Université de Montréal

TONGA - Un algorithme de gradient naturel pour les problèmes de grande taille

par
Pierre-Antoine Manzagol

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 Maître ès sciences (M.Sc.)
en informatique

Août, 2007

Université de Montréal
Faculté des études supérieures

Ce mémoire intitulé:

TONGA - Un algorithme de gradient naturel pour les problèmes de grande taille

présenté par:

Pierre-Antoine Manzagol

a été évalué par un jury composé des personnes suivantes:

Douglas Eck
président-rapporteur

Yoshua Bengio
directeur de recherche

Max Mignotte
membre du jury

Mémoire accepté le 20 novembre 2007

RÉSUMÉ

Les systèmes adaptatifs sont confrontés à des données qui évoluent rapidement en quantité et en complexité. Les avancées matérielles de l'informatique ne suffisent pas à compenser cet essor. Une mise à l'échelle des techniques d'apprentissage est nécessaire. D'une part, les modèles doivent gagner en capacité de représentation. De l'autre, les algorithmes d'apprentissage doivent devenir plus efficaces.

Nos travaux se situent dans ce contexte des problèmes de grande taille et portent sur l'amélioration des algorithmes d'apprentissage. Deux éléments de réponse sont déjà connus. Il s'agit des méthodes de second ordre et de l'approximation stochastique. Or, les méthodes de second ordre possèdent des complexités en calculs et en mémoire qui sont prohibitives dans le cadre des problèmes de grande taille. Également, il est notoirement difficile de concilier ces méthodes avec l'approximation stochastique. TONGA est un algorithme d'apprentissage conçu pour faire face à ces difficultés. Il s'agit d'une implantation stochastique et adaptée aux problèmes de grande taille d'une méthode de second ordre, le gradient naturel. Dans ce mémoire, nous examinons de près ce nouvel algorithme d'apprentissage en le comparant sur plusieurs problèmes au gradient stochastique, la technique d'optimisation communément utilisée dans le cadre des problèmes de grande taille. Nos expériences montrent que TONGA est au moins tout aussi efficace que le gradient stochastique, ce qui est un accomplissement en soit. Dans certains cas, TONGA offre une convergence nettement supérieure à celle du gradient stochastique.

Mots clés : Algorithme d'apprentissage, méthode de second ordre, gradient naturel, approximation stochastique

ABSTRACT

Machine learners are faced with data growing in quantity and complexity. Unfortunately, hardware advances are not enough to cope with this growth. Therefore, models must gain in expressiveness and learning algorithms in efficiency. In other words, we need techniques that scale.

Our work deals with improving learning algorithms in this respect. Indeed, for larger problems one still cannot resort to the more powerful classical second order techniques. They are too expensive to be practical and break down with the stochastic approximation. A well tuned standard gradient descent remains the method most commonly used.

TONGA is a new algorithm designed for the large scale setting. It is an online approximation of the natural gradient. In this thesis, we examine and evaluate this technique. Our results show TONGA to perform at least as well as standard gradient descent (which is already quite an accomplishment) and sometimes much better.

Keywords: Learning algorithm, second order method, natural gradient, online learning

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES APPENDICES

NOTATION

$\mathbb{R}$	L'ensemble des réels.
x	Un scalaire.
$\mathbf{x}$	Un vecteur colonne.
$\mathbf{X}$	Une matrice.
^{T}	La transposée. $\mathbf{x}^T$ est donc un vecteur ligne.
$\widetilde{\mathcal{L}}$, $\mathcal{L}$	$\mathcal{L}$ est un estimé empirique de $\widetilde{\mathcal{L}}$

REMERCIEMENTS

Cette thèse n'aurait été possible sans le soutien de mon directeur de recherches, Yoshua Bengio. J'aimerais particulièrement le remercier pour ses conseils et pour le formidable environnement de recherche qu'il a su créer au sein du Laboratoire d'Informatique des Systèmes Adaptifs (LISA). J'aimerais également remercier Nicolas Le Roux pour l'encadrement généreux qu'il m'a fourni dans son rôle de mentor, mais surtout en tant qu'ami. Olivier Delalleau a été d'une aide constante dans l'utilisation de la librairie PLearn, sur laquelle sont basées les expériences de ces travaux. Je souhaite également remercier de manière générale les membres du LISA, sans qui ces deux années n'auraient pas été les mêmes. Mes pensées vont à ma famille, tous les quatre, pour leurs encouragements de chaque instant.

Cette thèse a été réalisée avec le soutien financier du Conseil de Recherches en Sciences Naturelles et en Génie du Canada (CRSNG).

CHAPITRE 1

INTRODUCTION

L'apprentissage est une extraordinaire faculté biologique. Elle permet à l'être vivant d'incorporer de l'information sur son entourage, de façon à y être mieux adapté. Le domaine de l'Intelligence Artificielle a consacré de vastes efforts à tenter de reproduire cette faculté, donnant lieu à l'apprentissage machine. Cette branche est caractéristique des sciences modernes de par sa pluridisciplinarité : l'informatique fournit le support d'apprentissage, les statistiques de nombreux outils d'inférence et les sciences cognitives une bonne dose d'inspiration. Il en résulte une formidable approche aux problèmes d'automatisation pour lesquels on ne dispose pas de suffisamment de connaissances explicites dans les données. Il s'agit de découvrir automatiquement des régularitées présentes dans les données, régularités qui sont ensuites exploitées pour résoudre de la tâche.

Dans le but de bien motiver l'intérêt de l'approche et d'en présenter les notions de base, ce premier chapitre s'ouvre sur la description d'un problème type d'apprentissage machine (section 1.1). Par la suite, nous abordons le problème d'apprentissage sous l'angle de la minimisation d'un coût (section 1.2), puis passons brièvement en revue les algorithmes d'apprentissage à base de gradient (section 1.3). Nous insistons ensuite sur la nécessité d'une mise a l'échelle des techniques d'apprentissage machine (section 1.4). Enfin, nous présentons les contributions et la structure de ce mémoire (section 1.5).

1.1 Un petit exemple

La fraude par carte de crédit est un problème de taille : en 2005 elle représente au Canada plus de cinquante millions de dollars (RCMP, 2007). Étant donné le nombre considérable de transactions par carte de crédit, la tâche requiert l'automatisation. Or, bien qu'un humain sache facilement identifier une transaction suspecte, il lui

est parfois difficile d'en expliquer la raison. L'approche qui consiste a énumérer un ensemble de règles (le *knowledge engineering*) n'est donc pas à privilégier. De plus, toute règle non énoncée sera absente du système. On dispose cependant de banques de données de transactions dont on sait si elles sont frauduleuses. L'apprentissage machine est l'approche de choix pour ce type de problème.

L'objectif est donc l'élaboration d'un système adaptatif qui détermine si une transaction est frauduleuse ou pas. Ce système prend en entrée des transactions sous la forme de *caractéristiques* ou *traits* (par exemple le montant, l'heure et le lieu de la transaction, de même que les habitudes de consommation du client) et doit produire en sortie une décision (fraude ou pas). Chaque combinaison d'entrée-sortie constitue une instance du problème, un *exemple* dans le jargon. On dispose d'un certain nombre de ces exemples, qui constituent l'*ensemble d'apprentissage*. Un système adaptatif est la combinaison d'un *modèle* et d'une *technique d'apprentissage* qui contruit le modèle à partir des données. Le modèle peut avoir une forme spécifique déterminée par un faible nombre de paramètres et on parle alors de *modèle paramétrique*.

Une fois le système construit, il est nécessaire de l'évaluer. Pour ce faire, on calcule un *coût* qui est une mesure reflétant le nombre et la gravité des erreurs. Dans le cas de notre exemple, la gravité d'une erreur dépend fortement du montant de la transaction. Cependant, il est important de réaliser que l'on veut mesurer non pas le coût sur les données d'entraînement, mais plutôt le coût du système confronté à de nouveaux exemples. En d'autres termes, on désire évaluer la capacité du système à *généraliser*, et non pas à *mémoriser*. Idéalement le système devrait apprendre tout ce qui peut être appris du problème sur les données d'entraînement, sans pour autant en apprendre les spécifités propres à cet échantillon d'entraînement. C'est ainsi que le problème d'apprentissage met en jeu les phénomènes de sous-entraînement et de surentraînement. Pour illustrer le surentraînement, supposons que la seule transaction s'étant déroulée en Micronésie soit frauduleuse. Il ne faudrait pas pour autant en déduire que toute transaction s'y déroulant est frauduleuse. Il s'agit vraisemblablement d'un artifice lié à l'absence de plus de données d'entraînement

dans cette région du globe.

Cet exemple portant sur la fraude par carte de crédit se situe dans un cadre d'apprentissage particulier. Puisque l'on désire identifier une structure précise dans les données (reliée au concept de fraude), on dit qu'il s'agit d'un problème d'apprentissage *supervisé*. De plus, étant donné que la sortie prend des valeurs dans un ensemble fini (fraude ou pas), on parle également d'un problème de *classification*. Il existe divers cadres d'apprentissage, de même que plusieurs autres notions liées à l'apprentissage. Cependant, cette introduction suffit à la compréhension de la suite du développement. Nous référons le lecteur intéressé aux ouvrages de référence sur le sujet (Bishop, 1995, 2006; Duda et al., 2001).

1.2 Apprentissage et minimisation de coût

De nombreux problèmes d'apprentissage peuvent être posés sous la forme de la minimisation d'un coût $\widetilde{\mathcal{L}}$:

$$\widetilde{\mathcal{L}}(\theta) = \int L(\mathbf{x}, \theta)\widetilde{p}(\mathbf{x})d\mathbf{x} \tag{1.1}$$

où L est une fonction de coût dépendant des paramètres θ et des données $\mathbf{x}$ de distribution $\widetilde{p}(\mathbf{x})$. Ce qui fait de cette minimisation un problème d'*apprentissage* (mettant donc en jeu les notions de sous-entraînement et de surentraînement) et non d'*optimisation*, c'est le fait que $\widetilde{p}(\mathbf{x})$ n'est connue que via un certain nombre N d'échantillons $\mathbf{x}_i$. Ces échantillons définissent une distribution empirique $p(\mathbf{x})$, de même qu'un coût empirique (ou coût d'entraînement) :

$$\mathcal{L}(\theta) = \frac{1}{N}\sum_{i=1}^{N} L(\mathbf{x}_i, \theta) \tag{1.2}$$

C'est donc avec un (ou plusieurs) estimé du coût que l'on doit travailler.

Dans bien des cas il n'existe pas de solution analytique à la minimisation. De même, le coût est souvent une fonction non-linéaire et non convexe de ses paramètres. Les méthodes d'optimisation basées sur le gradient constituent alors un

choix judicieux. Ces méthodes procèdent à l'ajustement itératif des paramètres θ (que l'on doit alors indexer par l'itération, θ_t). Elles ont l'avantage d'exploiter la structure du coût (par rapport aux algorithmes évolutionnaires, par exemple), mais elles sont sensibles aux minima locaux.

1.3 Apprentissage basé sur le gradient

Cette section et la section suivante présentent brièvement les principales techniques d'apprentissage à base de gradient. Pour un traitement complet, nous référons le lecteur à Nocedal et Wright (2006).

1.3.1 Descente de gradient de premier ordre

La plus simple des méthodes à base de gradient est la *descente de gradient de premier ordre*, qui consiste à progressivement mettre à jour les paramètres θ suivant :

$$\theta_{t+1} = \theta_t - \phi_t \frac{\partial \mathcal{L}(\theta_t)}{\partial \theta} \tag{1.3}$$

$$= \theta_t - \phi_t \frac{1}{N} \sum_{i=1}^{N} \frac{\partial L(\mathbf{x}_i, \theta_t)}{\partial \theta} \tag{1.4}$$

où ϕ est le *facteur d'apprentissage* (*learning rate*), un scalaire qui régule l'apprentissage. Cette méthode a l'avantage d'être facile à implanter et de posséder des garanties de convergence sous réserve de certaines conditions sur ϕ_t. Elle souffre par contre d'une faible convergence dans le cas de problèmes dits mal conditionnés. Ces problèmes se manifestent par des surfaces d'erreur ressemblant à de longs ravins : la pente est forte dans certaines directions mais faible dans d'autres. Dans ces cas, le gradient ne pointe pas vers le minimum. Il pointe plutôt dans la direction du meilleur gain immédiat. Si le pas effectué dans cette direction est trop grand, l'optimisation va osciller entre les deux cotés du ravin, et réaliser peu de progrès dans les autres directions. De nombreuses heuristiques ont été développées pour tenter de limiter ces problèmes, avec des succès partiels et variables.

1.3.2 Méthode de Newton

La méthode de Newton est dite de *second ordre* car elle est basée sur une approximation quadratique locale de la fonction de coût. Elle peut donc tirer parti d'informations sur la courbure. Elle se formule ainsi :

$$\theta_{t+1} = \theta_t - \mathcal{H}^{-1} \frac{\partial \mathcal{L}(\theta_t)}{\partial \theta} \tag{1.5}$$

où $\mathcal{H}$ est la matrice hessienne, c'est-à-dire $\mathcal{H} = \frac{\partial^2 \mathcal{L}(\theta_t)}{\partial \theta^2}$. Dans le cas d'un coût quadratique, la méthode de Newton atteint le minimum en une étape. Dans les autres cas, elle peut grandement accélérer la convergence quand elle est positive définie (ce qui n'est pas toujours le cas). Essentiellement, il faut comprendre que l'utilisation de la hessienne revient à appliquer un facteur d'apprentissage différent suivant les directions de l'espace des paramètres. Ces facteurs d'apprentissage sont les inverses des valeurs propres de la hessienne et sont appliqués selon les directions des vecteurs propres correspondant. En effet, on sait qu'il s'agit là du choix optimal de facteur d'apprentissage dans le cas d'un coût quadratique. On sait également que pour éviter la divergence[1] pour un coût quadratique, le facteur d'apprentissage doit être inférieur à deux fois cette valeur. Dans le cas des problèmes mal conditionnés, cela a pour effet de redresser le gradient de façon à ce qu'il soit orienté vers le minimum.

1.3.3 Gradient conjugué

Le gradient conjugué est également basé sur l'idée d'utiliser un facteur d'apprentissage variable suivant la direction de l'espace des paramètres considérée, mais cette idée est déclinée différemment. Il s'agit d'optimiser complètement suivant une première direction, ce qui ce fait à l'aide d'une minimisation en ligne, puis de ne plus toucher aux paramètres suivant cette direction (d'après une approximation

[1] La divergence est un comportement indésirable qui se produit lorsque l'on a un facteur d'apprentissage trop grand. La correction appliquée a alors pour effet de déteriorer le coût.

quadratique local). La technique est donc censée requérir un nombre d'étapes limité par le nombre de paramètres dans le cas d'un coût quadratique.

