})$ est le trigramme.

Les $\alpha_i(v)$ sont des poids tels que $\forall v : \sum_{i=0}^{3} \alpha_i(v) = 1$. Ces poids sont entraînés par EM (« expectation-maximization ») afin de minimiser la perplexité sur l'ensemble de validation [3]

Partant d'un tel modèle pré-entraîné, nous cherchons une façon d'attribuer de nouveaux poids de mélange au modèle (4.4). Pour ce faire, définissons la distribution

$$Q^{(u)}(v_t|v_{t-2}v_{t-1}) = \sum_{i=0}^{3} \alpha_i^{(u)}(v_{t-1}) P_i(v_t|v_{t-2}v_{t-1}) \tag{4.5}$$

où $u \in [0,1]$ et les $\alpha_i^{(u)}(v_{t-1})$ sont des poids de mélange dépendants de u et tels que $\forall u, v : \sum_{i=0}^{3} \alpha_i^{(u)}(v) = 1$. Nous souhaitons que ces poids soient tels que

$$Q^{(0)} = P_0 \tag{4.6}$$

$$Q^{(1)} = P_{tri}. \tag{4.7}$$

Plus u est près de 1, plus le modèle $Q^{(u)}$ doit être près de P_{tri}, le modèle ayant la perplexité la plus faible ; plus il est près de 0, plus le modèle $Q^{(u)}$ est près de la distribution zéro-gramme (perplexité la plus élevée) P_0. La variable u devrait ainsi nous permettre de contrôler la perplexité du modèle.

[3] Se référer à (JELINEK 1990) pour une description de l'algorithme servant à entraîner les poids $\alpha_i(v)$.

La méthode proposée consiste à créer une série de modèles $Q^{(u)}$ pour différentes valeurs de u, de perplexité croissante. À chaque époque, on choisit comme proposition instrumentale le modèle $Q^{(u)}$ dont la perplexité est la plus proche de celle du NNLM sur un sous-ensemble de l'ensemble d'entraînement.

La formule que nous avons utilisée pour les $\alpha_i^{(u)}(v_{t-1})$ est

$$\alpha_i^{(u)}(v) = \frac{\alpha_i(v)w_i(u)}{\sum_{j=0}^{3}\alpha_j(v)w_j(u)} \tag{4.8}$$

où $w_0(u) = 1$ et $w_i(u) = u^{\frac{1}{4-i}}, i = 1, 2, 3$. Cette formule a l'avantage de répondre aux critères (4.6) et (4.7) et de passer de façon lisse de l'un à l'autre, à mesure que u augmente.

La figure 3 montre l'évolution des valeurs de $\alpha_i^{(u)}(v)$ dans le cas où les $\alpha_i(v)$ sont égaux pour tous i. Dans ce graphique on réfère à $\alpha_i^{(u)}(v)$ par $\alpha_i^{(u)}$ pour alléger la notation. On voit comment le poids $\alpha_0^{(u)}$ diminue alors que les poids $\alpha_1^{(u)}, \alpha_2^{(u)}, \alpha_3^{(u)}$ augmentent. On remarque également que $\alpha_1^{(u)}$ croît plus rapidement que $\alpha_2^{(u)}$ et $\alpha_3^{(u)}$ et que $\alpha_2^{(u)}$ croît plus rapidement que $\alpha_3^{(u)}$. Ceci afin que, alors que u augmente, on passe tour-à-tour d'une distribution zéro-gramme P_0 à une distribution proche de l'unigramme, puis du bigramme et enfin du trigramme interpolé P_{tri} (4.4).

Distribution instrumentale n-gramme adaptative

Avec un NNLM, le problème est que P (la distribution du NNLM) change à mesure que le réseau est entraîné. Aussi l'idée nous est-elle venue d'entraîner Q à estimer P ou Q^* (4.2) pendant que le NNLM est entraîné.

Dans un n-gramme traditionnel, les probabilités sont évaluées en comptant simplement la fréquence de chaque séquence observée dans le corpus d'entraînement (voir section B.4). Dans le cas qui nous intéresse, nous aimerions plutôt que la distribution instrumentale soit *adaptée* en cours d'entraînement pour approximer P ou Q^*. Pour y parvenir, il est clair que nous devons évaluer $e^{-\mathcal{E}(v,h)}$ (et $\|\frac{\partial \mathcal{E}(v,h)}{\partial \theta}\|$ si on souhaite approximer Q^*) ; nous souhaitons cependant, encore une fois, minimiser le nombre de mots v pour lesquels nous devons évaluer cette fonction.

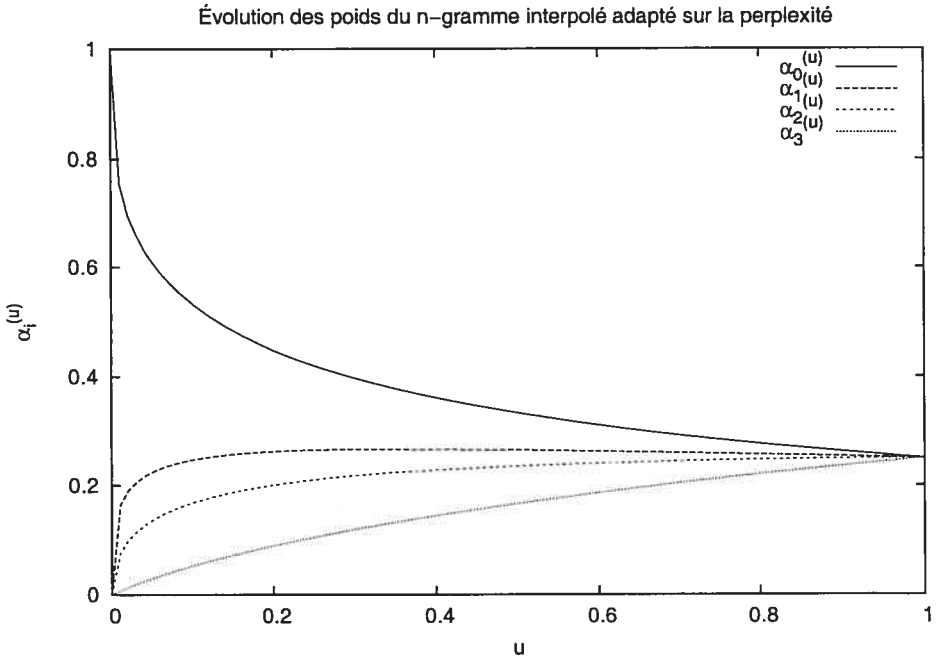


Figure 3 – Évolution des poids du n-gramme interpolé adapté sur la perplexité

Notre proposition est d'utiliser les valeurs de $e^{-\mathcal{E}(v,h)}$ (et de $\|\frac{\partial \mathcal{E}(v,h)}{\partial \theta}\|$) disponibles pour les mots échantillonnés seulement, puis de *redistribuer la masse de probabilité* parmi ces seuls mots dans le modèle de la distribution instrumentale.

Posons

$$Q_k(v|h_t) = \begin{cases} q_k(v|v_{t-k+1}^{t-1}) & \text{si } |v_{t-k+1}^{t-1}, v| > 0 \\ 0 & \text{sinon} \end{cases} \tag{4.9}$$

Les $q_k(v|v_{t-k+1}^{t-1})$ sont des paramètres à valeur réelle, tels que

$$\sum_{v \in \mathcal{V}} q_k(v|v_{t-k+1}^{t-1}) I_{|v_{t-k+1}^{t-1}, v|>0} = 1$$

où I est la fonction indicatrice.

Nous définissons la distribution instrumentale Q suivante :

$$Q(v|h_t) = \sum_{k=1}^{n} \alpha_k(h_t) Q_k(v|h_t) \tag{4.10}$$

où $\alpha_k(h_t)$ est une fonction de mixage telle que $\sum_{k=1}^{n} \alpha_k(h_t) = 1$.

Soit $\mathcal{W}$ l'ensemble des points tirés de Q. Soit $\bar{q}_k = \sum_{v \in \mathcal{W}} Q_k(v|h_t)$ la masse de probabilité du modèle k-gramme Q_k sur ces points et $\bar{y} = \sum_{v \in \mathcal{W}} y(v, h_t)$ leur masse non-normalisée dans la fonction à estimer. Ici $y(v, h_t) = e^{-\mathcal{E}(v,h_t)}$ si on veut approximer P et $y(v, h_t) = e^{-\mathcal{E}(v,h_t)} \| \frac{\partial \mathcal{E}(v,h_t)}{\partial \theta} \|$ si on veut approximer Q^*. Pour les mots $v \in \mathcal{W}$ tels que $Q_k(v|h_t)$ est non-nul, les paramètres $q_k(v|v_{t-k+1}^{t-1})$ sont adaptés de la façon suivante :

$$q_k(v|v_{t-k+1}^{t-1}) \leftarrow (1 - \eta) q_k(v|v_{t-k+1}^{t-1}) + \eta \frac{y(v, h_t)\bar{q}_k}{\bar{y}} \tag{4.11}$$

où η est un *taux d'apprentissage* (« learning rate ») qui doit être choisi ; dans nos expérience (chapitre 5), il est fixé à 0.001.

Les poids $\alpha_k(h_t)$ sont adaptés afin de minimiser la *divergence de Kullback-Leibler* sur l'ensemble $\mathcal{W}$. Cette divergence est définie par

$$\sum_{v \in \mathcal{W}} t(v|h_t) \log \frac{t(v|h_t)}{Q(v|h_t)} \tag{4.12}$$

où $t(v|h_t) = \frac{y(v,h_t)}{\bar{y}}$.

Il est évidemment peu pratique d'avoir un paramètre $\alpha_k(h_t)$ pour chaque contexte h_t. De plus cela n'est pas souhaitable car le modèle ne généralisera pas bien. La solution habituelle consiste à regrouper les contextes h de fréquence similaire. Dans nos expériences, nous avons considéré un seul mot d'historique. La méthode utilisée pour générer les groupes tente de s'assurer que chaque groupe contienne une fréquence totale à peu près égale. Soit $|v|$ la fréquence du mot v et soit $\{\mathcal{B}_i\}_{i=1}^{N}$ l'ensemble des groupes, on veut que

$$\forall i, j \sum_{v \in \mathcal{B}_i} |v| \approx \sum_{v \in \mathcal{B}_j} |v|$$

L'algorithme 5 explicite comment l'ensemble des mots $\mathcal{V}$ est partitionné en sous-ensembles $\mathcal{B}_i$ répondant à peu près à ce critère.

Algorithme 5 Génération des groupes de mots pour les poids α

(1) $i \leftarrow 0$
(2) $\mathcal{B}_0 \leftarrow \emptyset$
(3) $n_b \leftarrow 0$
(4) $\bar{n_b} \leftarrow \frac{1}{|\mathcal{V}|} \sum_{v \in \mathcal{V}} |v|$ {Fréquence moyenne}
(5) **tantque** $\mathcal{V} \neq \emptyset$ **faire**
(6) $v_{max} \leftarrow \text{argmax}_{v \in \mathcal{V}} |v|$ {v_{max} est le mot de fréquence maximale}
(7) $\mathcal{V} \leftarrow \mathcal{V} \setminus \{v_{max}\}$
(8) $\mathcal{B}_i \leftarrow \mathcal{B}_i \cup \{v_{max}\}$
(9) $n_b \leftarrow n_b + |v_{max}|$
(10) **si** $n_b > \bar{n_b}$ **alors**
(11) $n_b \leftarrow 0$
(12) $i \leftarrow i + 1$

Unigramme adaptatif

Nous commençons par un exemple simple de n-gramme adaptatif, utilisé dans nos expériences : un *unigramme adaptatif.*

Les probabilités pour chaque mot (indépendantes du contexte) sont maintenues dans une structure en arbre construit afin d'accélérer l'échantillonnage des points. Chaque feuille de l'arbre représente un mot particulier. Le principe consiste à maintenir les mots de forte probabilité plus près de la racine de façon à accélérer l'échantillonnage des mots. Une seconde structure de données permet de faciliter la recherche de mots : un simple tableau de pointeurs (*noeuds*) vers les noeuds de l'arbre, indexé par le mot en question.

Au début de l'entraînement, les probabilités de l'unigramme adaptatif sont initialisées à celles de l'unigramme classique. L'algorithme 6 décrit l'initialisation de l'arbre unigramme. Cet algorithme est directement inspiré de celui utilisé pour construire un arbre de Huffman (HUFFMAN 1952). La fonction retourne le noeud racine de l'arbre. L'algorithme 7 montre comment échantillonner un point de l'unigramme à l'aide de l'arbre en utilisant une valeur aléatoire u de loi uniforme $U(0, 1)$.

Après avoir propagé les valeurs énergétiques pour les mots échantillonnés $\mathcal{W}$, les probabilités du modèle sont réajustées selon la règle (4.11). Afin de maintenir les bonnes valeurs de probabilité dans les noeuds intérieurs – ce

Algorithme 6 Initialisation de l'arbre unigramme

(1) **fonction** *créerArbreUnigramme()*
(2) **pourtout** $v \in \mathcal{V}$ **faire**
(3) $noeud \leftarrow$ *créerFeuille()*
(4) $noeud.mot \leftarrow v$
(5) $noeud.probabilité \leftarrow noeuds[v].probabilité$
(6) $noeuds[v] \leftarrow noeud$
(7) $\mathcal{T} \leftarrow \mathcal{T} \cup \{noeud\}$
(8) **tantque** $|\mathcal{T}| > 1$ **faire**
(9) Choisir $noeud_1, noeud_2 \in \mathcal{T}$ de plus faible probabilité
(10) $noeud \leftarrow$ *créerNoeud()*
(11) $noeud.gauche \leftarrow noeud_1$
(12) $noeud.droit \leftarrow noeud_2$
(13) $noeud.probabilité \leftarrow noeud_1.probabilité + noeud_2.probabilité$
(14) $\mathcal{T} \leftarrow \mathcal{T} \setminus \{noeud_1, noeud_2\} \cup \{noeud\}$
(15) $\{\mathcal{T}$ contient un seul noeud, la racine de l'arbre i.e. $\mathcal{T} = \{racine\}\}$

qui est essentiel pour échantillonner – sans avoir à recréer l'arbre ni à parcourir tous ses noeuds, les noeuds parcourus pendant l'échantillonnage sont étiquetés ; après avoir modifié les probabilités des noeuds, seules les probabilités des noeuds intérieurs étiquetés sont recalculées. L'arbre unigramme est également complètement réinitialisé avec l'algorithme 6 à chaque époque, afin de conserver ses propriétés (mots plus probables en haut de l'arbre).

