Titre: Title:	Optimisation de réseaux de neurones profonds pour déploiement sur matériel incertain à faible consommation d'énergie
Auteur: Author:	Sébastien Henwood
Date:	2024
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Henwood, S. (2024). Optimisation de réseaux de neurones profonds pour déploiement sur matériel incertain à faible consommation d'énergie [Thèse de doctorat, Polytechnique Montréal]. PolyPublie. https://publications.polymtl.ca/59206/

URL de PolyPublie: PolyPublie URL:	https://publications.polymtl.ca/59206/
Directeurs de recherche: Advisors:	François Leduc-Primeau
Programme: Program:	Génie électrique

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Optimisation de réseaux de neurones profonds pour déploiement sur matériel incertain à faible consommation d'énergie

SÉBASTIEN HENWOOD
Département de génie électrique

Thèse présentée en vue de l'obtention du diplôme de *Philosophiæ Doctor*
Génie électrique

Juillet 2024

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Cette thèse intitulée :

Optimisation de réseaux de neurones profonds pour déploiement sur matériel incertain à faible consommation d'énergie

présentée par **Sébastien HENWOOD**
en vue de l'obtention du diplôme de *Philosophiæ Doctor*
a été dûment acceptée par le jury d'examen constitué de :

Jean-Pierre DAVID, président
François LEDUC-PRIMEAU, membre et directeur de recherche
James-A. GOULET, membre
Mounir BOUKADOUM, membre externe

DÉDICACE

Audrey,
on voit le bout :)

REMERCIEMENTS

Je souhaiterais remercier chaleureusement mon directeur de thèse, François Leduc-Primeau pour son support et ses conseils lors de mon aventure doctorale. Son accompagnement sur ces quatre années, les discussions et critiques portant sur les idées poursuivies et mon travail en découlant m'ont permis d'améliorer la qualité des travaux présentés dans cette thèse. Ces années furent riches d'enseignements et m'ont permis d'élargir mon horizon à la fois technique et humain et pour cela, je lui en suis reconnaissant. Je souhaiterais également remercier le Prof. Yvon Savaria pour ses conseils sur mon sujet de thèse lors de nos échanges.

Par ailleurs, je souhaiterais remercier mon ami et collègue Jonathan Kern avec qui j'ai partagé les joies de l'expérience doctorale. Je salue notamment sa force de proposition, inégalée en acronymes techniques, ainsi que ses légendaires capacités culinaires.

Je remercie également Gonçalo Mordido, chercheur postdoctoral à Polytechnique et au MILA pour notre collaboration lors de l'écriture du chapitre 4 de cette thèse. Son expertise et ses idées ont rehaussé de manière marquée la qualité des travaux produits.

J'adresse ensuite mes remerciements au Pr. Jean-Pierre David, qui a accepté de présider mon jury de thèse et celui de mon examen de synthèse. Par là même, je remercie les Pr. James Goulet et Mounir Boukadoum pour leur présence au sein de mon jury de thèse et ainsi de me faire bénéficier de leur expertise.

Je souhaite exprimer ma gratitude à l'Institut de Valorisation des Données (IVADO) pour m'avoir octroyé une bourse d'excellence au doctorat qui m'a permis de me concentrer sur mes travaux doctoraux lors de ces quatre années, ainsi que le ResmiQ pour les aides financières que j'ai reçues de leur part en entamant mon projet doctoral.

Pour ce long voyage de reprise d'études, j'exprime toute ma reconnaissance aux amis, familles, camarades, maîtres de stage, professeurs, collègues et directeurs qui m'ont motivé, fait grandir et m'ont permis d'en arriver là. Merci.

Je remercie particulièrement Anaïs Guitton pour avoir allumé l'étincelle m'ayant conduit jusqu'ici. Je remercie également l'entièreté du corps professoral du département QLIO de l'IUT de Lorient qui m'ont donné cette chance de reprendre mes études tout en m'y donnant goût.

Enfin, merci Audrey, pour ton affection (++), ton support (+) et ta patience inexorable (pas sûr) dans ce projet.

RÉSUMÉ

Nous sommes en 2024. Après une décennie enfiévrée de progrès en intelligence artificielle avec l'avènement des réseaux de neurones profonds, cette technologie tend à devenir omniprésente dans nos quotidiens. Gourmands en énergie, ces réseaux profonds participent à la croissance soutenue du poids environnemental du secteur informatique. Or, nous sommes dans un contexte dans lequel l'énergie se raréfie. En réponse à cette situation, on assiste au développement d'appareils à très faible consommation d'énergie. En particulier, les transactions mémoires constituent une large proportion de la consommation énergétique d'un système électronique et deviennent un élément critique lors du développement de systèmes numériques. Ainsi des techniques et des technologies novatrices apparaissent pour proposer des mémoires plus économes en énergie, dont certaines proposent d'effectuer un compromis contrôlable entre la fiabilité des éléments stockés et la consommation d'énergie de la mémoire.

Dans cette thèse, nous nous intéressons à l'obtention de réseaux de neurones profonds robustes, c'est-à-dire capables de proposer une performance intéressante en fonctionnant sur ces mémoires de fiabilité contrôlée. Par ailleurs, nous cherchons à minimiser conjointement la consommation d'énergie atteinte en exploitant le compromis énergie-fiabilité. Nous nous intéressons en particulier aux mémoires de type SRAM, dont le compromis est contrôlé par la tension d'alimentation, et à la technologie émergente de memristors, permettant un calcul en mémoire analogique au prix d'un bruit sur les éléments stockés. Le compromis y est contrôlé en réduisant artificiellement la plage de programmation de la mémoire.

Dans un premier temps, nous proposons un modèle théorique énergie-fiabilité d'une mémoire SRAM en nous basant sur des observations expérimentales. Les bits qui y sont stockés ont une probabilité d'être lus à une valeur incorrecte, probabilité que nous lions à la consommation d'énergie de la mémoire. Par la suite, nous étudions la relation de chaque couche d'un réseau profond avec ce type de panne matérielle et mettons en avant la sensibilité individuelle de chaque couche. De plus, nous montrons l'effet de la simulation de ces pannes lors de l'entraînement du réseau sur les paramètres de ce dernier. Enfin, nous proposons l'algorithme LaNMax pour optimiser conjointement le réseau de neurone et la fiabilité de chaque couche tout en minimisant la consommation d'énergie de la mémoire. Les résultats expérimentaux sur un réseau binarisé démontrent une réduction de la consommation énergétique de la mémoire d'un facteur trois tout en conservant une précision comparable sur la tâche de classification d'images CIFAR-10.

Dans un second temps, nous étudions le champ des algorithmes d'optimisation de réseau

de neurones dits *sharpness aware* et leur impact sur la robustesse des réseaux au bruit matériel d'un memristor servant à stocker leurs paramètres. Nous y présentons SAMSON, un algorithme dérivé de l'état de l'art, visant à bonifier la robustesse des réseaux entraînés par son biais. Nous démontrons l'intérêt de ces algorithmes *sharpness aware* pour la robustesse des réseaux sur une large série d'architectures de réseaux et de tâches, tout en soulignant l'amélioration permise par l'utilisation de SAMSON dans les cas fiables et de fiabilité limitée. Ces tests comprennent un modèle de bruit théorique et l'application de perturbations réalistes sur les paramètres du réseau.

Dans un troisième temps, nous étudions un modèle d'erreur détaillé de memristors comprenant un compromis énergie-fiabilité contrôlé par la plage de programmation utile. Le processus de transfert d'un réseau de neurone profond sur ces memristors y est détaillé, ainsi que des méthodes permettant de calculer la puissance dissipée par la mémoire et l'erreur sur la sortie du memristor comparativement à une sortie en environnement fiable. Ces approches sont implémentées dans une bibliothèque logicielle qui est ensuite utilisée pour optimiser conjointement les paramètres architecturaux du réseau et la plage de programmation utile. Les résultats expérimentaux obtenus sur la tâche de classification d'images ImageNet démontrent l'intérêt d'optimiser ces deux variables pour obtenir un compromis énergie-performance du réseau intéressant en régime de faible énergie.

Pour résumer, cette thèse présente des outils et des algorithmes permettant de préparer des réseaux de neurones profonds proposant une robustesse améliorée à des mémoires de fiabilité contrôlée, tout en optimisant l'énergie allouée à ces mémoires.

ABSTRACT

The year is 2024. Following a decade of buzzing developments in artificial intelligence with the emergence of deep neural networks, this technology usage is gradually becoming ubiquitous. Voracious energy consumers, these deep networks actively participate in the yearly growth of the IT sector's environmental impact. However, the global demand for energy tends to exceed the available capacity. This precarious situation is spurring the development of low-energy devices. Notably, memory transactions constitute a significant part of the energy consumption of an electronic system. Nonetheless, emerging memory technologies aiming to alleviate this issue were proposed. Moreover, some of these innovative memories allow us to trade the reliability of the stored elements to reduce memory energy consumption.

In this thesis, we study approaches to achieve robust deep neural networks that can deliver a compelling performance while operating on this memory with controlled reliability. Furthermore, we aim to jointly minimize the energy consumption of a deep neural network by capitalizing on this energy-reliability trade-off. Notably, we study the use of a voltage-scaled SRAM operating in the near-threshold regime on the one hand, and on the other hand, the emerging technology of memristors crossbars allowing low power processing in memory in the analog domain at the cost of hardware noise on the stored elements. For these memristors crossbar, the energy-reliability trade-off is controlled by artificially reducing the valid programmable range of the memory.

We first study an energy-reliability model for a voltage-scaled SRAM based on experimental data to do so. Bits stored on this memory may be read incorrectly, with a probability linked to the energy consumption of the memory. Later, we study the relation of a deep neural network's layers with this type of hardware fault, highlighting each layer's distinct sensitivity. Moreover, we assert the effect of simulating these failures during the training phase of the neural network and the impact on the final parameters. Finally, we propose the algorithm LaNMax to jointly optimize the neural network and the reliability of each of its layers while minimizing the energy allocated to the memory. Experimental results from a binary neural network show a threefold reduction in energy consumption while retaining a comparable accuracy on the image classification task CIFAR-10.

In the second phase, we interest ourselves in the sharpness-aware optimization of neural networks and the impact of this approach on the achieved robustness of a neural network whose parameters are subject to the noise of memristor-enabled storage. We propose SAMSON, a sharpness-aware optimization derived from state-of-the-art technology that aims to bolster

the robustness of a neural network. We experimentally demonstrate the improved robustness of neural network training with sharpness-aware algorithms on various neural network architectures and tasks. It is also shown that SAMSON's is beneficial in both reliable and limited reliability scenarios. Those experimental results include both a theoretical noise model and realistic hardware sampled perturbations on the parameters of the neural network.

In the third and last step, we study a precise energy-reliability failure model for a memristor crossbar whose programming range is artificially constrained. The programming step of neural networks, the power dissipation, and the error on the output of the neural network with respect to a reliable memory device are presented. These components are implemented in a software library and then used to jointly optimize the neural network architectural parameters and the useful programming range. Experimental results on the ImageNet image classification task confirm the added interest in optimizing them to reach an improved energy-performance trade-off in the low-energy domain.

In summary, this thesis presents tools and algorithms for implementing deep neural networks with improved robustness to memories and controlled reliability while optimizing the energy allocated to these memories.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES SIGLES ET ABRÉVIATIONS

AIHwKit	Artificial Intelligence Hardware Kit
ASAM	Adaptive Sharpness Aware Minimization
BC	Binary Connect
BSC	Binary Symmetric Channel
DNN	Deep Neural Network
FPGA	Field Programmable Gate Array
GPGPU	General Purpose Graphic Processing Unit
LDPC	Low Density Parity Check
LUT	Lookup Table
MVM	Matrix-vector Multiplications
NAS	Neural Architecture Search
NTC	Near Threshold Computing
PIM	Processing In Memory
SAM	Sharpness Aware Minimization
SAMSON	Sharpness Aware Minimization Scaled by Outlier Normalization
SGD	Stochastic Gradient Descent
SoC	System on a Chip
SRAM	Static Random Access Memory
SVD	Singular Value Decomposition
UL	Unrolled Linear (Convolution Implementation)
UR	Unfold Repeat (Convolution Implementation)

LISTE DES ANNEXES

CHAPITRE 1 INTRODUCTION

1.1 Les enjeux électriques de l'apprentissage profond

Le domaine de l'apprentissage automatique a évolué significativement ces dernières années, notamment grâce aux développements effervescents des réseaux de neurones profonds. Ces derniers démontrent une progression fulgurante dans de nombreux domaines d'application, au prix d'une complexité croissante et de besoins en ressources toujours plus importants. Ce constat est d'autant plus frappant quand on compare l'évolution des besoins en ressources pour entraîner un réseau de neurones à l'état de l'art d'une part et une courbe suivant la loi de Moore d'autre part. Dans le domaine des semi-conducteurs, la loi de Moore est un célèbre étalon de la progression technologique envisageable : à savoir un doublement du nombre de transistors dans un circuit intégré tous les deux ans. Le verdict est frappant : les besoins pour les réseaux de neurones observent une cadence d'augmentation effrénée dont la tendance dépasse largement celle d'une loi de Moore. Par exemple, l'évolution du nombre de paramètres dans les réseaux à l'état de l'art est comparée à la mémoire disponible sur le matériel de calcul dédié Figure 1.1 : on y observe la croissance effrénée de la taille des modèles de réseaux profonds dédiés au traitement du langage en rouge, dépassant en 2020 la capacité des accélérateurs de réseaux profonds en vert. Ces derniers observent une croissance plus mesurée de leur capacité au fil des ans. Couplé au ralentissement de la loi de Moore et à la fin de l'échelle de Dennard [1], les accélérateurs de réseaux de neurones seront bientôt fortement limités par leur seule consommation d'énergie.

Ces réseaux de neurones sont constitués de nombreux nœuds interconnectés nous permettant de transformer un signal d'entrée (images, texte, son, etc.) en un signal de sortie désiré (que représente l'image ? le son est-il un mot connu ?). Quand ces opérations sont nombreuses, on parle alors de réseau de neurones profond (en anglais, *Deep neural network*, DNN). Chacune de ces opérations peut faire appel à des paramètres qu'il est alors nécessaire d'optimiser : ce qu'on appelle dans le jargon une phase d'« entraînement ». Une fois cet entraînement terminé, le réseau est déployé sur un matériel final pour une phase d'utilisation où il sera amené à faire des « inférences » à répétition. Ces inférences sont le simple enchaînement des opérations du réseau pour une ou plusieurs entrées données. Il est important de noter que l'entraînement et l'inférence n'ont souvent pas le même contexte : on préférera effectuer l'entraînement sur un serveur de calcul à forte puissance pour réduire les délais ; tandis que le réseau ainsi entraîné pourra être déployé sur pléthore de matériels souvent moins puissants pour la phase d'inférence. Par exemple, il n'est pas rare de déployer des réseaux neuronaux

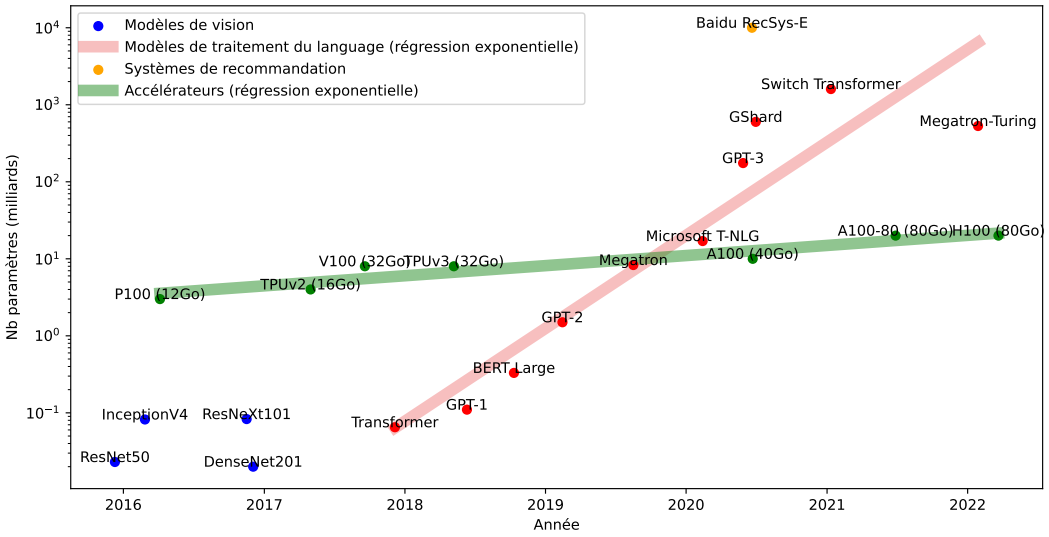


FIGURE 1.1 Croissance du nombre de paramètres pour différentes tâches (en bleu, en rouge, en orange) contre mémoire disponible sur divers accélérateurs (en vert) en se basant sur un format FP32 en fonction de l'année. Ce choix de format n'a pas d'impact sur la tendance observée. Adapté de [2].

sur des téléphones portables ou des ordinateurs portatifs dont le matériel s'adapte à ce nouvel usage, ou encore sur des dispositifs *Internet-of-Things* (IoT).

On peut ainsi dresser le portrait robot d'un DNN en répondant aux questions suivantes :

— quel enchaînement d'opération est effectué, c'est-à-dire quelle **architecture** ou encore **modèle** de DNN est utilisé,
— quel(s) **jeu(x) de données** a (ont) été utilisé(s) pour son **entraînement**,
— et quel algorithme a été utilisé pour son **entraînement**.

Or, si ce portrait robot évolue au fil des innovations techniques, il n'en reste pas moins que les réseaux à l'état de l'art requièrent une puissance de calcul significative pour fonctionner. Cette demande de puissance mène à se questionner sur les potentiels impacts environnementaux de l'apprentissage automatique et sa contribution au dérèglement climatique. Sous un angle moins vertueux, l'intérêt d'une technologie d'apprentissage profond à faible consommation d'énergie est notamment économique. Si l'on concentre sur le seul marché québécois de l'électricité, Hydro-Québec prévoyait en 2019 dans son « Portrait des ressources énergétiques » [3] un essoufflement des capacités d'approvisionnements en énergie dès 2026 face à la demande croissante. En langage économique, cette demande en augmentation face à une offre stagnante aura pour effet induit l'augmentation du prix de l'électricité. À l'échelle mon-

diale, l'énergie consommée est en augmentation constante avec l'accroissement du niveau de vie. D'autre part, les ressources énergétiques courantes sont menacées à moyen terme : les énergies fossiles se raréfient avec le passage du pic mondial de production de pétrole conventionnel dès 2008, mais également avec le retrait progressif de ces sources d'énergies dans certains pays pour tenter de respecter les engagements pris lors de Conférences des Parties (plus connues sous l'acronyme COP). Les énergies renouvelables requièrent des investissements massifs et sont limitées par leur faible densité énergétique. Les énergies nucléaires et hydrauliques ont des défis qui leur sont propres. En somme, le constat d'essouflement des capacités de production d'énergie dressé par Hydro-Québec peut être extrapolé au reste du monde.

Dans ce contexte, les solutions d'apprentissage automatique efficientes en énergie peuvent apparaître fort attrayantes. Par exemple, lors de calculs à large échelle en centre de données, mobilisant possiblement plusieurs accélérateurs de réseaux de neurones. Mais cette quête d'efficience peut également ouvrir de nouveaux champs applicatifs pour les réseaux neuronaux. Dans le domaine des applications embarquées, une diminution, même minime, de l'empreinte d'un réseau profond peut décider ou non de sa capacité à être déployé en conditions réelles. Un exemple frappant est à trouver dans le domaine de la vision assistée par ordinateur dont les applications de l'apprentissage profond couvrent désormais un large champ, de l'intelligence de terrain pour l'agriculture à la génération d'images sur téléphones.

Pour résumer, la consommation d'énergie d'un réseau de neurones profond est une faiblesse majeure de l'apprentissage automatique, autant d'un point de vue de l'impact environnemental que de la faisabilité technique et économique. Le développement d'algorithmes et de matériels efficients en énergie est crucial pour d'une part atténuer les externalités négatives de l'apprentissage machine sur l'environnement tout en rendant la technologie accessible à un plus large champ d'applications.