1.3.4 Méthodes quasi-Newton

Malheureusement, la hessienne n'est pas forcément positive définie, et la méthode de Newton peut faire des grands sauts incontrôlés. De plus, le coût du calcul et de l'inversion de la hessienne est très élevé : quadratique en mémoire et cubique en calculs. Il existe donc diverses approximations positives définies (dites quasi-Newton) de la méthode de Newton, que ce soit Gauss-Newton, Levenberg-Marquadt, BFGS ou encore le gradient naturel. Ces approximations ont toutes la forme suivante :

$$\theta_{t+1} = \theta_t - \Phi_t \frac{\partial \mathcal{L}(\theta_t)}{\partial \theta} \tag{1.6}$$

où Φ est maintenant une matrice dont le rôle effectif est d'affecter un facteur d'apprentissage variable suivant la direction du gradient considérée.

1.3.5 Approximation diagonale

Une approximation supplémentaire consiste à supposer que Φ est une matrice diagonale. On peut alors la représenter à faible coût. Ainsi, avec p le nombre de paramètres, l'approximation diagonale permet de passer à un coût de l'ordre de $\mathcal{O}(p)$ pour la mémorisation de Φ et sa multiplication avec le gradient. Cependant, l'approximation diagonale consiste à supposer que les vecteurs propres sont alignés sur les axes. Ce n'est en pratique pas le cas, et l'approximation est donc de qualité assez faible.

1.3.6 Approximation stochastique

Dans le cas où l'on dispose de nombreux exemples d'apprentissage, le calcul du coût d'entraînement 1.2, de son gradient et de la matrice Φ peut devenir considérable en raison de la somme sur les exemples. L'*approximation stochastique* consiste à faire chaque évaluation du coût, ou de toute autre mesure y étant reliée, sur la

base d'un ou de plusieurs exemples pris au hasard ou tout simplement en parcourant séquentiellement l'ensemble d'entraînement. En d'autres mots, on tente de faire plusieurs mises à jour possiblement moins précises, mais aussi moins coûteuses en calculs, plutôt qu'une seule mise à jour précise et coûteuse comme dans l'approche traditionnelle dite *par lot*. L'approximation stochastique est également sensée lorsque l'on fait face à un optimum mobile (données non stationnaires), lorsque l'on n'a pas accès à toutes les données d'un coup ou encore lorsqu'elles prennent la forme d'un flux. De plus, la stochasticité mise en jeu permet parfois d'échapper à des minima locaux de faible qualité. Cette approximation n'est cependant pas parfaite. Le bruit mis en jeu rend imprécises ou inefficaces les méthodes de second ordre, les minimisations de ligne (*line search*) et les calculs de directions conjuguées.

1.4 Nécessité d'une mise à l'échelle des techniques d'apprentissage

Un algorithme d'apprentissage peut être évalué en fonction de deux éléments. Le premier est l'erreur de généralisation à laquelle il mène, c'est-à-dire sa capacité à minimiser $\widetilde{\mathcal{L}}(\theta_t) - \widetilde{\mathcal{L}}(\theta_{optimal})$. Le second est sa vitesse de convergence, mesurée par l'évolution de $\theta_t - \theta_{optimal}$. Cette vitesse de convergence a longtemps été perçue comme une fonction du nombre de mises à jour de paramètres. Il est cependant plus judicieux de la considérer comme fonction du temps de calcul, car une technique qui converge plus lentement en termes de nombre de mises-à-jour peut tout de même converger plus vite en termes de temps (si chaque mise à jour est très rapide). C'est justement pour cette raison que la descente de gradient de premier ordre est utilisée en pratique dans le cadre des problèmes de grande taille.

En 1998, un jeu de recommandations quant au choix d'une technique d'apprentissage pour les réseaux de neurones était le suivant (LeCun et al., 1998) :

- les méthodes classiques de second ordre sont presque toujours impraticables
- si l'ensemble d'entraînement est grand et redondant, utiliser un gradient stochastique bien ajusté ou une variante stochastique diagonale de Levenberg-

Marquardt

– si l'ensemble d'apprentissage est petit[2], utiliser le gradient conjugué.

Durant les dix dernières années, le matériel informatique a considérablement évolué. Pourtant, les problèmes ont eux aussi évolué sous l'effet de la multiplication des capteurs et des applications gérées par des processus automatisés. Les problèmes de grande taille deviennent communs. Un problème de grande taille est à l'heure actuelle un problème pour lequel le nombre d'exemples et de paramètres est de l'ordre du million. Cela signifie que simplement mémoriser Φ de façon conventionnelle nécessite de l'ordre du téraoctet de mémoire. Il en résulte une importance accrue d'améliorer les techniques d'apprentissage. Cette amélioration doit passer par la conciliation des méthodes de second ordre avec l'approximation stochastique, de même que par l'adaptation de ces méthodes au cadre des problèmes de grande taille.

Notre compréhension et nos connaissances sur les algorithmes d'apprentissage ont également évoluées durant ce laps de temps. Au niveau théorique, Bottou et LeCun (2004) montrent que les algorithmes stochastiques pourtant reputés de convergence plus lente, ne le sont que lorsque l'on considère l'erreur d'apprentissage. En termes de convergence au point de meilleure généralisation, un algorithme stochastique bien conçu de la forme 1.6 est donc supérieur puisqu'il converge tout aussi vite en termes d'exemples, mais nécessite généralement moins de ressources par exemple. Il peut donc traiter plus d'exemples, en particulier les quelques exemples très informatifs. Au niveau pratique, des efforts considérables ont été consacrés à l'adaptation des méthodes classiques à l'approximation stochastique. Schraudolph (1999) a introduit le SMD, une nouvelle technique de la forme de 1.6 et qui semble prometteuse. Schraudolph (2002) traite de techniques pour accélérer le calcul matrice-vecteur dans le cas de nombreuses matrices Φ. Des versions stochastiques du gradient conjugué (Schraudolph et Graepel, 2003) et de BFGS (Schraudolph et al., 2007) ont été avancées. Il demeure cependant beaucoup de place à

[2]Également s'il s'agit d'une tâche de régression.

l'amélioration.

1.5 Contributions et structure du mémoire

Ce mémoire s'inscrit dans le cadre des travaux visant l'amélioration des techniques d'apprentissage. Il s'agit notamment de les rendre plus efficace dans le cadre des problèmes de grande taille, c'est-à-dire pour lesquels on dispose de nombreux exemples et pour lesquels les modèles possèdent de nombreux paramètres. Dans cette optique, nous étudions TONGA, un nouvel algorithme qui consiste en une approximation du gradient naturel conçue pour le cadre des problèmes de grande taille et de l'approximation stochastique. Nos expériences montrent que TONGA performe aussi bien en temps que le gradient stochastique, la méthode la plus communément utilisée dans ce cadre d'apprentissage. C'est là un accomplissement de taille. Dans certains cas, TONGA offre une performance supérieure.

Le chapitre 2 présente le gradient naturel, des arguments en faveur de son utilisation et les difficultées liées à son utilisation dans le cadre des problèmes de grande taille. En particulier, la section 2.6 traite d'efforts antérieurs pour obtenir une version moins coûteuse de l'algorithme. Le chapitre 3 présente ensuite l'intuition derrière la conception de TONGA, avant de décrire l'algorithme en détails. Le chapitre 4 fournit ensuite les détails et les résultats des expériences réalisées pour évaluer TONGA. Ces résultats sont ensuite discutés et des extensions envisagées au chapitre 5.

CHAPITRE 2

LE GRADIENT NATUREL

Le gradient naturel est une méthode de descente de gradient possédant la forme générale donnée par l'équation 1.6 avec pour choix particulier de la matrice Φ l'inverse de la matrice de covariance des gradients. C'est une matrice qui a l'avantage d'être positive définie, et plus facile à calculer que la hessienne. L'équation de mise à jour des paramètres du modèle devient alors

$$\theta_{t+1} = \theta_t - \mathbf{C}^{-1} \frac{\partial \mathcal{L}(\theta_t)}{\partial \theta} \tag{2.1}$$

où $\mathbf{C} = E\left[\frac{\partial \mathcal{L}(\theta)}{\partial \theta}^T \frac{\partial \mathcal{L}(\theta)}{\partial \theta}\right]$, l'espérance étant prise sur les données d'apprentissage. Dans ce chapitre, nous présentons d'abord trois perspectives sur ce choix particulier (sections 2.1, 2.2 et 2.3). Nous poursuivons en fournissant quelques intuitions sur le fonctionnement du gradient naturel (section 2.5). Nous abordons ensuite les difficultés liées à l'utilisation du gradient naturel dans le contexte des problèmes de grande taille et nous passons en revue les différentes techniques déjà mises de l'avant pour tenter de réduire le coût computationnel de la technique (section 2.6).

2.1 Le vrai gradient dans un espace de Riemann

Amari (1998) a dérivé le gradient naturel dans un contexte de géométrie de l'information pour un coût de type log-vraissemblance négative (NLL)[1]. En particulier, il a montré que l'espace des paramètres est un espace de Riemann de tenseur $\mathbf{C}$, la covariance des gradients (qui étant donné le coût de type NLL porte alors le nom de matrice d'information de Fisher). Cet espace de Riemann correspond en fait à l'espace des fonctions représentées par les paramètres. Dans cet espace, le

[1] Dans un cadre supervisé, avec $\mathbf{x}$ l'entrée d'un modèle probabiliste (c'est-à-dire qui fournit une distribution de probabilité sur sa sortie) et t la sortie désirée, la NLL est définie comme étant la fonction de coût $-\log p(t|x, \theta)$.

gradient naturel est introduit comme étant la direction de plus forte descente de l'erreur. Si l'espace des paramètres est euclidien (c'est-à-dire $\mathbf{C} = \mathbf{I}$), le gradient naturel correspond au gradient usuel.

2.2 Une approximation de la méthode de Newton

Le gradient naturel peut également être vu comme une approximation de la méthode de Newton (équation 1.5). En d'autres termes, on peut voir $\mathbf{C}$ comme étant une approximation de la matrice hessienne. Pour ce faire, considérons un système basé sur deux transformations. La première, $\mathcal{T}_{modele} : \mathbb{R}^i \rightarrow \mathbb{R}^o$, permet de passer de l'entrée $\mathbf{x}$ et des poids θ de dimensions i à la sortie $\mathbf{y}$ de dimension o. La seconde, $\mathcal{T}_{cout} : \mathbb{R}^o \rightarrow \mathbb{R}$, permet de passer de la sortie $\mathbf{y}$ au coût $\mathcal{L}(\theta)$ que l'on suppose être scalaire. $\mathcal{T}^i_{modele}$ représente la sous-transformation produisant chaque sortie, indexée par i. Intéressons-nous à la dérivée du coût $\mathcal{L}(\theta)$ par rapport aux paramètres θ, en considérant les dérivées partielles par rapport à $\mathbf{y}$, la sortie du système :

$$\mathcal{H} = \frac{\partial^2 \mathcal{L}(\theta)}{\partial \theta^2} \tag{2.2}$$

$$= \frac{\partial}{\partial \theta} \left(\frac{\partial \mathcal{L}(\theta)}{\partial y} \frac{\partial y}{\partial \theta} \right) \tag{2.3}$$

$$= \frac{\partial y}{\partial \theta}^T \frac{\partial^2 \mathcal{L}(\theta)}{\partial y^2} \frac{\partial y}{\partial \theta} + \sum_{i=1}^{o} \frac{\partial \mathcal{L}(\theta)}{\partial y_i} \frac{\partial^2 y_i}{\partial \theta^2} \tag{2.4}$$

$$= \frac{\partial y}{\partial \theta}^T \mathcal{H}_{\mathcal{T}_{cout}} \frac{\partial y}{\partial \theta} + \sum_{i=1}^{o} \frac{\partial \mathcal{L}(\theta)}{\partial y_i} \mathcal{H}_{\mathcal{T}^i_{modele}} \tag{2.5}$$

Considérons d'abord le second terme. Dans le cas d'un système où $\mathcal{T}_{modele}$ est une fonction linéaire de ses paramètres, ce terme est nul. En pratique, il s'agit d'une approximation courante. On peut également motiver cette approximation de la façon suivante dans le cas d'un coût NLL, une fois l'apprentissage complété. Considérons un système adaptatif probabiliste ayant pour sortie t, c'est à dire

modélisant $p(t|\theta, \mathbf{x})$. On a alors :

$$\int p(t|\theta, \mathbf{x})dt = 1$$

En dérivant par rapport à θ, on aboutit à :

$$\int \frac{\partial p(t|\theta, \mathbf{x})}{\partial \theta} dt = 0$$

On utilise alors l'astuce du logarithme pour obtenir :

$$\int p(t|\theta, \mathbf{x}) \frac{\partial \log p(t|\theta, \mathbf{x})}{\partial \theta} dt = 0$$

Ainsi, lorsque les données sont générées par le modèle, l'espérance du gradient de la NLL est nulle. Dérivons à nouveau par rapport à θ, en utilisant à nouveau l'astuce du logarithme :

$$\int \left(p(t|\theta, \mathbf{x}) \frac{\partial \log p(t|\theta, \mathbf{x})}{\partial \theta} \frac{\partial \log p(t|\theta, \mathbf{x})}{\partial \theta^T} + p(t|\theta, \mathbf{x}) \frac{\partial^2 \log p(t|\theta, \mathbf{x})}{\partial \theta^2} \right) dt = 0$$

En réordonnant et en posant $L = -\log p(t|\theta, \mathbf{x})$ ceci donne :

$$E\left[\frac{\partial^2 L}{\partial \theta^2}\right] = E\left[\frac{\partial L}{\partial \theta} \frac{\partial L}{\partial \theta^T}\right]$$

où l'espérance est prise selon la distribution du modèle. On voit donc que dans le cas d'un coût NLL et lorsque les données sont générées par le modèle, l'espérance de la covariance des gradients est égale à celle de la matrice hessienne. En pratique, on devrait obtenir une pseudo-égalité sous la distribution empirique une fois le modèle entraîné. Évidemment, ceci ne nous informe pas sur le comportement durant l'entraînement.

Considérons maintenant le premier terme de l'équation 2.5. Ce terme ne diffère de la matrice de covariance des gradients que par la présence de la hessienne $\mathcal{H}_{\mathcal{T}_{cout}}$ du coût par rapport aux sorties. Dans le cas d'un coût quadratique, cette hessienne

vaut l'identité. Dans le cas un peu plus général d'un système où la sortie est scalaire, la hessienne est également représentée par un scalaire, et sa présence n'affecte donc pas la forme générale de la matrice de covariance. Dans le reste des cas, on peut approximer cette hessienne par l'identité, aboutissant ainsi encore à la matrice de covariance des gradients.

Cette approximation de la hessienne est connue sous le nom d'approximation linéarisée, d'approximation du produit externe ou encore d'approximation du carré du jacobien. Lorsque le coût employé est la NLL, on parle de gradient naturel. Dans le cas de la MSE, on dit qu'il s'agit de l'approximation Gauss-Newton, ou encore Levenberg-Marquadt en version regularisée.