Bigramme adaptatif

Comme on le verra au chapitre 5, les résultats les plus intéressants ont été obtenus avec un bigramme adaptatif. Nous présentons ici les détails d'implantation propres à ce modèle.

Les valeurs de probabilité des bigrammes sont également stockés dans une structure en arbre. À chaque contexte $h = v_{t-1}$ (un mot) est associé un arbre binaire classique. Chaque noeud de cet arbre représente un mot $v \in \mathcal{V}$ ainsi que sa probabilité bigramme associée $q_2(v|v_{t-1})$. Un noeud contient également la probabilité cumulative du mot qu'il représente et de ses sous-arbres. Seuls les mots v observés à la suite de v_{t-1} dans le corpus d'entraînement ont un noeud associé dans l'arbre ; pour les autres, la probabilité est nulle (voir

Algorithme 7 Échantillonnage à partir de l'arbre unigramme

```
(1)  fonction échantillonnerArbreUnigramme(u)
(2)    noeud ← racine
(3)    boucle
(4)      si feuille(noeud) alors
(5)        retourner noeud
(6)      sinon
(7)        u' ← u − noeud.gauche.probabilité
(8)        si u' < 0 alors
(9)          noeud ← noeud.gauche
(10)       sinon
(11)         noeud ← noeud.droit
(12)         u ← u'
```

équation (4.9)), c'est-à-dire qu'il n'y a pas de noeud.

L'algorithme 8 résume les étapes effectuées pour tirer un mot d'un bigramme en contexte $h = v_{t-1}$ avec une valeur aléatoire u tirée de $U(0, 1)$.

Algorithme 8 Échantillonnage à partir du bigramme

```
(1)  fonction échantillonnerArbreBigramme(u, h)
(2)    noeud ← racine[h]
(3)    boucle
(4)      si noeud.gauche ≠ nil et u ≤ noeud.gauche.cumulatif alors
(5)        noeud ← noeud.gauche
(6)      sinon si noeud.droit ≠ nil alors
(7)        u' ← u − noeud.cumulatif + noeud.droit.cumulatif
(8)        si u' < 0 alors
(9)          retourner noeud
(10)       sinon
(11)         noeud ← noeud.droit
(12)         u ← u'
(13)     sinon
(14)       retourner noeud {Feuille}
```

Afin de s'assurer que la probabilité est non-nulle pour tout mot dans $\mathcal{V}$, les probabilités sont mixées avec celles d'un unigramme adaptatif (voir

section 4.1.1). La probabilité résultante est

$$Q(v|v_{t-1}) = (1 - \alpha(v_{t-1}))Q_1(v) + \alpha(v_{t-1})Q_2(v|v_{t-1}) \tag{4.13}$$

où $Q_1(v)$ est la distribution de l'unigramme adaptatif et $Q_2(v|v_{t-1})$ est celle du bigramme adaptatif.

L'échantillonnage de ce modèle est résumé dans l'algorithme 9.

Algorithme 9 Échantillonnage du bigramme adaptatif (mixture)

```
(1) fonction échantillonnerBigramme(h)
(2)   u ← U(0,1) {Échantillonnage uniforme}
(3)   si racine[h] ≠ nil alors {Vérifier si ∃v : |hv| > 0}
(4)     si u < α(h) alors
(5)       retourner échantillonnerArbreBigramme(u/α(h), h)
(6)     sinon
(7)       retourner échantillonnerArbreUnigramme((u − α(h))/(1 − α(h)))
(8)   sinon
(9)     retourner échantillonnerArbreUnigramme(u)
```

On s'assure que les $\alpha(v_{t-1})$ demeurent entre 0 et 1 en posant

$$\alpha(v_{t-1}) = \sigma(a_i)\ v_t \in \mathcal{B}_i$$

où $\sigma(\cdot)$ est la fonction sigmoïde et a_i est un paramètre à valeurs réelles.

Soit $\mathcal{W}$ l'ensemble des mots échantillonnés et soit i tel que $v_{t-1} \in \mathcal{B}_i$. Alors a_i est mis à jour par descente de gradient afin de minimiser la divergence (4.12). La règle de mise à jour résultante est

$$a_i \leftarrow a_i - \eta\alpha(v_{t-1})(1 - \alpha(v_{t-1})) \sum_{v \in \mathcal{W}} \frac{t(v|h_t)}{Q(v|v_{t-1})}(Q_1(v) - Q_2(v|v_{t-1})) \tag{4.14}$$

où $t(v|h_t)$ est le même que dans (4.12) et η est un taux d'apprentissage (dans nos expériences, fixé à 0.001). [4]

[4] Les détails des calculs ayant mené à ce résultat peuvent être trouvés en annexe, section C.3.

Au moment de l'échantillonnage, les noeuds traversés par l'algorithme 8 sont étiquetés ; par contre, les noeuds traversés par l'algorithme d'échantillonnage de l'arbre unigramme (algorithme 7) ne le sont pas. Après la mise-à-jour des probabilités, seuls les noeuds étiquetés dans le bigramme voient leurs probabilités (propres et cumulatives) recalculées. Les probabilités dans les noeuds de l'arbre unigramme sont, quant à elles, entièrement recalculées en traversant tous les noeuds de l'arbre. Pourquoi ne pas également étiqueter l'arbre unigramme ? La raison est que, dans le cas où on a réellement tiré un point du bigramme (étape 5 dans l'algorithme 9), il faut alors aller chercher la probabilité de l'unigramme pour ce mot. On peut le faire simplement avec le tableau indexé qui pointe vers les feuilles de l'arbre unigramme. Cependant, si on souhaite étiqueter les noeuds de l'arbre unigramme qu'il faut parcourir pour atteindre ce mot, il faut faire une recherche dans tout l'arbre unigramme. Cette recherche ne peut se faire en moins de temps que $O(|\mathcal{V}|)$, ce qui est du même ordre que recalculer les probabilités ! On n'a pas ce problème avec les arbres bigrammes car ces derniers ont une structure qui n'est pas faite pour optimiser l'échantillonnage, mais pour faciliter la recherche d'un mot particulier.

4.1.2 Choix de la taille d'échantillon

Nous avons considéré trois approches pour estimer la taille d'échantillon nécessaire pour obtenir une bonne approximation du gradient (2.14).

Dans l'application considérée, l'objectif est bien sûr de minimiser autant que possible le nombre d'évaluations de la fonction $\mathcal{E}(\cdot)$. Par contre, il est également important d'avoir une bonne estimation du gradient si on souhaite que la descente de gradient converge. Or, le gradient d'un NNLM évolue à mesure que le réseau est entraîné ; on peut conjecturer qu'à mesure qu'on l'entraîne, sa distribution devient de plus en plus complexe, ce qui a deux conséquences :

1. La distribution du réseau s'éloigne d'une distribution instrumentale ayant moins de capacité.
2. La variance du gradient augmente.

Méthode brute

La première méthode que nous avons considérée est très simple. L'idée est de conserver une copie des paramètres du modèle de la dernière époque afin de pouvoir revenir à cet état si le modèle diverge à cause d'une taille d'échantillon trop petite. La divergence du modèle est estimée en calculant l'erreur sur un petit sous-ensemble de l'ensemble d'entraînement. Cette méthode « brute » résout en même temps le problème du biais et celui de la variance mais elle le fait à beaucoup de frais.

L'algorithme 10 présente d'une façon générale l'entraînement d'un modèle avec cette méthode. L'ensemble D_M est un petit sous-ensemble de l'ensemble d'entraînement D_N ($M \ll N$). La variable m_0 est la taille d'échantillon initiale. La variable $k > 1$ est un multiplicateur pour augmenter la taille d'échantillon si le modèle diverge.

Algorithme 10 Méthode d'entraînement brute

(1) **fonction** *entraîner*$(D_N = \{(v_i, h_i)\}_{i=1}^{N}, D_M \subseteq D_N, m_0, k)$
(2) $m \leftarrow m_0$ {Taille d'échantillon}
(3) $\theta \leftarrow$ *init()* {Initialisation des paramètres}
(4) $\epsilon \leftarrow$ *erreur*(D_M, θ) {Calcul de la perplexité}
(5) $e \leftarrow 1$
(6) $a \leftarrow 0$ {Nombre total d'observations}
(7) **tantque** $e \leq N_e$ **faire** {Boucle sur les époques}
(8) $\theta_c \leftarrow \theta$ {Copie des paramètres}
(9) $\epsilon_c \leftarrow \epsilon$ {Copie de l'erreur}
(10) **pour** $t \leftarrow 1$ à N **faire** {Boucle sur l'ensemble d'entraînement}
(11) *observer*$((v_t, h_t))$
(12) $a \leftarrow a + 1$
(13) $\epsilon \leftarrow$ *erreur*(D_M, θ)
(14) **si** $\epsilon \geq \epsilon_c$ **alors** {La perplexité a augmenté (divergence)}
(15) $m \leftarrow \lceil km \rceil$ {Augmentation du nombre d'échantillons}
(16) $\theta \leftarrow \theta_c$
(17) $\epsilon \leftarrow \epsilon_c$
(18) $a \leftarrow a - N$
(19) **sinon** {La perplexité a diminué (convergence)}
(20) $e \leftarrow e + 1$

Bien que cette méthode soit coûteuse, elle fonctionne relativement bien. Cependant, la méthode que nous présentons dans la prochaine section est celle que nous avons finalement privilégiée car elle est vraiment plus efficace.

Taille d'échantillon effective

La présente section décrit une mesure heuristique pour contourner le problème du biais. Nous devons pour le moment oblitérer le problème de la variance du gradient en estimant la variance maximale atteinte par le réseau par essais-erreurs, ce qui est évidemment sous-optimal.

Pour estimer le nombre d'échantillons requis à chaque observation, nous proposons d'utiliser une méthode heuristique décrite dans (KONG, LIU et WONG 1994) : la *taille d'échantillon effective*. Cette méthode s'énonce comme suit.

Supposons qu'on utilise un échantillon de taille m pour estimer le gradient (2.14) avec l'estimateur pondéré biaisé (3.6). Soit P la distribution du NNLM et soit Q une distribution instrumentale. Soit V une variable aléatoire suivant la distribution Q. La taille d'échantillon effective (« effective sample size »), donnée par

$$ESS_m = \frac{m}{1 + \mathrm{Var}_Q\left[\frac{P(V|h_t)}{Q(V|h_t)}\right]} \tag{4.15}$$

correspond grosso-modo au *nombre de points, tirés de la distribution cible* $P(V|h_t)$*, qu'on aurait effectivement dû tirer pour atteindre une variance équivalente avec l'estimateur classique Monte Carlo* (3.3) [5].

Soit $\mathcal{W}$ l'ensemble des points tirés de la distribution instrumentale Q. Comme nous ne voulons pas calculer explicitement la variance $\mathrm{Var}_Q\left[\frac{P(V|h_t)}{Q(V|h_t)}\right]$ (encore une fois, trop coûteux), nous devons l'estimer par le carré du *coefficient de variation* (cv^2) des poids non-normalisés $w(v) = \frac{e^{-\mathcal{E}(v,h_t)}}{Q(v|h_t)}$ pour les

[5] La justification de cette heuristique est expliquée en détail dans (KONG 1992).

points $v \in \mathcal{W}$ [6] :

$$cv^2 = \frac{\sum_{v \in \mathcal{W}} (w(v) - \bar{w})^2}{m\bar{w}^2} \tag{4.16}$$

où $\bar{w} = \frac{1}{m} \sum_{v \in \mathcal{W}} w(v)$.

En remplaçant $\text{Var}_Q \left[\frac{P(V|h_t)}{Q(V|h_t)} \right]$ par cv^2 dans (4.15) on obtient l'approximation suivante pour ESS_m :

$$\widehat{ESS}_m = \frac{W^2}{S} \tag{4.17}$$

où $W = \sum_{v \in \mathcal{W}} w(v)$ et $S = \sum_{v \in \mathcal{W}} w^2(v)$.

Si on suppose que l'estimateur classique Monte Carlo, avec une taille d'échantillon de n [7], est toujours un bon estimateur du gradient, on peut utiliser cette mesure heuristique pour tenter de choisir m tel que $\widehat{ESS}_m \geq n$. Pour ce faire, à chaque nouvel exemple, nous tirons des points (par blocs de 10, par exemple) jusqu'à ce que cette condition soit atteinte (voir algorithme 11) ou que le nombre de points échantillonnés soit trop élevé pour que ça vaille la peine d'échantillonner (par exemple, s'il dépasse la taille du vocabulaire), auquel cas on oublie l'échantillonnage pour l'exemple courant et on calcule le gradient exact (rétro-propagation sur tous les mots de $\mathcal{V}$).

Taille d'échantillon effective alternative

L'estimateur de la taille d'échantillon effective donné en (4.17) a la particularité d'être indépendant de la fonction à estimer (ici, le gradient), ce qui est un avantage certain : le calcul explicite du gradient nécessite des ressources computationnelles additionnelles que l'on peut normalement s'éviter en mettant à jour les paramètres à la volée.

Cependant, comme nous l'avons vu dans la section 4.1.1, il peut être

[6] Le coefficient de variation est une mesure de dispersion relative à la moyenne. Il est utilisé quand on veut une mesure de variance unique pour des distributions ayant différentes moyennes (e.g., la distribution des $e^{-\mathcal{E}(v,h_t)}$, dont la moyenne $\frac{1}{|\mathcal{V}|} \sum_{v' \in \mathcal{V}} e^{-\mathcal{E}(v',h_t)}$ est différente pour différents contextes h_t (RICE 1988, p. 360).

[7] Dans le reste de ce mémoire, nous référons à cette taille d'échantillon par la taille d'échantillon cible.