Or, d'un point de vue matériel, les transactions mémoire induisent une large proportion de la consommation énergétique : ce phénomène est appelé « mur de la mémoire » (*« memory wall »*) [4, 5]. Sur certains accélérateurs de réseaux de neurones, la puissance dissipée par le stockage des paramètres du DNN se situe entre 31% et 59% [6] de la puissance totale. En règle générale, sur du matériel conventionnel avec architecture Von Neumann, la puissance dissipée pour la réalisation d'une opération sera dominée par les transactions mémoires [7]. Il est donc primordial d'aborder le problème de la consommation énergétique d'un réseau de neurones sous l'angle de l'énergie allouée à la mémoire : c'est l'objectif poursuivi dans cette thèse.

1.2 Approche proposée

En réponse à ces besoins et contraintes, de nombreux champs d'études connexes ont émergés pour proposer des algorithmes et matériels d'apprentissage profond à faible consommation d'énergie. Entre autres méthodes ayant fait leurs preuves, nous pouvons citer l'élagage (*pruning*) des paramètres, la quantification des paramètres et des activations ou encore la compression des paramètres qui nous permettent de réduire la consommation d'énergie du réseau neuronal dans sa phase d'inférence avec un sacrifice variable et surtout contrôlable de la performance initiale du réseau à l'issue de sa phase d'entraînement. Ces modifications ont trait à la définition même du réseau de neurones en tentant de réduire son empreinte mémoire.

D'autre part, il est possible de modifier l'implémentation matérielle de l'accélérateur de DNN. Sur matériel numérique conventionnel, la tension d'alimentation peut par exemple être réduite de sorte à opérer proche du seuil de tension (*Near-threshold Computing*, NTC). Des technologies émergentes peuvent également être envisagées comme l'opto-électronique ou encore le calcul en mémoire (*Processing In Memory*, PIM) basé par exemple sur l'utilisation de memristors.

Enfin, la co-conception du réseau de neurones et du matériel est une approche mixte visant à tirer parti au maximum des économies d'énergies matérielles tout en conservant une performance du réseau acceptable. L'exemple de la quantification ci-dessus en est un bon représentant : quantifier les paramètres va d'une part réduire la quantité totale de bits nécessaires pour représenter le DNN, mais également changer le matériel utilisé pour calculer le DNN. On pourra alors tirer parti de calculs à virgule fixe, bien plus économes en énergie et en aire de silicium que les calculs à virgule flottante utilisés par défaut.

Dans cette thèse, nous nous intéressons à la réduction de la consommation d'énergie de réseaux de neurones profonds par le biais de mémoires à faible consommation d'énergie. Nous nous concentrons sur deux modèles de mémoires : d'une part le standard actuel de type *static random-access memories*, SRAM opéré sous la tension nominale [8] avec une représentation numérique des nombres ; et d'autre part la technologie émergente de calcul en mémoire par memristors [9] avec une représentation analogique. Le gain d'énergie de l'approche SRAM est à trouver dans la relation entre puissance, résistance et intensité : si l'intensité diminue, des gains quadratiques seront observables sur la puissance. Dans notre cas, on considère la tension de consigne de la SRAM comme une variable libre qui peut être exploitée lors de la conception du DNN. Pour les memristors, s'agissant d'une technologie de calcul en mémoire le gain énergétique est double : d'une part une réduction des transactions mémoire en évitant

des aller-retours entre la mémoire et les unités de calcul, et d'autre part en effectuant les lourds calculs matriciels sur une technologie analogique peu gourmande en espace, temps et énergie.

Or, ces gains d'énergies se concrétisent dans les deux cas par un compromis énergie-fiabilité : la réduction d'énergie entraînera une augmentation de l'erreur dans les données stockées, et *in fine* les calculs délivrés. On peut alors parler de mémoires stochastiques, pour lesquelles un bruit aléatoire peut être modélisé afin de représenter l'erreur de stockage attendue.

Ce travail vise à étudier la relation entre les paramètres matériels de la mémoire et la conception du DNN et comment les exploiter au mieux pour obtenir un compromis énergie-fiabilité intéressant. On veut conserver un certain niveau de performance du DNN tout en proposant un niveau d'énergie moindre que dans une implémentation naïve, les DNN étant réputés pour avoir une certaine robustesse innée.

1.3 Structure de la thèse

Le reste de cette thèse est organisé comme suit :

- Premièrement, le chapitre 2 propose une revue de littérature sur les mécanismes de réduction d'énergie et sur le fonctionnement des modèles de mémoires utilisés. Nous considérons notamment les mémoires SRAM opérant à un niveau de tension proche du seuil et les memristors. Dans un second temps, nous nous intéressons aux mécanismes de tolérances aux erreurs matérielles, notamment aux capacités des réseaux de neurones à tolérer ces perturbations et aux mécanismes permettant de bonifier cette robustesse.
- Dans un second temps, le chapitre 3 procède à une présentation des méthodes de l'apprentissage profond (structure et entraînement d'un DNN, entraînement *hardware-aware* et *sharpness-aware*) et des modèles matériels utilisés dans les chapitres ultérieurs.
- Ensuite, dans le chapitre 4 nous nous intéressons aux mécanismes d'adaptation de réseaux de neurones profonds aux matériels fautifs pour un matériel numérique opérant en NTC. En nous basant sur un modèle de compromis énergie-fiabilité théorique, nous proposons l'algorithme LaNMax pour optimiser automatiquement ce compromis pour chaque couche d'un DNN lors de son entraînement. Nous étudions également l'impact de l'entraînement sur les propriétés du DNN ainsi obtenu. Enfin, nous réalisons des expériences pour juger des gains d'énergie atteignables.
- Dans le chapitre 5 la discussion se porte sur les mécanismes d'adaptation de réseaux de neurones profonds, mais cette fois-ci pour du matériel fautif analogique tel que

les memristors. Pour ce faire, nous nous tournons vers les techniques d'entraînement de DNN *sharpness-aware* et proposons SAMSON, une variante ayant pour objectif d'améliorer la robustesse des DNN ainsi obtenus. Nous expérimentons ensuite l'impact de SAMSON et des autres méthodes *sharpness-aware* sur la capacité d'un DNN à fonctionner en milieu fiable et de fiabilité limitée.

— Le chapitre 6 aborde finalement le problème de la co-optimisation des paramètres matériels et de l'architecture du DNN opérant sur memristors. Pour atteindre cet objectif, nous modélisons plus précisément le processus de transfert des paramètres d'un DNN sur un appareil de stockage memristif tout en estimant l'erreur et la puissance dissipée par les memristors. À l'aide de ces estimateurs, nous procédons à une optimisation de recherche d'architecture neuronale pour obtenir une architecture de DNN et certains paramètres matériels conjointement dans l'objectif d'améliorer le compromis énergie-performance atteint.

— Enfin, au chapitre 7 les différentes contributions de cette thèse sont résumées, une conclusion est proposée et les axes de travail futurs sont discutés.

1.4 Contributions et liste des publications

Les contributions scientifiques écrites durant cette thèse de doctorat sont listées dans le reste de cette section. Dans un premier temps, cette thèse propose une méthode systématique pour optimiser le compromis énergie-fiabilité d'une mémoire SRAM opérant sous le seuil de tension en différenciant la tension allouée à chaque couche d'un réseau de neurones profond. Ce travail a abouti à la publication suivante

— **S. Henwood**, F. Leduc-Primeau and Y. Savaria, « Layerwise Noise Maximisation to Train Low-Energy Deep Neural Networks, » 2020 2nd IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS), Genova, Italy, 2020, pp. 271-275, doi : 10.1109/AICAS48895.2020.9073854.

Une adaptation en français de cette publication constitue le chapitre 4 de cette thèse.

Dans un deuxième temps, nous nous sommes intéressés à la conception d'une méthode permettant de préparer un réseau de neurones profond aux memristors, dispositifs de calcul analogique peu gourmands en énergie. Ce travail a abouti à la publication suivante

— **S. Henwood**, G. Mordido, Y. Savaria and F. Leduc-Primeau, « Sharpness-Aware Minimization Scaled by Outlier Normalization for Improving Robustness on Noisy DNN Accelerators », Article soumis à IEEE Transactions on Neural Networks and Learning Systems et accepté pour présentation à la conférence 2024 Asilomar Conference on Signals, Systems, and Computers (ACSSC 2024), Pacific Grove, CA, USA.

Une adaptation en français de cette publication constitue le chapitre 5 de cette thèse.

Enfin, nous avons développé un cadre théorique pour calculer l'erreur introduite par l'utilisation d'un memristor, ainsi que l'usage d'énergie associé. En utilisant cet outil, une méthode permettant de générer conjointement des architectures de réseaux de neurones profonds et des configurations matérielles fut proposée dans l'objectif d'optimiser l'énergie allouée au réseau tout en maximisant sa performance. Ce travail a abouti aux publications suivantes :

1. **S. Henwood**, Y. Savaria and F. Leduc-Primeau, « MemNAS : Supernet Neural Architecture Search for Memristor-based DNN Accelerators », Article accepté à IEEE Workshop on Signal Processing Systems (SiPS) 2024.
2. J. Kern, **S. Henwood**, G. Mordido, E. Dupraz, A. Aïssa-El-Bey, Y. Savaria et F. Leduc-Primeau, "Fast and accurate output error estimation for memristor-based deep neural networks," IEEE Transactions on Signal Processing, p. 1–13, 2024.
3. J. Kern, **S. Henwood** et al., « MemSE : Fast MSE Prediction for Noisy Memristor-Based DNN Accelerators », 2022 IEEE 4th International Conference on Artificial Intelligence Circuits and Systems (AICAS), Incheon, Korea, Republic of, 2022, pp. 62-65, doi : 10.1109/AICAS54282.2022.9869978.

Pour les publications 2 et 3, les contributions sont dans la part expérimentale (implémentation, test, gestion des expériences et reporting) et dans le processus créatif (idéation, conception des expériences, amélioration des modèles utilisés) de la contribution. Le contenu des publications 2 et 3 est utilisé pour décrire le modèle matériel et les modèles de calcul de la sortie d'un DNN sur memristor section 3.3.2, tandis que les résultats obtenus dans ces deux publications alimentent les expériences réalisées publication 1 et chapitre 6.

CHAPITRE 2 MATÉRIEL EFFICIENT EN ÉNERGIE ET ROBUSTESSE DES RÉSEAUX DE NEURONES PROFONDS

Dans ce chapitre, nous nous intéressons aux solutions existantes nous permettant d'installer un compromis énergie-fiabilité dans la mémoire servant à stocker les paramètres des DNN. Dans un premier temps, nous clarifions la terminologie adoptée dans cette revue de littérature et par la suite dans cette thèse.

Dans un second temps, nous nous intéressons à la partie matérielle. Nous avons retenu deux plateformes de calcul proposant un compromis énergie-fiabilité : le calcul proche du seuil de tension (NTC) sur mémoire SRAM traditionnelle, et le calcul en mémoire (PIM) par memristors. Ces deux technologies seront exposées en détails dans ce chapitre. La consommation énergétique et la performance du système peuvent également être modulés par le choix de quantification pour représenter les nombres, domaine qui clôturera la première partie de cette revue de littérature.

Nous nous intéresserons enfin à la composante fiabilité du compromis, tout d'abord sous un angle matériel au travers de mécanismes de correction d'erreurs. Par la suite, nous étudierons la robustesse des réseaux de neurones et les méthodes existantes permettant de la bonifier.

2.1 Terminologie

Nous clarifions d'abord les termes employés dans cette thèse. Tout d'abord, la fiabilité que nous utilisons comme l'équivalent du terme anglais *reliability*. Nous nous basons sur la définition proposée par [10] donnant la fiabilité comme « la capacité d'un système ou composant de fonctionner dans des conditions données [...] ». Dans notre cas d'usage, la fiabilité devient la capacité d'un système de stockage numérique ou analogique à restituer les données qui y sont stockées.

Nous distinguons la fiabilité que nous associons au pur aspect matériel de la performance du DNN sur sa tâche. Ce dernier est considéré comme une construction abstraite, mathématique, avec une métrique donnée sur une tâche donnée. En pratique, nous parlerons de tâches de classification, ainsi la performance du DNN sera donnée par la précision des résultats produits suite au traitement d'un jeu de données.

Quand nous intégrons le DNN sur un matériel de fiabilité limitée, par exemple par l'utilisation d'un compromis énergie-fiabilité, les calculs théoriques du DNN que pourraient produire un matériel fiable ne seront plus une garantie. Nous adoptons le terme **erreur** pour qualifier

la différence entre les résultats obtenus avec le matériel de fiabilité limitée et les calculs théoriques du DNN.

Ces erreurs, au sens informationnel, sont notamment le fait de **pannes** matérielles dans la logique d'un circuit numérique. C'est l'équivalent du faux-ami anglais de *fault*. Or, ce terme n'est pas complètement adapté aux phénomènes observables sur le calcul en mémoire analogique qui introduisent parfois des erreurs au sens informationnel sur les résultats obtenus sans qu'il n'y ait de pannes matérielles fermes. On préférera donc parler dans le cas général de **bruit** matériel pour désigner les variabilités matérielles émergeant du compromis énergie-fiabilité.

Pour pallier ce bruit, nous cherchons à obtenir des DNN **robustes**, capables de maintenir un haut degré de performance dans un environnement augmentant l'erreur sur les calculs effectués.

Dans cette thèse, nous aborderons la technologie des memristors. Ces derniers sont notamment utilisés par groupes organisés suivant une structure d'interconnexion en tableau que nous appellerons **crossbar** par la suite, en accord avec la littérature anglophone sur le sujet.

2.2 Compromis énergie : le matériel et les algorithmes adaptés

2.2.1 Calcul proche du seuil de tension

Originellement proposé par Dreslinksi *et al.* en 2010 [8], le calcul proche du seuil de tension (*Near-threshold Computing*, NTC) vise à opérer le matériel avec une tension d'alimentation à un niveau proche du seuil de fonctionnement des transistors. La tension d'opération dans un circuit semi-conducteur influe directement sur l'énergie consommée par le circuit, mais également sur sa fréquence d'opération et donc sur le délai de calcul. Le délai de calcul observe une relation inverse avec la tension : la réduction de la tension entraîne dans un régime sous le seuil une augmentation exponentielle du délai. D'autre part, l'énergie consommée par les circuits intégrés provient de deux grandes composantes : premièrement, l'énergie de changement d'états et deuxièmement, l'énergie qui découle des courants de fuite. Avec la réduction de la tension, on observe une réduction quadratique de l'énergie de changement d'états associé au terme CV^2 jusqu'à un certain **niveau de tension** en dessous duquel on observe un terme d'augmentation exponentielle du délai de changement d'état des portes logiques qui induit en contrepartie une augmentation exponentielle de l'énergie consommée découlant du courant de fuite. Un compromis pertinent est alors d'opérer à une tension proche du seuil, là où l'énergie totale est minimisée, tel qu'illustré figure 2.1.

Or, dans des conditions de fonctionnement NTC un circuit exhibe une forte sensibilité aux

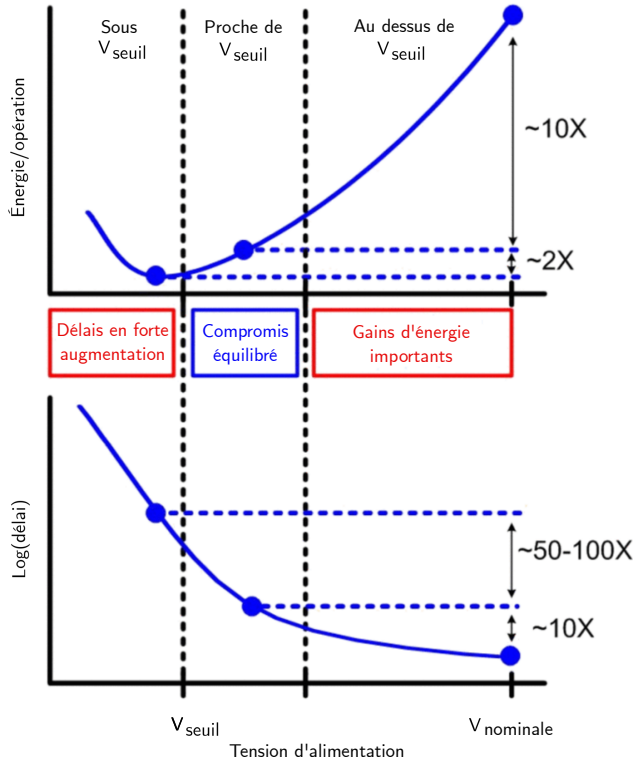
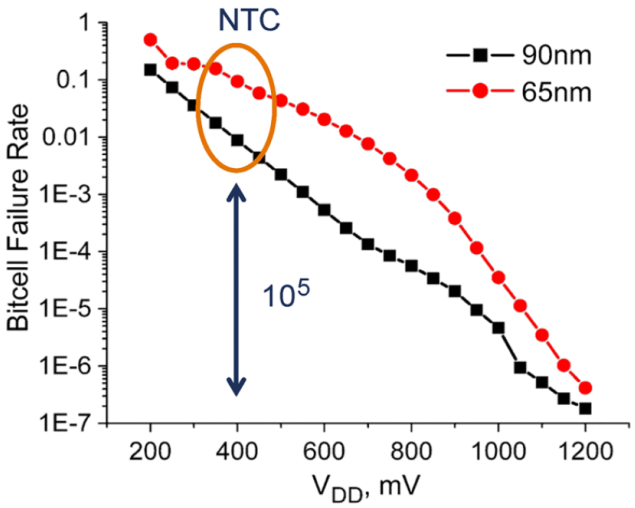


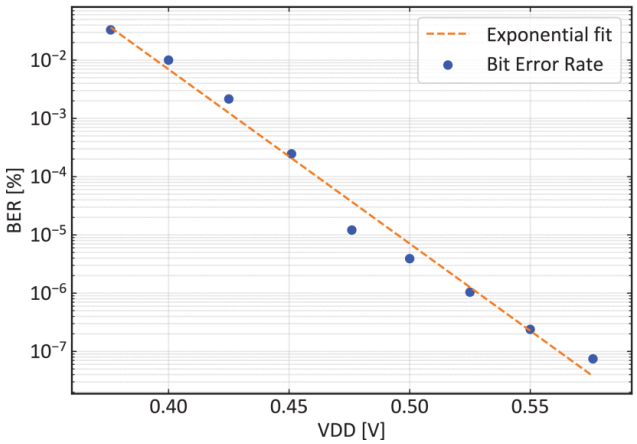
FIGURE 2.1 Énergie et délai d'un circuit selon la tension d'alimentation utilisée. Le calcul proche du seuil offre un compromis intéressant entre le délai et l'énergie consommée par le circuit. [8] © 2010, IEEE

variations des conditions de fabrication et de température, augmentant de fait les collages *stuck-at* mais aussi les erreurs de transmission découlant du fonctionnement des circuits soumis à la variation des délais des composants utilisés. Dans le cas des mémoires statiques, on observe l'apparition d'un taux d'erreurs aléatoires, communément appelées *bit-flips*. Selon Dreslinski, il existe un régime dans lequel ce taux augmente exponentiellement avec la réduction de tension d'alimentation, tel qu'observé pour des mémoires SRAM fabriquées à l'aide de deux technologies de résolutions lithographiques distinctes comme illustré figure 2.2a.

Plus récemment, ces résultats ont été répliqués sur une technologie de gravure à 22nm [11] comme l'illustre la figure 2.2b. En l'occurrence, un *System on Chip* (SoC) incorporant une mémoire SRAM a été opéré à divers niveaux de tension d'alimentation, et une régression confirme la relation exponentielle entre le taux d'erreur observé dans une mémoire SRAM et la tension d'alimentation.



(a) 90nm et 65nm [8] © 2010, IEEE



(b) 22nm [11] © 2020, IEEE

FIGURE 2.2 Relation entre la tension et le taux d'erreur sur circuit SRAM pour différentes technologies de gravure. VDD désigne la tension d'alimentation de la SRAM.

2.2.2 Calcul en mémoire

Pour une introduction au calcul en mémoire (*Processing in memory*, PIM), voir [9, 12]. À haut niveau, le PIM est une évolution de la traditionnelle architecture Von Neumann pour laquelle les données en mémoire sont acheminées jusqu'à une unité de calcul puis rapatriées en mémoire une fois traitées. Ce va-et-vient est coûteux, car il engendre de nombreuses transactions mémoire. Avec une architecture PIM, l'unité de calcul peut commander la mémoire pour effectuer sur place certaines opérations. Dans le contexte d'un DNN, une opération s'y prête : les calculs matrices-vecteurs (*Matrix-vector Multiplications*, MVM) prenant la forme $Ax = b$ avec A une matrice stockée dans la mémoire PIM-capable-de-calculs.