2.3 Minimiser le *surapprentissage*

Dans tout problème d'apprentissage, l'objectif est la minimisation du coût de généralisation. Or, on a vu qu'étant donné que la distribution des données n'est connue qu'à travers un certain nombre d'échantillons, on ne peut évaluer directement ce coût. Le danger réside dans le phénomène de *surapprentissage*, qui consiste à apprendre des particularités des données d'entraînement qui ne reflètent pas les données dans leur ensemble.

Au regard de l'optimisation, une technique usuelle pour combattre ce problème est l'arrêt préventif (*early stopping*). Il s'agit de mettre de côté une partie des exemples d'entraînement pour constituer un ensemble de *validation* qui fournit un estimé indépendant de l'erreur de généralisation. Lorsque l'erreur de validation commence à augmenter, il faut arrêter l'apprentissage.

Le Roux et al. (2008) proposent une perspective différente. Il s'agit de combattre le surentraînement tout au long de l'apprentisage. En effet, le surentraînement se produit tout au long de l'optimisation, mais ne se met à détériorer l'erreur de généralisation qu'à partir d'un certain point, déterminé par l'arrêt préventif. En termes de gradients, le gradient g du coût d'entraînement n'est jamais tout à fait colinéaire avec $\widetilde{g}$ le gradient du coût de généralisation, et au bout d'un moment

leur produit scalaire devient même négatif. L'article montre que le le gradient naturel représente soit la direction qui minimise la probabilité d'augmenter l'erreur de généralisation, soit la direction de plus forte descente de cette même erreur, suivant les hypothèses. Nous reprenons ici ces arguments en se plaçant dans les deux philosophies d'apprentissage, à savoir bayesienne et fréquentiste.

Le point de départ de cette argumentation est de faire l'approximation que les gradients sur chaque exemple sont des tirages indépendants et identiquement distribués de la loi des gradients de l'erreur de généralisation. Ceci nous permet d'appliquer le théorème central limite :

$$g \sim \mathcal{N}\left(\widetilde{g}, \frac{\widetilde{C}}{n}\right) \tag{2.6}$$

où $\widetilde{C}$ est la vraie matrice de covariance de $\frac{\partial L(x,\theta)}{\partial \theta}$.

2.3.1 Perspective bayesienne

Dans la perspective bayesienne, le gradient $\widetilde{g}$ de l'erreur de généralisation est une variable aléatoire et on désire évaluer sa distribution à posteriori étant donné les échantillons g_i. Ceci nous permet ensuite d'obtenir une distribution sur le produit scalaire $v^T\widetilde{g}$ entre la direction hypothétique de descente v et $\widetilde{g}$, quelle que soit v.

Supposons que la distribution à priori de $\widetilde{g}$ est une gaussienne centrée en 0 et de variance $\sigma^2 I$:

$$\widetilde{g} \sim \mathcal{N}\left(0, \sigma^2 I\right) \tag{2.7}$$

Par le théorème de Bayes et en se basant sur l'équation 2.6, on peut évaluer la distribution à posteriori de $\widetilde{g}$ étant donné les g_i (en supposant que la seule information fournie par les g_i sur $\widetilde{g}$ est donnée par la moyenne empirique g et la covariance empirique C). Pour ce faire, on utilise le fait que la multiplication de deux gaussiennes donne lieu (après normalisation) à une gaussienne de moyenne et

variance connues (Petersen et Pedersen, 2006), ce qui donne :

$$\widetilde{g}|g,\widetilde{C} \sim \mathcal{N}\left(\left(n\widetilde{C}^{-1}+\frac{I}{\sigma^2}\right)^{-1}\left(\frac{\widetilde{C}}{n}\right)^{-1}g, \left(\frac{I}{\sigma^2}+n\widetilde{C}^{-1}\right)^{-1}\right) \tag{2.8}$$

c'est-à-dire :

$$\widetilde{g}|g,\widetilde{C} \sim \mathcal{N}\left(\left(I+\frac{\widetilde{C}}{n\sigma^2}\right)^{-1}g, \left(\frac{I}{\sigma^2}+n\widetilde{C}^{-1}\right)^{-1}\right) \tag{2.9}$$

Pour la suite du développement, posons $\widetilde{C}_\sigma = I + \frac{\widetilde{C}}{n\sigma^2}$. La distribution sur $v^T\widetilde{g}$ est donc donnée par (il s'agit d'une combinaison linéaire, voir Petersen et Pedersen (2006)) :

$$v^T\widetilde{g}|g,\widetilde{C} \sim \mathcal{N}\left(v^T\widetilde{C}_\sigma^{-1}g, \frac{v^T\widetilde{C}_\sigma^{-1}\widetilde{C}v}{n}\right) \tag{2.10}$$

De là, diverses stratégies sont possibles. En particulier :

- choisir la direction v de façon à maximiser la diminution immédiate de l'erreur de généralisation, c'est-à-dire minimiser l'espérance de $v^T\widetilde{g}$. La directions v à prendre est alors

  $$v \propto -\widetilde{C}_\sigma^{-1}g. \tag{2.11}$$

  En retournant à la définition de $\widetilde{C}_\sigma$, on voit que si $\sigma < \infty$ alors il s'agit du gradient naturel régularisé. Si au contraire $\sigma = \infty$, alors $\widetilde{C}_\sigma = I$ et il s'agit de la descente de gradient standard.
- choisir la direction minimisant la probabilité d'augmenter l'erreur de généralisation, c'est-à-dire minimisant la probabilité que $v^T\widetilde{g}$ soit positif. Dans le cas d'une distribution gaussienne, cela revient à minimiser le quotient de la moyenne sur l'écart-type :

  $$\text{argmin}_v \frac{v^T\widetilde{C}_\sigma^{-1}g}{\sqrt{v^T\widetilde{C}_\sigma^{-1}\widetilde{C}v}}$$

où le n au dénominateur de la variance a été laissé de côté puisqu'il n'affecte pas la minimisation. En mettant cette quantité au carré et en dérivant par rapport à v, on obtient : $2\widetilde{C}_\sigma^{-1}g(v^T\widetilde{C}_\sigma^{-1}g)(v^T\widetilde{C}_\sigma^{-1}\widetilde{C}v) - 2\widetilde{C}_\sigma^{-1}\widetilde{C}v(v^T\widetilde{C}_\sigma^{-1}g)^2$ au numérateur. Oubliant les facteurs scalaires, on voit que le premier terme est dans la direction de $\widetilde{C}_\sigma^{-1}g$ et le second dans celle de $\widetilde{C}_\sigma^{-1}\widetilde{C}v$. Par conséquent, pour annuler la dérivée il faut avoir $g \propto \widetilde{C}v$, ou en termes de v (étant donné que les matrices de covariance $\widetilde{C}$ et $\widetilde{C}_\sigma$ sont inversibles),

$$v \propto -\widetilde{C}^{-1}g. \tag{2.12}$$

Cette direction correspond au gradient naturel. On remarque de plus qu'il n'y a pas de dépendance en σ utilisé dans la distribution a priori de $\widetilde{g}$.

2.3.2 Perspective fréquentiste

Dans la perspective fréquentiste $\widetilde{g}$ est perçu comme étant une quantité fixe indéterminée. Nous envisageons les directions de descente qui sont une transformation linéaire du gradient, c'est-à-dire $v = M^Tg$. Cette forme est générale et englobe par exemple la direction donnée par la méthode de Newton.

Étant donné que $g \sim \mathcal{N}\left(\widetilde{g}, \frac{\widetilde{C}}{n}\right)$, la quantité d'intérêt $v^T\widetilde{g}$ est donnée par

$$v^T\widetilde{g} = g^TM\widetilde{g} \sim \mathcal{N}\left(\widetilde{g}^TM\widetilde{g}, \frac{\widetilde{g}^TM^T\widetilde{C}M\widetilde{g}}{n}\right) \tag{2.13}$$

Comme dans le cas bayesien, nous cherchons à minimiser la probabilité que $v^T\widetilde{g}$ soit positif, mais cette fois en fonction de la matrice M. La matrice M^* répondant à ce critère est donnée par

$$M^* = \text{argmin}_M \frac{\widetilde{g}^TM\widetilde{g}}{\widetilde{g}^TM^TCM\widetilde{g}} \tag{2.14}$$

Le numérateur de la dérivée de cette quantité est $\widetilde{g}\widetilde{g}^TM^T\widetilde{C}M\widetilde{g}\widetilde{g}^T - 2\widetilde{C}M\widetilde{g}\widetilde{g}^TM\widetilde{g}\widetilde{g}^T$. Le premier terme est dans la direction de $\widetilde{g}$ et le second dans celle de $\widetilde{C}M\widetilde{g}$. Pour que

la dérivée s'annule peu importe la valeur de $\widetilde{g}$, il faut donc que $M \propto \widetilde{C}^{-1}$. On obtient donc le même résultat que dans le cadre bayesien : le gradient naturel représente la direction minimisant la probabilité d'augmenter l'erreur de généralisation.

2.4 Covariance et second moment

Bien que le gradient naturel soit basé sur l'emploi de la covariance, TONGA utilise plutôt la matrice des seconds moments (c'est-à-dire la covariance non-centrée, qui mesure l'écart à l'origine, plutôt qu'à la moyenne). Ce choix est motivé par la stabilité numérique, puisque la matrice des seconds moments a généralement des valeurs propres plus élevées que la matrice de covariance. Mentionnons d'ailleurs que c'est là une pratique courante (Amari et al., 2000).

Dans le cadre de l'apprentissage par lot, ce changement n'affecte pas la direction du gradient naturel. En effet, posant g la moyenne des gradients, C la matrice de covariance, $\bar{C}$ la matrice des seconds moments, $v = C^{-1}g$ le gradient naturel et $\bar{v} = \bar{C}^{-1}g$ on a :

$$\bar{C}v = (C + gg^T)v = g + gg^Tv = g(1 + g^Tv) \tag{2.15}$$

En multipliant par $\bar{C}^{-1}$, on obtient :

$$v = \bar{C}^{-1}g(1 + g^Tv) = (1 + g^Tv)\bar{v} \tag{2.16}$$

La direction demeure donc inchangée, et le redimensionnement de la norme est de $\frac{1}{1+\|g\|\|v\|\cos(g,v)}$

2.5 Intuition sur la technique

On peut se demander comment interpréter l'effet de la matrice des seconds moments dans l'équation 2.1. Concrètement, cela signifie un facteur d'apprentissage basé sur le second moment dans une direction donnée. Ainsi, une direction de fort second moment se voit attribuer un facteur d'apprentissage faible et inversement.

Ceci devrait donc diminuer le facteur d'apprentissage dans les directions de fort gradient, et l'augmenter dans les directions de faible gradient.

En pratique, le gradient naturel est reconnu comme traitant efficacement les plateaux présents dans la fonction de coût en découplant les paramètres. En effet, ces plateaux correspondent souvent à des symétries dans l'espace des paramètres. Ce sont des zones où le gradient est faible et où la progression de la diminution de l'erreur est faible avec un gradient ordinaire de premier ordre.

2.6 Tentatives antérieures

Malheureusement le gradient naturel possède un coût computationnel fort élevé qui le rend inapplicable dans le contexte des problèmes de grande taille. En effet, avec p le nombre de paramètres du modèle, une implantation classique a une consommation de ressources de l'ordre de $\mathcal{O}(p^3)$ en calculs et $\mathcal{O}(p^2)$ en mémoire. La complexité en calculs est dominée par l'inversion de la matrice de covariance. Au niveau de la mémoire, cette consommation constitue un vrai problème, car elle peut facilement dépasser la mémoire disponible sur un système. Différentes approches ont été avancées pour tenter de réduire cette complexité, et nous les passons maitenant en revue.

Une première façon de réduire la complexité consiste simplement en l'approximation diagonale de la covariance. Cela ramène la complexité en calculs et en mémoire à un maigre $\mathcal{O}(p)$. En pratique cependant, les paramètres sont corrélés et l'approximation diagonale se révèle assez pauvre.

Heskes (2000) envisage donc dans le cadre d'un réseau de neurones une approximation bloc diagonale de la matrice de covariance où un bloc est constitué des poids d'une même couche. Il s'agit donc d'ignorer les éléments croisés de la matrice qui représentent les interactions inter couche (un changement dans un poids d'une couche de réseau de neurones peut être compensé par un changement dans le poids d'une autre couche). L'idée est que ce sont là des phénomènes très non-linéaires, et que l'approximation (de la matrice hessienne) apportée par la matrice

de covariance n'est probablement pas intéressante à la base. La technique néglige également dans la rétropropagation les corrélations entre les dérivées et les entrées d'une couche. Il est ainsi possible d'obtenir une complexité réduite à la fois en calculs et en mémoire, à savoir de l'ordre du carré du nombre d'unités cachées de la couche la plus large. Il faut cependant réaliser qu'un réseau de neurones est bien souvent constitué d'une seule couche très large, et donc que le gain n'est pas forcément si intéressant en pratique.

Yang et Amari (1997) avancent également une approche conçue pour les réseaux de neurones. En supposant une distribution gaussienne sur les entrées, on est en mesure de calculer et d'inverser analytiquement la matrice de covariance en $\mathcal{O}(n^2)$ où n est le nombre d'entrées du réseau. Le gain est réalisé grâce à l'identification d'une structure dans la matrice[2] qui permet de réutiliser des calculs. Cependant, bien que l'approche permette une représentation compacte de la matrice de covariance, le requis en mémoire opérationnelle demeure $\mathcal{O}(p^2)$. De plus, la technique est complexe et basée sur deux suppositions : une distribution gaussienne sur les entrées et un nombre d'unités cachées largement inférieur au nombre d'entrées. Notons qu'en pratique, on peut tenter de changer la représentation des entrées de façon à ce que leur distribution soit proche d'une gaussienne.

Amari et al. (2000) offrent une approche plus générale basée sur l'emploi de la formule de Sherman-Morrison :

$$(\mathbf{C} + \mathbf{g}\mathbf{g}^T)^{-1} = \mathbf{C}^{-1} - \frac{\mathbf{C}^{-1}\mathbf{g}\mathbf{g}^T\mathbf{C}^{-1}}{1 + \mathbf{g}^T\mathbf{C}^{-1}\mathbf{g}} \tag{2.17}$$

Il s'agit donc de maintenir un estimé empirique de l'inverse de la covariance, estimé qui peut être mis à jour en $O(p^2)$. Mais là encore, le requis en mémoire demeure $O(p^2)$, ce qui est impraticable.

Il n'est cependant pas nécessaire de calculer explicitement l'inverse de la matrice

[2]On identifie des blocs qui correspondent à des groupements de paramètres dans un réseau de neurones de n entrées possédant une couche cachée de m unités et une seule unité de sortie : m groupes de n paramètres correspondant aux poids des neurones cachés, et 2 groupes de m paramètres correspondant aux biais des neurones cachés et aux poids du seul neurone de sortie.

de covariance, étant donné que l'on désire seulement connaître son produit avec le gradient. Yang et Amari (1998) proposent deux façons d'en tirer parti. La première utilise une méthode itérative (le gradient conjugué) pour resoudre $Cv = g$. La seconde tire parti des travaux de Yang et Amari (1997) et obtient ainsi une approche de complexité réduite mais qui souffre des mêmes faiblesses.