Algorithme 11 Propagation avec taille d'échantillon effective

(1) **fonction** *forward*$((v, h))$
(2) *fpropInput(h)* {Propagation du contexte à la couche cachée}
(3) *fprop(v)* {Propagation du mot observé}
(4) $ESS \leftarrow 0$
(5) $W \leftarrow 0$
(6) $S \leftarrow 0$
(7) $\mathcal{W} \leftarrow \emptyset$ {Ensemble des mots échantillonnés}
(8) **tantque** $ESS < n$ **faire**
(9) **pour** $i \leftarrow 1$ à b **faire**
(10) $v' \leftarrow Q(\cdot|h)$
(11) $\mathcal{W} \leftarrow \mathcal{W} \cup \{v'\}$
(12) *fprop(v')* {Propagation du mot échantillonné}
(13) $w(v') \leftarrow \frac{e^{-\mathcal{E}(v',h)}}{Q(v'|h)}$
(14) $W \leftarrow W + w(v')$
(15) $S \leftarrow S + w^2(v')$
(16) $ESS \leftarrow \frac{W^2}{S}$
(17) **retourner** $\mathcal{W}$

avantageux de définir une distribution instrumentale qui utilise la norme du gradient pour minimiser la variance de l'estimateur. Or il nous semble que l'estimateur de taille d'échantillon effective (4.15), puisqu'il ne tient pas compte de la norme du gradient, risque de systématiquement sous-estimer la taille d'échantillon effective si on tire des mots d'une distribution instrumentale qui approxime (4.2) ; par conséquent, malgré le fait que la variance soit réduite par le choix de la distribution instrumentale, le nombre d'échantillons nécessaire pourrait être sur-évalué. Bien entendu, il se peut que l'approximation de (4.2) – par un n-gramme adaptatif par exemple – ait une erreur élevée ; par conséquent, rien ne garantit que la variance de l'estimé du gradient obtenu en utilisant cette approximation comme distribution instrumentale soit réduite par rapport à un estimé du gradient obtenu en utilisant une autre distribution instrumentale. Mais de toute façon, cela n'empêche pas qu'il pourrait être utile d'utiliser un estimateur de taille d'échantillon effective qui prenne la variance dans la norme du gradient en considération, plutôt que seulement les poids $w(v)$.

Algorithme 12 Rétro-propagation avec taille d'échantillon effective

(1) **fonction** *backward*$((v,h),\mathcal{W})$
(2) *bprop*$((v,h))$
(3) **pourtout** $v' \in \mathcal{W}$ **faire**
(4) {On suppose que la fonction *forward* a été appelée au préalable}
(5) $w(v') \leftarrow \frac{e^{-\mathcal{E}(v',h)}}{Q(v'|h)}$
(6) *bprop*$((v',h))$
(7) $\frac{\partial P(v|h)}{\partial \theta} \leftarrow \frac{\partial \mathcal{E}(v,h)}{\partial \theta} - \frac{1}{W}\sum_{v' \in \mathcal{W}} w(v') \frac{\partial \mathcal{E}(v',h)}{\partial \theta}$
(8) **retourner** $\frac{\partial P(v|h)}{\partial \theta}$

Nous proposons donc d'estimer une taille d'échantillon effective alternative par l'approximation suivante :

$$\widehat{ESS'}_m = \frac{m^2(WV - U^2)}{S(mT - R^2)} \tag{4.18}$$

où m est la taille d'échantillon pour l'échantillonnage pondéré et

$$R = \sum_{v \in \mathcal{W}} \left\| \frac{\partial \mathcal{E}(v,h)}{\partial \theta} \right\| \qquad S = \sum_{v \in \mathcal{W}} w^2(v)$$

$$T = \sum_{v \in \mathcal{W}} \left\| \frac{\partial \mathcal{E}(v,h)}{\partial \theta} \right\|^2 \qquad U = \sum_{v \in \mathcal{W}} \left\| \frac{\partial \mathcal{E}(v,h)}{\partial \theta} \right\| w(v)$$

$$V = \sum_{v \in \mathcal{W}} \left\| \frac{\partial \mathcal{E}(v,h)}{\partial \theta} \right\|^2 w(v) \qquad W = \sum_{v \in \mathcal{W}} w(v)$$

où $\mathcal{W}$ est l'échantillon en question.

Une justification de cette heuristique peut être trouvée en annexe, section C.4.

4.2 Entraînement avec une chaîne de Markov Monte Carlo

L'algorithme Metropolis-Hastings indépendant, présenté dans la section 3.3, nous est d'abord apparu comme le choix le plus évident comme méthode d'approximation du gradient. Le principe de la divergence contrastive (HINTON 2000) utilise en effet une chaîne de Markov Monte Carlo (échantillonnage de Gibbs) pour entraîner des produits d'experts, avec des résultats étonnants. Nous pensions à tort que nous pouvions appliquer avec succès le principe de la divergence contrastive à l'entraînement d'un NNLM.

Ce principe se résume à estimer le gradient négatif par le gradient sur un point généré par une seule itération d'une chaîne de Markov Monte Carlo *initialisée avec la valeur observée.*

Les algorithmes présentés ici donnent les détails de l'activation et de la rétro-propagation du NNLM avec l'algorithme de Metropolis-Hastings indépendant, avec une taille d'échantillon de n. La constante n_{init} indique le nombre d'itérations à effectuer avant d'utiliser les points générés par la chaîne. Noter également que l'étape 4 de l'algorithme 13 peut être substituée par une initialisation aléatoire.

Si on souhaite utiliser le truc de la divergence contrastive, on choisit $n_{init} = 0$ et $n = 1$, i.e., on n'attends pas que la chaîne converge et on tire un seul point. Dans ce cas l'étape 4 doit demeurer telle quelle, i.e., on doit initialiser la chaîne avec la valeur observée.

4.3 Entraînement de la densité jointe

Dans un modèle à densité conditionnelle comme le NNLM, la fonction de partition dépend de l'historique des n derniers mots h (voir équation (2.10)). Par conséquent, la partie négative du gradient (voir équation (2.14)) dépend elle aussi de l'historique. Cela a un impact sur la façon dont on doit effectuer les approximations Monte-Carlo.

Algorithme 13 Metropolis-Hastings indépendant, propagation

(1) **fonction** *forward*$((v,h))$
(2) *fpropInput(h)* {Propagation du contexte à la couche cachée}
(3) *fprop(v)* {Propagation du mot observé}
(4) $\tilde{v}_0 \leftarrow v$ {Divergence constrastive}
(5) **pour** $i \leftarrow 1$ à $n_{init} + n$ **faire** {Metropolis-Hastings}
(6) $v' \leftarrow Q(\cdot|h)$
(7) *fprop(v')* {Propagation du mot échantillonné}
(8) $u \leftarrow U(0,1)$
(9) $r \leftarrow \min\left\{1, \frac{e^{-\mathcal{E}(v',h)}Q(\tilde{v}_{i-1}|h)}{e^{-\mathcal{E}(\tilde{v}_{i-1},h)}Q(v'|h)}\right\}$
(10) **si** $u \leq r$ **alors**
(11) $\tilde{v}_i \leftarrow v'$
(12) **sinon**
(13) $\tilde{v}_i \leftarrow \tilde{v}_{i-1}$
(14) **retourner** $\{\tilde{v}_{n_{init}+1}, \ldots, \tilde{v}_{n_{init}+n}\}$

Algorithme 14 Metropolis-Hastings indépendant, rétro-propagation

(1) **fonction** *backward*$((v,h),\mathcal{W})$
(2) *bprop*$((v,h))$
(3) **pourtout** $v' \in \mathcal{W}$ **faire**
(4) {On suppose que la fonction *forward* a été appelée au préalable}
(5) *bprop*$((v',h))$
(6) $\frac{\partial P(v|h)}{\partial \theta} \leftarrow \frac{\partial \mathcal{E}(v,h)}{\partial \theta} - \frac{1}{|\mathcal{W}|}\sum_{v' \in \mathcal{W}} \frac{\partial \mathcal{E}(v',h)}{\partial \theta}$
(7) **retourner** $\frac{\partial P(v|h)}{\partial \theta}$

Dans le cas de l'échantillonnage pondéré (voir section 3.2), on doit réestimer la fonction de partition à chaque nouvelle observation. Dans le cas d'une chaîne de Markov (voir section 3.3), un problème similaire se pose : on doit, à chaque nouvel exemple, réinitialiser la chaîne et attendre sa convergence. Dans les deux cas, il serait peut-être possible de tirer parti du fait que la fonction de partition de la densité jointe ne dépend pas, quant à elle, de l'historique. Bien qu'elle dépende des paramètres θ, elle varie peu d'une itération à l'autre (du moins, si le pas de la descente de gradient – le taux d'apprentissage – n'est pas trop élevé).

Supposons que l'on cherche à maximiser la *log-vraisemblance* de la densité

jointe (2.9) plutôt que la log-vraisemblance de la densité conditionnelle. Le gradient de la log-vraisemblance inverse sur l'observation (v_t, h_t) sera

$$\frac{\partial(-\log P(v_t, h_t))}{\partial\theta} = \underbrace{\frac{\partial\mathcal{E}(v_t, h_t)}{\partial\theta}}_{\text{renforcement positif}} - \underbrace{\sum_{(v',h')\in\mathcal{V}^{n+1}} P(v', h')\frac{\partial\mathcal{E}(v', h')}{\partial\theta}}_{\text{renforcement négatif}} \tag{4.19}$$

Le gradient comporte donc une partie *renforcement positif* et une partie *renforcement négatif* qui n'est autre que l'espérance du gradient sur la densité jointe. Encore une fois, il est possible d'estimer la partie négative par échantillonnage Monte Carlo. On se retrouvera toutefois à maximiser une *pseudo-vraisemblance*, ce qui pourrait avoir comme effet de ralentir l'entraînement ; mais peut-être ce ralentissement sera-t-il contrebalancé par une accélération au niveau de l'estimation par échantillonnage (estimateur moins biaisé, donc moins de points à échantillonner).

4.3.1 Échantillonnage pondéré sur la densité jointe

Le problème avec la méthode d'échantillonnage proposée en (3.4) est que nous devons calculer la fonction de partition. La méthode proposée en (3.6) est une façon de remédier à ce problème en l'estimant.

Une autre voie de solution consiste à remarquer que la fonction de partition, dans le cas de la distribution jointe, est une constante et qu'elle est par conséquent « absorbée » par le taux d'apprentissage (« learning rate ») dans la descente de gradient. Ainsi, on peut simplement ignorer cette fonction de partition et utiliser tout simplement

$$\frac{1}{m}\sum_{i=1}^{m}\frac{\partial\mathcal{E}(V_i, H_i)}{\partial\theta}\frac{e^{-\mathcal{E}(V_i,H_i)}}{Q(V_i, H_i)} \tag{4.20}$$

comme renforcement négatif, plutôt que

$$\frac{1}{m}\sum_{i=1}^{m}\frac{\partial\mathcal{E}(V_i, H_i)}{\partial\theta}\frac{e^{-\mathcal{E}(V_i,H_i)}}{ZQ(V_i, H_i)} \tag{4.21}$$

où Z est la fonction de partition de la jointe et $(V_1, H_1), \ldots, (V_m, H_m)$ est un échantillon de variables aléatoires tirées iid de $Q(V, H)$.

Le problème ici est que si Z est différent de 1, alors la partie négative du gradient aura soit trop de poids (ce qui emmènerait $\mathcal{E}(v_t, h_t)$ vers zéro) ou pas assez (ce qui l'emmènerait vers l'infini).

La solution que nous proposons est de garder les gradients positif et négatif « balancés ». Pour ce faire, on garde en mémoire une moyenne mobile des gradients positifs et négatifs et on estime Z par le ratio de leurs normes ou par la moyenne de leurs ratios. Cette heuristique est justifiée par l'observation que l'espérance du gradient tend asymptotiquement vers zéro dans une descente de gradient, ainsi

$$\mathrm{E}\left[\frac{\partial \mathcal{E}(v,h)}{\partial \theta}\right] = \mathrm{E}\left[\frac{\partial \mathcal{E}(v,h)}{\partial \theta}\frac{e^{-\mathcal{E}(V,H)}}{ZQ(V,H)}\right]. \tag{4.22}$$

En particulier

$$Z = \frac{\left\|\mathrm{E}\left[\frac{\partial \mathcal{E}(V,H)}{\partial \theta}\frac{e^{-\mathcal{E}(V,H)}}{Q(V,H)}\right]\right\|}{\left\|\mathrm{E}\left[\frac{\partial \mathcal{E}(V,H)}{\partial \theta}\right]\right\|}. \tag{4.23}$$

Mis à part le fait que nous maximisons une pseudo-vraisemblance plutôt que la véritable vraisemblance, il y a un autre problème : c'est qu'en tirant le mot suivant v *et* son contexte h, on introduit une variabilité supplémentaire qui pourrait ralentir l'apprentissage.

4.3.2 Chaîne de Markov Monte Carlo sur la densité jointe

Quand nous entraînons sur la densité conditionnelle avec l'algorithme de Metropolis-Hastings, il faut réinitialiser la chaîne de Markov à chaque nouvel exemple tiré, car le contexte h change. Si on cherche plutôt à entraîner sur la densité jointe, nous pouvons prendre avantage du fait que la chaîne ne dépend pas de h mais seulement de θ. Or, θ change lentement à travers le temps dû à la nature de la descente de gradient (qui procède en effectuant de petits changements locaux). Donc une heuristique raisonnable consiste à

utiliser la même chaîne tout au long de l'apprentissage, c'est-à-dire à ne pas réinitialiser la chaîne à chaque nouvel exemple.

L'algorithme résultant se présente de la même façon que l'algorithme 13. Soit $Q(\cdot)$ une densité instrumentale jointe. La fonction *init()* retourne l'état actuel de la chaîne, c'est-à-dire :

- Une valeur aléatoire lors du premier appel.
- Le dernier couple $(\tilde{v}_n, \tilde{h}_n)$ retourné lors du dernier appel à *forward* lors des appels subséquents.