Dans [9], une typologie des matériels disponibles pour une architecture PIM est proposée : on y retrouve les matériels basés sur la charge tels que les SRAM abordés dans la section sur le NTC, et les matériels basés sur la résistance. Les matériels basés sur la charge sont techniquement matures, mais ils souffrent d'une large surface requise et d'une sensibilité à la chaleur et aux variations dans le procédé de fabrication, limitant les avancées en miniaturisation. Ces deux facteurs les rendent peu propices à l'apprentissage profond qui requière des capacités de traitement toujours plus importantes.

D'autre part, les matériels basés sur la résistance sont une technologie émergente, qui pourrait atteindre les dimensions de l'ordre de quelques nanomètres [9]. On y retrouve par exemple les memristors. Or, cette relative immaturité technique apporte un problème majeur : les memristors souffrent de variations paramétriques importantes rendant leurs sorties stochastiques.

Memristors et réseaux profonds

Proposés conceptuellement en 1971 [13] avec de premières démonstrations technologiques dans les années 2000 [14–16], les memristors bénéficient d'un regain de popularité avec l'émergence de l'apprentissage profond. Ainsi, plus récemment, une puce pouvant accueillir de petits réseaux de neurones (4 millions de paramètres) est proposée [17]. Un memristor est un composant passif qui est notamment utilisé comme mémoire non volatile en y appliquant une tension permettant de changer la conductance du memristor. Organisés en tableaux de deux dimensions nommés « *crossbar* », les memristors permettent entre autres applications d'effectuer un calcul MVM en y stockant une matrice de conductances. Cette architecture est présentée figure 2.3. Le vecteur d'entrée est transformé en tension présentée aux lignes du tableau, et la tension lue en sortie des colonnes est le produit de ce vecteur d'entrée et de la conductance stockée dans les memristors. Cette opération repose sur l'application de la loi d'Ohm et de la loi des nœuds de Kirchhoff et s'effectue en temps constant [18]. Les memristors

sont également adaptés aux calculs logiques et arithmétiques [19–21].

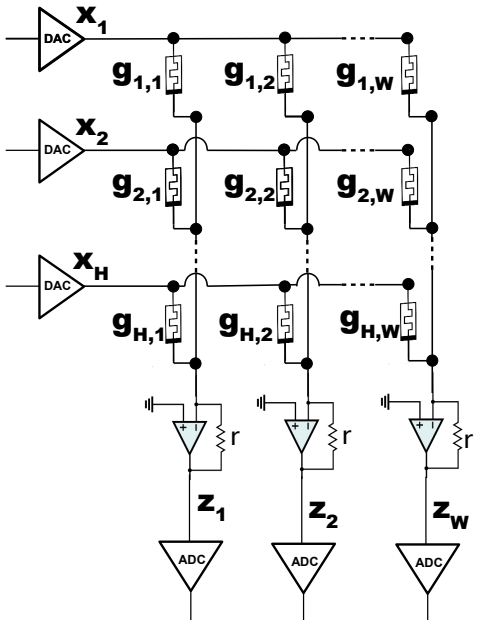


FIGURE 2.3 Architecture de crossbar de memristors pour calculs matriciels.

Ces qualités font du memristor un matériel important dans la recherche en apprentissage profond, avec d'une part une recherche en conception matérielle, et d'autre part une recherche ayant pour vocation la simulation du comportement de DNN construits avec des memristors.

Du point de vue matériel, plusieurs implémentations dédiées aux DNN ont été proposées. Dans [22], un circuit permettant de calculer et d'entraîner un réseau neuronal peu profond est conçu en se basant exclusivement sur des memristors : pour chaque couche du réseau un crossbar est alloué et les crossbars sont directement connectés entre eux. Une approche similaire [23] utilisant deux memristors par paramètre à stocker permet d'une part de réduire la consommation d'énergie du système par un facteur de 9, tout en augmentant la précision des paramètres stockés d'autre part. Dans [24], le circuit de programmation des memristors est augmenté d'un transistor afin d'améliorer la précision lors de la phase de programmation en limitant la tension d'entrée, permettant la programmation d'un DNN entraîné en dehors de la puce. Ce circuit atteint une efficacité de 115 tera-operations par seconde par watts (TOPs W^{-1}), à comparer aux 7TOPs W^{-1} d'un circuit numérique équivalent avec une précision de 4 bits. Il est aussi noté qu'une limitation technique en termes d'efficacité provient des circuits permettant de lier les domaines numérique et analogique, d'où l'intérêt de pouvoir opérer en restant dans le domaine analogique comme discuté dans [22]. Cette idée est réutilisée pour un réseau convolutionnel de 5 couches [25]. Des applications expérimentales ont démontré l'utilisation de memristors pour une variété d'applications de l'apprentissage profond sor-

tant du traditionnel réseau *fully-connected* ou convolutionnel : par exemple pour les réseaux LSTM (*« Long short term memory »*) [26] ou encore pour les réseaux auto-encodeurs [27]. On distingue le cas usuel de calcul d'une opération MVM présente dans le DNN simplement programmée dans un tableau de memristors, du cas avancé où l'on souhaite effectuer la phase d'apprentissage du DNN sur le tableau de memristor [22, 28].

Afin de faciliter l'utilisation des memristors, trois bibliothèques ont été développées ces dernières années pour simuler le fonctionnement des réseaux qui les exploitent : *Analog Hardware Acceleration Kit* d'IBM [29], DNN+Neurosim [30] et RxNN [31]. Chacune implémente des modèles d'erreur pour les memristors, et elles permettent à l'utilisateur de tester un DNN en simulant l'erreur sur ses paramètres résultant de l'utilisation de memristors. Des travaux notables ont été effectuées avec RxNN [31] qui a été utilisé pour simuler six DNN différents effectuant une tâche de classification d'image de la base de données ImageNet-1K [32]. Ces auteurs mesurent une perte de précision de l'ordre de 9% à 32% pour les DNN considérés.

Programmation et usage d'un memristor

La programmation d'un memristor est un processus itératif avec une part d'aléatoire [33]. Le changement de la conductance s'effectue par une série de pulsations électriques [34] : une tension positive entraînant une augmentation de la conductance et une tension négative la réduisant comme illustré figure 2.4.

Chaque pulsation permet de faire tendre la conductance vers une valeur désirée, avec une vérification de la valeur lue après chaque pulsation jusqu'à ce que la valeur souhaitée soit atteinte, plus ou moins une marge d'erreur arbitraire. Ce fonctionnement par approximations successives est coûteux en temps et énergie et il est donc possible d'augmenter la tolérance sur la cible à atteindre au prix d'une augmentation de variance sur les conductances programmées. Ainsi, en réduisant l'énergie allouée à la phase de programmation, les conductances programmées auront tendance à se confondre, augmentant par là même l'erreur sur les calculs produits par le crossbar. Ce compromis est illustré pour un memristor stockant des valeurs binaires figure 2.5, dans laquelle on observe un plus grand chevauchement des distributions de deux valeurs stockées en utilisant moins d'énergie pour la programmation des memristors.

Pour palier ce phénomène, il est possible de quantifier les valeurs cibles de conductance. Bien que la conductance programmée soit *in fine* analogique, la quantification de la cible permet de réduire l'impact du bruit gaussien dû à l'erreur de programmation en réduisant la probabilité que les distributions de deux valeurs distinctes se chevauchent.

Un choix courant de quantification est une représentation binaire, également très étudiée dans

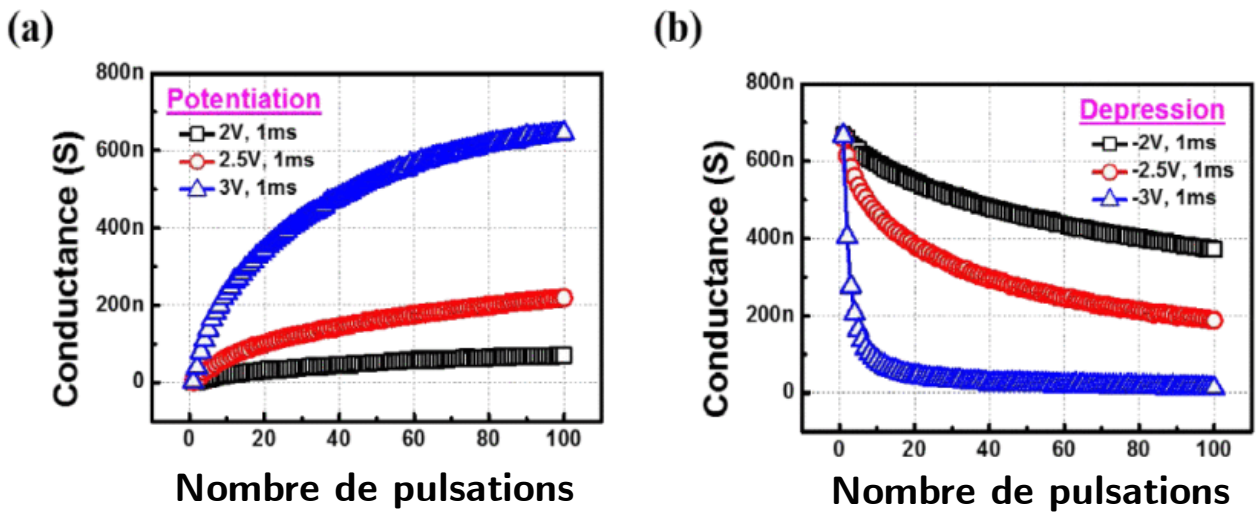


FIGURE 2.4 Programmation de memristor : (a) augmentation de la conductance par *potentiation* (b) réduction par *dépression* [35] Copyright © 2016, IEEE

le domaine des DNN [33, 36, 37] en sus des choix de quantification intermédiaires usuels [38]. Pour ces memristors quantifiés, il est possible de moduler l'énergie de programmation en fonction de la valeur pour obtenir un compromis énergie-fiabilité plus granulaire. Dans [33], il est montré que le meilleur compromis est atteint quand une énergie plus importante est dédiée à la programmation de l'état faible que de l'état fort dans un memristor binarisé. Similairement, dans [36] le choix de niveau de tension des pulsations de programmation de memristor binarisés est étudié. Ainsi, une plus petite tension augmente le chevauchement des distributions entre les deux états binaires programmables, tout en réduisant l'énergie de programmation et en améliorant la durée de vie du matériel.

Problème de variations paramétriques des memristors

Outre la question de l'erreur de programmation, les memristors ont d'autres problèmes matériels affectant la conductance de telle sorte qu'elle soit différente de celle souhaitée, ce qui aboutit à impacter les calculs effectués [22, 38, 39]. On compte premièrement les problèmes de variations de puce à puce (*device to device*) ou encore de variations de cycle à cycle dues aux procédés de fabrication encore peu matures [40]. Pour mieux comprendre ces variations de cycle à cycle, il faut regarder le memristor de près : un élément primordial nommé filament, pour lequel des impuretés et autres aléas dans le procédé de fabrication mèneront à ces variations cycle à cycle [41, 42]. On compte également le problème des *sneak paths* que l'on peut retrouver dans du matériel conventionnel [43] où l'électricité passe par des chemins alternatifs

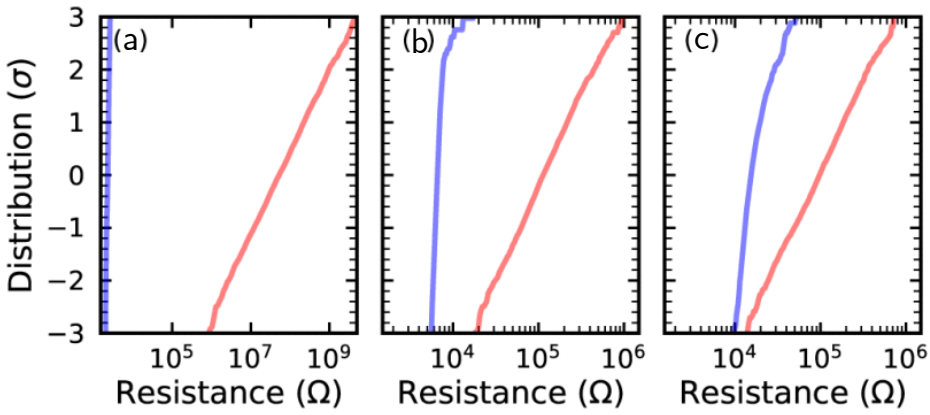


FIGURE 2.5 Distribution des résistances d'un memristor de type *resistive-RAM* binaire (état fort en rouge, état faible en bleu) sous conditions d'énergie (a) très forte (b) forte (c) faible [36] Copyright © 2019, IEEE

qui n'étaient pas désirés ; ou encore le problème de diaphonie (*crosstalk*) [44], c'est-à-dire des interférences électromagnétiques intra-puce.

Ces problématiques peuvent être partiellement corrigées en améliorant l'architecture des memristors utilisés. Comme vu plus haut, on peut par exemple utiliser un memristor couplé à une résistance [24] ou alternativement à une diode [45]. Ce sont deux modifications effectuées à haut niveau sur la structure d'une puce. Or, on peut s'intéresser directement aux structures élémentaires permettant de construire un memristor. Ainsi, on peut modifier la structure du filament pour utiliser du graphène et améliorer les propriétés physiques des memristors ainsi obtenues [44]. Les modifications architecturales réduisent cependant l'efficacité des memristors en termes d'aire de la puce : c'est le cas avec l'ajout d'une résistance ou d'une diode. Cependant, ceci rend la technologie plus attrayante à court terme en palliant les problèmes que nous venons d'aborder [46].

Ces variations sont étudiées dans plusieurs modèles statistiques, ensuite repris dans les bibliothèques de simulation [29–31]. Un modèle courant est celui d'une erreur gaussienne sur les conductances programmées [47], avec une dépendance à certains facteurs : par exemple sur la technique de programmation du memristor [33]. Il existe des modèles plus avancés permettent de simuler la sortie du DNN en tenant également compte des interactions avec le matériel intermédiaire comme les convertisseurs analogique-numérique [29].

2.2.3 Compromis énergie-précision dans les systèmes numériques

Les circuits numériques exploitent systématiquement une représentation binaire et les algorithmes de calculs utilisent toujours une représentation comportant un nombre de bits fixé

en amont. Une variable peut ainsi être exprimée avec une plus grande résolution quand on l'exprime avec un plus grand nombre de bits. Ce nombre de bits disponibles peut être vu comme une contrainte à laquelle les algorithmes doivent se plier et qui détermine le compromis énergie-précision. En effet, effectuer un calcul avec des nombres définis sur plus de bits permet une plus grande précision, mais augmente généralement la consommation énergétique en requérant des circuits plus complexes pour traiter ces bits plus nombreux comme représenté tableau 2.1.

Tableau 2.1 Énergie par opération pour une technologie 45nm selon le nombre de bits alloués à la représentation des nombres [7]

Addition	
8 bits	0,03pJ
32 bits	0,1pJ
Multiplication	
8 bits	0,2pJ
32 bits	3,1pJ

(a) Calcul en nombre entiers

Addition	
16 bits	0,4pJ
32 bits	0,9pJ
Multiplication	
16 bits	1,1pJ
32 bits	3,7pJ

(b) Calcul à virgule flottante

Cache	(64 bits)
8 kB	10pJ
32 kB	20pJ
1 mB	100pJ
DRAM	1,3-2,6nJ

(c) Transactions mémoire

Un domaine de recherche très actif consiste à quantifier correctement les algorithmes de traitement de signal afin de trouver le meilleur compromis complexité-précision [48]. Sous cet angle, la quantification est une contrainte omniprésente dans les circuits numériques ou de traitement du signal et représente un compromis énergétique répandu. En effet, un choix de quantification apportera une erreur sur la représentation des nombres, mais un nombre restreint de bits ou une représentation à virgule fixe au lieu d'une virgule flottante permettent des gains appréciables d'efficacité énergétique que ce soit par une réduction de l'empreinte mémoire ou une réduction de la complexité des circuits utilisés. L'étude et la correction des erreurs de quantification est un prérequis pour permettre à certains algorithmes de fonctionner correctement dans un environnement quantifié.

Étant donné que la quantification ne concerne que la définition du format des nombres, les gains énergétiques se cumulent avec les autres choix de conception. Par exemple, dans [49], une méthodologie est proposée pour optimiser les paramètres d'une mémoire SRAM opérée en NTC servant à stocker les paramètres quantifiés d'un filtre de Kalman [50]. L'étude conjointe des modèles d'erreurs de la quantification et de la mémoire NTC est utilisée pour corriger les équations théoriques de l'algorithme du filtre de Kalman. Cette co-conception algorithme-système permet une optimisation de la tension d'alimentation mémoire et du choix de quantification de façon à minimiser la consommation d'énergie tout en respectant

une contrainte d'erreur sur les calculs effectués par le système. Des gains énergétiques de 56% sont mentionnés.

La quantification est aussi une méthode éprouvée pour obtenir des DNN sobres en énergie et un domaine de recherche actif avec de nombreuses publications visant à optimiser la représentation des nombres constituant les paramètres et les sorties intermédiaires du DNN que ce soit lors de l'entraînement [51, 52] et/ou lors de son déploiement [1, 53, 54].

Un défi important est de réduire autant que possible le nombre de bits nécessaires pour représenter un DNN sur les systèmes numériques traditionnels étant donné la croissance du nombre de paramètres, menant à des quantifications extrêmes pour le déploiement de DNN [55, 56] ou encore pour son entraînement [52, 57]. Le cas le plus extrême est celui d'un réseau dans lequel les paramètres et les sorties intermédiaires sont binarisés : dans ce cas de figure, la quantification permet de remplacer les lourds calculs matriciels par des opérations `XNOR` et `popcount` dont l'évaluation est peu gourmande en énergie [55, 56]. Les gains en termes de mémoire utilisée et de calculs évités pour ce cas de figure extrême sont disponibles tableau 2.2 : on y observe notamment le compromis effectué entre la précision atteinte par le DNN et sa simplification.

Tableau 2.2 Gains mémoire et de calculs pour une convolution selon la binarisation des entrées et des paramètres [56]

Configuration des entrées et paramètres	Opérations réalisées par la convolution	Gains de mémoire à l'inférence	Gains de calculs à l'inférence	Précision sur ImageNet-1K avec le DNN AlexNet
Entrées à virgule flottante, paramètres à virgule flottante	$+, -, \times$	1x	1x	56,7%
Entrées à virgule flottante, paramètres binarisés	$+, -$	32x	2x	56,8%
Entrées binarisées, paramètres binarisés	`XNOR`, `popcount`	32x	58x	44,2%

Plusieurs auteurs ont observé un effet de régularisation par lequel des modèles quantifiés peuvent fournir une performance supérieure à celle obtenue avec des réseaux exploitant une

représentation comportant plus de bits, voire une représentation en virgule flottante [36, 58].

Pour l'utilisateur, il s'agit alors de trouver un compromis entre la taille du DNN et le choix de quantification, ou en d'autres termes, de maximiser la performance du DNN tout en conservant une consommation énergétique faible. Selon [59], il est plus intéressant d'utiliser une quantification sur une plage de 1 à 4 bits plutôt que la traditionnelle quantification sur 8 bits utilisée pour le déploiement de DNN afin d'atteindre le meilleur compromis énergie-performance. Dans ce cas de figure, il est observé que la perte de performance due à la quantification peut être réduite en augmentant la taille du DNN tout en obtenant un meilleur compromis énergie-performance.

Comme nous l'avons vu lors de la section sur les memristors, la quantification demeure une méthode intéressante en dehors du matériel numérique. La quantification sur n états permet de réduire la probabilité de chevauchement des distributions des n états du memristor, et augmente ainsi la fiabilité des calculs effectués. Étant donné l'intérêt d'utiliser une faible tension de programmation du memristor pour augmenter sa durée de vie [36], la quantification permet de réduire la perte de performance tout en conservant une durée de vie du memristor intéressante.

2.2.4 Méthodes d'optimisation de l'empreinte mémoire de DNN

La co-conception d'un modèle de DNN et du matériel le calculant ne se limite pas à la seule quantification de ses paramètres et de ses activations. Cette section vise à présenter les approches les plus plébiscitées visant à changer la représentation d'un DNN pour alléger l'empreinte mémoire de ce dernier. On peut remarquer une approche tierce, visant à modifier l'architecture même du modèle de DNN considéré : ce choix de conception sera abordé section 2.3.3. Comme pour la quantification, ces approches proposent de réduire l'empreinte mémoire du DNN en essayant de maintenir une performance intéressante. Souvent, un compromis intéressant peut être atteint comme nous allons maintenant le voir.