Schraudolph (2002) propose également une approche itérative basée sur la minimisation d'un coût quelque peu différent. Cette approche est utilisée dans le cadre minibatch, où Cv peut-être calculé à faible coût via deux multiplications matrice-vecteur. Cependant, la covariance des gradients n'est estimée que sur un faible nombre de points et se solde par un estimé instable.

CHAPITRE 3

TONGA

Les techniques existantes ne fournissent pas de solution au problème de l'application du gradient naturel aux problèmes de grande taille. Elles ne parviennent pas à surmonter les deux principales embûches. La première est la qualité de l'approximation de la matrice de covariance. Bien que la matrice de covariance évolue de façon lisse au cours de l'apprentissage, elle nécessite plusieurs gradients pour être correctement estimée. La seconde embûche est celle de la complexité computationnelle. On ne peut se permettre des complexités de l'ordre de $O(N^2)$ en calculs et en mémoire lorsque N est de l'ordre de 10^5.

Ce chapitre présente TONGA, un nouvel algorithme d'apprentissage conçu dans le but de surmonter ces deux embûches à la fois. C'est donc un algorithme qui est conçu avec une complexité applicable aux problèmes de grande taille, et pensé pour le cadre de l'approximation stochastique. Nous présentons d'abord l'intuition derrière sa conception (section 3.1), avant de poursuivre avec sa formulation détaillée (section 3.2). Nous présentons ensuite la complexité de la technique (section 3.3). Enfin, nous présentons une variante dans l'application de l'algorithme pour les modèles neuronaux ou de mixtures (section 3.4).

3.1 Intuition

Dans bien des applications, on peut approximer une matrice par ses vecteurs propres de plus grandes valeurs propres. Dans le cadre de la descente de gradient, il n'y a à priori aucune raison de penser que cette approximation est justifiée pour la matrice $\mathbf{\Phi}$. En effet, on a vu dans l'équation 1.6 que celle-ci permet d'appliquer un facteur d'apprentissage différent dans chaque direction de l'espace des paramètres. Dans le cas du gradient naturel, ces facteurs sont les inverses des valeurs propres de la matrice de covariance. Rien ne semble indiquer que certaines valeurs soient

plus importantes que d'autres, en particulier les valeurs propres les plus fortes, qui correspondent donc aux facteurs d'apprentissages les plus faibles.

Pourtant, l'argumentation développée par LeCun et al. (1998) va justement en ce sens dans le cas des matrices hessiennes de réseaux de neurones multi-couche. Ce qui pose problème dans l'apprentissage, c'est l'écart entre les grandes et les petites valeurs propres, mesuré par le ratio de la plus grande valeur propre et de la plus petite. Lorsque ce nombre est grand, la surface d'erreur prend la forme de longs ravins étroits, où le gradient ne pointe pas en direction du minimum. La méthode de Newton s'attaque à ce problème en affectant à chaque direction de l'espace des paramètres un facteur d'apprentissage qui lui est propre. Or, les observations de LeCun et al. (1998) montrent que le spectre de la matrice hessienne de réseaux de neurones multi-couche présente quelques grandes valeurs propres, beaucoup de moyennes et quelques petites. Ainsi, si on est en mesure de traiter séparément les quelques directions de fortes valeurs propres, on pourra ensuite utiliser un facteur d'apprentissage qui conviendra à la majeure partie des directions restantes, diminuant ainsi grandement la dégénérescence du problème. Étant donné l'étroite relation entre la hessienne et la matrice de covariance des gradients, on peut supposer que ce même discours s'applique au gradient naturel.

TONGA adopte cette approche : il s'agit de maintenir une approximation de faible rang de la matrice de covariance de façon à connaître les directions de grandes valeurs propres et les facteurs d'apprentissage à leur appliquer. Cependant, la décomposition en valeurs et vecteurs propres de la matrice de covariance est à priori une opération aussi coûteuse que son inversion. Elle nécessite également le calcul explicite de la matrice de covariance. On peut donc tenter de réaliser ces opérations dans un espace de dimension plus faible : l'espace engendré par les directions de plus forte variance et par le gradient considéré. La complexité des opérations s'en trouve fortement diminuée. La section suivante présente TONGA en détails.

3.2 TONGA

La spécification de l'algorithme TONGA repose sur trois éléments. Le premier est la définition de l'approximation de la matrice de covariance, le second est le calcul du gradient naturel à partir de cette approximation et le troisième est la réévaluation de de l'approximation de faible rang. Nous passons en revue chacun de ces éléments avant de présenter TONGA dans son ensemble dans l'algorithme 1.

3.2.1 Approximation de la matrice de covariance

Supposons que l'on dispose au temps t d'une approximation de faible rang $\hat{C}_t$ de la matrice de covariance des gradients :

$$\hat{C}_t = U_t U_t^T \tag{3.1}$$

où U est la matrice constituée des vecteurs propres non-normalisés de l'approximation. Lorsque l'on observe un gradient au temps $t+1$, on va intégrer son information ainsi :

$$C_{t+1} = \gamma \hat{C}_t + g_{t+1} g_{t+1}^T \tag{3.2}$$

où γ est un facteur d'oubli. On remarque ici deux choses. D'abord, le second terme n'est pas pondéré par $1-\gamma$. La raison est que l'on désire que la contribution de tous les gradients ayant servi dans l'approximation ait la même forme, en particulier ceux ayant servi à produire l'approximation initiale (qui seraient sans cela seuls à fournir une contribution non pondérée par $1-\gamma$). Il est donc nécessaire de normaliser notre approximation (ou encore le gradient naturel obtenu grâce à cette approximation) par $\gamma+1$. Ensuite, on remarque que l'on a utilisé la notation C_{t+1} plutôt que $\hat{C}_{t+1}$. Ceci vise à indiquer que le rang de l'approximation a peut-être augmenté. Il faudrait normalement procéder à une réévaluation de l'approximation de faible rang pour maintenir le rang. En pratique, on laisse plutôt croître le rang de l'approximation pendant b étapes pour des raisons de complexité sur lesquelles nous reviendrons

dans la section 3.3. Ainsi, pendant b étapes indexées chacune par n, l'approximation de la matrice de covariance est donnée par :

$$C_{t+n} = \gamma^n \hat{C}_t + \sum_{i=1}^{n} \gamma^{n-i} g_{t+i} g_{t+i}^T \tag{3.3}$$

On va reformuler ceci en posant l'écriture $X_{t+n} = [U_t \;\; \gamma^{\frac{-1}{2}} g_{t+1} \;\; \ldots \;\; \gamma^{\frac{-n+1}{2}} g_{t+n-1} \;\; \gamma^{\frac{-n}{2}} g_{t+n}]$, ce qui donne la formule matricielle :

$$C_{t+n} = \gamma^n X_{t+n} X'_{t+n} \tag{3.4}$$

Notons qu'il faut maintenant normaliser par $\gamma^t + \ldots + \gamma + 1$, c'est-à-dire par $(1 - \gamma^{t+1})/(1 - \gamma)$. Une fois que b étapes sont complétées, on procède à la réévaluation de l'approximation de faible rang (section 3.2.3).

Ceci nous fournit donc une approximation de la matrice de covariance qui peut être utilisée dans un contexte d'apprentissage stochastique. En pratique, on va utiliser une version régularisée de 3.4, en ajoutant une petite constante λ sur la diagonale de l'approximation

$$C_{t+n} = \gamma^n X_{t+n} X'_{t+n} + \lambda I \tag{3.5}$$

Ceci a pour effet de limiter les valeurs propres de la matrice à au moins λ, ce qui fait que le facteur d'apprentissage lié à notre approximation de la matrice de covariance dans n'importe quelle direction sera d'au plus $1/\lambda$.

3.2.2 Calcul du gradient naturel entre deux décompositions

Comme on l'a vu dans l'équation 2.1, le calcul du gradient naturel revient à appliquer un redimensionnement (facteur d'apprentissage) variable au gradient stochastique standard, suivant les directions de l'espace des paramètres. Avec TONGA, il s'agit d'un redimensionnement spécifique pour quelques directions (celles correspondant aux plus grandes valeurs propres de la matrice de covariance des gradients)

et d'un même redimensionnement pour toutes les autres directions. Il suffit donc de travailler dans le sous espace engendré par notre approximation de la matrice de covariance et par le gradient en cours. C'est d'ailleurs l'intérêt de l'inclusion du vecteur g_{t+n} dans l'évaluation de C_{t+n} (équation 3.2).

Calculons ainsi le gradient naturel v_{t+n} basé sur notre formulation de la matrice de covariance (équation 3.5) :

$$v_{t+n} = C_{t+n}^{-1} g_{t+n} = (\gamma^n X_{t+n} X_{t+n}^T + \lambda I)^{-1} g_{t+n} \tag{3.6}$$

Étant donné la formulation de X_{t+n}, on va remplacer g_{t+n} par l'expression suivante :

$$g_{t+n} = \gamma^{\frac{n}{2}} X_{t+n} y_n \text{ avec } y_n = [0, \dots 0, 1]^T \tag{3.7}$$

On peut alors reformuler le calcul du gradient naturel :

$$v_{t+n} = (\gamma^n X_{t+n} X_{t+n}^T + \lambda I)^{-1} \gamma^{\frac{n}{2}} X_{t+n} y_n \tag{3.8}$$

Utilisons maintenant l'identité de Woodbury pour les matrices définies positives (Petersen et Pedersen, 2006). Avec P et R deux matrices positives définies, cette identité dit :

$$(B^T R^{-1} B + P^{-1})^{-1} B^T R^{-1} = P B^T (B P B^T + R)^{-1}$$

Dans notre cas, on a $B = \gamma^{\frac{n}{2}} X_{t+n}^T$, $P^{-1} = \lambda I$ et $R = I$, ce qui nous donne :

$$v_{t+n} = \frac{1}{\lambda} I \gamma^{\frac{n}{2}} X_{t+n} (\gamma^{\frac{n}{2}} X_{t+n}^T \frac{1}{\lambda} I \gamma^{\frac{n}{2}} X_{t+n} + I)^{-1} y_n \tag{3.9}$$

$$v_{t+n} = \gamma^{\frac{n}{2}} X_{t+n} (\gamma^n X_{t+n}^T X_{t+n} + \lambda I)^{-1} y_n \tag{3.10}$$

où $X_{t+n}^T X_{t+n}$ est connue sous le nom de matrice de Gram, que l'on identifie par G_{t+n}. Mentionnons d'ailleurs que le calcul de G_{t+n+1} est très peu coûteux si on a mémorisé G_{t+n}. L'équation 3.10 est le résultat de l'approche décrite ci-haut, à savoir calculer le gradient naturel en passant par une autre base (de dimension plus faible) correspondant à l'espace engendré par l'approximation de la matrice

de covariance.

3.2.3 Réévaluation de l'approximation de faible rang

Chaque gradient supplémentaire pris en compte dans notre estimé de la covariance augmente la dimension des éléments mis en jeu dans l'équation du calcul du gradient naturel (équation 3.10). Il faut donc périodiquement réévaluer l'approximation de faible rang, une opération qui consiste à résumer l'information. Ceci peut se faire à faible coût grâce au lien entre la décomposition de la matrice de covariance et celle de la matrice de Gram, tel que montré par Schölkopf et al. (1998). Étant donné la décomposition de la matrice de Gram :

$$G_{t+n} = VDV^T \tag{3.11}$$

la décomposition de la matrice de covariance est donnée par :

$$C_{t+n} = (X_{t+n}VD^{-\frac{1}{2}})D(X_{t+n}VD^{-\frac{1}{2}})^T \tag{3.12}$$

3.3 Complexité

La complexité de TONGA dépend de la complexité du calcul du gradient naturel et de celui de la réévaluation de l'approximation de faible rang. Soient k le rang de l'approximation suite à une évaluation, b le nombre d'étapes entre chaque réévaluation et p le nombre de paramètres du modèle. Le coût du calcul du gradient naturel dans l'équation 3.10 semble d'abord être $O(p(k+b)^2 + (k+b)^3)$, lié au calcul et à l'inversion de la matrice de Gram. Cependant, le coût du calcul de G_t sachant G_{t-1} est réduit, et le coût devient $O(p(k+b) + (k+b)^3)$. Le coût de la réévaluation de l'approximation est quant à lui de l'ordre de $O(k(k+b)^2+p(k+b)k)$, respectivement pour la décomposition de la matrice de Gram, et le calcul des équivalents de la matrice de covariance. La consommation en mémoire est quant à elle de $O(p(k+b) + (k+b)^2)$ associée respectivement à la mémorisation de $\mathbf{X}$ et de

Algorithm 1 TONGA

{$\mathbf{g}$ est le gradient courant}
{p est la dimension du gradient}
{k est le rang de l'approximation de la matrice de covariance}
{b est le nombre d'étapes (de gradients) entre les réévaluations de l'approximation.}
{i est l'index des étapes entre les réévaluations de l'approximation.}
{λ est un paramètre de régularisation}
{γ est le facteur d'oubli}
{$\mathbf{d}$ est un vecteur contenant les valeurs propres de $\hat{C}_t$}
{$\mathbf{U}$ est la matrice des k vecteurs propres non-normalisés de l'approximation}
{$\mathbf{X}$ contient les k vecteurs propres suivi des i gradients accumulés}
{$\mathbf{v}$ est le gradient naturel résultant}
{$\mathbf{G}$ est la matrice de gram contenant $\mathbf{X}^T\mathbf{X}$}
{$\mathbf{a}$ et $\mathbf{r}$ sont des vecteurs temporaires de dimension $k+i$}
{$\mathbf{V}$ et $\mathbf{D}$ sont respectivement les vecteurs et valeurs propres de la matrice de Gram.}
Redimensionner X à $p \times k$ et initialiser à 0
Redimensionner G à $k \times k$ et initialiser à 0
Redimensionner $\mathbf{r}$ et $\mathbf{a}$ à k et initialiser à 0
for all t **do**
  $i \leftarrow (t \bmod b) + 1$
  Redimensionner $\mathbf{X}$ à $p \times k + i$
  $\mathbf{X}[:, k+i] \leftarrow \gamma^{\frac{-i}{2}} g$
  Redimensionner $\mathbf{G}$ à $(k+i) \times (k+i)$
  $\mathbf{G}[k+i, :] \leftarrow \mathbf{X}^T[k+i, :]\mathbf{X}$
  $\mathbf{G}[:, k+i] \leftarrow \mathbf{G}[k+i, :]^T$
  Redimensionner r et a à $k+i$
  $\mathbf{r}[k+i-1] \leftarrow 0$
  $\mathbf{r}[k+i] \leftarrow \gamma^{\frac{-i}{2}}$
  Résoudre le système linéaire $(\mathbf{G} + \gamma^{-i}\lambda\mathbf{I})\mathbf{a} = \mathbf{r}$ en $\mathbf{a}$
  $v \leftarrow \mathbf{X}\mathbf{a} \times (1-\gamma)/(1-\gamma^{t+1})$
  if $i == b$ **then**
    {Refaire l'approximation de rang k}
    $(\mathbf{V}, \mathbf{d}) \leftarrow$ eigendecomposition(G)
    $\mathbf{U} \leftarrow \gamma^{b/2} XV$
    Redimensionner $\mathbf{X}$, $\mathbf{G}$, $\mathbf{r}$ et $\mathbf{a}$ en remplaçant la dimension $k+i$ par k
    $\mathbf{X} \leftarrow \mathbf{U}$
    $\mathbf{G} \leftarrow \mathbf{U}^T\mathbf{U}$
    $\mathbf{r} \leftarrow \mathbf{0}$
  end if
end for
Retourner $\mathbf{v}$

$\tilde{\mathbf{G}}$. On comprend donc que TONGA peut permettre des gains substantiels dans la mesure où une approximation de faible rang est adéquate.