Algorithme 15 Metropolis-Hastings indépendant sur la densité jointe, propagation

(1) **fonction** *forward*$((v, h))$
(2) *fpropInput(h)* {Propagation du contexte à la couche cachée}
(3) *fprop(v)* {Propagation du mot observé}
(4) $(\tilde{v}_0, \tilde{h}_0) \leftarrow$ *init()*
(5) **pour** $i \leftarrow 1$ à n **faire** {Metropolis-Hastings}
(6) $(v', h') \leftarrow Q(\cdot)$
(7) *fpropInput(h')* {Propagation du contexte échantillonné}
(8) *fprop(v')* {Propagation du mot échantillonné}
(9) $u \leftarrow U(0, 1)$
(10) $r \leftarrow \min\left\{1, \frac{e^{-\mathcal{E}(v', h')} Q(\tilde{v}_{i-1}, \tilde{h}_{i-1})}{e^{-\mathcal{E}(\tilde{v}_{i-1}, \tilde{h}_{i-1})} Q(v', h')}\right\}$
(11) **si** $u \leq r$ **alors**
(12) $(\tilde{v}_i, \tilde{h}_i) \leftarrow (v', h')$
(13) **sinon**
(14) $(\tilde{v}_i, \tilde{h}_i) \leftarrow (\tilde{v}_{i-1}, \tilde{h}_{i-1})$
(15) **retourner** $\{(\tilde{v}_1, \tilde{h}_1), \ldots, (\tilde{v}_n, \tilde{h}_n)\}$

Pour clarifier, supposons qu'au temps a les paramètres du réseau soient θ_a et soit $P_{\theta_a}(v, h)$ la densité jointe du réseau de neurones. Supposons que $(\tilde{v}_0, \tilde{h}_0)$ soit la réalisation d'une variable aléatoire (V, H) issue de $P_{\theta_a}(\cdot)$. Alors l'échantillon $\{(\tilde{v}_1, \tilde{h}_1), \ldots, (\tilde{v}_n, \tilde{h}_n)\}$ produit par l'algorithme est une réalisation de n variables aléatoires iid de $P_{\theta_a}(V, H)$.

Après la mise-à-jour de θ_a, on obtient les paramètres θ_{a+1} qui définissent la fonction $P_{\theta_{a+1}}(v, h)$. Puisque $\theta_a \approx \theta_{a+1}$ alors $P_{\theta_a} \approx P_{\theta_{a+1}}$. Ainsi, on peut considérer $(\tilde{v}_n, \tilde{h}_n)$ comme une réalisation d'une variable aléatoire (V, H) issue

Algorithme 16 Metropolis-Hastings indépendant sur la densité jointe, rétro-propagation

(1) **fonction** *backward*$((v,h),\mathcal{W})$
(2) *bprop*$((v,h))$
(3) **pourtout** $(v',h') \in \mathcal{W}$ **faire**
(4) {On suppose que la fonction *forward* a été appelée au préalable}
(5) *bprop*$((v',h'))$
(6) $\frac{\partial P(v,h)}{\partial \theta} \leftarrow \frac{\partial \mathcal{E}(v,h)}{\partial \theta} - \frac{1}{|\mathcal{W}|} \sum_{(v',h') \in \mathcal{W}} \frac{\partial \mathcal{E}(v',h')}{\partial \theta}$
(7) **retourner** $\frac{\partial P(v,h)}{\partial \theta}$

de $P_{\theta_{a+1}}(\cdot)$. En posant $(\tilde{v}_0, \tilde{h}_0) = (\tilde{v}_n, \tilde{h}_n)$ lors de l'appel subséquent à l'algorithme, on obtient une réalisation de n variables aléatoires iid de $P_{\theta_{a+1}}(\cdot)$.

CHAPITRE 5

Expérimentations et résultats

Ce chapitre présente les différentes expériences que nous avons menées et les résultats associés. Nous commençons par décrire les deux bases de données sur lesquelles nous avons effectué les expérimentations. Ensuite, nous présentons les expériences qui ont bien fonctionné et leurs résultats. Notre objectif principal est de montrer qu'on peut atteindre une perplexité équivalente ou inférieure à celle atteinte par un NNLM standard en moins de temps avec une méthode d'échantillonnage. Notre objectif secondaire est de montrer que nous pouvons atteindre une perplexité inférieure à celle atteinte par une méthode traditionnelle (trigramme interpolé) avec un NNLM entraîné avec une méthode d'échantillonnage (ce qui sera vrai si on atteint l'objectif principal). Nous montrons également le rôle de la taille d'échantillon et de la distribution instrumentale dans la qualité du modèle. Nous terminons par la présentation des expériences qui n'ont pas fonctionné.

Note : Étant donné la nature de ces expériences, les temps de calcul peuvent devenir excessivement longs. Par conséquent, nous n'avons pas de résultats exhaustifs pour les expériences qui ont mal fonctionné, c'est-à-dire celles pour lesquelles le modèle divergeait très rapidement, car nous nous sommes contentés d'en faire la constatation et d'interrompre les expériences

en question afin de nous concentrer sur d'autres problèmes.

5.1 Bases de données

5.1.1 Le corpus Brown

Le corpus **Brown** consiste en 1 181 041 mots tirés de documents en Anglais (Américain) imprimés en 1961. Les documents proviennent de 15 catégories différentes afin de faire du corpus une bonne référence.

Le corpus a d'abord été divisé en trois sous-ensembles : un corpus de 800 000 mots (entraînement), un de 200 000 mots (validation) et un de 181 041 mots (test). Le vocabulaire a été généré en prenant tous les mots des ensembles d'entraînement et de validation et en enlevant les mots « rares » i.e. en ajoutant un mot « bidon » pour les mots apparaissant 3 fois ou moins dans les deux sous-ensembles. Le vocabulaire final consiste en 14 857 mots.

5.1.2 Le corpus AP-News

Le corpus **AP-News** consiste en un large ensemble de mots tirés de documents de l'Associated Press (AP) de 1995 à 1996. L'ensemble d'entraînement contient 13 994 528 mots, l'ensemble de validation, 963 138 mots et l'ensemble de test, 963 071 mots. Une fois le nombre de mots réduits en conservant seulement les mots les plus fréquents, en ignorant les majuscules, en remplaçant les symboles numériques par des symboles spéciaux, les mots rares par un autre symbole spécial ainsi que les noms propres, la taille du vocabulaire est de 17 964 mots.

5.2 Hyperparamètres

Afin de faciliter les comparaisons, nous utilisons toujours les mêmes hyperparamètres pour les modèles lorsqu'entraînés sur un problème particulier

(Brown ou AP-News). Dans le cas de Brown, ces hyperparamètres ont été repris du meilleur modèle présenté dans (BENGIO, DUCHARME, VINCENT et JAUVIN 2003) [1]. Dans le cas de AP-News, elles sont arbitraires (en fait, nous avons simplement augmenté le nombre de neurones cachées par rapport à Brown).

Hyperparamètre	Brown	AP-News
Taux d'apprentissage initial	3e-3	
Constante de décroissement	1e-8	
Pénalité sur les poids	1e-4	
Nombre de mots de contexte	3	
Dimension des vecteurs de caractéristiques (mots de contexte)	30	
Dimension des vecteurs de caractéristiques (mot de sortie)	30	
Nombre de neurones cachées	**80**	**400**

Tableau I – Hyperparamètres des réseaux de neurones

5.3 Présentation des résultats

Nous présentons les résultats de différentes expériences en commençant par celles qui ont fonctionné. Les expériences qui ont mal – ou pas – fonctionné sont présentées en second lieu, de façon brève et succincte.

5.3.1 Entraînement par échantillonnage pondéré biaisé

Nous présentons d'abord les résultats pour l'algorithme de l'échantillonnage pondéré biaisé sur la densité conditionnelle, avec différentes variantes.

[1] Quelques expériences préliminaires avec des hyperparamètres légèrement différents se sont avérées moins bonnes sur Brown et ce, que l'on entraîne de façon classique ou avec une méthode d'échantillonnage pondéré.

Le tableau II résume chacun des modèles testés dans cette section. [2]

Nous présentons dans un premier temps nos résultats les plus encourageants. Nous comparons un entraînement classique (`Brown-Classique`) à un entraînement par échantillonnage pondéré (`Brown-EP-Bi-50`).

La figure 4 présente l'évolution de la perplexité d'entraînement des deux modèles par rapport au nombre d'époques. On constate que le modèle entraîné par échantillonnage pondéré converge plus vite que le modèle entraîné de façon classique.

La figure 5 présente les erreurs de validation et de test par rapport au temps de calcul. On remarque d'abord que le modèle entraîné par échantillonnage aléatoire atteint son point de surentraînement (« overfitting ») [4] à une perplexité moindre que le modèle standard. À ce point, la perplexité de test pour le modèle entraîné par échantillonnage pondéré est également moindre : 196,6 alors que celle de l'autre modèle est de 204, ce qui constitue une amélioration inespérée de 3,8%. Par rapport à un trigramme interpolé (perplexité de 253,8 en test), c'est une amélioration de 29%.

Le temps d'entraînement total pour atteindre une perplexité en test de 204 (perplexité du modèle classique) sur un P4 2 GHz (temps machine) est de 25 heures 40 minutes (9 époques) pour l'algorithme avec échantillonnage et de 113 jours pour l'autre (18 époques). Le modèle entraîné par échantillonnage est donc plus de **100 fois** plus rapide que le modèle entraîné classiquement.

En regardant la figure 5, on réalise que le réseau entraîné par échantillonnage aléatoire complète son entraînement avant que le modèle standard n'ait le temps de compléter une première passe à travers l'ensemble d'entraînement.

Est-il avantageux d'utiliser une taille d'échantillon effective cible plus petite ? Dans les figures 6, 7 et 8, on compare le modèle `BrownEPBi50` à un modèle similaire mais avec une taille cible de 10 (`Brown-EP-Bi-10`) [5]. La

[2] Pour faciliter les explications, chaque condition expérimentale est affublée d'un code.

[4] On notera que le point de surentraînement survient à la même époque (la 18e époque) dans les deux cas.

[5] **Rappel :** La taille d'échantillon cible est le nombre de mots que nous supposons suffisants pour assurer une bonne convergence si l'on utilisait la méthode classique de

Tableau II – Conditions expérimentales

Code	Modèle	Méthode utilisée
Brown-Classique	Brown	Classique (pas d'échantillonnage) [3]
Brown-EP-Bi-50	Brown	– Échantillonnage pondéré biaisé – Bigramme adaptatif sur P – Taille d'échantillon effective (cible : 50)
Brown-EP-Bi-10	Brown	– Échantillonnage pondéré biaisé – Bigramme adaptatif sur P – Taille d'échantillon effective (cible : 10)
Brown-EP-Uni-10	Brown	– Échantillonnage pondéré biaisé – Unigramme adaptatif sur P – Taille d'échantillon effective (cible : 10)
APNews-EP-Bi-50	AP-News	– Échantillonnage pondéré biaisé – Bigramme adaptatif sur P – Taille d'échantillon effective (cible : 50)
APNews-EP-Uni-100	AP-News	– Échantillonnage pondéré biaisé – Unigramme adaptatif sur P – Taille d'échantillon effective (cible : 100)

Figure 4 – Erreur d'entraînement vs nombre d'époques pour un entraînement classique et un entraînement par échantillonnage pondéré biaisé (Brown)

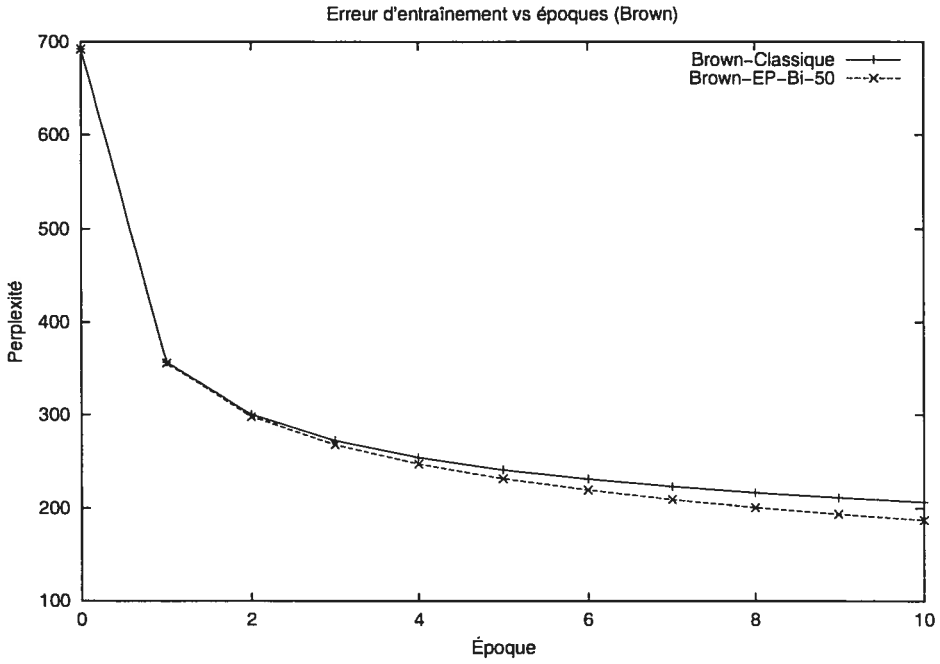


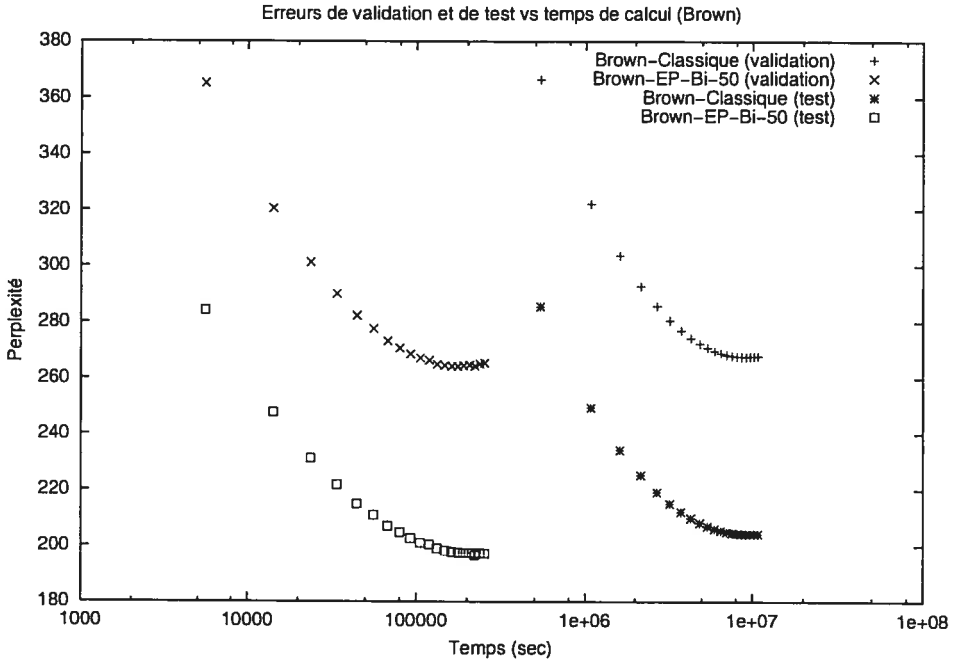
figure 6 montre que l'évolution de l'erreur d'entraînement est exactement la même pour les deux modèles. Chaque époque étant moins coûteuse avec le modèle `Brown-EP-Bi-10` (moins de points à échantillonner), ces résultats suggèrent qu'une taille d'échantillon effective cible de 10 est suffisante pour une bonne approximation du gradient.