Élagage

Afin de faciliter le processus d'entraînement d'un DNN, ces derniers sont en règle générale sur-paramétrisés : ils possèdent un nombre de paramètres plus grand que nécessaire. Ainsi, une large portion des paramètres d'un DNN sont redondants et peuvent être élagués (*pruned*), c'est-à-dire mis à la valeur zéro. On obtient ainsi des tenseurs de paramètres creux (*sparse*). Ce procédé d'élagage fut exploré historiquement dès 1989 [60]. Un critère, par exemple la faible magnitude des paramètres [61], permet de décider quels paramètres sont élagués, et

un compromis entre le niveau d'élagage et la performance du DNN est à trouver. En règle générale, un DNN élagué est soumis à un ajustement (*finetuning*) pour retrouver une portion de la performance perdue.

Avec ce critère de magnitude pour un élagage du DNN AlexNet [62], reconnu pour être sur-paramétrisé, environ 50% des paramètres peuvent être élagués sans heurter de manière significative la performance du DNN sans ajustement ; 80% avec ajustement [61]. On observe néanmoins que l'élagage est plus approprié pour les couches linéaires d'un DNN (réduction d'un facteur 9,9 du nombre de paramètres) que pour les couches de convolutions (réduction d'un facteur 2,7) [61]. Malheureusement, pour l'architecture AlexNet de cette étude, l'énergie allouée aux couches convolutionnelles est bien plus importante que celle allouée aux couches linéaires [63].

Sur la base de cette observation, un élagage guidé par l'énergie plutôt que par le nombre de paramètres fut proposé [64]. Des gains d'énergie d'un facteur 3,7 pour AlexNet sont observés, en comparaison d'un gain d'un facteur 1,74 avec le critère basé sur la magnitude des paramètres du paragraphe précédent. Cette approche basée sur l'énergie fut également appliquée à l'architecture GoogleNet [65], moins sujette au phénomène de sur-paramétrisation, car plus compacte, avec un gain d'énergie d'un facteur 1,6.

Plus récemment, l'hypothèse du ticket de loterie [66] répète le cycle entraînement-élagage-remise en état initial du DNN. Les performances atteignent alors celle du DNN non élagué tout en élaguant 85% à 95% des paramètres. Un élagage intermédiaire, entre 50% et 90% des paramètres du DNN, obtient des performances plus intéressantes que le DNN initial. Les DNN élagués identifiés avec cette procédure, les gagnants de la loterie, n'obtiennent une bonne performance qu'en étant entraîné depuis l'état initial du DNN avant son entraînement-élagage. Néanmoins, ce cycle comportant des entraînements répétés est lourd en calculs pour l'utilisateur.

Aujourd'hui, on distingue deux approches d'élagage : l'élagage non structuré, tel que décrit ci-dessus, et l'élagage structuré. La différence fondamentale entre ces deux approches est la facilité de tirer des gains d'énergie sur le matériel calculant le DNN. Dans le cas structuré, on vise par exemple à retirer des filtres de convolutions plutôt qu'un paramètre isolé dans une couche. Le retrait d'un ou de plusieurs filtres de convolution allège immédiatement l'architecture du DNN, ce faisant réduisant l'énergie pour calculer ce dernier. L'élagage structuré est donc une approche très intéressante pour obtenir des gains d'énergie sans modifier le matériel. Cependant, il est aussi plus difficile de réaliser un élagage structuré sans dégrader la performance du DNN, ce qui en fait un domaine de recherche très actif [67].

Inversement, l'élagage non structuré nécessite un matériel adapté pour exploiter pleinement

le caractère creux des paramètres d'un DNN élagué. Une question prépondérante est le format de stockage des tenseurs creux. Par exemple, un codage de Huffman sur un DNN d'architecture AlexNet quantifié et élagué permet de réduire de 20% à 30% les besoins mémoires [68] tout en requérant une étape de décodage des paramètres du DNN. En comparaison, [69] développe un matériel dédié permettant de traiter les couches linéaires d'un DNN dont les paramètres sont stockés dans un format compressé *Compressed-Sparse Columns* sans décompresser ces derniers.

Enfin, on distingue une approche complémentaire d'élagage dynamique des activations d'un DNN. Par exemple, on peut tirer parti de la fonction d'activation ReLU. Cette dernière ne conserve que la part positive de ses entrées : une partie de ses sorties est alors fixée à la valeur zéro. Sur la base de cette observation, [63] propose un mécanisme de codage par plage (*run length encoding*) pour compresser la sortie d'une couche ReLU. Les accès mémoire sont ainsi réduits d'un facteur 1,5. En plus de cette compression, le matériel peut être modifié pour éviter la lecture des paramètres du DNN et le calcul de la couche suivante dépendants des activations de valeur zéro, réduisant l'énergie nécessaire de 45% [63]. En plus de quoi, [70] propose un mécanisme de passage du cycle matériel pour ces activations de valeur zéro, améliorant ainsi la vitesse d'exécution du DNN par un facteur 1,37. L'élagage des activations est également proposé : comparativement à l'exploitation de ReLU qui se veut être un mécanisme sans perte pour le DNN, une portion des activations dans une plage de faible magnitude peut être forcée à zéro, ou élaguée, comme pour [61]. En exploitant ce mécanisme, des gains de 11% sur la vitesse d'exécution sont observés [70], ou encore une réduction d'un facteur 2 de l'énergie consommée [71]. Pour cet élagage des activations, comme pour l'élagage non structuré des paramètres, on remarque que nous sommes dans un cas particulier qui requiert un matériel adapté pour obtenir des gains d'énergie.

L'élagage est donc une méthode pertinente pour réduire les besoins mémoires d'un DNN. Dans sa forme structurée, l'architecture même du DNN est allégée et les gains sont universels. Dans sa forme non structurée, les gains sont amoindris d'étapes de codage et décodages supplémentaires pour compresser les paramètres creux d'un DNN. Alternativement, le DNN creux pourrait ne pas être compressé, mais alors il n'y aurait pas de gain mémoire du fait de l'élagage. Ce codage et décodage font de l'élagage non structuré une méthode peu adaptée pour l'utilisation de crossbar de memristor dans le format de calcul vecteur-matrice que nous avons présenté plus tôt. Sur matériel numérique non fiable, l'impact d'une panne matérielle sur les valeurs des paramètres creux du DNN dans un format compressé pourrait changer de manière significative les résultats d'une opération, sans mécanisme de correction d'erreur approprié.

Distillation et entraînement de DNN amélioré

Une autre approche vise à exploiter des DNN plus compacts pour atteindre une performance similaire au DNN de plus grande taille. Étant donné que cela impacte l'architecture de DNN utilisée, les gains sont immédiats. Cela en fait aussi l'approche la plus générale qui soit. Il est également évident que ce procédé est lié au processus de recherche d'architecture neuronale qui sera décrit section 2.3.3.

Nous attirons l'attention vers un procédé en particulier : la distillation (du savoir). Ce dernier est pertinent dans cette revue de littérature, car il consiste à entraîner un DNN compact (l'élève) à imiter les sorties d'un ensemble de DNN de plus grande taille (les maîtres). Cette méthode pourrait donc être interprétée comme une approche de compression de DNN. Par exemple, pour une tâche de classification, la distillation consiste à remplacer les étiquettes « fermes » du jeu de données (valeur 1 pour la classe cible, 0 sinon) par les scores par classe (valeur entre 0 et 1 pour chaque classe) précédant l'opération *softmax* en sortie du DNN maître. L'objectif pour le DNN élève est alors de minimiser la différence entre les scores du DNN maître et ses propres scores [72]. Cet entraînement est donc réalisé en amont de l'opération *softmax*, cette dernière compressant l'information en sortie du DNN pour transformer les scores en probabilité par classe. En adaptant cette opération de *softmax* pour générer des probabilités par classe moins fermes, réduisant ainsi la perte d'information, la sortie de cette opération peut être utilisée directement [73]. Enfin, les sorties intermédiaires du DNN maître peuvent aussi être utilisées comme signal pour l'entraînement du DNN élève [74].

Ce procédé de distillation permet au DNN élève d'atteindre une performance qui serait plus difficile à atteindre s'il était entraîné avec le jeu de données initial, toutes choses égales par ailleurs [72, 75]. Ainsi, [73] démontre un gain de 2% sur la précision atteinte par un DNN élève sur une tâche de reconnaissance vocale atteignant ainsi la performance d'un ensemble de 10 DNN maîtres.

Partage de paramètres et factorisation

Enfin, une forme notable de compression de DNN consiste à adopter un schéma de partage des paramètres. La populaire opération de convolution en est un bon exemple : cette dernière peut être interprétée comme une opération linéaire dont les paramètres sont partagés entre chaque *patch* de l'image d'entrée. Ainsi, l'opération de convolution permet de traiter une même entrée tout en nécessitant moins de paramètres. De la même manière, des travaux proposent de partager les paramètres entre toutes les couches d'un DNN [76].

Ce procédé peut être appliqué plus finement aux paramètres. Par exemple, ces derniers

peuvent être représentés indirectement en utilisant un indice vers un tableau de valeurs fixes. L'indice utilise une représentation sur peu de bits, et le tableau de valeur fixe est de taille limitée mais avec des représentations sur plus de bits. En l'occurence, Deep Compression [68] utilise un algorithme des k-moyennes pour grouper les paramètres sur 256 valeurs pour les couches convolutionnelles et sur 16 pour les couches linéaires d'un DNN AlexNet. Les indices sont ainsi définis sur 8 bits pour les couches convolutionnelles et 4 bits pour les couches linéaires. Contrairement à la quantification pure et bien que le format de stockage des paramètres en mémoire soit allégé, les opérations résultant de l'utilisation d'une représentation indicielle sont réalisées en pleine précision. On remarque également que l'utilisation de paramètres binarisés avec un coefficient multiplicatif [56] peut être interpretée comme une représentation indicielle sur 1 bit. Néanmoins, des travaux visent à étendre cette approche indicielle en remplaçant l'opération d'inférence par une opération de lecture de table de correspondance (*lookup table*, LUT) [77]. Ce dernier choix de conception rend l'inférence de DNN propice à l'utilisation de FPGA, et des gains d'énergie de 47% sont rapportés.

L'utilisation d'une représentation indicielle est néanmoins soumise aux mêmes observations qu'à la section sur l'élagage pour le déploiement sur memristors et matériel numérique de fiabilité limitée.

De manière orthogonale aux représentations indicielles, un tenseur de paramètre peut être représenté en plusieurs sous-tenseurs, collectivement moins gourmands en espace mémoire que le tenseur originel. Par exemple, un choix de conception de la populaire architecture MobileNet [78] est l'utilisation de convolution *depthwise separable* : on effectue d'abord une convolution dans le domaine spatial de l'entrée (hauteur, largeur), puis une convolution dans le domaine des canaux de l'entrée. Ce procédé réduit l'empreinte mémoire et la complexité des calculs d'un DNN. On peut également représenter l'entièreté d'un DNN sous la forme d'un tenseur en 8 dimensions de rang réduit, et remplaçant chaque convolution par 3 sous-convolutions [79] pour compresser un DNN d'un facteur 3 à 6 à niveau de performance similaire. Ces exemples s'intègrent facilement dans l'architecture de DNN et donc dans son entraînement. On note cependant qu'il est possible de réaliser une décomposition tensorielle d'un tenseur après l'entraînement du DNN pour réduire a posteriori l'empreinte mémoire. De plus, cette décomposition tensorielle peut être soumise à une approximation de rang réduit (*low-rank approximation*) pour plus encore réduire l'empreinte mémoire, au prix d'une perte de performance du DNN pré-entraîné, pouvant parfois être recouvrée par ajustement, comme pour l'élagage. Ainsi, en utilisant une décomposition tensorielle de type *canonical polyadic* et une approximation de rang réduit, les calculs de DNN sur CPU sont accélérés d'un facteur 4,5 [80]. Cette approche de décomposition tensorielle est intimement liée à la définition même de l'architecture du DNN [78, 79].

2.2.5 L'efficience de DNN : une approche multimodale

Dans cette section, nous nous sommes intéressés à diverses méthodes et matériels populaires permettant de réduire l'empreinte mémoire d'un DNN. Si ces approches sont variées, elles ne sont pas pour autant exclusives. Par exemple, [68] utilise conjointement l'élagage et le partage des paramètres pour réduire les besoins de stockage d'un DNN d'un facteur 35 à 49 ; tout en réduisant ses besoins d'énergie d'un facteur 3 à 7. De la même manière, [36] emploie un format de paramètre binaire pour un DNN déployé sur memristor.

Les méthodes et les gains d'énergie associés peuvent ainsi se cumuler. L'enjeu devient alors de tirer le maximum de performance du DNN tout en réduisant à un minimum son empreinte mémoire, en associant les méthodes pertinentes. L'espace d'optimisation est ainsi très varié.

Le compromis énergie-fiabilité d'une mémoire servant à stocker les paramètres du DNN que nous souhaitons exploiter pour améliorer l'efficience d'un DNN n'échappe pas à cette dimension multimodale. Ainsi, la mémoire de fiabilité limitée utilisée pour stocker les paramètres du DNN peut s'appliquer à des DNN dont les paramètres ont subi une quantification plus ou moins agressive, comme c'est le cas dans les travaux de [36]. De la même manière, le réseau pourrait avoir été élagué, entraîné par distillation ou encore soumis à un partage des paramètres.

On peut cependant discerner des limites au cumul des approches abordées en présence d'une mémoire de fiabilité limitée. Notamment, les approches qui touchent au format en mémoire des paramètres du DNN sont directement impactées par l'utilisation d'une mémoire non fiable. Par exemple, un élagage non structuré abouti à des paramètres de DNN creux. Sur un memristor, cela est peu intéressant du fait de la structure contrainte de crossbar pour stocker les paramètres. Sur matériel numérique, cela pose la question du format de représentation des paramètres creux et de l'impact significativement différent qu'une erreur mémoire pourrait avoir sur les calculs effectués. Cette dernière question se pose également en présence de paramètres quantifiés de manière agressive ou avec un partage des paramètres, de telle sorte qu'un encodage visant à réduire les besoins mémoire soit effectué pour stocker le DNN. L'utilisation d'un format de quantification pose également la question du poids différencié qu'une erreur mémoire pourrait avoir sur les calculs effectués selon que cette erreur affecte le bit de poids fort ou le bit de poids faible.

2.3 Compromis fiabilité : comment l'améliorer

2.3.1 Paradigmes de tolérance d'erreurs

On dénombre deux paradigmes de tolérance d'erreurs [81] : le calcul approximatif utilisant une approche déterministe produisant des résultats peu précis, et le calcul stochastique visant à remplacer les calculs arithmétiques par des calculs probabilistes. Le NTC et la quantification que nous avons abordés plus tôt sont des exemples d'implémentations de calcul approximatif : ce sont des scénarios dans lesquels l'exactitude n'est pas un prérequis pour obtenir des résultats corrects. Ces définitions ne brident en rien les implémentations qui en découlent : par exemple, des memristors peuvent servir à implémenter un calculateur stochastique [82]. De la même manière, un réseau de neurones profond est un outil de calcul approximatif : les entrées et sorties ne sont pas toujours précises, et c'est d'ailleurs un avantage majeur des DNN. On peut alors parler d'algorithmes robustes, tolérant les erreurs et les approximations.

Les algorithmes itératifs qui visent à améliorer la solution à chaque itération sont particulièrement adaptés au calcul approximatif [81]. On relaxe les conditions d'arrêts de l'algorithme, ce faisant, cela réduit le nombre d'itérations et donc le nombre de calculs, au profit d'un résultat moins précis. C'est par exemple le cas d'un DNN qui peut être entraîné sur un plus petit nombre d'époques [83] ou sur un jeu de données plus petit [84].

Le calcul stochastique a une définition plus restreinte et opère exclusivement sur des séquences de bits aléatoires, la taille de cette séquence étant directement liée à la précision du résultat obtenu. Dans [85], une implémentation de réseau de croyances profond sur calculateur stochastique en nombres entiers atteint entre autres des gains d'énergie de l'ordre de 21% à 33% par rapport à une approche binaire standard.

2.3.2 Détection et correction d'erreurs

Codes de correction d'erreurs

Un cas de figure particulier concerne les codes de correction d'erreurs, traditionnellement utilisés dans le domaine des télécommunications pour fonctionner sur un canal de transmission bruité. Le fonctionnement est en deux étapes : un encodage de la source de données permettant d'ajouter de la redondance, puis un décodage sur le message transmis, ce dernier étant potentiellement altéré. Ce procédé d'encodage et de décodage permet de tolérer plus facilement une certaine quantité d'erreurs sur le message reçu : c'est par exemple utilisé dans les barrettes de DRAM pour les serveurs d'applications professionnelles sensibles. La problématique de stockage d'une information fiable à partir de composants non fiables fut explorée

dès 1968 [86].

Plus récemment dans [87], une machine à état fiable est construite à partir de composants non fiables en répliquant les calculs sur plusieurs machines parallèles. Les résultats sont choisis avec un vote majoritaire, et les entrées sont encodées avec un code *Low Density Parity Check* (LDPC) [88].

Les codes LDPC sont intéressants par leur complexité de décodage limitée et leurs faibles besoins de redondance. Un décodeur LDPC peut opérer sur du matériel non fiable [89,90], par exemple en utilisant le NTC. Dans [91], la quantification, la longueur des mots de code et le taux d'erreur d'une mémoire NTC sont optimisés pour minimiser la consommation d'énergie d'un décodeur LDPC tout en atteignant une performance de décodage donnée.

En s'inspirant de codes de correction d'erreurs pour modéliser la dernière couche d'un DNN comme un problème de décodage, [92] démontre une robustesse améliorée en comparaison d'un DNN traditionnel tout en tenant compte des erreurs produites par les memristors.

Tolérance au bruit algorithmique

La tolérance au bruit algorithmique (*« Algorithmic Noise Tolerance »*, ANT) est une méthode visant à compenser les erreurs qui a été employé avec succès dans le domaine du traitement de signal et de l'apprentissage automatique [93–95]. Introduit pour contrecarrer les effets négatifs du NTC, l'ANT consiste à adosser à un système de calcul non fiable un estimateur et un correcteur d'erreurs matériel fiable, mais léger, et donc biaisé, en se basant sur la connaissance des calculs effectués par le système, de ses entrées et sorties attendues. Par exemple, dans [94], l'ANT est appliqué aux unités MAC calculant un filtre numérique. En se basant sur l'évolution de la sortie du filtre à chaque pas, les auteurs proposent de remplacer une sortie détectée comme erronée par la précédente sortie du filtre, une estimation biaisée, mais précise et surtout peu coûteuse en calculs de la vraie sortie. Des gains de l'ordre de 60% à 81% quant à l'énergie consommée sont notés, tout en observant une dégradation des résultats de l'ordre de 0,5dB.

La conception de l'estimateur et d'un correcteur optimal ne sont pas des questions avec une réponse unique : l'enjeu principal est de concevoir un estimateur et un correcteur d'erreur aussi performant que possible tout en consommant le moins d'énergie pour une application donnée. On compte ainsi des applications variées comme le calcul de chaînes de Markov [96], ou encore à un décodeur de Viterbi [97]. C'est donc sans surprise que ce modèle a été appliqué au domaine de l'apprentissage profond.

Ainsi dans [95], l'ANT est appliquée aux lourds calculs MVM d'un DNN convolutionnel. Un

estimateur est obtenu en utilisant des manipulations algébriques pour décomposer les calculs matriciels et un arrondi à la puissance de 2 la plus proche pour substituer les opérations de multiplications par des `décalages` plus efficaces en énergie. Pour un réseau convolutionnel peu profond, cela équivaut à un surplus de calculs de l'ordre de 5% pour les couches convolutionnelles et 15% pour les couches linéaires (*fully connected*), mais en pouvant supporter onze fois plus d'erreurs. Une approche tierce pour réseaux convolutionnels [98] propose d'avoir un estimateur regroupant les calculs par un algorithme de k-moyennes [99], en obtenant un surplus de calcul de 15% pour le même type de réseau que la précédente approche, mais avec cette fois-ci une tolérance à dix fois plus d'erreurs.

2.3.3 Robustesse de DNN et architectures dédiées

Les DNN constituent une classe d'algorithmes robustes. Selon le contexte, la robustesse peut désigner la capacité du DNN d'opérer dans un contexte de réseaux compétitifs dits antagonistes [100] ou sur un matériel incertain [101].