Si l'on suppose que $k \leq b$ et que $b^2 \leq p$, le coût total en calculs de TONGA par calcul de gradient naturel[1] est alors de $O(pb)$, et de même pour la mémoire. Si on travaille de plus sur des mini-lots de gradients de gradients de taille b', on aboutit à un coût par exemple de l'ordre de $O(\frac{bp}{b'})$. Avec un choix de $b = b'$, on obtient un coût en calculs de $O(p)$ par exemple, tout comme pour le gradient stochastique. TONGA possède donc un coût fortement réduit par rapport aux diverses autres tentatives visant à diminuer la complexité du gradient naturel (section 2.6).

3.4 Variante bloc-diagonale

Collobert (2004) montre que dans les mixtures d'experts et dans les modèles neuronaux multi-couche ayant un coût de type NLL, la matrice hessienne devient rapidement bloc-diagonale. Ces blocs sont formés des poids d'un même neurone. Ainsi, une fois les symétries dans l'espace des paramètres brisées, les paramètres des différents neurones deviennent localement indépendants. Là encore, le lien entre la hessienne et la matrice de covariance des gradients nous permet d'envisager une variante bloc-diagonale de TONGA. Cette variante consiste à maintenir une approximation de faible rang pour chaque bloc constitué des poids d'un même neurone. Cette approche est valable à la fois dans le cas des modèles neuronaux et dans celui de n'importe quel modèle impliquant des mélanges de plusieurs modèles indépendants (un bloc consiste alors des paramètres d'un même modèle).

Dans le cas des modèles neuronaux, on peut se demander si tous les blocs ont la même importance. En effet, LeCun et al. (1998) font l'observation que la hessienne possède des valeurs propres généralement plus faibles dans les couches proches de l'entrée et que ces valeurs deviennent plus fortes lorsque l'on approche des couches de sortie. C'est ce qui explique le phénomène connu de l'apprentissage rapide et oscillant observé dans les couches de sortie.

[1] Le coût de la réévaluation de l'approximation est donc porté sur b calculs de gradient naturel.

CHAPITRE 4

EXPÉRIENCES

Nous présentons maintenant les détails et les résultats de l'évaluation empirique de l'algorithme TONGA présenté au chapitre 3. La section 4.1 présente et justifie le choix du modèle d'apprentissage utilisé dans les expériences. La section 4.2 passe ensuite en revue les hyperparamètres mis en jeu à la fois par le modèle et par les algorithmes d'apprentissage. Nous présentons ensuite à la section 4.3 les problèmes sur lesquels TONGA est évalué, avant de présenter les résultats généraux obtenus (section 4.4). La section 4.5 présente quelques résultats additionels.

4.1 Choix du modèle d'apprentissage

En tant qu'algorithme d'apprentissage, TONGA doit être appliqué à l'optimisation d'un modèle d'apprentissage. La seule restriction quant au choix de ce modèle est le requis qu'il possède une paramétrisation différentiable.

Les modèles neuronaux constituent un vaste ensemble de modèles répondant à ce critère. De plus, ces modèles se prêtent naturellement à la variante bloc-diagonale envisagée précédemment (section 3.4). Sur le plan théorique, le théorème de Kolmogorov stipule qu'un réseau de neurones peut représenter n'importe quelle fonction continue. La difficulté liée à leur utilisation réside cependant dans leur optimisation, qui bien que fortement étudiée demeure difficile (LeCun et al., 1998; Tesauro, 1992). C'est d'ailleurs pourquoi ces modèles sont tombés quelque peu en défaveur avec l'arrivée de modèles à base de vecteurs de supports (SVM) (Schölkopf et Smola, 2002) dont l'optimisation a l'avantage d'être facile (convexe). Pourtant, l'exploration récente des limitations des SVM (notamment dans Bengio et Le Cun (2007)) a ravivé l'intérêt pour les modèles neuronaux, de même que la découverte de nouvelles stratégies d'optimisation[1] (Hinton et al., 2006; Bengio et al., 2007). Pour

[1] Ces stratégies sont basées sur l'utilisation des techniques à base de gradient.

toutes ces raisons, nous avons choisi d'évaluer TONGA sur les modèles neuronaux.

Considérons un réseau de neurones possédant C couches cachées, prenant en entrée $\mathbf{x}$ et produisant une sortie $\mathbf{y}$. Chaque couche cachée opère sur sa propre entrée $\mathbf{e}^c$ qui correspond à la sortie de la couche cachée précédente ou à $\mathbf{x}$ dans le cas de la première couche cachée. Elle calcule d'abord une activation $\mathbf{a}^c$ selon

$$\mathbf{a}^c = \mathbf{W}^c \mathbf{e}^c \tag{4.1}$$

où $\mathbf{W}^c$ est la matrice des poids de la couche cachée. La couche cachée applique ensuite une non-linéarité f^c (dite fonction de transfert) pour produire sa sortie $\mathbf{s}^c$

$$\mathbf{s}^c = f^c(\mathbf{a}^c) \tag{4.2}$$

Le réseau possède également une couche de sortie responsable du calcul de la sortie du réseau de neurones et qui opère suivant le même fonctionnement, à partir de la sortie de la dernière couche cachées (la non-linéarité n'est cependant pas toujours appliquée).

Notre implantation des réseaux de neurones est standard. Les fonctions de tranfert des couches cachées correspondent toutes à la tangente hyperbolique (appliquée individuellement à chaque élément de $\mathbf{a}^c$) et nous appliquons une non-linéarité de type *softmax* dans la couche de sortie

$$\mathbf{y}_i = \frac{\exp^{\mathbf{a}_i}}{\sum_j \exp^{\mathbf{a}_j}} \tag{4.3}$$

où i est un indexe sur les éléments des vecteurs d'activation et de sortie. La sortie est donc une probabilité. La descente de gradient est effectuée sur un coût de log-vraissemblance négative (NLL), c'est-à-dire avec t la sortie désiré : $L(\theta) = -\log p(t|\mathbf{x}, \theta)$.

4.2 Hyperparamètres

Les hyperparamètres sont des paramètres qui ne sont pas ajustés automatiquement et qui affectent le comportement du modèle ou de l'algorithme auquel ils se rapportent. Notre cadre expérimental en met en jeu plusieurs, et il est nécessaire de spécifier la stratégie utilisée pour les déterminer.

Le premier groupe d'hyperparamètres a trait à l'architecture du réseau de neurone. Il s'agit du nombre de couches et de neurones (unités cachées) par couche. Nos expériences se concentrent sur les réseaux à une seule couche cachée, car cette configuration est celle utilisée le plus souvent en pratique. Cependant, nous envisageons également un réseau à deux couches cachées dans une expérience, car d'une part ces réseaux peuvent théoriquement être plus expressifs (bien que leur optimisation soit plus difficile) et de l'autre ils possèdent plus de niveaux de non-linéarités. Cette dernière caractéristique affecte donc la qualité de l'approximation de la matrice hessienne fournie par la matrice de covariance (section 2.2).

Les autres hyperparamètres concernent les algorithmes d'apprentissage. Dans le cas du gradient stochastique, il s'agit de la taille des lots (mbs) utilisés pour estimer le gradient, du facteur d'apprentissage (lr), de sa constante de décroissance (dc) et du facteur de régularisation L_1 ($L1$). On a déjà mentionné le rôle de la taille des lots utilisés pour le calcul du gradient (1.3.6) et du facteur d'apprentissage (1.3). La constante de décroissance du facteur d'apprentissage est utilisée pour rendre compte de l'observation que le facteur d'apprentissage doit être grand en début d'optimisation (alors que les valeurs des paramètres sont "loins" des valeurs optimales et que les symétries dans l'espace des paramètres causent des plateaux dans la fonction de coût) et faible vers la fin. Le taux d'apprentissage effectif est alors donné par

$$lr_{effectif} = \frac{1}{1 + dc \times u} lr$$

où u est le nombre de mises à jour de paramètres effectuées. Le facteur de régularization a pour rôle de limiter le surentraînement. Dans les cas où le modèle risque d'apprendre des particularités des données d'entraînement, on peut rajouter un

terme au coût pour pénaliser la complexité du modèle. Dans le cas de la régularisation L_1 ce terme est proportionel à la norme L_1 du vecteur de paramètres :

$$|\mathbf{x}|_1 = \sum_{d=1}^{D} |\mathbf{x}_d| \tag{4.4}$$

Le facteur de régularisation est utilisé pour moduler l'importance du terme de complexité dans le coût.

TONGA possède tous les hyperparamètres du gradient stochastique ordinaire, mais aussi quatre de plus. Le premier est le facteur d'oubli γ utilisé dans la formulation de l'approximation de la matrice de covariance (équation 3.5). Ce paramètre contrôle la stabilité de l'estimé. Le second hyperparamètre est le rang de l'approximation de faible rang de la covariance (k), c'est-à-dire le nombre de directions retenues suite à une réévaluation de l'approximation de la matrice de covariance. Un autre hyperparamètre est le nombre d'étapes entre chaque réévaluation de l'approximation. Cet hyperparamètre, combiné au précédent, détermine le nombre de directions qui recevront un facteur d'apprentissage particulier. Enfin, le dernier hyperparamètre additionnel est le terme de régularisation λ de l'approximation de la matrice de covariance. En pratique, il détermine le facteur d'apprentissage maximal appliqué par TONGA dans toute direction, en particulier dans les directions autres que celles données par les premiers vecteurs propres de la décomposition de la matrice de covariance des gradients.

On remarque que TONGA possède un grand nombre d'hyperparamètres. Idéalement, on aimerait avoir des valeurs par défaut pour ces paramètres, ou alors des façons de les ajuster automatiquement. Dans le cas du terme de régularisation λ, on peut envisager la variante suivante. Au lieu d'employer un terme fixe (le même dans tous les blocs pour la variante bloc-diagonale), on pourrait envisager l'ajuster à chaque réévaluation de l'approximation de faible rang à la valeur de la dernière valeur propre. C'est un ajustement conservateur, mais spécifique à chaque neurone. En effet, on a mentionné les observations de LeCun et al. (1998) selon lesquelles la forme de la hessienne est différente suivant les couches du réseau de neurones.

En pratique, on a fait les mêmes observations avec le spectre de la matrice de covariance : les valeurs propres des neurones de sorties sont d'un autre ordre que celles des neurones cachés. On adresse alors possiblement mieux le rôle de fournir un facteur d'apprentissage pour les directions autres que celles données par les premiers vecteurs propres de la décomposition de la matrice de covariance des gradients (sous réserve que cette approche ne soit pas trop conservatrice). Cependant, un autre hyperparamètre devient alors nécessaire pour effectuer le rôle habituel d'un paramètre de régularisation : une valeur minimale λ_{min} de λ pour fournir en quelque sorte un garde-fou.

4.2.1 Sélection des hyperparamètres

Le protocole expérimental suivant a été appliqué à la sélection des hyperparamètres. Le nombre d'unitées cachées est fixé à une valeur qui donne des résultats satisfaisants sur la base d'un entraînement par gradient stochastique. Les meilleurs jeux d'hyperparamètres des algorithmes d'apprentissage ont été déterminés sur la base de la combinaison fournissant la meilleure erreur de classification sur l'ensemble de validation pour des combinaisons de valeurs tirées d'intervalles choisis, et ce pour une initialisation aléatoire donnée des paramètres et à l'intérieur d'un laps de temps fixé. Une fois la sélection des hyperparamètres complétée, ces hyperparamètres ont été testés sur neuf initialisations aléatoires supplémentaires. Les dix exécutions permettent ensuite de tracer des courbes munies de barres d'erreurs représentant les écart-types.

Notre procédure de sélection des hyperparamètres sur la base de la meilleure erreur de validation vise à nous permettre de comparer des algorithmes en fonction de la qualité de la solution obtenue en terme de généralisation. En effet, l'erreur de validation est un estimateur non-biaisé de l'erreur de généralisation. La procédure permet également de comparer la vitesse de convergence liée à ce choix d'hyperparamètres. Il faut cependant se montrer méfiant des comparaisons générales de la capacité à réduire rapidement l'erreur de généralisation. Il existe vraissemblablement des jeux d'hyperparamètres diminuant cette erreur plus rapidement, mais ils

ne parviennent pas à une erreur aûssi faible. Cependant, en pratique on applique également cette procédure de sélection, et c'est donc la vitesse de convergence liée au jeu d'hyperparamètres de meilleure validation qui nous intéresse.

4.3 Description des jeux de données

Cette section décrit les trois problèmes sur lesquels nous avons évalué TONGA.

4.3.1 Letter

Letter est un jeu de données de taille moyenne. Le problème consiste à identifier les lettres de l'alphabet à partir de seize attributs (ce qui est peu). Pour ce faire on dispose de 20000 exemples, les 16000 premiers servant à l'entraînement. Nous avons divisé les exemples restant en deux, la première moitié formant l'ensemble de validation (servant à la sélection des hyperparamètres) et la seconde celui de test.

4.3.2 USPS

Le jeu de données *United States Postal Service* (USPS) comporte 9298 images 16×16 de chiffres manuscrits ayant été normalisées et recentrées. Celles-ci sont réparties en 7291 exemples d'entraînement et 2007 exemples de tests. Nous avons mis de côté les 1000 derniers exemples d'entraînement pour constituer un ensemble de validation.

4.3.3 MNIST

MNIST (LeCun et Cortes) est problablement le jeu de données de reconnaissance de caractères le plus utilisé. Il s'agit de 70000 images (28×28 pixels) de chiffres manuscrits, normalisées selon la taille du chiffre, puis recentrées. Nous avons divisé les exemples ainsi : 50000 exemples d'entraînement, 10000 exemples de validation et 10000 exemples de test.