Par contre, si on regarde l'évolution des erreurs de validation (figure 7) et de test (figure 8), on constate qu'un modèle entraîné avec une taille d'échantillon effective cible de 10 (`Brown-EP-Bi-10`) converge moins vite et conduit rapidement à un point de surentraînement plus élevé que pour `Brown-EP-Bi-50` et `Brown-Classique` i.e. le modèle généralise moins bien.

La figure suivante (9) compare deux modèles similaires entraînés par échantillonnage pondéré, mais avec des distributions instrumentales différentes. La taille d'échantillon effective ciblée est fixée à 10 et on compare un

Monte Carlo (la variable n dans l'algorithme 11).

Figure 5 – Erreur de validation et de test vs temps de calcul pour un entraînement classique et un entraînement par échantillonnage pondéré biaisé (Brown)



unigramme adaptatif (`Brown-EP-Uni-10`) à un bigramme adaptatif (`Brown-EP-Bi-10`). L'utilisation du bigramme semble conduire à une meilleure approximation du gradient si on se fie à l'évolution de l'erreur.

En observant le comportement de l'erreur de validation (figure 10), on constate qu'au point de surentraînement, elle est plus élevée pour `Brown-EP-Uni-10` que pour `Brown-EP-Bi-10`. Il en est de même pour l'erreur de test au point de surentraînement (figure 11). On pourrait objecter qu'il n'y a qu'à augmenter la taille d'échantillon cible pour contourner ce problème : il est en effet plus facile (moins coûteux) d'échantillonner un point de l'unigramme et peut-être que le surplus de points à tirer pour obtenir une bonne convergence avec une distribution instrumentale unigramme n'affecterait pas singulièrement les performances. La figure 12 nous permet de douter de cette affirmation. On remarque en effet qu'en utilisant l'unigramme, on met généralement plus de temps à atteindre une perplexité donnée qu'en utilisant

Figure 6 – Erreur d'entraînement vs nombre d'époques pour deux entraînements par échantillonnage pondéré biaisé avec des tailles cibles différentes (Brown)

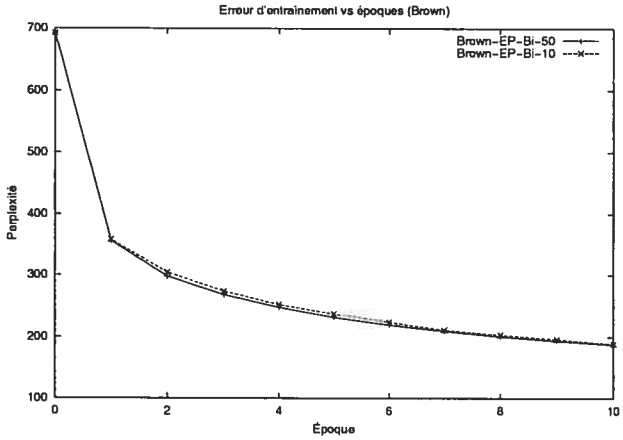
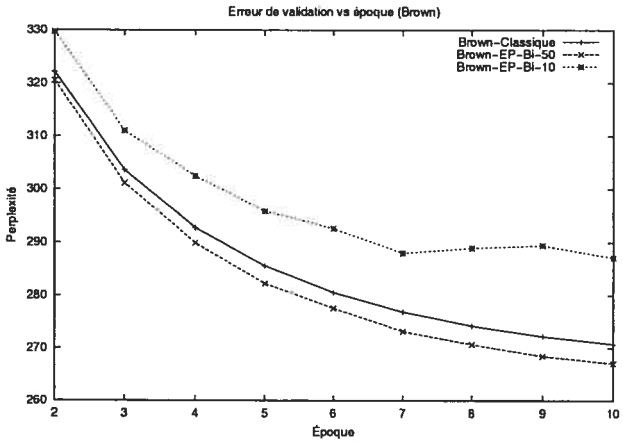


Figure 7 – Erreur de validation vs nombre d'époques pour deux entraînements par échantillonnage pondéré biaisé avec des tailles cibles différentes (Brown)



le bigramme, dû au fait que la taille d'échantillon nécessaire augmente plus rapidement, comme le montre la figure 13.

La majorité des tests ont été effectués sur Brown. La raison principale était que nous voulions tester plusieurs configurations afin de les comparer les unes aux autres. Brown a l'avantage d'être un corpus relativement petit, ce qui facilite les tests ; de plus, il est reconnu comme un corpus « étalon » dans le domaine de la modélisation du langage.

Une autre raison est que la seconde base de données considérée, AP-News, contient beaucoup plus de mots que Brown (plusieurs millions). De plus, comme les modèles sont beaucoup plus gros (400 neurones cachées plutôt que 80), les temps de calculs sont faramineux. Les expériences ont été parallélisées sur plusieurs processeurs, mais avec les méthodes d'échantillonnage on doit paralléliser au niveau de la couche cachée, ce qui est beaucoup moins avantageux que de paralléliser au niveau de la couche de sortie, comme on le fait avec les NNLM classiques (BENGIO, DUCHARME, VINCENT et JAUVIN 2003). Lors de la rétro-propagation, ce sont les mises-à-jour des vecteurs de caractéristiques qui sont les plus coûteux (ils consomment environ 50% du temps de calcul du gradient).

Au moment de rédiger ce mémoire, nous avons seulement deux résultats expérimentaux partiels qui nous permettent de douter de l'extensibilité de la méthode proposée utilisée telle quelle.

La première expérience a consisté à entraîner le modèle `APNews-Uni-100` sur un « cluster » de 20 processeurs Athlon 1GHz à mémoire partagée, équipés de cartes Myrinet. Après avoir complété 60% d'une époque (plus de 5 jours de calculs), l'expérience a planté parce que certains paramètres avaient divergé. À la fin de l'expérience, la taille d'échantillon moyenne était passée autour de 800–900 mots, ce qui explique la lenteur du calcul. Quand à la divergence des paramètres, nous pensons qu'elle peut être expliquée par une trop grande variance dans l'estimé du gradient.

La seconde expérience (`APNews-Bi-50`) a été menée dans les mêmes conditions (20 processeurs Athlon 1GHz). Cependant, nous avons cette fois-ci testé le modèle sur de petits sous-ensembles (100 000 exemples) des ensembles

Figure 8 – Erreur de test vs nombre d'époques pour deux entraînements par échantillonnage pondéré biaisé avec des tailles cibles différentes (Brown)

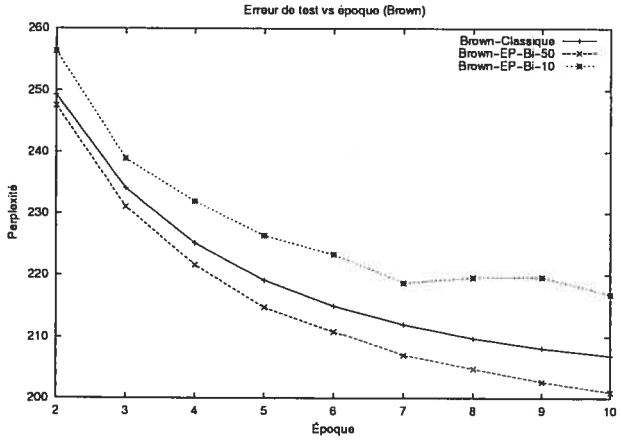


Figure 9 – Erreur d'entraînement vs nombre d'époques pour deux entraînements par échantillonnage pondéré biaisé avec deux distributions instrumentales différentes (Brown)

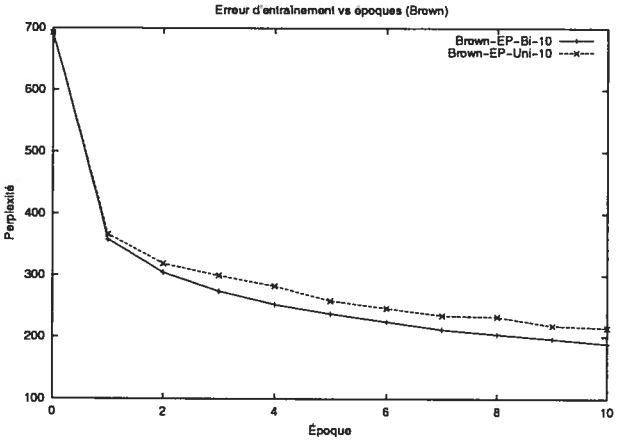


Figure 10 – Erreur de validation vs nombre d'époques pour deux entraînements par échantillonnage pondéré biaisé avec deux distributions instrumentales différentes (Brown)

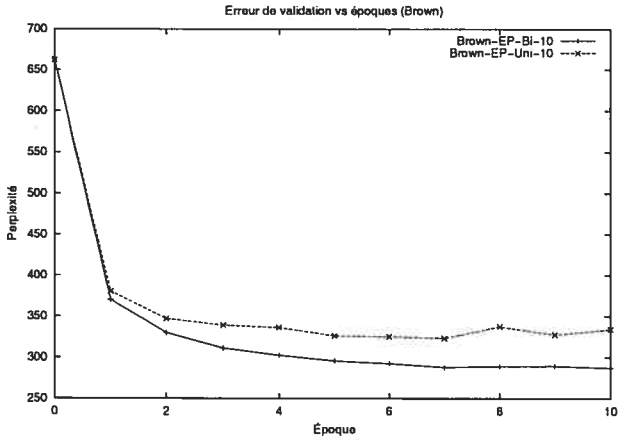


Figure 11 – Erreur de test vs nombre d'époques pour deux entraînements par échantillonnage pondéré biaisé avec deux distributions instrumentales différentes (Brown)

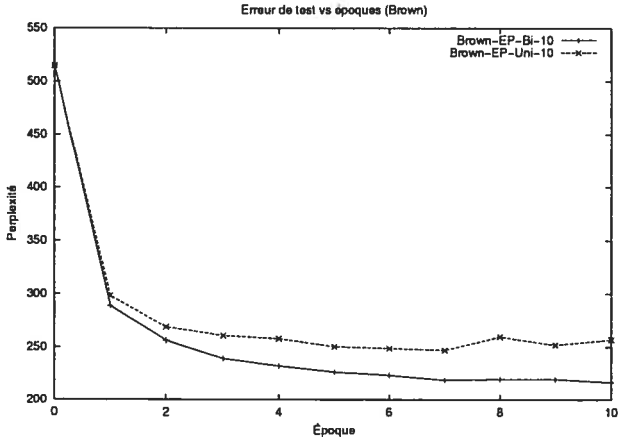


Figure 12 – Temps de calcul vs erreur d'entraînement pour deux entraînements par échantillonnage pondéré biaisé avec des distributions instrumentales différentes (Brown)

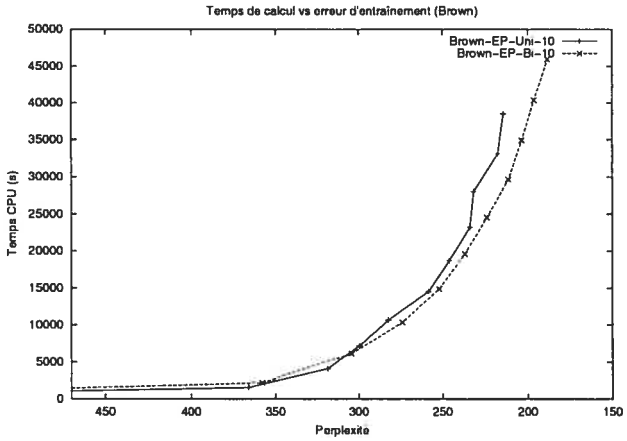
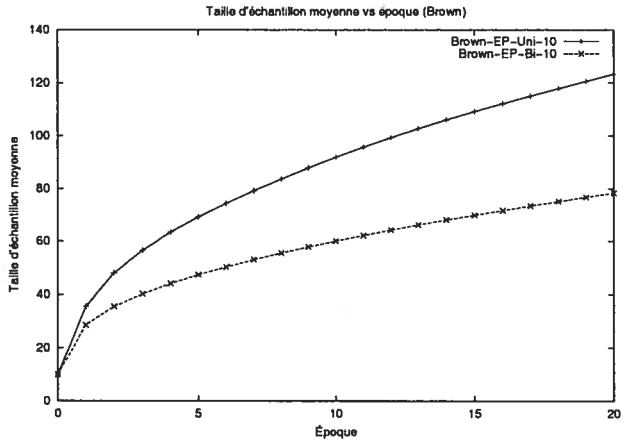


Figure 13 – Taille d'échantillon moyenne vs nombre d'époques pour deux entraînements par échantillonnage pondéré biaisé avec des distributions instrumentales différentes (Brown)



d'entraînement, de validation et de test, à tous les 500 000 mots. Comme on le voit dans la figure 14, le modèle semble converger de façon normale au début de l'entraînement. De plus, la relation nombre d'exemples vs taille d'échantillon moyenne semble être relativement linéaire (figure 15). Si on projette linéairement, on peut estimer le nombre d'échantillons moyens total à environ 400 à la fin de l'époque (i.e. en moyenne, environ 400 mots sont tirés pour estimer le gradient à chaque observation pendant cette époque). Rien ne garantit, par contre, que l'erreur continuerait de descendre.