Dans le cas de matériel incertain, [101] démontre une capacité innée de réseaux peu profonds à résister à de haut taux d'erreurs matériels (de l'ordre de 10^{-3}) que l'erreur soit représentée avec une valeur aléatoire ou une valeur neutre (soit zéro). Dans ce cas, la robustesse est la capacité du DNN à conserver une portion de sa performance tandis que les conditions matérielles se dégradent. Cette même robustesse peut être améliorée en jouant sur l'architecture du réseau pour que ce dernier ait plus de paramètres.

Cette observation s'étend à des réseaux ayant une quantification extrême [102, 103]. Il est donc possible de combiner la quantification et un matériel stochastique pour atteindre un nouveau compromis énergie-performance pour le DNN. L'accumulation d'erreurs à chaque couche du DNN [102] dépend non seulement de la stochasticité matérielle, mais également de l'architecture du DNN : des couches de *pooling* réduisant l'erreur relativement aux couches précédentes. Il est également montré que le DNN observe une meilleure robustesse lorsque l'erreur matérielle impacte les résultats intermédiaires du DNN plutôt que ses paramètres.

Préparation de DNN robustes

Pour un DNN entraîné sur un matériel distinct de son déploiement, ce qui est souvent le cas au vu de la lourdeur de l'algorithme d'entraînement, on observe un maximum de performance quand le DNN aura été entraîné en simulant la quantification du matériel cible autrement appelé *quantization-aware training*. Couplé avec une troncature (*clipping*) pour trouver l'échelle des paramètres du DNN [104], on obtient un DNN robuste au bruit de quantification. Ces

paramètres de troncature et de quantification peuvent être appris pour améliorer la robustesse du DNN à des erreurs matérielles [105] en simulant les erreurs du matériel cible lors de l'entraînement du DNN (*hardware-aware training*, ou plus précisément *error-aware training*), les erreurs pouvant provenir de choix de conception tel que l'utilisation du NTC. Cette méthode est connue de longue date pour améliorer la robustesse de DNN [106]. La simulation des erreurs pour chaque bit peut cependant induire de sérieux surplus de calculs et de besoins mémoires lors de l'entraînement du DNN.

Cette simulation des erreurs matérielles lors de l'entraînement permet de mieux les tolérer lors du déploiement du DNN. Il est même possible de simuler de potentiels mécanismes de détection et de correction d'erreurs. Ainsi, dans [107], un modèle de détection et de correction d'erreurs sur une mémoire SRAM opérant en NTC stockant les paramètres d'un DNN est considéré : quand un *bitflip* est détecté par le matériel, la correction appliquée fait tendre la valeur fautive vers zéro. Ce modèle de fonctionnement est ensuite incorporé dans l'entraînement du DNN en simulant les pannes par un mécanisme proche de Dropout [108], ce qui est plus rapide que le modèle de pannes par bit. Il est enfin montré qu'un tel mécanisme permet d'augmenter la robustesse du réseau si ce dernier est entraîné à un taux d'erreur représentatif du matériel, et que ce taux d'erreur peut être modulé selon un découpage arbitraire du DNN pour améliorer le compromis énergie-fiabilité. Dans certaines configurations, il est ainsi plus économe en énergie de réduire la tension que d'augmenter la taille du DNN jusqu'à un certain seuil où des pertes de performance critique sont observables. Dans une approche similaire, [6] entraîne un DNN pour déploiement dans un environnement NTC en simulant des erreurs par bits lors de l'entraînement. Pour s'assurer que le taux d'erreur par bit utilisé lors de l'entraînement est respecté lors du déploiement, un mécanisme de canari est intégré dans la mémoire SRAM stockant les paramètres du DNN pour contrôler et corriger au besoin la tension d'alimentation.

Des expériences similaires ont montré l'intérêt de simuler les erreurs pour un déploiement sur memristors binarisés [36] ou analogiques [109]. Dans [109], l'erreur des memristors est modélisée par un bruit gaussien additif sur les paramètres du DNN lors de son entraînement. Un réseau de type ResNet32 entraîné avec cette procédure peut atteindre des pertes de performance inférieure au pourcent tout en étant conditionné à un bruit gaussien d'écart type 5%. Or, ce modèle gaussien simple est critiquable par son inexactitude. Dans [110], un modèle plus exhaustif des variations inhérentes aux memristors est utilisé pour l'entraînement de DNN. Les divers DNN entraînés avec cette procédure observent une amélioration de la perte de performance sur memristor dans le pire des cas de 21,81% à un modeste 2,65%.

Une alternative à la simulation du matériel, qui demeure coûteuse en temps et en mémoire

lors de l'entraînement du DNN, est d'utiliser des régularisations ou des fonctions de pertes adaptées [111, 112]. En s'inspirant de la fonction de perte des *Supports Vector Machines* visant à maximiser la marge dans un problème de classification (*Hinge loss*), [111] obtient des DNN binarisés avec une robustesse améliorée par rapport à la classique fonction de perte par entropie croisée. Enfin, il est montré que cette approche de maximisation de la marge est bonifiée avec une simulation des erreurs matérielles, ce qui demeure coûteux lors de l'entraînement du DNN.

Recherche d'architectures neuronales

Dans le contexte de DNN, il est possible de jouer sur l'architecture pour maximiser la performance tout en incluant des contraintes ou des objectifs tiers à minimiser tel que le nombre de bits composant le DNN, la latence matérielle, etc. On entre alors dans le domaine de la recherche d'une architecture neuronale (*Neural Architecture Search*) [113,114]. De nombreuses architectures efficientes de DNN ont vu le jour ces dernières années avec l'engouement autour du NAS : entre autres, on peut citer les architectures Swin [115], FBNet [116] et S3 [117].

Les principaux critères utilisés dans ces travaux qui considèrent une plateforme de calcul traditionnelle sont la latence, le nombre d'opérations et le nombre de paramètres du DNN. Ces approches se combinent facilement avec des définitions plus précises du budget matériel, prenant par exemple en compte la quantification du DNN ou la plate-forme de calcul utilisée [118, 119]. Néanmoins, une procédure NAS demeure lourde en calcul et des efforts du point de vue algorithmique sont entrepris pour réduire cette charge [120, 121].

Dernièrement, le domaine s'est étendu à des architectures matérielles non-conventionnelles : memristors, FPGA et autres unités de calculs dédiées [122–124]. Mieux encore, des approches adaptées aux matériels sujets aux pannes, tels que les memristors, sont proposées [125, 126]. Dans cette optique, la co-conception du matériel et du DNN par un algorithme de NAS englobant ces deux dimensions est une direction de recherche émergente [127, 128].

En intégrant un entraînement simulant grossièrement les erreurs produites par le matériel dans une procédure de NAS, [126] obtient une architecture de DNN avec une robustesse améliorée. En étudiant de près les architectures obtenues, il s'avère que cette robustesse est atteinte en ajoutant une certaine redondance dans les calculs du DNN, entre autres améliorations architecturales. Des résultats similaires [125] sont observés en simulant précisément le bruit matériel avec la librairie *Analog Hardware Acceleration Kit* [29] (AIHwKit). Un exemple de compromis architectural obtenu est visible figure 2.6, en comparaison avec la traditionnelle architecture ResNet32.

En particulier, la simulation précise des perturbations d'un memristor effectuée par [125] est utilisée pour améliorer les métriques d'énergie, de débit et la précision atteinte sur la tâche de classification CIFAR-10, sur la tâche de *Visual Wake Word* ou de détection de mot-clef (*Keyword spotting*). Sur la tâche de classification CIFAR-10, cette approche permet une réduction d'énergie de 17% tout en améliorant la précision obtenue de 2%.

Il est également remarqué que cette redondance obtenue dépend de la tâche du DNN et des contraintes matérielles choisies : pour des tâches plus complexes, il est plus difficile de respecter un nombre de paramètres du DNN maximal, auquel cas la procédure de NAS converge vers des réseaux profonds avec peu de redondances. Les DNN convolutionnel jouissant d'une forte efficience paramètres-performance sont ainsi plus prompts à bénéficier de la redondance dans les architectures obtenues.

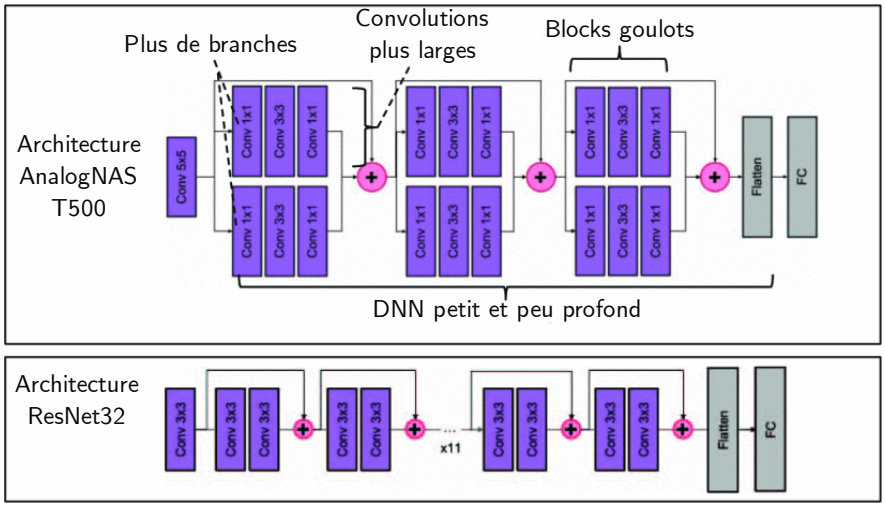


FIGURE 2.6 Architectures optimisées pour déploiement sur memristor avec la méthode AnalogNAS [125] Copyright © 2023, IEEE

2.4 Synthèse des travaux antérieurs

Dans ce chapitre, nous avons passé en revue les mécanismes de réduction d'énergie d'une part et les mécanismes permettant d'augmenter la fiabilité d'un système ou algorithme fonctionnant dans un contexte incertain d'autre part. En liant les deux approches, on peut obtenir des systèmes plus efficients énergétiquement, tout en minimisant les pertes de performance. Si ces mécanismes de fiabilité sont répandus, ils sont cependant dépendants d'une quelconque modification matérielle ou algorithmique venant éroder le gain d'énergie initial et augmentant la complexité du système utilisé.

Dans le reste de cette thèse, nous visons à tirer parti au maximum du compromis énergie-fiabilité matériel en construisant un DNN robuste intégrant les paramètres matériels de la mémoire dans sa définition et son optimisation. Ces paramètres matériels peuvent tantôt être la tension d'alimentation d'une mémoire SRAM opérant en NTC [49] ou encore la plage de conductance utile d'un memristor [129]. Plutôt que de modifier les architectures matérielles ou de complexifier les calculs du DNN, nous visons à incorporer la robustesse dans les paramètres de ce dernier. En utilisant des modèles de pannes théoriques pour modifier l'entraînement du DNN, les méthodes proposées permettent d'obtenir des DNN proposant un compromis énergie-performance contrôlable et peu coûteux en temps de calculs. Nous étendons ensuite les paramètres architecturaux du DNN dans l'optimisation pour obtenir un compromis énergie-fiabilité plus représentatif de la multitude de cas d'usage possibles.

CHAPITRE 3 MÉTHODES DE L'APPRENTISSAGE PROFOND ET MODÈLES MATÉRIELS

Nous cherchons maintenant à introduire les notations, symboles et concepts pertinents de l'apprentissage profond pour la bonne compréhension des chapitres suivants. Pour ce faire, une section 3.1 présente les notations et les symboles utilisés. Dans un deuxième temps, la section 3.2 présente l'entraînement d'un DNN en abordant la structure attendue d'un DNN, l'apprentissage par descente de gradient, la méthode d'entraînement *hardware-aware*, et enfin le concept de l'apprentissage conscient de la régularité de la fonction de perte du DNN. Dans un troisième et dernier temps, la section 3.3 présente le procédé de programmation d'un DNN sur une mémoire SRAM sous-alimentée et sur memristor.

3.1 Notations et symboles

Par la suite, nous exprimerons des opérations dont les notations sont maintenant présentées.

3.1.1 Structures

$\{x, y\}$ Ensemble composé des éléments x et y
$\{x, .., y\}$ Ensemble composé de l'énumération des entiers entre x et y
$\mathbf{x}$ Vecteur de variables x (1 dimension)
$\mathbf{X}$ Matrice de variables x (2 dimensions)
$\boldsymbol{\mathcal{X}}$ Tenseur de variables x (n dimensions)

Indexage de structure

Nous utilisons la notation d'indexage avec l'indice i tel que $\mathbf{x}_i$. Pour une structure multidimensionnelle, on pourra enchaîner les indices. Par exemple, pour une matrice $\mathbf{X}$ nous pouvons utiliser la notation $\mathbf{X}_{i,j}$ avec i l'indice de la première dimension et j l'indice de la deuxième dimension. En l'absence de précision sur une ou plusieurs dimensions, on considérera que tous les éléments de cette dimension sont utilisés.

3.1.2 Opérations

Opérations sur des scalaires ou appliquées sur chaque élément d'un ensemble pris indépendamment :

$\log(x)$ Logarithme naturel de x

$|x|$ Valeur absolue de x

Opérations sur des structures :

$\texttt{count}(\boldsymbol{\mathcal{X}})$ Nombre d'éléments du tenseur $\boldsymbol{\mathcal{X}}$. Par extension, peut être appliqué aux matrices et vecteurs.

$\|\boldsymbol{\mathcal{X}}\|$ norme-2 du tenseur $\boldsymbol{\mathcal{X}}$, que l'on obtient en convertissant $\boldsymbol{\mathcal{X}}$ en vecteur et en calculant la norme-2 de ce vecteur.

$\|\boldsymbol{\mathcal{X}}\|_p$ norme-p du tenseur $\boldsymbol{\mathcal{X}}$, que l'on obtient en convertissant $\boldsymbol{\mathcal{X}}$ en vecteur et en calculant la norme-p de ce vecteur.

$x \circ y$ Produit matriciel d'Hadamard (*elementwise product*) entre x et y

$x \circledast y$ Produit de convolution entre x et y

3.1.3 Symboles

La liste suivante recense les symboles généraux employés dans cette thèse. Des symboles spécifiques sont également introduits au fil de ce chapitre.

$x \triangleq y$ On définit x tel qu'il soit égal à y

$\nabla f(x)$ Gradient de la fonction $f(x)$

$\frac{dy}{dx}$ Dérivée de y par rapport à x

$\nabla_w f(x)$ Gradient de la fonction $f(x)$ par rapport à l'élément w

$\mathbb{E}(x)$ Espérance de x

$x^\star = \operatorname{argmin}_x f(x)$ À lire comme suit : $x^\star$ est la valeur de l'argument x minimisant la fonction f

$\tilde{x}$ Variante bruitée de l'élément x

$\boldsymbol{\mathcal{W}}$ Désigne un tenseur de paramètres d'une couche de DNN (en utilisant la première lettre du terme *weight*). Par extension,

$\mathbf{W}$ désigne une projection en deux dimensions de $\boldsymbol{\mathcal{W}}$,

w un élément de $\boldsymbol{\mathcal{W}}$.

Selon le contexte, on abusera de la notation $\boldsymbol{\mathcal{W}}$ pour désigner par extension tous les paramètres d'un DNN. On pourra alors utiliser $\boldsymbol{\mathcal{W}}^{(\ell)}$ pour désigner le tenseur de paramètres de la couche ℓ.

$\boldsymbol{\mathcal{X}}$ Désigne l'entrée d'une opération, soumis aux mêmes variantes que $\boldsymbol{\mathcal{W}}$.

$\boldsymbol{\mathcal{Y}}$ Désigne la sortie d'une opération, soumis aux mêmes variantes que $\boldsymbol{\mathcal{W}}$.

$(\boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{O}})$ Désigne un jeu de données composé du tuple entrées $\boldsymbol{\mathcal{X}}$ et étiquettes $\boldsymbol{\mathcal{O}}$

ϵ Désigne une perturbation additive

δ Désigne une perturbation multiplicative

$x \sim \mathcal{U}\{y_1, .., y_n\}$ À lire comme suit : x suit une distribution discrète uniforme des éléments

pris dans l'ensemble $\{y_1, .., y_n\}$

$\mathcal{L}$ Fonction de perte, suivie de ses entrées. Par concision, ces derniers pourront être omis.

3.2 Entraînement d'un DNN

3.2.1 Structure d'un DNN

Dans cette thèse, nous considérons un DNN composé de ℓ opérations paramétriques distinctes, appelées « couches » du DNN. Chacune de ces ℓ opérations est donc paramétrée par un tenseur $\boldsymbol{\mathcal{W}}^{(i)}\, \forall i \in \{1, .., \ell\}$. En règle générale, nous considérons deux opérations paramétriques : les couches dites linéaires (*linear* ou *fully-connected* dans certaines références) et les couches dites de convolution. Des opérations paramétriques tierces existent : par exemple, l'opération dite de *batch-normalization* (normalisation par mini-lot). Il existe également des opérations non paramétriques : par exemple, l'opération ReLU (*Rectified Linear Unit*).

Pour notre problématique, les couches linéaires et de convolutions sont particulièrement cruciales, car le tenseur de paramètres $\boldsymbol{\mathcal{W}}$ atteint une taille importante (plusieurs millions d'éléments) dans les architectures de DNN modernes, et constitue donc une part prépondérante des besoins de stockage mémoire associés à un DNN. Par ailleurs, des techniques d'optimisations permettent de fusionner les paramètres de couches adjacentes comme de *batch-normalization* dans le tenseur $\boldsymbol{\mathcal{W}}$, renforçant la part mémoire d'un DNN alloué à ces deux types de couches. Enfin, on peut noter que ces deux opérations nécessitent des calculs matriciels, comme nous allons maintenant le voir.

Plus précisément, une couche linéaire effectue l'opération suivante :

$$\mathbf{y} = \mathbf{x}\mathbf{W}^T + \mathbf{b}\,, \tag{3.1}$$

c'est-à-dire un produit vecteur-matrice entre le vecteur $\mathbf{x}$ de dimension (H) et la matrice $\mathbf{W}$ de dimension (W, H), avec un potentiel vecteur de biais $\mathbf{b}$ de dimension (W), pour obtenir un vecteur $\mathbf{y}$ de dimension (W). On remarque que la formulation générale du tenseur n-dimensionnel $\boldsymbol{\mathcal{W}}$ est substituée par celle plus précise d'une matrice $\mathbf{W}$ en deux dimensions. De plus, on souligne que le vecteur de biais $\mathbf{b}$ fait partie intégrante des paramètres d'une couche linéaire.

D'autre part, une couche de convolution est souvent utilisée pour traiter des données ayant une dépendance spatiale ou temporelle. C'est par exemple le cas pour les images, organisées en tenseur d'entrée $\boldsymbol{\mathcal{X}}$ de dimensions (C^i, H, W), avec la dimension C^i les canaux de l'image, H la hauteur et W la largeur. L'opération de convolution consiste à réaliser un produit

de convolution $\circledast$ entre chaque *patch* possible de l'image et un tenseur de paramètres $\boldsymbol{\mathcal{W}}$, aussi appelés « filtres de convolution », de dimensions (C^o, C^i, H_c, W_c). On remarque que ce tenseur $\boldsymbol{\mathcal{W}}$ possède une dimension sur le nombre de canaux de sortie C^o, opère sur le nombre de canaux de l'image C^i et sur une surface de hauteur $H_c \leq H$ et une largeur $W_c < W$. Comme pour la couche linéaire, un vecteur biais $\mathbf{b}$ de dimension (C^o) peut-être utilisé et fait partie de l'ensemble des paramètres de la couche. On obtient un tenseur de sortie $\boldsymbol{\mathcal{Y}}$ de dimensions (C^o, H^o, W^o).

Pour mieux illustrer ces éléments, un exemple jouet est des tenseurs d'entrée et de sortie est montré figure 3.1a, et le tenseur de paramètres de la couche de convolution figure 3.1b.