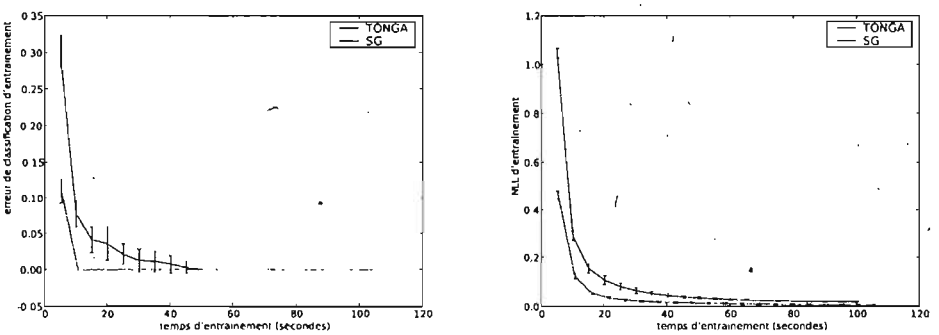
4.4 Résultats généraux

Cette section dresse le portrait général de la performance de TONGA. Nous y présentons les résultats pour les meilleurs candidats du gradient stochastique et de TONGA (parmi toutes ses variantes), en se basant sur la sélection d'hyperparamètres évoquée précédemment (4.2.1). Les hyperparamètres sélectionnés sont présentés à l'annexe I.

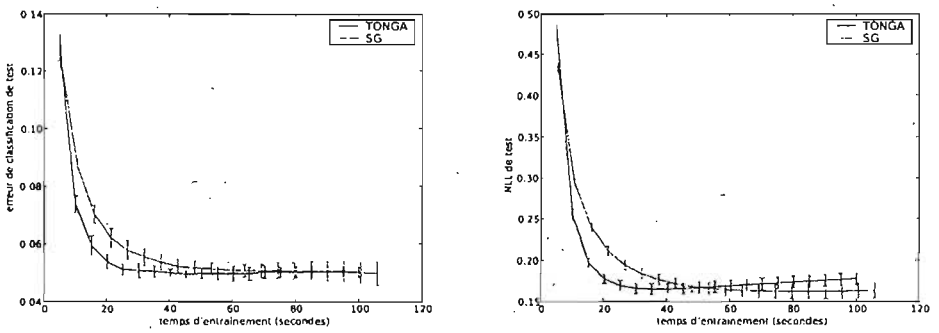
4.4.1 Letter

Après avoir fait quelques essais avec un réseau de neurones à une couche cachée entraîné par gradient stochastique, nous avons fixé le nombre d'unités cachées à 100. Il s'agit là d'un nombre assez petit d'unités cachées qui permet d'obtenir des résultats corrects. Le temps d'entraînement alloué était de 100 secondes. La comparaison entre TONGA et le gradient stochastique est présentée à la figure 4.1. La meilleure performance pour TONGA est obtenue dans sa variante bloc-diagonale avec adaptation de λ.

TONGA apparaît moins rapide à diminuer les erreurs d'entraînement, tant pour la classification que pour la NLL. Les écarts types dans le cas de l'erreur de classification sont également supérieurs pour TONGA. On peut cependant supposer un biais de cette mesure. En effet, ces courbes représentent en pratique l'erreur sur l'ensemble d'entraînement telle qu'estimée à travers le comportement du modèle en cours d'optimisation (alors que la valeur des paramètres change constamment). Cette pratique vise à obtenir un estimé gratuit de l'erreur sur l'ensemble d'entraînement, c'est-à-dire sans avoir à l'inspecter entre chaque étape d'optimisation. Étant donné le grand nombre d'exemples d'entraînement, l'évaluation exacte de cette erreur est coûteuse. Dans le cas de techniques fonctionnant avec des tailles de mini-lots différentes, la comparaison des estimés résultant s'avère biaisée. Pour le voir, considérons deux techniques possédant des tailles de mini-lots de 1 et 100. Si on s'intéresse à l'estimé fourni après 100 exemples, il se basera sur des résultats obtenus avec 100 valeurs différentes des paramètres dans le cas du mini-lot de taille



(a) Erreur de classification d'entraînement (b) NLL d'entraînement



(c) Erreur de classification de test (d) NLL de test

FIG. 4.1 – Comparaison entre le gradient stochastique (SG) et TONGA sur la base de données LETTER, sur la base de l'erreur de classification et de la log-vraissemblance négative (NLL), pour 10 initialisations aléatoires des paramètres.

1 mais sur les paramètres d'initialisation dans le cas des paramètres associés au mini-lot de taille 100. Cet effet s'estompe cependant au fur et à mesure que l'on observe plus d'exemples.

Au niveau des erreurs de test on voit que TONGA offre une performance supérieure. TONGA minimise plus rapidement ces erreurs et commence déjà à montrer les signes du surapprentissage en dépit de la faible capacité du modèle (qui possède seulement 100 unités cachées). Cette remontée des erreurs est plus forte dans le cas de la NLL, ce qui indique que le modèle paie plus chèrement en terme de

confiance dans ses erreurs, qu'en terme de nombre d'erreurs. En effet, l'erreur de classification assigne un coût unitaire à chaque exemple mal classifié tandis que la NLL pénalise en fonction de la probabilité assignée à la décision. Ainsi, un faible nombre d'exemples peut suffire à déteriorer fortement la NLL.

4.4.2 USPS

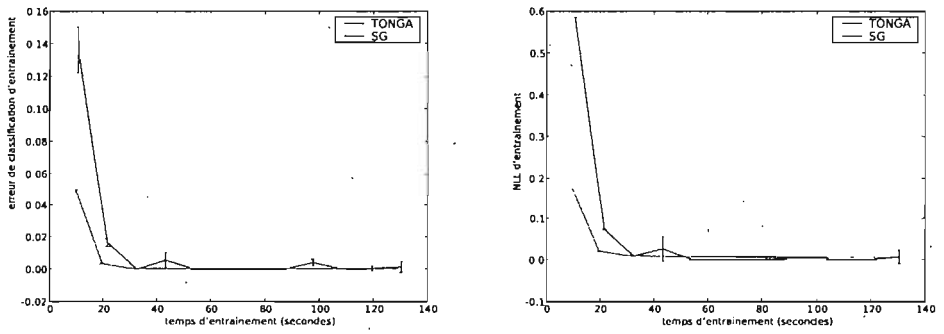
Suite aux tests préliminaires avec le gradient stochastique, nous avons fixé le nombre d'unités cachées à 300. Le temps d'entraînement alloué était de 130 secondes. Les résultats sont présentés à la figure 4.2.

Les courbes d'entraînement obtenues sont sujettes au même commentaire que celles obtenues sur Letter. On remarque cependant une performance similaire. Au niveau des erreurs de test, la principale différence se situe au niveau de la NLL, pour laquelle l'erreur de TONGA se met à augmenter après peu de temps, alors que l'erreur de classification demeure stable. C'est là encore le phénomène observé sur Letter : le modèle devient très confiant sur quelques exemples sur lesquels sa décision est mauvaise. Cela se paye cher en log-vraissemblance négative, mais pas en nombre d'erreurs. Au niveau de l'erreur de classification, TONGA semble un peu plus lent à diminuer l'erreur, mais rattrape le retard après 40 secondes. La courbe de l'erreur de classification en test est quant à elle favorable à TONGA qui réussi à atteindre une erreur plus faible d'un demi pourcent en moyenne après une minute d'optimisation.

4.4.3 MNIST

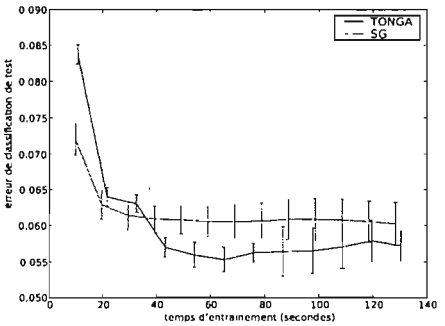
Les expériences sur ce jeu de données ont été menées avec 800 unités cachées avec un temps d'optimisation accordé de 5000 secondes. Les résultats pour TONGA ont été obtenus avec la variante bloc-diagonale et l'adaptation de λ. Ces résultats sont présentés à la figure 4.3.

TONGA performe ici nettement mieux que le gradient stochastique. Il est plus rapide à réduire les erreurs d'entraînement, et l'algorithme atteint après moins

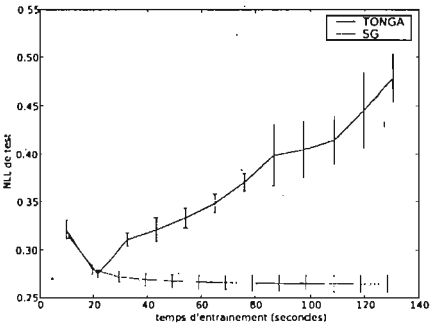


(a) Erreur de classification d'entraînement

(b) NLL d'entraînement



(c) Erreur de classification de test



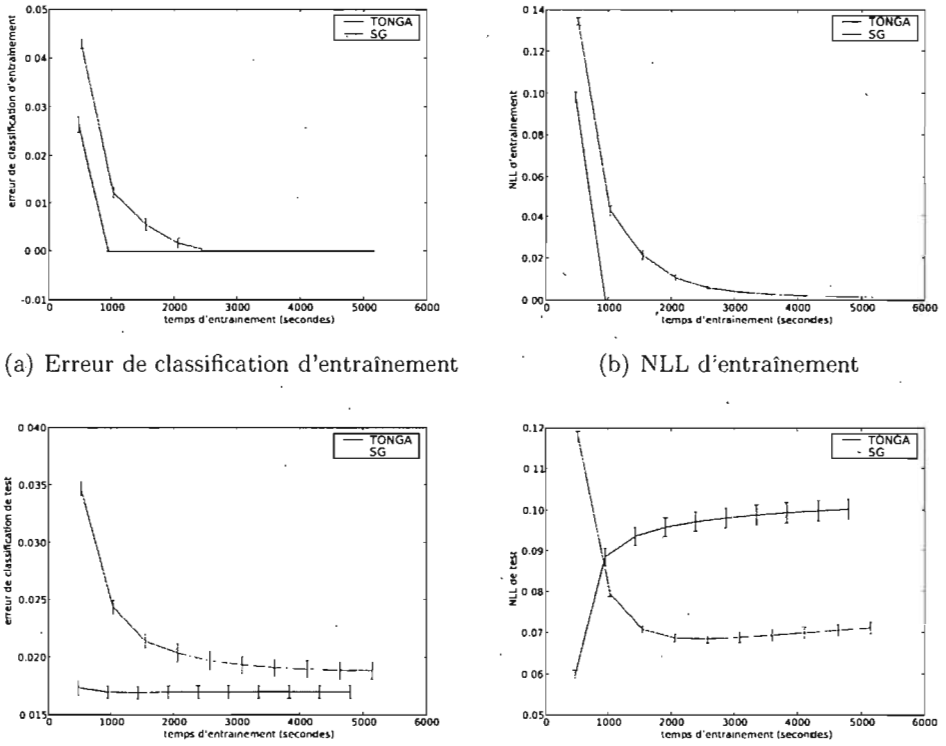
(d) NLL de test

FIG. 4.2 – Comparaison entre le gradient stochastique (SG) et TONGA sur la base de données USPS, sur la base de l'erreur de classification et de la log-vraissemblance négative (NLL), pour 10 initialisations aléatoires des paramètres.

de 1000 secondes un minimum en erreur de classification de test que le gradient stochastique n'atteint pas au bout de 5000 secondes. Au niveau de la NLL en test, on observe le même phénomène que précédemment décrit (le modèle paye cher une grande confiance dans un nombre petit d'erreurs).

4.5 Résultats additionels

Nous présentons dans cette section quelques résultats additionnels. D'abord, une expérience sur un réseau à deux couches cachées. Ensuite, des résultats com-



(a) Erreur de classification d'entraînement

(b) NLL d'entraînement

(c) Erreur de classification de test

(d) NLL de test

FIG. 4.3 – Comparaison entre le gradient stochastique (SG) et TONGA sur la base de données MNIST, sur la base de l'erreur de classification et de la log-vraissemblance négative (NLL), pour 10 initialisations aléatoires des paramètres.

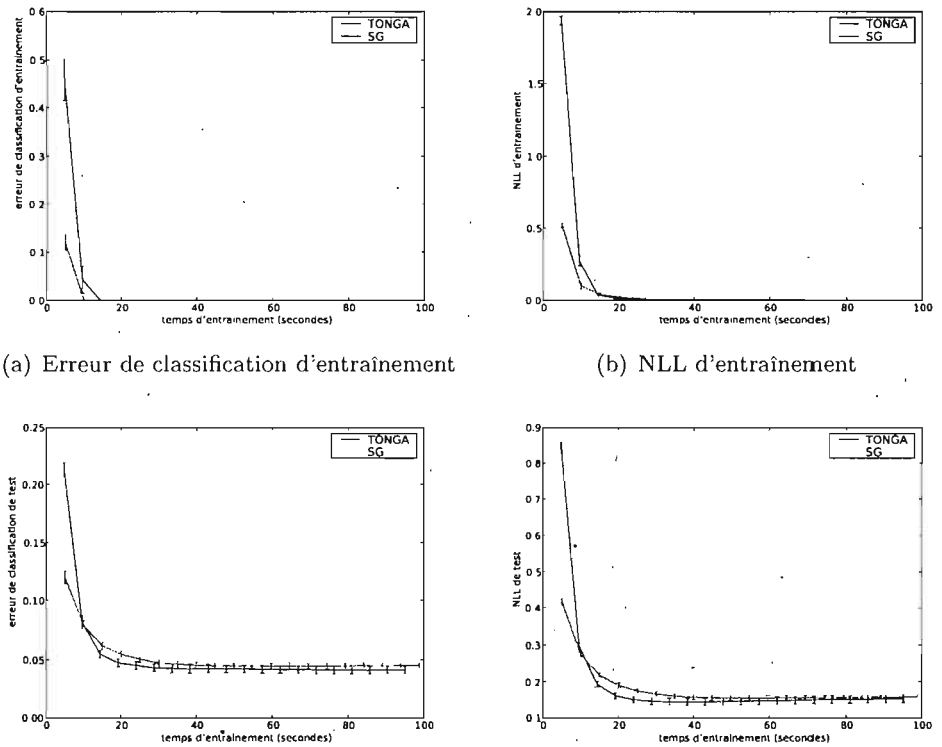
paratifs des différentes variantes de TONGA. Les hyperparamètres sélectionnés sont également présentés à l'annexe I.

4.5.1 Réseau à deux couches cachées

Nous envisageons maintenant une architecture à deux couches cachées pour les raisons évoquées précédemment (section 4.2), en particulier le fait que la présence de multiples couches de non-linéarités peut affecter l'hypothèse évoquée dans la section 2.2 justifiant le gradient naturel comme étant une approximation de la

méthode de Newton. On peut donc se demander si la performance de TONGA (en tant que gradient naturel) s'en voit diminuée.

L'expérience a été menée sur la base de données LETTER avec deux couches cachées de 100 et 50 unités cachées respectivement. Les résultats sont présentés à la figure 4.4.