5.4 Autres méthodes

Nous consignons ici brièvement les différentes expériences que nous avons tentées et qui ont échoué. Toutes ces expériences ont été effectuées sur Brown. Pour toutes ces expériences, les perplexités ont été évaluées sur de petits sous-ensembles (1 000 mots) des ensembles d'entraînement, de validation et de test. Ces estimés nous permettent de constater rapidement la convergence ou la divergence des différents algorithmes.

Pour toutes les expériences, les valeurs des hyperparamètres sont les mêmes que dans le tableau I pour le corpus Brown.

Le tableau III résume les résultats d'expériences menées avec un unigramme « classique » comme distribution instrumentale (voir équation 4.3). Tel qu'attendu, la convergence est bonne au début de l'entraînement, car le NNLM a la même distribution que l'unigramme. Par contre, à mesure qu'on entraîne le réseau, sa distribution s'éloigne de celle de l'unigramme et la taille d'échantillon requise pour avoir un bon estimé du gradient augmente trop rapidement.

Le tableau IV présente les résultats pour des modèles entraînés par échantillonnage pondéré avec, pour distribution instrumentale, des n-grammes adaptatifs entraînés à approximer la distribution instrumentale Q^*, définie dans l'équation (4.2), qui minimise la variance de l'échantillonnage pondéré. Les résultats ne sont pas désastreux mais le calcul de la norme du gradient,

Figure 14 – Erreurs d'entraînement, de validation et de test vs nombre d'époques pour entraînement par échantillonnage pondéré biaisé (AP-News)

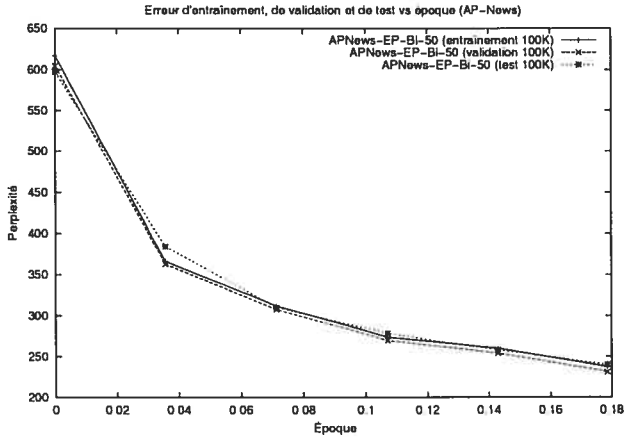
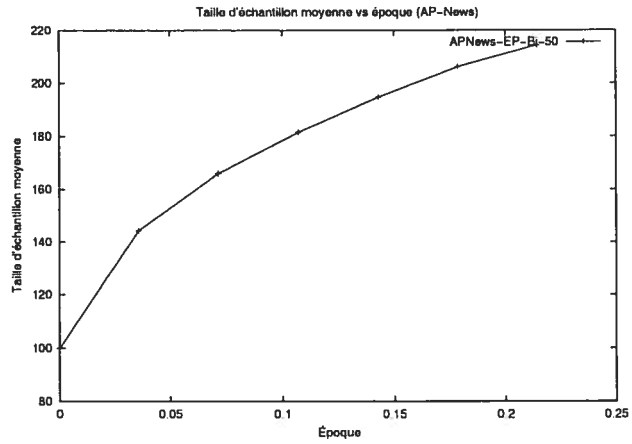


Figure 15 – Taille d'échantillon moyenne vs nombre d'époques pour entraînement par échantillonnage pondéré biaisé (AP-News)



que nécessite l'approximation de Q^*, sont très coûteux en temps de calcul pour une amélioration douteuse. La taille d'échantillon effective alternative (voir section 4.1.2) sous-estime le nombre d'échantillons nécessaires.

Dans le tableau V, on résume les résultats des expériences menées avec Metropolis-Hastings. On utilise un unigramme classique comme distribution instrumentale. Aucune de ces expériences n'a fonctionné. Le tableau VI montre les résultats tout aussi infructueux de modèles entraînés avec un trigramme adapté sur la perplexité.

Le tableau VII résume les résultats des expériences menées avec échantillonnage pondéré biaisé sur la distribution jointe. La distribution instrumentale est :

$$Q(v_1, \ldots, v_n) = \prod_{i=1}^{n} P_1(v_i) \tag{5.1}$$

où P_1 est la distribution unigramme telle que définie dans (4.3). Aucun des modèles n'a convergé.

Enfin, le dernier tableau (VIII) présente les résultats pour des expériences sur la densité jointe, mais cette fois-ci avec la méthode de Metropolis-Hastings. La distribution instrumentale utilisée est (5.1).

Tableau III – Entraînement par échantillonnage pondéré biaisé, unigramme

Méthode utilisée	Résultats
Taille d'échantillon : méthode brute ($m_0 = 10$)	Divergence dès la première époque.
Taille d'échantillon : méthode brute ($m_0 = 100$)	Bonne convergence, mais le nombre d'échantillons requis augmente si rapidement que les performances ne sont pas bonnes : après 4 époques, le nombre d'échantillons requis passe à 3200.
Taille d'échantillon effective (cible : 10)	Divergence dès la première époque.
Taille d'échantillon effective (cible : 100)	Bonne convergence, mais le nombre d'échantillons requis augmente si rapidement que les performances ne sont pas bonnes : après une époque, le nombre d'échantillons moyen requis est environ 700.

Tableau IV – Entraînement par échantillonnage pondéré biaisé, unigrammes et bigrammes adaptés sur Q^*

Méthode utilisée	Résultats
– Unigramme adaptatif – Taille d'échantillon effective (cible : 10)	Les résultats ne sont pas meilleurs qu'avec un unigramme adaptatif sur P, alors que le temps de calcul est de 3 à 4 fois plus élevé.
– Bigramme adaptatif – Taille d'échantillon effective (cible : 10)	Les résultats ne sont pas meilleurs qu'avec un bigramme adaptatif sur P, alors que le temps de calcul est de 2 fois plus élevé.
– Unigramme adaptatif – Taille d'échantillon effective alternative (cible : 10)	Diverge dès la première époque. La taille d'échantillon moyenne reste autour de 10 (insuffisant).

Tableau V – Entraînement par Metropolis-Hastings indépendant, unigramme classique

Méthode utilisée	Résultats
– Taille d'échantillon : méthode brute ($m_0 = 1$) – $m_{init} = 0$	Diverge dès la première époque.
– Taille d'échantillon : méthode brute ($m_0 = 100$) – $m_{init} = 0$	Diverge dès la première époque.
– Taille d'échantillon : méthode brute ($m_0 = 100$) – $m_{init} = 1000$	Diverge dès la première époque.

Tableau VI – Entraînement par Metropolis-Hastings indépendant, trigramme adapté sur la perplexité

Méthode utilisée	Résultats
– Taille d'échantillon : taille fixe (10) – $m_{init} = 0$	Divergence dès la première époque.
– Taille d'échantillon : taille fixe (100) – $m_{init} = 0$	Divergence dès la première époque.

Tableau VII – Entraînement par échantillonnage pondéré biaisé, distribution jointe, unigramme classique

Méthode utilisée	Résultats
Taille d'échantillon : taille fixe (100)	Diverge dès la première époque.
Taille d'échantillon : taille fixe (1000)	Diverge dès la première époque.

Tableau VIII – Entraînement par Metropolis-Hastings indépendant, distribution jointe, unigramme classique

Méthode utilisée	Résultats
– Taille d'échantillon : taille fixe (1) – $m_{init} = 0$	Diverge dès la première époque.
– Taille d'échantillon : taille fixe (100) – $m_{init} = 100$	Diverge dès la première époque.

CHAPITRE 6

Conclusion

Nous concluons ce mémoire en rappelant brièvement les résultats des expériences que nous avons menées. Nous terminons en discutant des possibles extensions à ce projet.

6.1 Discussion

Les résultats de nos expérimentations nous permettent de conclure qu'il est possible d'obtenir une accélération significative de l'entraînement d'un réseau de neurones probabiliste dans un problème avec beaucoup de classes (ici, la modélisation du langage) en estimant le gradient par échantillonnage pondéré biaisé. Sur un problème typique (corpus Brown), l'accélération est de l'ordre de deux magnitudes (100 fois plus rapide) si on entraîne sur un seul processeur.

Nous avons également démontré l'importance de choisir une taille d'échantillon suffisante. Nous pouvons ajuster la taille d'échantillon grâce à une mesure heuristique, la taille d'échantillon effective ; cependant, il est important que la taille d'échantillon ciblée soit suffisante afin de compenser pour

l'augmentation du bruit dans le gradient.

Nous avons montré l'importance, également, d'utiliser une distribution instrumentale qui s'adapte à la distribution du modèle au cours de l'entraînement. Nous avons développé une méthode qui permet d'adapter ainsi des modèles n-gramme (unigramme et bigramme) de façon efficace. Nous avons vu qu'il valait la peine d'utiliser un bigramme adaptatif plutôt qu'un unigramme adaptatif.

Nous n'avons pas sû montrer si la technique pouvait s'étendre à des problèmes plus complexes, en particulier à des problèmes sur des corpus comportant plusieurs millions de mots. Nous conjecturons cependant qu'elle nécessite des améliorations supplémentaires, ne serait-ce que du point de vue de la parallélisation sur plusieurs machines.

Enfin, nous avons écarté un vaste ensemble de méthodes et d'ajustements qui ne sont pas parvenus à assurer la convergence des modèles ou qui se sont avérés très inefficaces ; en particulier, l'algorithme de Metropolis-Hastings et la maximisation d'une pseudo-vraisemblance (densité jointe).

6.2 Travaux futurs

Le premier défi qui nous attend, si nous souhaitons poursuivre dans cette voie de recherche, est d'étendre la méthode développée à des problèmes plus difficiles. Nous conjecturons que des modèles distribués comme le NNLM sont en mesure de mieux apprendre les dépendances de haute dimensionalité présentes dans la langue que des modèles locaux plus traditionnels, comme les n-grammes : ils sont en mesure d'extraire plus d'information que ces derniers sur de grosse bases de données. Or, quelques problèmes demeurent lorsqu'on entraîne un gros modèle sur une grosse base de données.

Le premier problème survient au niveau de la parallélisation sur plusieurs machines. D'abord, l'utilisation de la méthode nécessite de paralléliser au niveau de la couche cachée. Le désavantage est que le coût des communications est généralement plus élevé dans ce cas. En effet, le gros de ces échanges se

fait au niveau des vecteurs de caractéristiques pour les mots de sortie. Si le nombre moyen de mots tirés est de l'ordre de 100 et que la dimension des vecteurs de caractéristiques est de l'ordre de 10 alors le coût des communications est de l'ordre de 1000. Si on parallélise sur la couche de sortie, les communications sont plutôt dans $O(n_h)$ où n_h est le nombre de neurones cachées, généralement quelques dizaines ou centaines. Cette difficulté se ressent lorsqu'on entraîne en parallèle : le coût des communications devient si élevé qu'il ne vaut souvent même pas la peine de paralléliser. Une première optimisation consiste à réduire au maximum le nombre de communications, chaque communication ayant un coût de base élevé. Une seconde idée serait d'effectuer la mise-à-jour des vecteurs de façon asynchrone. Le transfert des données sur le gradient des vecteurs de caractéristiques serait exécuté par une sous-routine, pendant que les calculs se poursuivent dans le processus parent. Étant donné que les mises-à-jour sont relativement petites, ça ne devrait pas fausser de beaucoup les calculs effectués par le processus parent.

Le second problème vient du fait que, d'après ce que nous avons observé, la méthode demeure extrêmement sensible à la qualité de la distribution instrumentale et à sa capacité à s'adapter à la distribution cible, ainsi qu'au nombre d'échantillons tirés. Par exemple, bien qu'un unigramme adaptatif soit échantillonnable et adaptable à peu de frais, la taille de l'échantillon nécessaire pour avoir une bonne estimation du gradient rend cette économie négligeable par rapport au coût qu'engendre le calcul du gradient sur un plus grand nombre de points. Le problème peut s'énoncer ainsi : existe-t-il une méthode générale qui permette de s'adapter à l'augmentation inévitable de la complexité de la distribution du réseau afin d'éviter à la fois de diverger et de devoir tirer un trop grand nombre de points ? Nous croyons qu'il serait intéressant de considérer d'autres types de distributions instrumentales adaptatives, avec plus de capacité (ou dont le nombre de paramètres augmente au besoin). Nous pensons également qu'il serait bon d'explorer certaines méthodes Monte Carlo supplémentaires. En particulier, nous sommes intéressés à essayer la méthode de la pondération dynamique (Liu, Liang et Wong 2001) qui est une méthode « hybride » entre une chaîne de Mar-

kov Monte Carlo et l'échantillonnage pondéré biaisé. Nous n'avons pas utilisé de distribution instrumentale adaptative avec Metropolis-Hastings : il serait intéressant de voir si cela peut améliorer les résultats de cet algorithme.