Nous supposons que la fonction $\texttt{patch}_{\boldsymbol{\mathcal{X}}}(c^o, h^o, w^o)$ pour n'importe quelle combinaison $c^o \in \{1, .., C^o\}$, $h^o \in \{1, .., H^o\}$, $w^o \in \{1, .., W^o\}$ nous renvoie un tenseur de dimension (C^i, H_c, W_c), le *patch* correspondant dans $\boldsymbol{\mathcal{X}}$. Le calcul de la convolution suit ensuite pour chaque canal de sortie

$$\boldsymbol{\mathcal{Y}}_{c^o,h^o,w^o} = \mathbf{b}_{c^o} + \sum_{c^i=1}^{C^i} \texttt{patch}_{\boldsymbol{\mathcal{X}}}(c^o, h^o, w^o)_{c^i} \circledast \boldsymbol{\mathcal{W}}_{c^o,c^i} \,. \tag{3.2}$$

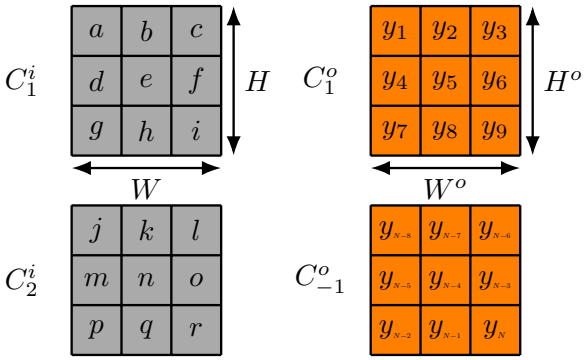
Les couches de convolution proposent plusieurs paramètres permettant de moduler le nombre de *patchs* possibles : le décalage entre deux *patchs* successifs (*stride*), l'écart entre les pixels d'un *patch* (*dilation*), ou encore l'ajout de p pixels sur le contour de l'image (*padding*).

En pratique sur GPGPU, chaque *patch* est déplié (*unfolding*) en vecteur pour permettre de réaliser l'opération de convolution par une simple opération de produit scalaire matriciel de la forme

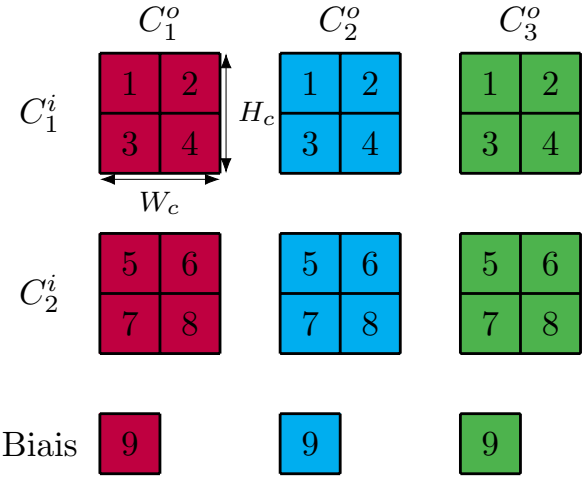
$$\mathbf{Y} = \mathbf{X}\mathbf{W} \,, \tag{3.3}$$

avec un repliage de la matrice $\mathbf{Y}$ résultante en le tenseur originel $\boldsymbol{\mathcal{Y}}$. Avec cette formulation $\mathbf{X}$ est une matrice comprenant $(H + 2p - H_c)(W + 2p - W_c)$ lignes, le nombre de *patchs*, et $(H_c W_c C^i)$ colonnes, le nombre d'éléments dans un *patch*. Ainsi, chaque colonne corresponds à un filtre de convolution aplati. Les colonnes calculent ainsi simultanément chaque canal pour un pixel de sortie. On voit apparaître une colonne supplémentaire remplie de la valeur 1 si le biais est utilisé. La matrice de paramètres $\mathbf{W}$ est de dimensions $(H_c W_c C^i, C^o)$, avec l'insertion du biais dans la première dimension si ce dernier est utilisé, soit une matrice de dimensions $(H_c W_c C^i + 1, C^o)$.

À noter que ces formulations sont souvent généralisées pour le traitement de lots en préfixant l'entrée $\mathbf{x}$ ou $\boldsymbol{\mathcal{X}}$ d'une dimension sur le lot. Par exemple, on trouvera pour le traitement d'images un tenseur $\boldsymbol{\mathcal{X}}$ de dimensions (N, C^i, H, W) avec N la taille de lot utilisée.



(a) Tenseur d'entrées $\boldsymbol{\mathcal{X}}$ et de sorties $\boldsymbol{\mathcal{Y}}$. Le dernier canal de sortie est d'index -1. Le nombre d'éléments de $\boldsymbol{\mathcal{Y}}$ est associé à la variable $y_N = C^o H^o W^o$ pour simplifier la représentation.



(b) Tenseur de paramètres $\boldsymbol{\mathcal{W}}$ et vecteur de biais $\mathbf{b}$ d'une couche de convolution

FIGURE 3.1 Illustration des tenseurs d'entrée, de sortie et des paramètres d'une couche de convolution jouet.

3.2.2 Apprentissage par descente de gradient

En général, un DNN dispose de deux modes d'opération : passe avant et passe arrière (ou encore rétropropagation du gradient, *backpropagation*). Lors de la passe avant, une entrée sous forme de tenseur $\boldsymbol{\mathcal{X}}^{(1)}$ est présentée à la première couche du DNN et successivement propagée aux couches suivantes jusqu'à la ℓ-ème couche du DNN de sorte à obtenir la sortie $\boldsymbol{\mathcal{Y}}^{(\ell)}$ du DNN. Avec cette notation, la sortie de la i-ème couche représente également l'entrée de la (i+1)-ème couche, soit $\boldsymbol{\mathcal{Y}}^{(i)} = \boldsymbol{\mathcal{X}}^{(i+1)}$. Par simplicité, nous pouvons représenter cette opération par une fonction f encapsulant l'entièreté du DNN, masquant les indices de couches et concaténant tout les paramètres du DNN en un tenseur $\boldsymbol{\mathcal{W}}$ tel que

$$\boldsymbol{\mathcal{Y}} = f(\boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{W}})\,. \tag{3.4}$$

Lors de la passe arrière, une fonction de perte à minimiser $\mathcal{L}(\boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{Y}})$ est calculée selon une sortie voulue pour le DNN $\boldsymbol{\mathcal{O}}$ et la sortie produite par le DNN $\boldsymbol{\mathcal{Y}}$. Cette sortie voulue $\boldsymbol{\mathcal{O}}$ prend plusieurs appellations dans la littérature selon les cas d'usage : étiquette (*label*), cible (*target*), etc. Par simplicité à nouveau, nous pouvons exprimer la fonction de perte selon les seuls paramètres du modèle de DNN, $\boldsymbol{\mathcal{W}}$, tel que $\mathcal{L}(\boldsymbol{\mathcal{W}})$.

Ensuite, le gradient de chaque tenseur de paramètre par rapport à la fonction de perte $\nabla_{\boldsymbol{\mathcal{W}}^{(i)}}\mathcal{L}(\boldsymbol{\mathcal{W}})\,\forall i \in \{1, .., \ell\}$ est obtenu par l'application de dérivées en chaîne, c'est-à-dire l'application de dérivées successives depuis la dernière couche du DNN vers la première couche.

Sans perte de généralité, l'entraînement d'un DNN consiste à alterner ces deux étapes et à mettre à jour les paramètres du DNN selon la règle de descente du gradient à la fin de chaque passe-arrière comme suit :

$$w_{(t)} = w_{(t-1)} - \iota \nabla_w \mathcal{L}(w)\,, \tag{3.5}$$

en adoptant ici le point de vue d'un paramètre $w \in \boldsymbol{\mathcal{W}}$ pour simplifier les notations utilisées. On remarque l'indice $(t-1)$ précisant l'état initial de w et l'indice (t) le nouvel état de w.

À terme, ce procédé doit permettre d'atteindre un minima de la fonction de perte $\mathcal{L}(\boldsymbol{\mathcal{W}})$. On note l'apparition d'un facteur ι dans l'équation 3.5, communément appelé taux d'apprentissage (*learning rate*). Ce taux d'apprentissage est souvent choisi avec précaution, car une trop grande valeur ne permettrait pas de converger vers un minima de la fonction de perte $\mathcal{L}(\boldsymbol{\mathcal{W}})$. Des techniques avancées d'entraînement de DNN considèrent l'utilisation d'un planning pour la valeur de ι, par exemple en réduisant la valeur de ce dernier par un facteur < 1 à intervalles réguliers, afin de converger vers des minima de la fonction de perte plus fins.

D'autre part, on distingue plusieurs phases lors de l'entraînement d'un DNN : les phases

d'entraînement et les phases de test. À chacune de ces phases est associée une portion distincte du jeu de données, « ensemble d'entraînement » ou « ensemble de test ». Lors de la phase de test, seule la passe avant est calculée et la performance sur la tâche d'intérêt est rapportée. Ce procédé permet de nous assurer de la performance de généralisation du DNN à des données nouvelles. À noter ici que la fonction de perte $\mathcal{L}$ est souvent une fonction substitut permettant une meilleure optimisation que la tâche d'intérêt finale. Dans le domaine de la classification, on sera par exemple intéressé par une bonne précision de classification (la tâche d'intérêt) rapportée en pourcentage de bonne classification. Or, il est courant d'entraîner un DNN pour une tâche de classification en utilisant une fonction de perte d'entropie croisée. Ainsi, il est d'usage de rapporter la performance sur l'ensemble de test d'un jeu de données pour juger de la performance de généralisation d'un DNN.

On peut également distinguer une phase tierce d'ajustement (*finetuning*). Pour un DNN *pré-entraîné* sur une tâche distincte, souvent plus complexe, on procède à une nouvelle phase d'entraînement, souvent plus courte, sur le jeu de données d'ajustement ou en modifiant le contexte d'exécution. Par exemple, on pourrait réaliser un ajustement *hardware-aware* comme nous le verrons section 3.2.3. On pourrait également pré-entraîner un DNN sur un jeu de données de grande taille tel que ImageNet-1K [32] puis procéder à un ajustement sur le relativement petit jeu de données CIFAR-10 [130] afin d'améliorer les performances sur ce dernier. Enfin, lors d'un ajustement, les paramètres de SGD sont choisi avec soin pour ne pas totalement effacer les effets du pré-entraînement.

En pratique, le jeu de données composé du tuple $(\boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{O}})$ est peu susceptible d'être traité en une passe avant et une passe arrière sur les accélérateurs de DNN modernes de type GPGPU étant donné les importants besoins mémoires que ce choix induirait. Par exemple, les populaires jeux de données d'images CIFAR-10 et CIFAR-100 [130] sont composés de 50,000 images d'entraînement de 32 pixels de côté sur 3 canaux (RVB) et sont aujourd'hui considérés comme des jeux de données jouets. Comme mentionné au paragraphe précédent, on peut souligner de manière plus représentative pour ce problème de mémoire le jeu de données ImageNet-1K [32] composé de 1,281,167 images d'entraînement de taille variées. Pour pallier cette situation, le jeu de données est divisé en mini-lots (*mini-batchs*), des sous-ensembles d'une taille donnée. Par exemple, il est couramment utilisé une taille de mini-lot la plus grande possible permettant de réaliser la passe avant et la passe arrière avec la capacité mémoire disponible de l'accélérateur utilisé. On peut noter que ce procédé n'est pas toujours possible avec la croissance en complexité des DNN modernes et des jeux de données associés : des techniques tierces que nous ne décrirons pas ici sont alors utilisées. Le gradient est ainsi estimé pour chacun des mini-lots de manière indépendante.

Bien qu'il serait possible d'accumuler le gradient sur l'ensemble des mini-lots pour obtenir un estimé plus précis de ce dernier, l'algorithme de descente de gradient stochastique (*stochatic gradient descent*, SGD) propose d'appliquer l'étape de descente de gradient décrit par l'équation 3.5 après chaque mini-lot. L'entraînement d'un DNN est alors organisé en époques pendant lesquelles tous les mini-lots disponibles sont utilisés les uns à la suite des autres pour réaliser une descente de gradient par mini-lot. Ainsi, à la fin d'une époque tous les éléments de l'ensemble d'entraînement du jeu de données auront été utilisés pour réaliser une descente de gradient.

Ce procédé à un effet de régularisation sur le DNN : la performance de généralisation sera bonifiée. De nombreuses autres méthodes de régularisation sont couramment employées. Le but de cette section n'est pas d'en faire une présentation exhaustive, mais nous présentons cependant quelques méthodes qui seront couramment utilisées dans les chapitres à venir. La première est l'utilisation d'augmentations aléatoires sur le jeu de données. En transformant de manière cohérente l'entrée ou la sortie attendue, par exemple pour des images en retournant aléatoirement l'image autour de son axe vertical ou en recadrant l'image, l'entraînement du DNN se retrouve bonifié. Deuxièmement, l'utilisation de terme de régularisation dans la fonction de perte du DNN. Par exemple, il est courant d'employer un terme d'atténuation des pondérations (*weight decay*) de la façon suivante :

$$\mathcal{L}(\boldsymbol{\mathcal{W}}) + \frac{\lambda}{2}\|\boldsymbol{\mathcal{W}}\|\,, \tag{3.6}$$

avec $\|\boldsymbol{\mathcal{W}}\|$ la norme-2 du tenseur de paramètres du DNN $\boldsymbol{\mathcal{W}}$. Dans ce cas, la passe arrière du DNN s'effectue sur la sortie de l'équation 3.6. L'ajout de ce terme d'atténuation des pondérations va naturellement guider les paramètres du DNN vers la valeur 0, un procédé à l'efficacité démontrée pour améliorer les performances de DNN.

Enfin, on peut noter que l'algorithme SGD peut être amélioré par l'utilisation des moments des gradients de la fonction de perte, comme par exemple la méthode des moments de Nesterov [131]. Pour ce faire, on introduit une variable intermédiaire ν permettant de conserver une somme glissante des gradients. La mise à jour des paramètres de l'équation 3.5 devient alors :

$$\nu_{(t)} = \nabla_w \mathcal{L}(w) + \beta\nu_{(t-1)} \tag{3.7}$$

$$w_{(t)} = w_{(t-1)} - \iota\nu_{(t)}\,, \tag{3.8}$$

avec $0 \leq \beta < 1$ un scalaire permettant de maîtriser l'importance donnée à l'historique du gradient dans les mises à jour. On remarque qu'un choix de $\beta = 0$ aboutit à la situation

initiale. L'engouement autour des DNN et des propriétés de l'algorithme d'optimisation sous-jacent ont conduit à l'émergence d'algorithmes dérivés : par exemple le populaire Adam [132], réputé pour accélérer la convergence du DNN lors de son entraînement. En pratique, SGD demeure un algorithme de référence dans l'entraînement de DNN, permettant d'obtenir des résultats marginalement supérieurs dans certains cas de figure.

3.2.3 Entraînement *hardware-aware*

Après l'entraînement d'un DNN vient sa phase d'utilisation. Lors de cette phase, des inférences sont réalisées en grand nombre : des données $\boldsymbol{\mathcal{X}}$ pour lesquelles on ne dispose pas d'étiquettes $\boldsymbol{\mathcal{O}}$ sont présentées au DNN, et la sortie $\boldsymbol{\mathcal{Y}}$ ainsi obtenue est un estimé de $\boldsymbol{\mathcal{O}}$. Or, le contexte d'utilisation peut varier selon les besoins de l'utilisateur. Naturellement, nous aurons besoin d'un DNN dont l'inférence sera peu coûteuse en temps et énergie. Sur un matériel disposant de peu de ressources, ce besoin de rapidité et de frugalité se transforme en contrainte.

Dans la revue de littérature du chapitre précédent, plusieurs méthodes permettant de réduire les besoins d'un DNN furent présentées. Entre autres, nous nous intéressons dans cette section aux méthodes modifiant le contexte d'exécution entre la phase d'entraînement et la phase d'utilisation. C'est par exemple le cas de la quantification, méthode consistant à réduire le nombre de bits alloués pour représenter les nombres du DNN. Plus particulièrement, le nombre de bits alloués aux paramètres du DNN lors de sa phase d'utilisation peut être grandement réduit pour répondre à notre problématique de besoins mémoires trop importants.

Néanmoins, lors de la phase d'apprentissage sur GPGPU une représentation sur 32 ou 16 bits (FP32, FP16, BF16, TF32) est communément utilisée, avec l'objectif d'avoir une granularité importante pour permettre la bonne convergence de l'entraînement du DNN. Or, le passage d'une représentation aussi précise à une représentation sur moins de bit peut entraîner une lourde perte de performance pour le DNN lors de la phase d'utilisation sans action supplémentaire.

En la matière, une méthode plébiscitée est d'entraîner le DNN en simulant les conditions du matériel cible, et donc la quantification, lors de la passe avant : on parle d'entraînement *hardware-aware*. Ceci n'impose pas de modifications sur le format des paramètres w qui demeurent stockés dans un format de plus haute précision. La quantification dispose alors de pas intermédiaires, ou *buffers* invisibles, pour éviter l'explosion de la fonction de perte du DNN à chaque mise à jour des paramètres par descente de gradient, ou inversement d'observer un effet nul lors de la descente de gradient.

Plus précisément, si l'on reprend notre modèle de passe avant de l'équation 3.4, nous pouvons maintenant introduire une opération de quantification $q(\boldsymbol{\mathcal{W}})$. De cette manière, on obtient la sortie

$$\boldsymbol{\mathcal{Y}} = f(\boldsymbol{\mathcal{X}}, q(\boldsymbol{\mathcal{W}}))\,. \tag{3.9}$$

Dans la suite de cette thèse, nous utiliserons notamment une binarisation des paramètres du DNN. Une méthode populaire pour y parvenir est *Binary connect* (BC) [133] considérant que les valeurs binaires permettent de représenter le signe de chaque paramètre du DNN, qui sont alors pris dans l'ensemble $\{-1, 1\}$. Sur le matériel cible, les gains sur la mémoire nécessaire au stockage des paramètres du DNN sont importants par rapport à une représentation à haute précision, mais cela simplifie également les opérations de calculs matriciels en remplaçant les multiplications par des changements de signe et additions. Avec BC, lors de la préparation du DNN sur matériel conventionnel un entraînement *hardware-aware* est utilisé. Pour ce faire, la passe avant est obtenue en utilisant simplement une fonction de signe pour simuler la binarisation des paramètres du DNN comme suit :

$$\boldsymbol{\mathcal{W}}_b = \texttt{sign}(\boldsymbol{\mathcal{W}})\,, \tag{3.10}$$

$$\boldsymbol{\mathcal{Y}} = f(\boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{W}}_b)\,. \tag{3.11}$$

Néanmoins, l'utilisation de cette fonction de quantification introduit une opération supplémentaire dans le graphe de calcul du DNN. Afin de permettre la passe arrière, il est nécessaire de définir une dérivée de q, soit pour notre cas de figure $\frac{d\boldsymbol{\mathcal{W}}_b}{d\boldsymbol{\mathcal{W}}}$. Une approche populaire pour certaines fonctions stochastiques ou de discrétisation, et utilisée par BC, est l'utilisation de l'estimateur de gradient *straight-through* [134] (estimateur direct). Cet estimateur ignore simplement la fonction de quantification, et permet le transit des gradients de l'aval du DNN vers l'amont sans les modifier. On obtient ainsi

$$\frac{d\boldsymbol{\mathcal{W}}_b}{d\boldsymbol{\mathcal{W}}} \triangleq 1\,. \tag{3.12}$$

De plus, [133] applique une troncature aux paramètres $\boldsymbol{\mathcal{W}}$ après chaque mise à jour de SGD pour les ramener dans l'intervalle $[-1, 1]$, ce qui est sans impact sur leur représentation binarisée par l'opération `sign`. Par ailleurs, nous améliorons BC selon les suggestions de [56] : la sortie d'une couche dont les paramètres sont binarisés est mise à l'échelle par une constante, comme par exemple celle utilisée pour initialiser un DNN aléatoirement [135].

Enfin, on peut remarquer que l'entraînement simulant les conditions du matériel cible, ou *hardware-aware*, ne se limite pas à la quantification mais également à répliquer plus ou moins

précisément l'ensemble des conditions du matériel cible. En présence d'un matériel bruité comme les mémoires sous-alimentées ou les memristors, il est ainsi souhaitable de procéder à une simulation du bruit matériel lors de l'entraînement du DNN [109]. Pour ce faire, nous introduisons une fonction de bruitage n venant s'appliquer à la représentation quantifiée du DNN de telle sorte que

$$\boldsymbol{\mathcal{W}}_q = q(\boldsymbol{\mathcal{W}})\,, \tag{3.13}$$

$$\tilde{\boldsymbol{\mathcal{W}}}_q = n(\boldsymbol{\mathcal{W}}_q)\,, \tag{3.14}$$

$$\boldsymbol{\mathcal{Y}} = f(\boldsymbol{\mathcal{X}}, \tilde{\boldsymbol{\mathcal{W}}}_q)\,. \tag{3.15}$$

De la même manière que pour la fonction de quantification, il est nécessaire de définir une dérivée pour $\frac{d\tilde{\boldsymbol{\mathcal{W}}}_q}{d\boldsymbol{\mathcal{W}}_q}$. Un choix valide consiste encore à utiliser l'estimateur *straight-through*, de telle sorte que

$$\frac{d\tilde{\boldsymbol{\mathcal{W}}}_q}{d\boldsymbol{\mathcal{W}}_q} \triangleq 1\,. \tag{3.16}$$

Avec cet entraînement *hardware-aware*, la performance du DNN une fois déployée sur le matériel cible est améliorée.