(a) Erreur de classification d'entraînement

(b) NLL d'entraînement

(c) Erreur de classification de test

(d) NLL de test

FIG. 4.4 – Comparaison entre le gradient stochastique (SG) et TONGA sur la base de données LETTER 2 LAYERS, sur la base de l'erreur de classification et de la log-vraissemblance négative (NLL), pour 10 initialisations aléatoires des paramètres.

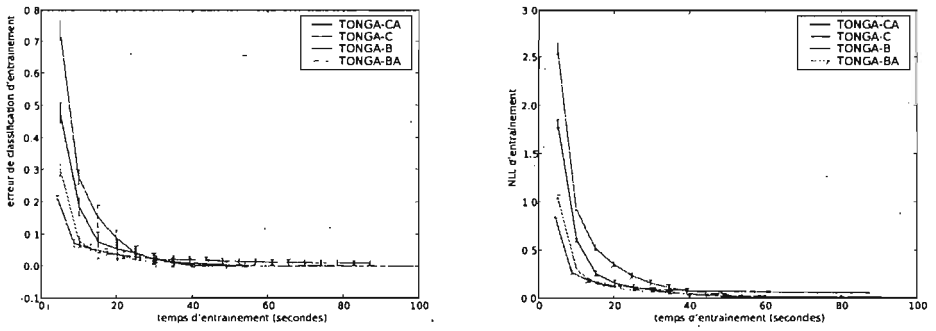
On observe ici essentiellement la même performance que dans le cas du réseau à une seule couche cachée. TONGA est plus rapide à diminuer les erreurs de test et

se met à montrer des signes de surapprentissage alors que le gradient stochastique finit par rattraper.

4.5.2 Performance des différentes variantes de TONGA

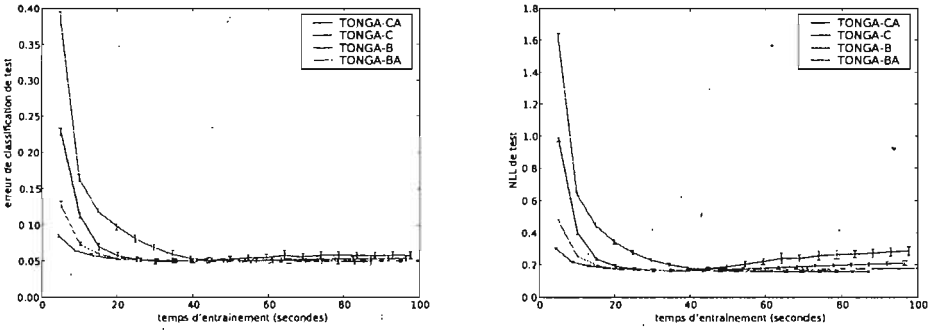
Nous présentons maintenant des résultats comparatifs des différentes variantes de TONGA. La figure 4.5 donne les résultats sur LETTER dans le cas du réseau à une couche cachée. Des résultats sur MNIST sont présentés à la figure 4.6.

On remarque que les variantes bloc-diagonales convergent plus rapidement. Par contre, la variante adaptative de λ ne semble pas améliorer (ni détériorer) les résultats. La courbe de TONGA en version pleine avec adaptation de λ présente de grandes variations. En observant les résultats pour chacune des initialisations, on découvre que quatre de ces initialisations mènent à des divergences. Il faut en déduire que le facteur d'apprentissage appliqué est trop élevé.



(a) Erreur de classification d'entraînement

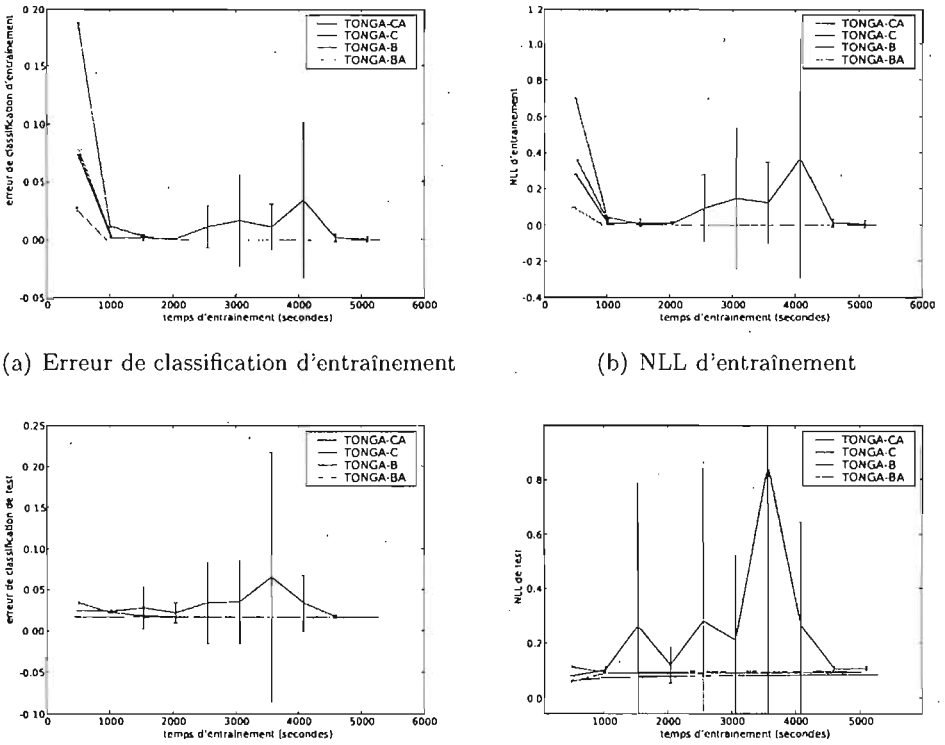
(b) NLL d'entraînement



(c) Erreur de classification de test

(d) NLL de test

FIG. 4.5 – Comparaison entre les différentes variantes de TONGA (selon que TONGA est appliqué à la matrice de covariance complète (TONGA-C) ou sur les blocs formés des paramètres de chaque neurone (TONGA-B) et suivant l'utilisation de l'ajustement automatique de λ (extension A) ou pas) sur la base de données LETTER, sur la base de l'erreur de classification et de la log-vraissemblance négative (NLL), pour 10 initialisations aléatoires des paramètres.



(a) Erreur de classification d'entraînement

(b) NLL d'entraînement

(c) Erreur de classification de test

(d) NLL de test

FIG. 4.6 – Comparaison entre les différentes variantes de TONGA (selon que TONGA est appliqué à la matrice de covariance complète (TONGA-C) ou sur les blocs formés des paramètres de chaque neurone (TONGA-B) et suivant l'utilisation de l'ajustement automatique de λ (extension A) ou pas) sur la base de données MNIST, en fonction de l'erreur de classification et de la log-vraissemblance négative (NLL), pour 10 initialisations aléatoires des paramètres.

CHAPITRE 5

DISCUSSION

Les méthodes classiques d'optimisation de second ordre ne sont pas applicables au cadre des problèmes de grande taille, c'est-à-dire comportant un grand nombre d'exemples et nécessitant des modèles possédant de nombreux paramètres. TONGA est un algorithme d'apprentissage qui vise à tirer profit de ces techniques en contournant l'obstacle de leur complexité. Nos résultats montrent que TONGA est au moins aussi efficace que le gradient stochastique ordinaire, la technique à base de gradient communément employée dans un tel contexte. En d'autres mots, TONGA obtient des gains en convergence qui lui permettent de compenser le surcoût lié aux calculs additionnels mis en jeu. C'est déjà là un grand pas vers l'adaptation des techniques de second ordre au cadre des problèmes de grande taille. De plus, dans une des expériences réalisées, TONGA se montre significativement plus performant que le gradient stochastique ordinaire.

Dans ce chapitre nous analysons d'abord les résultats obtenus au niveau de l'ajustement des hyperparamètres, de la performance de TONGA et de l'efficacité de la variante bloc diagonale. Nous envisageons ensuite des pistes pour l'amélioration de TONGA.

5.1 Ajustement des hyper-paramètres de TONGA

L'ajustement des hyperparamètres de TONGA n'a pas posé de difficulté particulière. La taille des lots (*mbs*) utilisés a été ajustée grossièrement en essayant des valeurs parmi $\{100, 500, 1000\}$. Les principales considérations face à cet ajustement sont d'une part le rôle de l'hyperparamètre dans la complexité (équation 3.3) et de l'autre la convergence plus rapide obtenue avec des petits lots (à cause de la fréquence plus élevée de mises à jour des paramètres). La constante de décroissance (*dc*) n'a nécessité aucune attention particulière dans son ajustement et

la sélection a été faite parmi de petites valeurs par défaut, à savoir l'ensemble $\{0, 1e-8, 1e-7, 1e-6, 1e-5\}$. Des essais sommaires ont montrés que le paramètre de régularisation L_1 ne semblait pas améliorer les résultats, et il a donc été fixé à zéro pour les expériences sur USPS et MNIST. Il en va de même pour le facteur d'oubli γ dont l'influence semblait faible : sa valeur a été fixée à 0.995 dans toutes nos expériences.

Les derniers hyperparamètres, à savoir le facteur d'apprentissage (lr), le rang de l'approximation de la matrice de covariance(k), le nombre d'étapes entre les réévaluations de cette approximation ($cmbs$) et le facteur de régularisation (λ) sont l'objet d'une interaction. Le fait de modifier l'un affecte le choix optimal pour les autres. En effet, tous ces hyperparamètres affectent les facteurs d'apprentissage appliqués dans les diverses directions de l'espace des paramètres. Le facteur d'apprentissage (lr) a une action globale. Le rang de l'approximation de la matrice de covariance suite à une réévaluation (k) et le nombre d'étapes entre les réévaluations de cette approximation ($cmbs$) affectent le rang de l'approximation de la covariance, c'est-à-dire le nombre de directions qui recevront un facteur d'apprentissage spécifique (sous réserve que ce facteur soit inférieur au facteur seuil imposé par la régularisation). Enfin, le facteur de régularisation (λ) affecte un facteur d'apprentissage aux directions restantes.

L'ajustement automatique de ce dernier groupe d'hyperparamètres (ou de certains d'entre eux à partir des autres) serait un réel atout pour rendre TONGA plus sympathique auprès du nouvel usager. Notre première tentative, qui consiste à ajuster automatiquement λ à la dernière valeur propre obtenue dans la décomposition n'est pas concluante. TONGA repose sur l'idée que le spectre de la matrice de covariance comporte quelques grandes valeurs propres, un grand nombre de moyennes et quelques petites, ce que nous observons en pratique. Le problème réside cependant dans l'identification de la transition entre les grandes et les moyennes valeurs propres. Au cours de l'optimisation, le spectre peut varier (comporter plus ou moins de grandes valeurs propres). Si l'on fixe le nombre de directions que l'on considère être de grandes valeurs propres, deux cas de figures sont envisageables. Soit ce

nombre est trop petit face au vrai spectre et si l'on se base sur la dernière valeur propre pour établir λ on sera trop conservateur (facteur d'apprentissage trop faible dans les directions restantes). Soit ce nombre est trop grand, et on sera alors potentiellement trop aggressif (divergence) si apparaît par la suite dans le spectre une nouvelle direction de grande valeur propre (cette apparition sera constatée par le mécanisme d'ajustement à posteriori). Notre impression est donc que ces hyperparamètres sont actuellement ajustées de façon sous optimale. Pourtant, même avec ces ajustements on observe des performances intéressantes. Un meilleur ajustement serait donc très prometteur.

5.2 Performance de TONGA

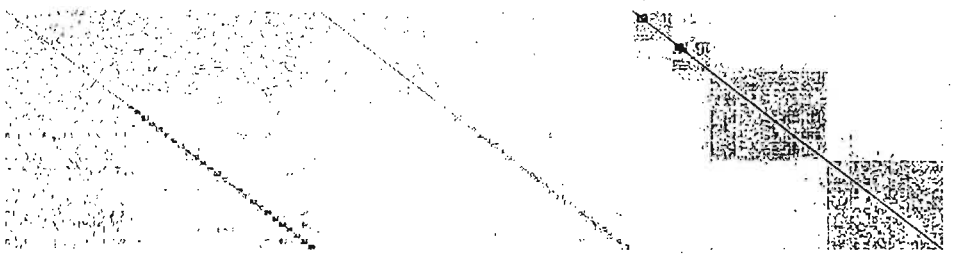
Pour tenter d'expliquer la différence de performance de TONGA sur LETTER et USPS d'une part, et MNIST de l'autre, nous offrons l'explication suivante. Les gains de convergence en temps de TONGA dépendent du coût de calcul du gradient naturel. On a vu que ce coût est basé sur le rang de l'approximation de la matrice de covariance (k) et sur le nombre d'étapes entre les réévaluations de cette approximation (*cmbs*). Suivant les problèmes, le ratio du nombre de grandes valeurs propres dans le spectre de la matrice de covariance sur le nombre de dimensions de l'espace des paramètres n'est pas le même. Ainsi, les résultats sur MNIST sont obtenus avec un coût de calcul du gradient naturel[1] quasi identique à celui dans le cas de LETTER (les valeurs de k et *cmbs* utilisées sont proches). On en retire donc un plus grand bénéfice.

5.3 Efficacité de la variante bloc diagonale

Nos expériences montrent que la variante bloc diagonale de l'algorithme fonctionne très bien, peu importe que l'on adapte λ ou pas. L'approximation résultante de la matrice de covariance doit donc être bonne. Dans le but de vérifier cette affirmation, nous avons calculé la matrice de covariance des gradients des paramètres

[1] Dans la version bloc diagonale il s'agit d'un coût par bloc.

d'un petit réseau de neurones (de façon à ce que la mémorisation de la matrice de covariance ne dépasse pas la mémoire du système) entraîné sur LETTER. Il s'agit donc d'un réseau à seize entrées, cinquante unitées cachées et vingt-six sorties. La figure 5.1 montre la matrice de corrélation correspondante. Les premiers blocs représentent les poids des unités d'entrées vers chaque unité cachée (cinquante blocs) et les suivants les poids des unités cachées vers les sorties (vingt-six blocs). On peut voir que l'approximation bloc diagonale est raisonable, même après seulement un passage sur les données d'entraînement. On voit aussi que la matrice de covariance obtenue en optimisant avec TONGA est davantage bloc-diagonale, confirmant l'efficacité de la technique (tout comme la formulation classique du gradient naturel) à briser rapidement les symétries.

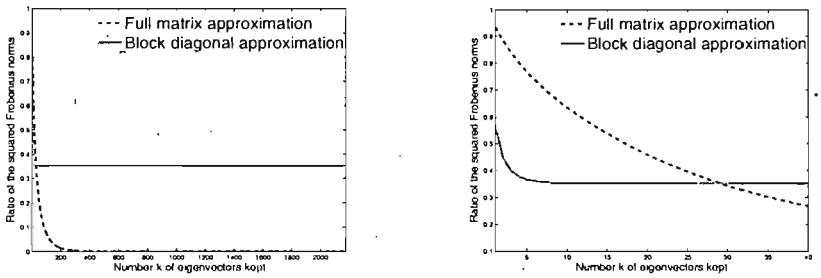


(a) Gradient stochastique (b) TONGA (c) TONGA - zoom

FIG. 5.1 – Valeur absolue de la corrélation entre les gradients stochastiques standards après un passage sur l'ensemble d'entraînement dans un réseau de neurones de cinquantes unités cachées entraîné sur LETTER, en optimisant selon le gradient stochastique (gauche) et selon le gradient naturel (centre et droite).