Le dernier problème est que l'utilisation de l'estimateur de taille d'échantillon effective nécessite une taille d'échantillon cible qui correspond à la taille d'échantillon requise par l'estimateur Monte Carlo classique. Cela ne règle pas un problème : nous devons décider arbitrairement de cette taille cible. De plus, nous savons qu'à mesure que les paramètres du réseau sont entraînés, le réseau devient de plus en plus bruité et nécessite par conséquent une plus grande taille cible. Au début de l'entraînement, quand tous les paramètres sont initialisés aléatoirement selon une loi normale, la variance est faible et on peut utiliser une taille d'échantillon très faible ; mais plus tard pendant l'entraînement, il faudrait normalement augmenter la taille cible à mesure que la variance du gradient augmente. On peut utiliser la méthode brute pour l'estimer, mais cela nécessite d'évaluer la perplexité sur un sous-ensemble, ce qui est assez coûteux. Une idée serait d'estimer la perplexité « à la volée » : il suffit d'estimer la fonction de partition, ce qu'on peut faire en réutilisant les échantillons déjà tirés. Une autre façon – plus simple – d'adapter la taille cible est d'utiliser une fonction croissante dans le temps (un genre de taux d'apprentissage), e.g., $n_a = \lceil n_0 + a\epsilon \rceil$ où a est le nombre d'observations, n_0 est la taille cible initiale et $\epsilon > 0$ est une constante qui détermine la croissante de la taille cible dans le temps.

Nous avons été surpris de constater que notre meilleur modèle a convergé en moins d'époques – et vers un meilleur taux d'erreur – que le modèle classique. La différence en erreur de test est significative. Nous nous devons de vérifier si ce phénomène se répète sur d'autres problèmes : il est étonnant qu'une approximation se comporte mieux que la méthode exacte.

Pour terminer, la méthode que nous avons développée peut en fait s'étendre à d'autres problèmes que la modélisation de la langue. En fait, on peut l'utiliser pour tout problème de classification ou d'estimation de probabilité avec un grand nombre de classes. Par exemple, on pourrait l'utiliser pour entraîner un réseau de neurones à reconnaître les caractères chinois.

Références

BENGIO, S. et Y. BENGIO (2000), « Taking on the Curse of Dimensionality in Joint Distributions Using Neural Networks », *IEEE Transaction on Neural Networks special issue on data mining and knowledge discovery 11*(3), p. 550–557.

BENGIO, Y. (2002), « New Distributed Probabilistic Language Models », Rapport technique 1215, Département d'Informatique et de Recherche Opérationnelle, Université de Montréal.

BENGIO, Y., R. DUCHARME, P. VINCENT et C. JAUVIN (2003), « A Neural Probabilistic Language Model », *Journal of Machine Learning Research 3*, p. 1137–1155.

BENGIO, Y. et J.-S. SENÉCAL (2003), « Quick Training of Probabilistic Neural Nets by Sampling », *Proceedings of the Ninth International Workshop on Artificial Intelligence and Statistics*,

BISHOP, C. (1995), *Neural Networks for Pattern Recognition*. Oxford University Press.

CHARNIAK, E. (1999), « A Maximum-entropy-inspired Parser », Rapport technique CS-99-12, Brown University.

FOSTER, G. (2002), *Text Prediction for Translators*, Ph. D. thesis, Département d'Informatique et de Recherche Opérationnelle, Université de Montréal.

HASTINGS, W. K. (1970), « Monte Carlo Sampling Methods Using Markov Chains and their Applications », *Biometrika 57*(1), p. 97–109.

HINTON, G. (1986), « Learning Distributed Representations of Concepts », *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*, Lawrence Erlbaum, Hillsdale, p. 1–12.

HINTON, G. (1999), « Products of Experts », *Proceedings of the Ninth International Conference on Artificial Neural Networks*, p. 1–6.

HINTON, G. (2000), « Training Products of Experts by Minimizing Contrastive Divergence », Rapport technique 2000-004, Gatsby Computational Neuroscience Unit, University College London.

HUFFMAN, D. A. (1952), « A Method for the Construction of Minimum-Redundancy Codes », *Proceedings of the IRE*, p. 1098–1101.

JELINEK, F. (1990), *Readings in Speech Recognition*, p. 405–506. Morgan Kaufmann, voir la référence (WALBEL et LEE 1990).

KONG, A. (1992), « A Note on Importance Sampling using Standardized Weights », Rapport technique 348, Department of Statistics, University of Chicago.

KONG, A., J. S. LIU et W. H. WONG (1994), « Sequential Imputations and Bayesian Missing Data Problems », *Journal of the American Statistical Association 89*, p. 278–288.

LIU, J. S. (2001), *Monte Carlo Strategies in Scientific Computing*. Springer.

LIU, J. S., F. LIANG et W. H. WONG (2001), « A Theory for Dynamic Weighting in Monte Carlo Computation », *Journal of the American Statistical Association 96*, p. 561–573.

LUO, Z. Q. (1991), « On the Convergence of the LMS Algorithm with Adaptive Learning Rate for Linear Feedforward Networks », *Neural Computation 3*(2), p. 226–245.

MANNING, C. D. et H. SCHÜTZE (1999), *Foundations of Statistical Natural Language Processing*. MIT Press.

MINSKY, M. L. et S. A. PAPERT (1969), *Perceptrons*. MIT Press.

RICE, J. A. (1988), *Mathematical Statistics and Data Analysis*. Wadsworth and Brooks/Cole.

RIEGER, B. B. (1991), « On Distributed Representation in Word Semantics », Rapport technique TR-91-012, International Computer Science Institute, UC Berkeley.

ROBERT, C. P. et G. CASELLA (2000), *Monte Carlo Statistical Methods.* Springer, Springer texts in statistics.

ROSENBLATT, F. (1957), « The Perceptron – A Perceiving and Recognizing Automaton », Rapport technique 85-460-1, Cornell Aeronautical Laboratory.

RUMELHART, D. E., G. E. HINTON et R. J. WILLIAMS (1986), « Learning Representations by Back-propagating Errors », *Nature* (323), p. 533–536.

SCHWENK, H. et J.-L. GAUVAIN (2002), « Connectionist Language Modeling for Large Vocabulary Continuous Speech Recognition », *Proceedings of ICASSP*, p. 765–768.

SHANNON, C. E. (1948), « A Mathematical Theory of Communication », *Bell Systems Technical Journal 27*, p. 379–423.

WALBEL, A. et K.-F. LEE (Édits.) (1990), *Readings in Speech Recognition.* Morgan Kaufmann.

WITTGESTEIN, L. (1961), *Tractatus logico-philosophicus suivi de Investigations philosophiques*, Collection Tel. Gallimard.

ANNEXE A

Glossaire

Descente de gradient Algorithme d'entraînement permettant de minimiser une fonction de coût L en calculant son gradient par rapport à ses paramètres θ. À chaque étape de l'algorithme, le gradient $\frac{\partial C}{\partial \theta}$ est calculé. Les paramètres sont mis à jour en descendant dans la direction inverse à leur gradient. Le gradient sur le coût total est généralement estimé en prenant le gradient sur un seul exemple à la fois, une méthode connue sous le nom de *descente de gradient stochastique*. Pour plus de détails, voir la section 1.3.

Époque En apprentissage-machine, une époque correspond à un parcours de tous les points de l'ensemble d'entraînement, pendant l'entraînement. Typiquement, un entraînement par descente de gradient stochastique (voir section 1.3) nécessite de visiter plusieurs fois l'ensemble d'entraînement, en mettant à jour à chaque fois les paramètres pour chaque point de l'ensemble.

Pénalité sur les poids Constante utilisée dans la mise-à-jour des paramètres d'un modèle entraîné par descente de gradient, permettant de relaxer une explosion de la norme des paramètres qui introduit de l'instabilité dans la fonction d'activation du modèle. Pour plus de détails, voir la section 1.3.

Réseau de neurones Modèle bien connu dans le domaine de l'apprentissage machine. Un réseau de neurones est un modèle non-paramétrique défini par une fonction d'activation

$$f(\mathbf{x}) = \mathbf{w} \cdot h(\mathbf{W} \cdot \mathbf{x} + \mathbf{d}) + b$$

où $\mathbf{x}$ est l'entrée du réseau, $\mathbf{w}$ est un vecteur de *poids de sortie*, b est un *biais de sortie*, h est une fonction vectorielle non-linéaire, $\mathbf{W}$ est une matrice de *poids cachés* et $\mathbf{d}$ est un vecteur de *biais de sortie*. Pour plus de détails, voir la section 1.2.

Rétro-propagation Algorithme bien connu en apprentissage machine qui permet de calculer le gradient de façon efficace dans un réseau de neurones. Pour plus de détails, voir la section 1.3.

Taux d'apprentissage Le pas employé lors de la descente de gradient. Dans une descente de gradient stochastique, les mises-à-jour sur les paramètres sont de la forme

$$\theta \leftarrow \theta - \eta \frac{\partial C}{\partial \theta}$$

où C est une fonction de coût et θ est un paramètre à optimiser. La variable η est le taux d'apprentissage. Pour plus de détails, voir la section 1.3.

ANNEXE B

Traitement de la langue naturelle

Cet annexe présente rapidement le domaine du traitement de la langue naturelle et introduit les modèles de langue à base markovienne.

B.1 Introduction : le modèle du canal bruité

Le problème de base du traitement de la langue naturelle se pose dans les termes suivants. Considérons $s = v_1^N$ une séquence de symboles générée par une distribution inconnue $P(S)$. Cette séquence nous parvient indirectement, sous la forme d'une séquence d'observations $o = o_1^M$, en passant à travers un *canal bruité* (SHANNON 1948) de distribution (inconnue) $P(O|S)$. L'objectif du traitement de la langue naturelle est de reconstituer la séquence originale s en se basant sur l'observation o.

Parmi toutes les séquences s' possibles, on choisira la séquence $\hat{s}$ la plus vraisemblable étant donné l'observation o, i.e. on prendra $\hat{s}$ qui maximise

$P(s'|o)$. En reprenant le théorème de Bayes et en notant que $P(o)$ est constant :

$$\hat{s} = \underset{s'}{\operatorname{argmax}} P(s'|o) = \underset{s'}{\operatorname{argmax}} \frac{P(o|s')P(s')}{P(o)} = \underset{s'}{\operatorname{argmax}} P(o|s')P(s') \quad \text{(B.1)}$$

Dans (B.1), $P(o|s')$ est la *probabilité de canal* et $P(s')$ est le *modèle de langue.*

Le modèle du canal bruité est en effet applicable à une variété de problèmes. Si on considère seulement les problèmes où les séquences s sont des phrases dans une langue naturelle, la traduction automatisée (l'observation est alors une séquence de mots dans une autre langue) et la reconnaissance vocale sont les exemples les plus communs.

B.2 Modélisation de la langue naturelle

Le problème de la modélisation du langage consiste principalement à trouver une distribution $\hat{P}$ qui approxime le mieux possible la vraie distribution P du langage.

La distance entre deux distributions peut être mesurée par la *divergence de Kullback-Leibler*, aussi appelée *entropie relative* :

$$KL(P||\hat{P}) = \mathrm{E}_P \left[\log \frac{P(X)}{\hat{P}(X)} \right]. \quad \text{(B.2)}$$

Malheureusement, nous ne connaissons pas la vraie distribution P donc il est impossible de calculer cette distance. Cependant, il est possible de calculer une mesure intimement reliée, appelée *entropie croisée.* L'entropie croisée entre X, une variable aléatoire tirée de la vraie distribution P, et la distribution $\hat{P}$, est donnée par

$$H(X,Q) = H(X) + KL(P||\hat{P}) = \mathrm{E}_P \left[\log \frac{1}{\hat{P}(X)} \right] \quad \text{(B.3)}$$

où $H(X) = \mathrm{E}_P\left[\log \frac{1}{P(X)}\right]$ est appelé l'*entropie* de X.

Si on assume que la séquence de symboles (mots) x_1^N est générée par un processus ergodique et stationnaire, on peut estimer (B.3) par

$$H(X, \hat{P}) \approx -\frac{1}{N} \log \hat{P}(x_1^N). \tag{B.4}$$

En modélisation de la langue, on mesure généralement le taux d'erreur avec la *perplexité* qui n'est que l'exponentielle de l'entropie croisée :

$$\text{perplexité}(x_1^N, \hat{P}) = e^{H(x_1^N, \hat{P})} = \sqrt[N]{\hat{P}(x_1^N)}. \tag{B.5}$$

De façon intuitive, cette mesure indique grosso-modo le nombre moyen de symboles entre lesquels le modèle « hésite » à chacune des N étapes.

B.3 L'hypothèse markovienne

Il existe plusieurs approches pour modéliser la langue. Les modèles les plus populaires sont basés sur une approximation markovienne. Soit une séquence de mots v_1^N tirée de la distribution $P(S)$. On a que

$$P(v_1^N) = \prod_{t=1}^{N} P(v_t | v_1^{t-1}). \tag{B.6}$$

Intuitivement, ce résultat vient de la remarque suivante : il est possible de générer une séquence v_1^N de façon itérative, en générant un mot v_t à la fois issu de $P(V|v_1^{t-1})$. En répétant ce processus N fois, on obtient une séquence v_1^N issue de $P(S)$.

L'hypothèse markovienne consiste à reprendre la forme de (B.6), mais en considérant à chaque étape une fenêtre de n mots :

$$P(v_1^N) \approx \prod_{t=1}^{N} P(v_t | v_{t-n+1}^{t-1}). \tag{B.7}$$

Cette approximation présuppose que ce sont les mots les plus près qui aident le mieux à prédire le prochain mot, une assomption qui est vraie dans beaucoup de cas.

B.4 Le modèle n-gramme

Un n-gramme est un modèle bien connu dans le domaine du traitement de la langue qui fait l'utilisation de l'hypothèse markovienne pour modéliser le langage. Les prédictions d'un n-gramme sont basées sur de simples comptes de fréquence de cooccurrences de mots.

Soit $D_N = (v_1, \ldots, v_N)$ une séquence de mots (corpus) $v_t \in \mathcal{V}$. Un tel ensemble de mots en séquence est appelé un *corpus*. Les probabilités P_n associées à un modèle n-gramme sont définies par

$$P_n(v|h) = \begin{cases} \frac{|h,v|}{|h|} & \text{si } |h,v| > 0 \\ 0 & \text{sinon} \end{cases} \tag{B.8}$$

où h est la séquence des $n-1$ derniers mots et $|s|$ est la *fréquence* de la séquence s dans le corpus D_N, i.e., le nombre de fois que cette séquence apparaît dans ce corpus.