3.2.4 Troncature et robustesse

La présence de bruit dans les calculs du DNN n'est pour autant pas solutionnée avec un entraînement *hardware-aware*. Ce dernier améliore la rétention de performance du DNN en présence de bruit sans modifier l'architecture du DNN ou le matériel utilisé : le DNN obtenu est plus robuste. Or, des méthodes supplémentaires peuvent être déployées pour bonifier la robustesse, comme nous avons pu l'apercevoir dans la revue de littérature du chapitre précédent. En particulier, l'utilisation de troncature basée sur la magnitude (*clipping*) permet d'améliorer significativement la robustesse d'un DNN en incitant l'entraînement de ce dernier à introduire de la redondance dans les résultats intermédiaires [105, 136, 137].

Pour ce faire, on introduit un paramètre de magnitude maximale c et l'opération $\texttt{clip}(x, c)$ sur une entrée x définie comme suit :

$$\texttt{clip}(x, c) = \min(\max(x, -c), c)\,, \tag{3.17}$$

restreignant le domaine de $x \in [-c, c]$ et n'ayant aucun effet si c'est déjà le cas. Cette opération est utilisée dans une version adaptée de la descente de gradient que nous avions

équation 3.5 de la manière suivante :

$$w_{(t)} = \texttt{clip}(w_{(t-1)} - \iota \nabla_w \mathcal{L}(w),\, c)\,. \tag{3.18}$$

3.2.5 Régularité de la fonction de perte

Néanmoins, l'utilisation de ces méthodes d'entraînement *hardware-aware* et de *clipping* pour contrer un bruit matériel introduit un compromis entre la performance du DNN et la robustesse atteinte, réduisant la première pour améliorer la seconde. Pour pallier cette situation, le chapitre 5 propose une approche basée sur l'optimisation de DNN conscient de la régularité de la fonction de perte, notion que nous allons maintenant définir.

Le bénéfice usuel d'une plus grande régularité dans la fonction de perte du DNN est associé dans la littérature à une augmentation de la performance de généralisation du DNN [138–141]. D'abord introduit par [142] pour tirer parti de cette observation, *Sharpness-Aware Minimization* (SAM) est une méthode alternative de calcul des gradients pour un DNN visant à maximiser la régularité de la fonction de perte $\mathcal{L}$ utilisée, tout en optimisant simultanément cette dernière. L'objectif est d'améliorer la performance de généralisation d'un DNN en convergeant vers des zones de la fonction de perte localement plus plates, et donc plus régulières. À noter que l'on peut observer l'emploi de l'expression « paysage » de la fonction de perte du DNN (*loss landscape*) dans la littérature. Dit autrement, pour un paramètre w et son voisinage, on cherche à trouver une région de faible valeur pour le paysage de la fonction de perte $\mathcal{L}(w)$.

La régularité peut être définie comme une mesure du changement relatif dans la fonction de perte du DNN avec l'ajout d'une perturbation sur les paramètres w du DNN. Ainsi, un DNN a une forte régularité quand l'ajout d'une perturbation sur ses paramètres a un impact moins fort sur la valeur de la fonction de perte que la même perturbation affectant un DNN d'architecture similaire observant une dégradation plus marquée de la fonction de perte. Plus formellement, pour une perturbation aléatoire ϵ telle que :

$$\tilde{w} = w + \epsilon\,, \tag{3.19}$$

on obtient une valeur la fonction de perte du DNN $\mathcal{L}(\tilde{w})$. On suppose que la direction aléatoire de ϵ induit $\mathcal{L}(w) \leq \mathcal{L}(\tilde{w})$.

La régularité est alors définie par la différence

$$\mathcal{L}(\tilde{w}) - \mathcal{L}(w)\,, \tag{3.20}$$

un DNN plus régulier observant une valeur plus faible de cette équation. À noter que cette définition de la régularité fait écho aux problématiques de robustesse que nous avons abordé à la section 3.2.4. C'est la motivation des travaux du chapitre 5.

Plus concrètement, on peut comparer deux DNN avec la même architecture et atteignant une valeur de $\mathcal{L}(w)$ similaire pour un paramètre w de ces deux DNN, soit $w^{(1)}$ et $w^{(2)}$. Pour un même ϵ, on compare la valeur de $\mathcal{L}$ et l'on observe

$$\mathcal{L}(w^{(1)+\epsilon}) < \mathcal{L}(w^{(2)} + \epsilon)\,. \tag{3.21}$$

Sur la base de cette observation, on conclut que $w^{(1)}$ est un DNN plus régulier que $w^{(2)}$.

Cet exemple est illustré figure 3.2.

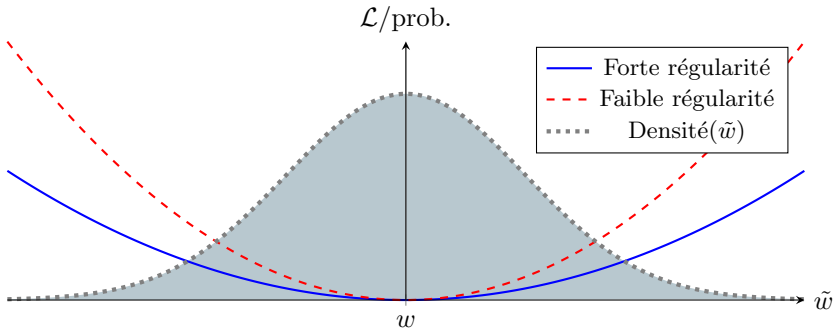


FIGURE 3.2 La robustesse : un sous-produit de la régularité de la fonction de perte ? On observe un paramètre w perturbé par une fonction de bruit d'allure gaussienne pour obtenir $\tilde{w}$. La fonction de perte $\mathcal{L}$ est plus impactée pour un même $\tilde{w}$ avec un réseau ayant une plus faible régularité (en rouge) comparativement à un réseau ayant une meilleure régularité (en bleu). Dans ce cas de figure, le réseau avec une plus grande régularité est également plus robuste au bruit gaussien sur w.

Avec l'amélioration notable de la performance de généralisation atteinte par SAM, de nombreux travaux se sont intéressés à augmenter l'efficience [143–147], la performance [148–150] ou encore la compréhension [151] de la méthode. Par ailleurs, SAM fut étendu à des usages spécifiques tels que la quantification [152] ou encore pour le traitement des jeux de donnés déséquilibrés (*data imbalance*) [153]. Enfin, plusieurs travaux [150, 154] ont mis en lumière l'importance d'une mesure de la régularité invariante à l'échelle des paramètres du DNN, notamment dans le cas de la robustesse d'un DNN à des données antagonistes [155]. Plusieurs travaux [149, 156] mettent en évidence un lien entre la régularité de la fonction de perte et la robustesse à des perturbations antagonistes du DNN. La régularité de la fonction de perte d'un DNN robuste est ainsi étudiée [155] dans le contexte d'un DNN entraîné dans un contexte antagoniste avec des données bruitées [157]. Le problème du sur-apprentissage

(*overfitting*) robuste [158] sous le prisme de la régularité de la fonction de perte y est définit : ce phénomène intervient lorsqu'un DNN est robuste aux données antagonistes observées lors de l'entraînement, mais avec une faible généralisation à de nouvelles données antagonistes lors de la phase de test.

SAM

En particulier, SAM propose de reformuler l'entraînement d'un DNN comme un problème d'optimisation *min-max* de la forme

$$\min_{\boldsymbol{\mathcal{W}}} \max_{\|\boldsymbol{\epsilon}\| \leq \rho} \mathcal{L}(\boldsymbol{\mathcal{W}} + \boldsymbol{\epsilon}) \,, \tag{3.22}$$

un terme d'atténuation des pondérations étant omis par simplicité. On remarque que le domaine de l'opérateur max, ou de manière équivalente région de maximisation, est une sphère définie par le tenseur $\boldsymbol{\epsilon}$ en haute dimension dont le rayon, mesuré par la norme-2, est limité à la valeur ρ. Le choix de la norme-2 fut démontré supérieur à d'autres choix de norme expérimentalement [142]. Avec cette formulation, une valeur approximative du $\boldsymbol{\epsilon}$ optimal de l'équation 3.22, $\boldsymbol{\epsilon}^{\star}_{\text{SAM}}$ fut proposée :

$$\boldsymbol{\epsilon}^{\star}_{\text{SAM}}(\boldsymbol{\mathcal{W}}) = \rho \frac{\nabla \mathcal{L}(\boldsymbol{\mathcal{W}})}{\|\nabla \mathcal{L}(\boldsymbol{\mathcal{W}})\|} \,. \tag{3.23}$$

Cette valeur de perturbation optimale $\boldsymbol{\epsilon}^{\star}$, obtenue par un développement en série de Taylor au premier degré du sous-problème de maximisation de l'équation 3.22, peut être interprétée comme la perturbation au pire des cas pour la fonction de perte du DNN.

On remarque ensuite que la minimisation de $\mathcal{L}(\boldsymbol{\mathcal{W}} + \boldsymbol{\epsilon})$ s'effectue sur les paramètres $\boldsymbol{\mathcal{W}}$, avec la descente de gradient traditionnelle que nous avons abordé plus tôt. Or, lors de l'étape de maximisation de l'équation 3.23 le gradient $\nabla \mathcal{L}(\boldsymbol{\mathcal{W}})$ nécessite d'être évalué : SAM induit donc un budget calculatoire plus élevé avec une passe avant et une passe arrière supplémentaire. Afin d'atténuer les effets associés aux surplus de calculs, les auteurs proposent de recourir à un entraînement distribué sur plusieurs machines [142].

Par ailleurs, une métrique de m-irrégularité (m-*sharpness*) issue de l'équation 3.22 est proposée. Cette dernière vise à estimer de manière pratique la régularité d'un DNN sur un jeu de données pour lequel n mini-lots S de taille m sont observés en séquence, et pour lesquels il serait donc peu pratique de calculer un $\boldsymbol{\epsilon}$ global. L'opération de maximisation de la fonction de perte $\mathcal{L}$ est donc répétée pour chaque mini-lot évalué $\mathcal{L}_S$ afin d'obtenir la m-irrégularité

comme suit :

$$\frac{1}{n}\sum_{S}\frac{1}{m}\max_{\|\boldsymbol{\epsilon}\|\leq\rho}\mathcal{L}_S(\boldsymbol{\mathcal{W}}+\boldsymbol{\epsilon})-\mathcal{L}_S(\boldsymbol{\mathcal{W}}). \tag{3.24}$$

Les bienfaits de SAM pour la performance de généralisation d'un DNN, et donc de l'intérêt d'avoir une fonction de perte régulière, est montré expérimentalement dans [142]. Cependant, on remarque que la mesure de la régularité de la fonction de perte de SAM n'est pas indépendante de l'échelle des paramètres du DNN. Cette particularité ouvre la voie à des manipulations de la régularité calculée en changeant l'échelle des paramètres du DNN sans pour autant que les prédictions en sortie du DNN soient altérées [154, 155].

SAM adaptatif (ASAM)

Afin de pallier cette caractéristique, une version adaptative de SAM fut proposée par [150]. En introduisant un opérateur de mise à l'échelle sans impacts sur la fonction de perte du DNN, ASAM atteint une définition de régularité adaptative invariante à l'échelle des paramètres du DNN, au contraire de SAM. Cela se traduit par le problème d'optimisation *min-max* suivant :

$$\min_{\boldsymbol{\mathcal{W}}}\max_{\|\boldsymbol{\epsilon}/|\boldsymbol{\mathcal{W}}|\|\leq\rho}\mathcal{L}(\boldsymbol{\mathcal{W}}+\boldsymbol{\epsilon}), \tag{3.25}$$

où $|\boldsymbol{\mathcal{W}}|$ représente la valeur absolue du tenseur de paramètre $\boldsymbol{\mathcal{W}}$. Le domaine de l'opérateur max demeure le tenseur $\boldsymbol{\epsilon}$, mais ce dernier est transformé par le produit avec $|\boldsymbol{\mathcal{W}}|$ qui demeure fixe. Avec ASAM, des voisinages de tailles distinctes sont utilisés pour chaque paramètre, en fonction de leur magnitude. L'intuition ici est que des paramètres de forte magnitude peuvent subir des perturbations plus importantes que des paramètres de faible magnitude. Cette formulation de la régularité adaptative impacte également la forme du voisinage qui devient ellipsoïdale au lieu de sphérique. La perturbation optimale $\boldsymbol{\epsilon}^*_{\text{ASAM}}$ est alors définie par

$$\boldsymbol{\epsilon}^*_{\text{ASAM}}(\boldsymbol{\mathcal{W}})=\rho\,\frac{\boldsymbol{\mathcal{W}}\circ\boldsymbol{\mathcal{W}}\,\nabla\mathcal{L}(\boldsymbol{\mathcal{W}})}{\||\boldsymbol{\mathcal{W}}|\nabla\mathcal{L}(\boldsymbol{\mathcal{W}})\|}\,. \tag{3.26}$$

En pratique, la régularité adaptative introduite par ASAM est plus corrélée avec la performance de généralisation et mène à des entraînements de DNN dont la convergence est facilitée en utilisant de plus grands voisinages pour les paramètres de grande magnitude.

3.3 Modèles de programmation du matériel

Nous définissons dans un premier temps les éléments communs à toutes les plateformes matérielles envisagées : les mémoires sous-alimentées et les memristors.

Dans chacun des cas, nous aurons besoin d'un paramètre matériel permettant d'installer un compromis énergie-fiabilité. Nous souhaiterons également mesurer un degré de fiabilité matériel de façon à pouvoir simuler le comportement de la mémoire soit lors d'un entraînement *hardware-aware* soit lors de l'évaluation de l'ensemble de test. Enfin, nous souhaiterons mesurer le niveau d'énergie de la mémoire.

3.3.1 Modèle de mémoires sous-alimentées

Nous considérons qu'un DNN pourra être programmé sur un système numérique avec une mémoire fonctionnant en NTC. La réduction de la tension d'alimentation des cellules mémoires fera apparaître des pannes à un taux p lors de la lecture des paramètres du DNN. Nous chercherons à minimiser la consommation d'énergie totale de la mémoire servant à stocker le DNN E, tout en maximisant la performance de ce dernier.

Selon des observations expérimentales sur des circuits SRAM pour plusieurs technologies CMOS, la relation entre l'énergie et la fiabilité des cellules mémoires suit un modèle exponentiel [8, 11] que l'on peut représenter comme suit pour $p > 0$:

$$\eta(p) = -\frac{\log(p)}{a}\,, \tag{3.27}$$

avec η la consommation d'énergie moyenne et a un paramètre dépendant de la technologie utilisée. De plus, nous supposons que les bits stockés sont indépendamment affectés par les erreurs. En pratique, cette supposition peut être approchée même en présence d'erreurs corrélées dans les cellules mémoires en entrelaçant le stockage des bits.

On remarque que l'équation 3.27 n'est pas définie pour $p = 0$, puisque la probabilité d'erreur ne peut jamais atteindre cette valeur en pratique dans un matériel moderne. Cependant, nous considérons par souci de simplicité que les implémentations usuelles disposent d'une configuration permettant d'obtenir $p = 0$, avec une consommation d'énergie $\eta(0) \triangleq 1$.

Pour programmer un DNN sur cette mémoire numérique, un format de quantification ζ est à préciser : ce dernier définit le nombre de bits alloué par paramètre du DNN. Entre autres, nous considérerons qu'un DNN peut être quantifié sur 16 bits sans heurter la performance de généralisation observée lors de l'entraînement du DNN. Nous considérerons par ailleurs un format alternatif sur 1 bit avec BC comme présenté section 3.2.3 pour programmer le DNN sur la mémoire en NTC.

Ainsi, un paramètre d'un bit $\tilde{w_b}$ lu sur cette mémoire non fiable peut être décrit avec une

modèle de canal binaire symétrique (*Binary Symmetric Channel*, BSC), défini comme

$$\tilde{w_b} = \texttt{BSC}(w_b, p) = \begin{cases} -w_b & \text{avec probabilité } p, \\ w_b & \text{avec probabilité } 1-p, \end{cases} \tag{3.28}$$

pour $0 \leq p \leq \frac{1}{2}$. Étant donné que la mémoire est modélisée par un canal de communication, nous pouvons nous référer à p comme un niveau de bruit sur le canal. Nous remarquons que sous cette formulation la lecture de la mémoire est un processus de Bernoulli.

On obtient ainsi

$$\tilde{\boldsymbol{\mathcal{W}}}_b = \texttt{BSC}(\boldsymbol{\mathcal{W}}_b, p)\,, \tag{3.29}$$

qui peut être utilisé dans la passe avant de l'équation 3.15.

Or, si la dérivée $\frac{d\tilde{\boldsymbol{\mathcal{W}}}_b}{d\boldsymbol{\mathcal{W}}}$ est définie par l'estimateur *straigth-through* comme proposé équation 3.16, il n'en est pas de même pour $\frac{d\tilde{\boldsymbol{\mathcal{W}}}_b}{dp}$. C'est le propos de l'algorithme *Layerwise noise maximization* présenté au chapitre 4.

Pour aller plus loin dans notre modélisation et exprimer la consommation d'énergie totale de la mémoire servant à stocker le DNN, notée E, nous supposons que le tenseur de paramètre $\boldsymbol{\mathcal{W}}^{(i)} \forall i \in \{1, .., \ell\}$ est stocké sur une cellule mémoire disposant de son propre compromis énergie-fiabilité p_i. Pour un réseau composé de ℓ couches, nous obtenons donc le vecteur $\mathbf{p}$ de taille ℓ en concaténant les $p_i \forall i \in \{1, .., \ell\}$. Ceci nous mène à la définition de E suivante :

$$E(\mathbf{p}) = \zeta * \sum_{i=1}^{\ell} \eta(p_i) \texttt{count}(\boldsymbol{\mathcal{W}}^{(i)})\,, \tag{3.30}$$

où $\texttt{count}(\boldsymbol{\mathcal{W}}^{(i)})$ représente le nombre de paramètres dans la i-ème couche du DNN.

Résumé du modèle de mémoires sous-alimentées

Nous obtenons ainsi les trois éléments composant notre modèle de mémoire sous-alimentée :

- p : le paramètre matériel permettant de réaliser un compromis énergie-fiabilité,
- le degré de fiabilité matériel, directement contrôlé par p et simulé par un BSC,
- enfin, le niveau d'énergie de la mémoire tel que mesuré par l'équation 3.30.

3.3.2 Modèle de memristors

D'autre part, nous considérons qu'un DNN sera programmé sur un crossbar de memristors effectuant du calcul en mémoire dans le domaine analogique. Ce matériel émergent constitue

une direction prometteuse pour la conception de matériel spécialisé dans l'accélération de DNN avec un faible budget d'énergie [109,159]. On cherchera à minimiser la puissance dissipée par les crossbars de memristors opérant le DNN, notée P, tout en maximisant la performance de ce dernier en présence de bruit matériel.

Format de calcul des couches linéaires et de convolution

En particulier, les opérations de produit vecteur-matrice constituent une cible privilégiée pour le calcul sur memristor lorsque ces derniers sont organisés en crossbar comme nous l'avons observé dans le chapitre 2. C'est le cas des opérations des couches linéaires et de convolutions que nous avons présenté section 3.2.1.

Or, le modèle de calculs des équations 3.1 et 3.3 ne se prête pas immédiatement à un simple produit vecteur matrice. Nous pouvons les reformuler pour nous y conformer.

Pour l'équation 3.1, le problème est que le vecteur optionnel de biais additif $\mathbf{b}$ est distinct du produit vecteur matrice. Deux options se présentent à l'utilisateur : déléguer le traitement de l'addition à la partie numérique du système accueillant le memristor, ou fusionner le biais dans l'opération vecteur matrice. Le premier choix n'est pas dénué de sens : une opération d'addition à un faible coût énergétique sur matériel numérique, relativement à une multiplication, et permet de retirer une source de bruit sur les résultats du DNN. Le second choix s'effectue en concaténant le vecteur de biais dans la première dimension de la matrice de paramètres $\mathbf{W}$, comme nous l'avions fait pour obtenir l'équation 3.3 pour les convolutions, et en concaténant une valeur un à l'entrée $\mathbf{x}$. On obtient alors une matrice de paramètres de dimensions $(W, H+1)$. Nous supposons que ce second choix est effectué, dans le but de réduire la consommation énergétique du système opérant le DNN.