Pour tenter de quantifier la ressemblance entre la matrice de covariance des gradients et ses approximations, on peut mesurer le ratio des normes de Frobenius suivant : $\frac{\|C-\hat{C}\|_F^2}{\|C\|_F^2}$, avec $\hat{C}$ l'approximation pleine ou bloc diagonale de TONGA. Cette comparaison est présentée à la figure 5.2, en fonction du nombre de vecteurs propres k utilisé dans l'approximation. On remarque que l'approximation des blocs mène à un ratio de 0.35 (mentionnons aux fins de la comparaison que l'approximation diagonale donne un ratio de 0.8) alors qu'on ne prend en compte que $1,7\%$ des

éléments de la matrice (82176 sur les 4734976 éléments). Ce ratio est quasiment atteint avec $k = 6$. On remarque également que pour $k < 30$ l'approximation bloc diagonale est supérieure à l'approximation pleine, pour une même valeur de k. Au regard de ce critère, l'approximation bloc diagonale est donc une solution offrant un très bon rapport qualité-coût pour de petites valeurs de k. En effet, le rang de l'approximation bloc diagonale de la matrice de covariance est en fait de k fois le nombre de blocs.



(a) Vue d'ensemble (b) Aggrandissement

FIG. 5.2 – Qualité de l'approximation $\hat{C}$ de la covariance C en fonction du nombre de vecteurs propres (k) utilisés dans l'approximation, mesurée selon le ratio des normes de Frobenius $\frac{\|C-\hat{C}\|_F^2}{\|C\|_F^2}$, pour l'approximation pleine ou bloc diagonale de TONGA.

En pratique cependant, on ne doit pas comparer les deux approximations sur la base d'un même k, puisque la complexité des deux approches n'est pas la même : la version bloc incorpore le nombre de bloc dans sa complexité. Vaut-il donc mieux faire un calcul plus coûteux de gradient naturel ou plutôt plusieurs calculs moins coûteux ? En termes de complexité du calcul du gradient naturel, avec k le rang de l'approximation, b le nombre d'étapes entre les réévaluations et n le nombres de blocs, on compare $\mathcal{O}(p_{plein}(k_{plein} + b_{plein}) + (k_{plein} + b_{plein})^3)$ et $\mathcal{O}(np_{bloc}(k_{bloc} + b_{bloc}) + n(k_{bloc} + b_{bloc})^3)$.

5.4 Travaux futurs

L'algorithme TONGA tel que présenté ici est un accomplissement réel : la technique est en mesure de puiser dans la force des algorithmes classiques de second ordre en demeurant applicable dans le contexte des problèmes de grande taille. Cependant, la technique mérite d'être améliorée. Premièrement, le fait que la technique possède un aussi grand nombre d'hyperparamètres est un obstacle à son utilisation. L'ajustement automatique de certains hyperparamètres est envisageable et il faudrait donc y travailler. Ensuite, la technique ne semble pas fournir tous les gains qu'on est en mesure d'espérer. Le problème se situe au niveau du facteur d'apprentissage appliqué dans les directions n'obtenant pas un facteur spécifique. La solution passe ici encore par l'amélioration de l'ajustement des hyperparamètres. Celle amélioration est donc critique. De manière plus générale, il serait informatif d'avoir une meilleure idée de la qualité des approximations de la matrice de covariance obtenues. Le ratio des normes de Frobenius est certes un indicateur, mais il ne nous informe pas sur les écarts entre valeurs et vecteurs propres. Au niveau pratique, on pourrait envisager des utilisation partielles de TONGA, en appliquant seulement la technique sur les unités de sorties (réputées plus mal conditionnées) dans le but d'améliorer le rapport du coût au gain.

CHAPITRE 6

CONCLUSION

Notre objectif était d'accéder à la puissance des algorithmes classiques de second ordre tout en conservant une complexité applicable aux problèmes de grande taille. À cette fin, nous avons présenté TONGA un nouvel algorithme de descente de gradient basé sur le gradient naturel. TONGA opère en s'attaquant directement à la dégénérescence des problèmes : il s'agit de fournir un facteur d'apprentissage spécifique dans quelques directions de l'espace des paramètres, de façon à pouvoir appliquer un facteur d'apprentissage général qui fonctionne bien dans les directions restantes. TONGA est en mesure de faire ceci à faible coût en s'intéressant au sous-espace de l'espace des paramètres dans lequel est concentré la variance. Notre évaluation sur trois problèmes classiques montre que TONGA est en mesure de soutenir la comparaison avec la technique communément utilisée dans ce cadre, le gradient stochastique. Dans l'une de nos expériences, TONGA offre même une performance nettement supérieure.

Bien que la version actuelle soit une réussite en soit, il y a une nécessité d'améliorer l'ajustement des hyperparamètres. Cette amélioration, qui passe par l'automatisation de certains de ces ajustements, permettra de faciliter l'utilisation de TONGA, mais permet aussi d'envisager des performances supérieures.

BIBLIOGRAPHIE

Shun-ichi Amari. Natural gradient works efficiently in learning. *Neural Computation*, 10(2):251–276, 1998. URL `citeseer.ist.psu.edu/article/amari98natural.html`.

Shun-ichi Amari, Hyeyoung Park et Kenji Fukumizu. Adaptive method of realizing natural gradient learning for multilayer perceptrons. *Neural Computation*, 12(6): 1399–1409, 2000. URL `citeseer.ist.psu.edu/amari98adaptive.html`.

Y. Bengio, P. Lamblin, D. Popovici et H. Larochelle. Greedy layer-wise training of deep networks. Dans B. Schölkopf, J. Platt et T. Hoffman, éditeurs, *Advances in Neural Information Processing Systems 19*, pages 153–160. MIT Press, 2007.

Yoshua Bengio et Yann Le Cun. Scaling learning algorithms towards AI. Dans L. Bottou, O. Chapelle, D. DeCoste et J. Weston, éditeurs, *Large Scale Kernel Machines*. MIT Press, 2007.

Christopher Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, London, UK, 1995.

Christopher Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

Leon Bottou et Yann LeCun. Large-scale on-line learning. Dans *Advances in Neural Information Processing Systems 15*. MIT Press, 2004.

R. Collobert. *Large Scale Machine Learning*. Thèse de doctorat, Université de Paris VI, LIP6, 2004.

R.O. Duda, P.E. Hart et D.G. Stork. *Pattern Classification, Second Edition*. Wiley and Sons, New York, 2001.

Tom Heskes. On natural learning and pruning in multilayered perceptrons. *Neural Computation*, 12(4):881–901, 2000.

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

Nicolas Le Roux, Pierre-Antoine Manzagol et Yoshua Bengio. Topmoumoute online natural gradient algorithm. Dans *Advances in Neural Information Processing Systems 20*. MIT Press, Cambridge, MA, 2008.

Y. LeCun, L. Bottou, G.B. Orr et K.-R. Müller. Efficient backprop. Dans G.B. Orr et K.-R. Müller, éditeurs, *Neural Networks : Tricks of the Trade*, pages 9–50. Springer, 1998.

Yann LeCun et Corinna Cortes. The mnist database of handwritten digits. http ://yann.lecun.com/exdb/mnist/.

J. Nocedal et S. Wright. *Numerical Optimization*. Springer, 2006.

K. B. Petersen et M. S. Pedersen. The matrix cookbook, feb 2006. Version 20051003.

RCMP. Counterfeiting and credit card fraud. http ://www.rcmp-grc.gc.ca/scams/ccandpc_e.htm, 2007.

B. Schölkopf, A. Smola et K.-R. Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, 10:1299–1319, 1998.

B. Schölkopf et A.J. Smola. *Learning with Kernels : Support Vector Machines, Regularization, Optimization and Beyond*. MIT Press, Cambridge, MA, 2002.

N. N. Schraudolph. Local gain adaptation in stochastic gradient descent. Dans *Proceedings of the 9th International Conference on Artificial Neural Networks*, pages 569–574, 1999.

Nicol N. Schraudolph. Fast curvature matrix-vector products for second-order gradient descent. *Neural Computation*, 14(7):1723–1738, 2002.

Nicol N. Schraudolph et Thore Graepel. Combining conjugate direction methods with stochastic approximation of gradients. Dans Christopher M. Bishop et Brendan J. Frey, éditeurs, *Proc. 9th Intl. Workshop Artificial Intelligence and Statistics (AIstats)*, pages 7–13, Key West, Florida, 2003. Society for Artificial Intelligence and Statistics. ISBN 0-9727358-0-1.

Nicol N. Schraudolph, Jin Yu et Simon Günter. A stochastic quasi-Newton method for online convex optimization. Dans *Proc. 11th Intl. Conf. Artificial Intelligence and Statistics (AIstats)*, pages 433–440, San Juan, Puerto Rico, 2007. Society for Artificial Intelligence and Statistics. ISBN 0-9727358-2-8.

G. Tesauro. Practical issues in temporal difference learning. *Machine Learning*, 8: 257–277, 1992.

H. H. Yang et S. Amari. Natural gradient descent for training multi-layer perceptrons, 1997. URL `citeseer.ist.psu.edu/hua96natural.html`.

Howard Hua Yang et S. Amari. Complexity issues in natural gradient descent method for training multi-layer perceptrons. *Neural Computation*, 10(8):2137–2157, 1998. URL `citeseer.ist.psu.edu/91462.html`.

Annexe I

Hyperparametres

Algorithme	Valeur des hyperparamètres
Gradient stochastique	$mbs = 1$, $lr = 1e-2$, $dc = 1e-6$, $L1 = 0.0$
TONGA (variante bloc avec adaptation de λ)	$mbs = 500$, $lr = 1e{-}5$, $dc = 1e{-}6$, $L1 = 0.0$, $k = 2$, $cmbs = 15$, $\lambda_{initial} = 1e-2$, $\lambda_{min} = 1e-2$

TAB. I.1 – Valeur des hyperparamètres obtenus par validation pour chacun des algorithmes d'apprentissage sur LETTER, avec un réseau de neurones possédant une couche de 100 unités cachées. Pour TONGA, seule la variante offrant la meilleure performance en validation est présentée.

Algorithme	Valeur des hyperparamètres
Gradient stochastique	$mbs = 1$, $lr = 5e{-}2$, $dc = 1e{-}5$, $L1 = 1e{-}4$
TONGA (variante bloc avec adaptation de λ)	$mbs = 500$, $lr = 1e{-}5$, $dc = 1e{-}6$, $L1 = 0.0$, $k = 6$, $cmbs = 30$, $\lambda_{initial} = 1e-2$, $\lambda_{min} = 1e-4$

TAB. I.2 – Valeur des hyperparamètres obtenus par validation pour chacun des algorithmes d'apprentissage sur USPS, avec un réseau de neurones possédant une couche de 300 unités cachées. Pour TONGA, seule la variante offrant la meilleure performance en validation est présentée.

Algorithme	Valeur des hyperparamètres
Gradient stochastique	$mbs = 200$, $lr = 1e-2$, $dc = 0$, $L1 = 1e-6$
TONGA (variante bloc avec adaptation de λ)	$mbs = 500$, $lr = 1e{-}5$, $dc = 1e{-}5$, $L1 = 0.0$, $k = 5$, $cmbs = 15$, $\lambda_i = 1e-2$, $\lambda_m = 1e-5$

TAB. I.3 – Valeur des hyperparamètres obtenus par validation pour chacun des algorithmes d'apprentissage sur MNIST, avec un réseau de neurones possédant une couche de 800 unités cachées. Pour TONGA, seule la variante offrant la meilleure performance en validation est présentée.

Algorithme	Valeur des hyperparamètres
Gradient stochastique	$mbs = 1000$, $lr = 5e-2$, $dc = 1e-7$, $L1 = 0$
TONGA (variante bloc sans adaptation de λ)	$mbs = 1000$, $lr = 1e-6$, $dc = 1e-6$, $L1 = 0$, $k = 2$, $cmbs = 15$, $\lambda_i = 1e-3$

TAB. I.4 – Valeur des hyperparamètres obtenus par validation pour chacun des algorithmes d'apprentissage sur LETTER, avec un réseau de neurones possédant deux couches cachées de 100 et 50 unités respectivement. Pour TONGA, seule la variante offrant la meilleure performance en validation est présentée.

Algorithme	Valeur des hyperparamètres
TONGA variante sans adaptation de λ	$mbs = 500$, $lr = 1e-5$, $dc = 0.0$, $k = 3$, $cmbs = 15$, $\lambda_i = 1e-2$
TONGA variante avec adaptation de λ	$mbs = 500$, $lr = 1e-4$, $dc = 1e-7$, $k = 3$, $cmbs = 30$, $\lambda_i = 1e-2$, $\lambda_m = 1e-2$
TONGA variante bloc sans adaptation de λ	$mbs = 100$, $lr = 1e-5$, $dc = 1e-5$, $k = 2$, $cmbs = 15$, $\lambda_i = 1e-2$
TONGA variante bloc avec adaptation de λ	$mbs = 500$, $lr = 1e-5$, $dc = 1e-6$, $k = 2$, $cmbs = 15$, $\lambda_i = 1e-2$, $\lambda_m = 1e-2$

TAB. I.5 – Valeur des hyperparamètres obtenus par validation pour chacune des variantes de TONGA sur LETTER, avec un réseau de neurones possédant une couche de 100 unités cachées. La taille des lots a été fixée à $mbs = 500$, et la régularisation $L1$ à 0.

Algorithme	Valeur des hyperparamètres
TONGA variante sans adaptation de λ	$lr = 1e-5$, $dc = 1e-7$, $k = 20$, $cmbs = 30$, $\lambda_i = 1e-3$
TONGA variante avec adaptation de λ	$lr = 1e-4$, $dc = 1e-7$, $k = 10$, $cmbs = 30$, $\lambda_i = 1e-2$, $\lambda_m = 1e-3$
TONGA variante bloc sans adaptation de λ	$lr = 1e-7$, $dc = 1e-7$, $k = 5$, $cmbs = 30$, $\lambda_i = 1e-5$
TONGA variante bloc avec adaptation de λ	$lr = 1e-5$, $dc = 1e-5$, $k = 5$, $cmbs = 15$, $\lambda_i = 1e-2$, $\lambda_m = 1e-5$

TAB. I.6 – Valeur des hyperparamètres obtenus par validation pour chacune des variantes de TONGA sur MNIST, avec un réseau de neurones possédant une couche de 800 unités cachées. La taille des lots a été fixée à $mbs = 500$, et la régularisation $L1$ à 0.