Le problème le plus évident avec ces modèles est leur faible capacité de généralisation. Par exemple, plus n est grand, plus il y aura des cooccurrences de mots pour lesquelles la probabilité estimée (B.8) sera nulle car elles n'auront pas été observées dans le corpus d'entraînement. C'est pourquoi plusieurs méthodes ont été développées afin de redistribuer la masse de probabilité dans le but de donner des probabilités non-nulles à des cooccurrences non-observées.

Une méthode bien connue consiste à mixer plusieurs modèles n-grammes. La probabilité du modèle devient :

$$P'_n(v|h) = \sum_{i=0}^{n} \alpha_i(h) P_i(v|h). \tag{B.9}$$

Les $\alpha_i(h)$ sont des poids tels que $\forall h \sum_{i=0}^{n} \alpha_i(h) = 1$. Le modèle P_0 est le « zéro-gramme » ; il attribue une probabilité égale à tous les mots du vocabulaire, peu importe le contexte i.e. $P_0(v|h) = P_0 = \frac{1}{|\mathcal{V}|}$.

Les poids $\alpha_i(h)$ sont adaptés par EM (« expectation-maximisation ») afin de minimiser la perplexité sur un ensemble disjoint de l'ensemble d'entraînement (JELINEK 1990) [1].

[1] EM est un algorithme d'entraînement utilisé pour entraîner des modèles paramétriques comme les mixtures de gaussienne ou les poids d'un trigramme interpolé. L'algorithme EM comprend deux phases. Lors de la première phase (E), les probabilités a posteriori des modèles sont calculés en utilisant les paramètres actuels du modèle. Lors de la seconde (M), les paramètres sont mis à jour en fonction des nouvelles probabilités a posteriori calculées pendant la phase E.

ANNEXE C

Développements et démonstrations

Cet annexe présente les développements mathématiques et les démonstrations trop substantiels pour être intégrés directement au document principal.

C.1 Distribution instrumentale de variance minimale pour l'échantillonnage pondéré

Soit l'ensemble $\mathcal{X}$ et une loi de distribution P. Soit X une variable aléatoire définie sur l'espace de probabilité $(\mathcal{X}, P)$. Soit $h(x)$ une fonction de domaine $\mathcal{X}$. On cherche à estimer $\mathrm{E}_P[h(X)]$ par échantillonnage pondéré (voir équation (3.4)). La distribution instrumentale Q^* qui minimise la va-

riance de cet estimateur est [1]

$$Q^*(x) = \frac{|h(x)|P(x)}{\sum_{x' \in \mathcal{X}} |h(x')|P(x')}. \tag{C.1}$$

Cependant, dans le cas qui nous intéresse, on cherche à estimer l'espérance du gradient. Or, le gradient est en fait un vecteur (dont chaque élément correspond à la dérivée partielle par rapport à un paramètre) et nous voulons utiliser la même distribution instrumentale pour tous les éléments du vecteur. Soit $\mathbf{h}(x)$ ce vecteur (le gradient). Ré-écrivons la variance de l'estimateur par échantillonnage pondéré pour estimer $\mathrm{E}_P[\mathbf{h}(X)]$ (voir équation (3.4)) :

$$\begin{aligned}
\mathrm{Var}_Q[\hat{I}_m] &= \mathrm{Var}_Q\left[\frac{1}{m}\sum_{i=1}^{m}\mathbf{h}(X_i)\frac{P(X_i)}{Q(X_i)}\right] \\
&= \frac{1}{m}\mathrm{Var}_Q\left[\mathbf{h}(X)\frac{P(X)}{Q(X)}\right] \\
&= \frac{1}{m}\left(\mathrm{E}_Q\left[\|\mathbf{h}(X)\|^2\frac{P^2(X)}{Q^2(X)}\right] - \left\|\mathrm{E}_Q\left[\mathbf{h}(X)\frac{P(X)}{Q(X)}\right]\right\|^2\right).
\end{aligned} \tag{C.2}$$

Puisque

$$\left\|\mathrm{E}_Q\left[\mathbf{h}(X)\frac{P(X)}{Q(X)}\right]\right\| = \|\mathrm{E}_P[\mathbf{h}(X)]\|$$

(indépendant de Q) minimiser $\mathrm{Var}_Q[\hat{I}_m]$ revient à minimiser $\mathrm{E}_Q\left[\|\mathbf{h}(X)\|^2\frac{P^2(X)}{Q^2(X)}\right]$. Ré-écrivons ce terme en ajoutant un lagrangien de constante λ, afin de s'assurer que $\sum_{x' \in \mathcal{X}} Q(x') = 1$, i.e.,

$$\sum_{x' \in \mathcal{X}} \|\mathbf{h}(x')\|^2\frac{P^2(x')}{Q^2(x')}Q(x') + \lambda\left(\sum_{x' \in \mathcal{X}} Q(x') - 1\right). \tag{C.3}$$

Dérivons maintenant cette expression par rapport à $Q(x)$ et résolvons

[1] Une preuve de ce théorème peut être trouvée dans (Robert et Casella 2000, pp. 84–85).

pour que la dérivée soit égale à zéro en $Q(x) = Q^*(x)$. On obtient

$$-\|\mathbf{h}(x)\|^2 \frac{P^2(x)}{Q^{*2}(x)} + \lambda = 0. \tag{C.4}$$

Si on résout pour $Q^*(x)$ on a

$$Q^*(x) = \frac{1}{\sqrt{\lambda}} \|\mathbf{h}(x)\| P(x). \tag{C.5}$$

Trouvons maintenant λ tel que $\sum_{x' \in \mathcal{X}} Q^*(x') = 1$:

$$\begin{aligned} 1 &= \sum_{x' \in \mathcal{X}} Q^*(x') \\ &= \sum_{x' \in \mathcal{X}} \frac{1}{\sqrt{\lambda}} \|\mathbf{h}(x')\| P(x') \\ &= \frac{1}{\sqrt{\lambda}} \sum_{x' \in \mathcal{X}} \|\mathbf{h}(x')\| P(x') \\ \iff \sqrt{\lambda} &= \sum_{x' \in \mathcal{X}} \|\mathbf{h}(x')\| P(x') \end{aligned} \tag{C.6}$$

d'où

$$Q^*(x) = \frac{\|\mathbf{h}(x)\| P(x)}{\sum_{x' \in \mathcal{X}} \|\mathbf{h}(x')\| P(x')}. \square \tag{C.7}$$

C.2 Équivalence entre l'échantillonnage pondéré biaisé et la méthode classique

Soit $X_1, \ldots, X_n$ un ensemble de variables aléatoires tirées iid de la distribution instrumentale $Q = P$ où $P(x) = \frac{1}{Z} f(x)$. Alors l'estimateur par échantillonnage pondéré biaisé $\tilde{I}_n$ (voir équation (3.6)) est équivalent à l'estimateur Monte Carlo classique $\hat{h}_n$ (voir équation (3.3)).

Preuve :

$$\begin{aligned}
\tilde{I}_n &= \frac{1}{W}\sum_{i=1}^{n} h(X_i)w(X_i) \\
&= \frac{\sum_{i=1}^{n} h(X_i)\frac{f(X_i)}{Q(X_i)}}{\sum_{i=1}^{n}\frac{f(X_i)}{Q(X_i)}} \\
&= \frac{\sum_{i=1}^{n} h(X_i)\frac{f(X_i)}{P(X_i)}}{\sum_{i=1}^{n}\frac{f(X_i)}{P(X_i)}} \\
&= \frac{\sum_{i=1}^{n} h(X_i)\frac{ZP(X_i)}{P(X_i)}}{\sum_{i=1}^{n}\frac{ZP(X_i)}{P(X_i)}} \\
&= \frac{Z\sum_{i=1}^{n} h(X_i)}{Zn} \\
&= \frac{1}{n}\sum_{i=1}^{n} h(X_i) \\
&= \bar{h}_n. \square
\end{aligned}$$

C.3 Règle d'apprentissage des poids $\alpha(\cdot)$ (bigramme adaptatif)

Nous donnons ici les développements pour arriver au résultat (4.14). Nous voulons minimiser la divergence de Kullback-Leibler sur un échantillon $\mathcal{W}$ de points tirés de Q :

$$C = \sum_{v\in\mathcal{W}} t(v|h_t)\log\frac{t(v|h_t)}{Q(v|h_t)} \tag{C.8}$$

On a que $\alpha(v_{t-1}) = \sigma(a_i)$ où $v_t \in \mathcal{B}_i$. Pour minimiser la divergence C, nous effectuons une descente de gradient sur les a_i. Il ne reste qu'à évaluer

le gradient $\frac{\partial C}{\partial a_i}$.

$$\begin{aligned}
\frac{\partial C}{\partial a_i} &= \frac{\partial C}{\partial \alpha(v_{t-1})} \frac{\partial \alpha(v_{t-1})}{\partial a_i} \\
&= \frac{\partial C}{\partial \alpha(v_{t-1})} \alpha(v_{t-1})(1 - \alpha(v_{t-1})) \\
&= \alpha(v_{t-1})(1 - \alpha(v_{t-1})) \sum_{v \in \mathcal{W}} t(v|h_t) \frac{\partial(-\log Q(v|v_{t-1}))}{\partial \alpha(v_{t-1})} \\
&= \alpha(v_{t-1})(1 - \alpha(v_{t-1})) \sum_{v \in \mathcal{W}} \frac{t(v|h_t)}{Q(v|v_{t-1})} \frac{\partial(-Q(v|v_{t-1}))}{\partial \alpha(v_{t-1})} \\
&= \alpha(v_{t-1})(1 - \alpha(v_{t-1})) \sum_{v \in \mathcal{W}} \frac{t(v|h_t)}{Q(v|v_{t-1})} (Q_1(v) - Q_2(v|v_{t-1})).
\end{aligned} \tag{C.9}$$

C.4 Taille d'échantillon effective alternative

On souhaite que notre estimateur soit une mesure approximative du nombre d'échantillons qu'il aurait fallu tirer de la distribution cible P pour obtenir avec l'estimateur Monte Carlo classique (3.3) une variance égale à celle de l'estimateur échantillonnage pondéré biaisé (3.6) avec une distribution instrumentale Q.

Soit X une variable aléatoire issue de la distribution instrumentale Q et Y une variable aléatoire issue de la distribution cible P. Soit $X_1, \ldots, X_m$ un ensemble de variables aléatoires iid de Q et $Y_1, \ldots, Y_m$ un ensemble de variables aléatoires iid de P. Soit $W = \sum_{i=1}^{m} w(X_i)$. Soit h une fonction scalaire. Le résultat qui suit se généralise facilement au cas où h est une fonction vectorielle (e.g., un gradient) en remplaçant $h^2(x)$ par $\|h(x)\|^2$.

On a que

$$\begin{aligned}
\mathrm{Var}_P[\bar{h}_n] &= \mathrm{Var}_P\left[\frac{1}{n}\sum_{i=1}^{n} h(Y_i)\right] \\
&= \frac{1}{n}\left[\mathrm{E}_P[h^2(Y)] - \mathrm{E}_P^2(h(Y))\right] \\
&= \frac{1}{n}\left[\mathrm{E}_Q[h^2(Y)w(Y)] - \mathrm{E}_Q^2(h(Y)w(Y))\right] \\
&\approx \frac{1}{n}\left[\frac{1}{W}\sum_{i=1}^{m} h^2(X_i)w(X_i) - \left(\frac{1}{W}\sum_{i=1}^{m} h(X_i)w(X_i)\right)^2\right] \\
&= \frac{1}{nW^2}\left[W\sum_{i=1}^{m} h^2(X_i)w(X_i) - \left(\sum_{i=1}^{m} h(X_i)w(X_i)\right)^2\right]
\end{aligned} \tag{C.10}$$

et que

$$\begin{aligned}
\mathrm{Var}_Q[\tilde{I}_m] &= \mathrm{Var}_Q\left[\frac{1}{W}\sum_{i=1}^{m} h(X_i)w(X_i)\right] \\
&= \sum_{i=1}^{m} \mathrm{Var}_Q\left[\frac{1}{W}h(X_i)w(X_i)\right] \\
&\approx \mathrm{Var}_Q[h(X)]\sum_{i=1}^{m}\left(\frac{w(X_i)}{W}\right)^2 \\
&= \left[\mathrm{E}_Q[h^2(X)] - \mathrm{E}_Q^2[h(X)]\right]\sum_{i=1}^{m}\left(\frac{w(X_i)}{W}\right)^2 \\
&\approx \left[\frac{1}{m}\sum_{i=1}^{m} h^2(X_i) - \left(\frac{1}{m}\sum_{i=1}^{m} h(X_i)\right)^2\right]\sum_{i=1}^{m}\left(\frac{w(X_i)}{W}\right)^2 \\
&= \frac{1}{(mW)^2}\left[m\sum_{i=1}^{m} h^2(X_i) - \left(\sum_{i=1}^{m} h(X_i)\right)^2\right]\sum_{i=1}^{m} w^2(X_i).
\end{aligned} \tag{C.11}$$

D'où

$$\frac{\mathrm{Var}_P[\bar{h}_n]}{\mathrm{Var}_Q[\tilde{I}_m]} \approx \frac{m(WV - U^2)}{nS(mT - R^2)} \tag{C.12}$$

avec

$$R = \sum_{i=1}^{m} h(X_i, h) \qquad S = \sum_{i=1}^{m} w^2(X_i)$$
$$T = \sum_{i=1}^{m} h^2(X_i) \qquad U = \sum_{i=1}^{m} h(X_i)w(X_i)$$
$$V = \sum_{i=1}^{m} h^2(X_i)w(X_i) \qquad W = \sum_{i=1}^{m} w(X_i).$$

En posant $\widehat{ESS'}_m = m\frac{\mathrm{Var}_P[\bar{h}_n]}{\mathrm{Var}_Q[\bar{I}_m]}$ on obtient une forme finale pour l'estimateur de taille d'échantillon effective alternative :

$$\widehat{ESS'}_m = \frac{m^2(WV - U^2)}{S(mT - R^2)}. \tag{C.13}$$