Pour l'équation 3.3, le problème est que l'opération est un produit matrice-matrice. Nous considérons deux possibilités pour reformuler cette opération en produit vecteur-matrice prêt à l'implémentation sur crossbar de memristor : *unrolled-linear* (linéaire déroulé, UL) et *unfold-repeat* (déplié-répété, UR).

La première approche, UL, propose de reformuler la couche de convolution en une couche linéaire. Pour ce faire, le tenseur d'entrée $\boldsymbol{\mathcal{X}}$ de la convolution (avant sa projection matricielle) est « déroulé » en un vecteur $\mathbf{x}$ de taille $(C^i(H+p^2)(W+p^2)$ d'une part. D'autre part, le tenseur de paramètres $\boldsymbol{\mathcal{W}}$ est projeté dans une matrice $\mathbf{W}_{\text{UL}}$ de dimensions $(C^i(H+p^2)(W+p^2)$, $C^oH^oW^o)$ où les paramètres de convolution sont répétés dans un motif clairsemé. On obtient ainsi une matrice creuse (*sparse*). Avec l'utilisation du biais additif, une ligne supplémentaire est ajoutée à la matrice $\mathbf{W}_{\text{UL}}$, cette ligne contenant le vecteur $\mathbf{b}$ dans un

motif répété, et un coefficient de valeur 1 est concaténé à la fin du vecteur $\mathbf{x}$. L'avantage principal de cette approche UL est que les calculs sont effectués en une opération, au prix de la taille prohibitive de la matrice $\mathbf{W}_{\text{UL}}$. Selon les paramètres de la couche de convolution et la taille de l'entrée $\mathbf{x}$, l'approche UL peut rapidement devenir impossible à implémenter sur un crossbar avec la technologie actuelle de par la taille de la matrice $\mathbf{W}_{\text{UL}}$. Par ailleurs, ce problème de taille et l'utilisation d'une matrice creuse introduit une problématique toute autre : plus la matrice $\mathbf{W}_{\text{UL}}$ sera creuse et contiendra de lignes, plus il y aura de bruit sur les résultats obtenus. L'approche UL est illustrée figure 3.3.

La deuxième approche, UR, tente de répondre à cette problématique en conservant la matrice $\mathbf{W}$ de l'équation 3.3. Dans ce cas, on programme sur le crossbar $\mathbf{W}_{\text{UR}} = \mathbf{W}$ une matrice pleine (*dense*) de taille minimale contenant les paramètres de la couche de convolution [109, 160]. Avec cette taille minimale, l'approche UR permet de répondre aux problématiques soulevées par l'approche UL de taille et de bruit sur les résultats. Or, ce changement s'effectue au prix de l'augmentation du nombre d'opérations à réaliser : avec l'approche UR, chaque utilisation du crossbar permet de calculer les canaux de sortie d'un seul pixel, ou de manière équivalente à traiter un *patch* de l'entrée. On doit donc faire autant d'utilisation du crossbar qu'il y a de *patch* pour l'approche UR, en comparaison avec l'unique opération de l'approche UL. Cependant, il est possible de mitiger l'impact en termes de temps d'usage de l'approche UR, sous réserve de crossbars supplémentaires disponibles, en répliquant la matrice $\mathbf{W}_{\text{UR}}$ sur plusieurs crossbars. Chacun de ces crossbars traite alors en parallèle un *patch*, donnant un facteur d'accélération linéaire en le nombre de crossbar utilisé. On remarque que chaque usage d'un crossbar pourra mener en pratique à une réalisation distincte du bruit matériel, dont nous détaillerons le modèle plus loin. Or, lors de la simulation des calculs sur memristors sur un matériel traditionnel tel qu'un GPGPU, la représentation de ces aléas matériel d'utilisation à utilisation serait peu pratique d'un point de vue du temps d'exécution. Ainsi, nous utiliserons une stratégie de partage du bruit échantillonné entre tous les *patchs* d'une convolution lors d'une passe avant. Cette seconde approche UR est illustrée figure 3.4.

Sur une architecture de type VGG [161] avec le jeu de données CIFAR-10, [159] démontre l'intérêt de l'implémentation de convolution UR pour maintenir une MSE faible entre la sortie voulue et la sortie de crossbar, en comparaison avec l'implémentation UL.

Calcul d'opérations tierces

En dehors de ces opérations de convolutions et linéaires, qui composent la majeure partie des paramètres d'un DNN, d'autres opérations paramétriques et non paramétriques composent les DNN modernes comme mentionné section 3.2.1. Des manipulations algébriques de ces

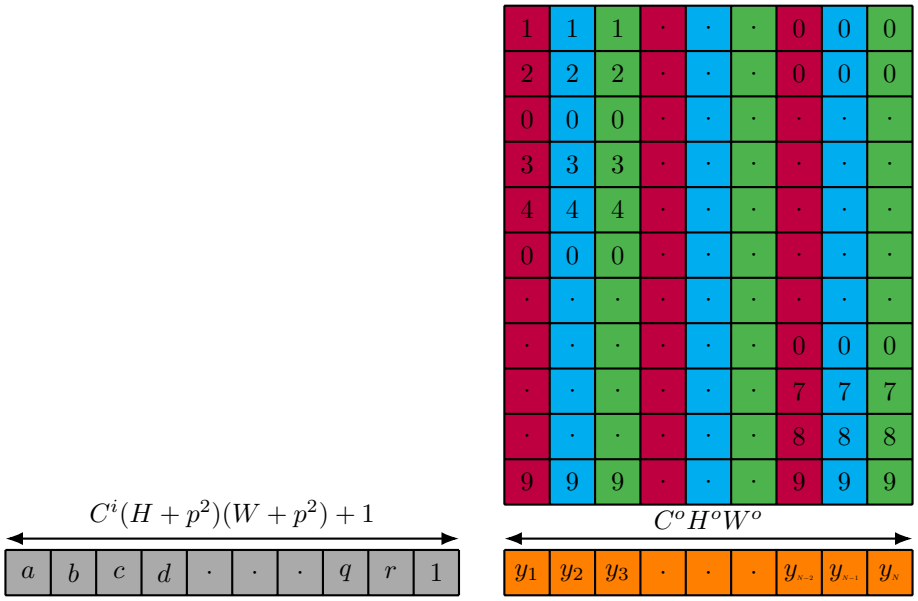


FIGURE 3.3 Réalisation d'une convolution avec l'approche *Unrolled-Linear*.

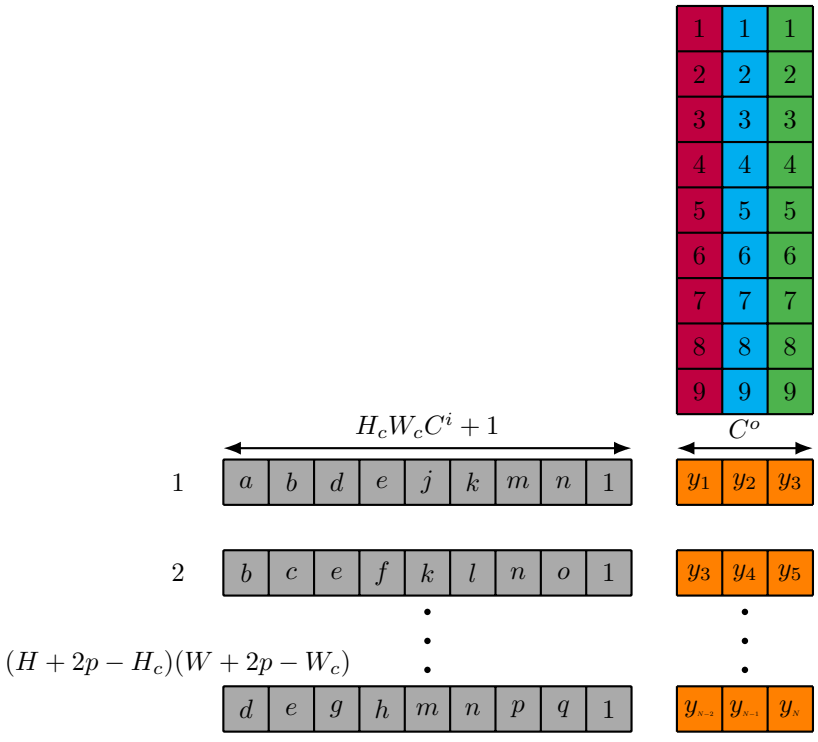


FIGURE 3.4 Réalisation d'une convolution avec l'approche *Unfold-Repeat*.

opérations permettent dans certains cas d'obtenir une formulation vecteur-matrice, et donc déléguer plus de calculs du DNN aux crossbars de memristors dans le but de réduire encore l'énergie allouée à l'inférence.

Une autre possibilité, tirée du processus d'optimisation d'un DNN pour son déploiement, consiste à fusionner les couches linéaires en séquences. C'est par exemple le cas des opérations de normalisation par lot (*batch-normalization*, BN) qui sont usuellement fusionnées avec une couche linéaire ou de convolution adjacente. Ce procédé de fusion permet de transformer deux opérations en une et est donc obligatoire pour tout usage sérieux de DNN : on gagne ainsi de la mémoire si l'opération fusionnée requiert des paramètres, du temps de calcul en réduisant les transactions mémoires et opérations, et *in fine* de l'énergie en combinant tous ces facteurs. Cette approche sera privilégiée dans les travaux présentés au chapitre 6. Au chapitre 5, la problématique étant l'entraînement de DNN on considère que ces opérations linéaires hors couches linéaires et de convolution sont réalisées sur un matériel numérique fiable.

On considérera par ailleurs que les opérations de réduction (somme, moyenne, etc.) seront réalisées sur matériel numérique fiable de par leur complexité réduite en comparaison des lourds produit vecteur-matrice. Cela comprend également les couches de *pooling* dont l'usage est répandu pour les DNN utilisés sur des tâches de vision.

D'autre part, certaines opérations d'un DNN moins adaptées au calcul sur memristor : c'est le cas par exemple des opérations non-linéaires, comme les activations d'un DNN, ou encore d'opérations à deux opérandes dynamiques (par exemple, certains calculs de l'architecture Transformers). Pour plus d'explications sur ce dernier point, l'usage d'un memristor tel que décrit consiste à programmer dans un crossbar une valeur connue en avance : une des opérandes est donc fixe par construction. Ainsi, pour ces deux types d'opérations nous considérons que le calcul sera opéré par un matériel numérique fiable [25, 162].

Programmation

Pour calculer les couches linéaires et de convolution d'un DNN, nous considérons donc l'utilisation de crossbars de memristors. Par mimétisme avec les notations d'une couche linéaire, directement transposable sur memristor, nous supposons que ce dernier est représenté par une matrice $\mathbf{W}$ de taille (H, W). Un vecteur $\mathbf{x}$ de taille (H) est présenté aux lignes du crossbar, transformé dans le domaine analogique, et sert de signal pour toutes les colonnes du crossbar. En sortie de ces colonnes, un vecteur $\mathbf{y}$ de taille (W) est lu.

Intéressons-nous maintenant aux memristors composant le crossbar, pour chaque ligne et

chaque colonne composant ce dernier. Pour représenter les paramètres w du DNN, ils sont programmés avec une valeur de conductance g. Grâce à la loi d'Ohm et la loi des nœuds de Kirchhoff, la tension en sortie d'un memristor est donnée par la conductance programmée du memristor multipliée par la tension en entrée de la ligne du crossbar. Le produit individuel réalisé par chaque memristor est ensuite sommé le long de chaque colonne. Enfin, un amplificateur à transimpédance (*transimpedance amplifier*, TIA) permet de convertir le courant en tension à la fin de chaque colonne [163]. Ces opérations ont lieu en parallèle pour chaque ligne et chaque colonne du crossbar ce qui rend l'opération de calcul vecteur-matrice très succincte. La figure 2.3 illustre cette architecture (adapté de [164]). La sortie y_j de la j-ème colonne est donnée par

$$y_j = r \sum_{i=1}^{H} g_{i,j} x_i \,, \tag{3.31}$$

où x_i est la tension en entrée de la ligne i, $g_{i,j}$ la conductance du memristor pour la ligne i et la colonne j du crossbar et r est la résistance de réponse du TIA. Nous supposons par la suite qu'un TIA idéal est utilisé pour nous concentrer sur les variations liées au memristor, soit $r = 1$.

En pratique, la valeur de conductance g est limitée par les capacités physiques de la technologie de memristor considérée. Ainsi, on suppose que la conductance observable est un réel positif comprit dans l'intervalle $[g_{\min}, g_{\max}]$. Or, la puissance dissipée par les memristors dépend de la magnitude des conductances utilisées : une conductance plus haute mène à une plus grande puissance. À l'inverse, une conductance plus faible nécessitera moins de puissance, et donc d'énergie, mais augmentera l'impact relatif du bruit matériel sur les calculs produits. Nous pouvons donc introduire un compromis énergie-fiabilité contrôlable par le choix d'une plage de conductance restreint. Pour ce faire, nous utilisons une variable $g_u \in]g_{\min}, g_{\max}]$, de telle sorte que l'intervalle de conductance utilisé devient $[g_{\min}, g_u]$. La variable g_u est nommée paramètre de mise à l'échelle des memristors.

Par ailleurs, les conductances g sont physiquement restreintes aux réels positifs, ce qui n'est pas une hypothèse réaliste pour les paramètres $\mathbf{W}$ du DNN qui sont couramment composés de valeurs positives et négatives. Pour résoudre ce problème, nous supposons que deux crossbars sont utilisés pour stocker et calculer $\mathbf{W}$: un pour les valeurs positives $g^{(+)}$ et un pour les valeurs absolues des valeurs négatives $g^{(-)}$. Les paramètres ayant le signe opposé sont fixés à la plus petite valeur de conductance admise par le crossbar, $g_{\min}$. Dans cette formulation, deux colonnes de crossbar servent à calculer en parallèle le résultat d'une colonne j de la matrice $\mathbf{W}$. La valeur négative résultante est ensuite soustraite de la valeur positive pour

obtenir la sortie y comme suit :

$$y_j = \sum_{i=1}^{H} g_{i,j}^{(+)} x_i - \sum_{i=1}^{H} g_{i,j}^{(-)} x_i \,, \tag{3.32}$$

que ce soit par l'usage d'un amplificateur différentiel ou sur un matériel numérique fiable.

Il est maintenant nécessaire de transformer linéairement les paramètres du DNN w depuis leur domaine d'origine vers le domaine de conductance utilisé pour obtenir la valeur de conductance souhaitée g avant de procéder à l'étape de programmation. Pour ce faire, une étape de mise à l'échelle est utilisée. Les paramètres du DNN sont d'abord normalisés dans l'intervalle $[0, 1]$ puis projetés dans l'intervalle de conductance utilisé. On obtient ainsi g tel que

$$g = \frac{|w|(g_u - g_{\min})}{\|\mathbf{W}\|_\infty} + g_{\min} \,, \tag{3.33}$$

avec $\|\mathbf{W}\|_\infty$ la norme-infinie de la matrice $\mathbf{W}$, soit la valeur maximale de la matrice $\mathbf{W}$.

On peut également obtenir la valeur de conductance souhaitée pour programmer les crossbars positives et négatives en explicitant le rôle de $|w|$ tel que :

$$g^{(+)} = \frac{\max(0, w)\,(g_u - g_{\min})}{\|\mathbf{W}\|_\infty} + g_{\min} \tag{3.34}$$

$$g^{(-)} = \frac{|\min(0, w)|\,(g_u - g_{\min})}{\|\mathbf{W}\|_\infty} + g_{\min} \,. \tag{3.35}$$

À noter qu'en sortie des crossbars, l'opération inverse de remise à l'échelle est appliquée afin d'obtenir le résultat originel. On remarque également que plusieurs choix sont possibles pour déterminer g_u : pour l'entièreté du DNN, pour chaque couche ℓ, ou encore pour chaque colonne de $\mathbf{W}$. De la même manière, on pourrait choisir un critère de normalisation au dénominateur de l'équation 3.33 basé sur l'entièreté du DNN ou par colonne de $\mathbf{W}$ plutôt que sur $\mathbf{W}$. Sur une architecture de type VGG [161] avec le jeu de données CIFAR-10, [159] démontre l'intérêt d'une assignation par couche.

Modèle de bruit matériel

Plusieurs aspects physiques peuvent conduire les calculs réellement produits par le crossbar à différer du cas idéal présenté par l'équation 3.31. En pratique, les valeurs de conductance utilisées lors des calculs $\tilde{g}$ diffèrent des valeurs de conductance souhaitées g par des mécanismes de bruit ou de perturbation lors de la programmation ou l'utilisation de la puce [165], par des variations de puce à puce lors de la fabrication des memristors [18,166,167] ou encore

par une dégradation des conductances au fil du temps [168].

Pour plusieurs technologies de memristors, le bruit matériel sur ces valeurs de conductances peut être représenté par un modèle de bruit additif gaussien de moyenne nulle pouvant être adapté aux sources de bruits considérées tel que :

$$\tilde{g} = g + \epsilon^v \,, \tag{3.36}$$

où g est la conductance désirée et ϵ^v est une variable aléatoire gaussienne de moyenne nulle et de variance σ_c^2 [109]. En pratique, σ_c^2 peut varier avec la conductance souhaitée g [164, 166].

Pour représenter ce cas de figure, on peut employer un modèle de bruit gaussien multiplicatif tel que

$$\tilde{g} = g \times \delta \,, \tag{3.37}$$

avec δ une variable aléatoire de moyenne un et de variance σ_c^2. Ce modèle multiplicatif est équivalent à un bruit gaussien additif comme présenté à l'équation 3.36 de moyenne nulle et de variance $g^2\sigma_c^2$, avec la définition supplémentaire de $\epsilon \triangleq 0$ pour le cas particulier de $g = 0$. En somme, l'utilisation du modèle de bruit multiplicatif peut être compris comme un cas particulier du bruit gaussien additif précédemment introduit.

Le modèle de bruit multiplicatif est particulièrement important pour évaluer la robustesse d'un DNN : avec ce modèle, le rapport signal sur bruit du matériel ne dépend pas de la conductance désirée g, et ne peut ainsi pas être amélioré en augmentant g_u. De plus, il permet de simuler efficacement l'effet du bruit sur performance d'un DNN lors de l'évaluation sur GPGPU en appliquant directement aux paramètres w du DNN le modèle de bruit multiplicatif pour obtenir

$$\tilde{w} \triangleq w\delta \,. \tag{3.38}$$

On utilise ensuite la définition de $\tilde{w}$ comme fonction de bruitage n pour l'équation 3.15. Enfin, opérant dans le domaine analogique, on suppose qu'il n'est pas nécessaire de procéder à une opération de quantification q.

On remarque que la simulation de bruit appliquée immédiatement aux paramètres du DNN de l'équation 3.38 ne représente pas totalement le procédé de programmation des memristors, notamment la séparation des valeurs positives et négatives en deux crossbars que nous avons introduit équations 3.34 et 3.35. De plus, l'indépendance à la valeur de g_u du bruit multiplicatif ne permet pas d'optimiser la plage de conductance utile.

Ainsi, nous utiliserons ces deux simulations de bruit selon le contexte.

Puissance dissipée par un crossbar

Dans [159], la puissance théorique P ainsi que l'erreur quadratique moyenne sont estimées pour un crossbar. Cette estimation est obtenue en propageant les moments d'ordre un et deux à travers les couches d'un DNN dans une passe avant adaptée. Ainsi, la puissance est obtenue en sommant l'apport de chaque couche d'un DNN assignée à un crossbar, tout en ignorant les couches opérant sur un matériel fiable.

La bibliothèque Python *MemSE*[1] propose une implémentation de ce procédé. Fonctionnant comme un greffon sur la bibliothèque d'apprentissage profond Pytorch [169], MemSE permet de réécrire à la volée le graphe de calcul d'un DNN pour y intégrer la propagation des moments et de la puissance. La quantification des paramètres du DNN y est configurable, ainsi que le paramètre d'échelle g_u. Cette approche basée sur les moments permet d'estimer précisément l'erreur et la puissance d'un DNN opérant sur crossbar en une passe avant. Cette dernière est plus lourde qu'une passe avant traditionnelle, mais évite de très nombreuses répétitions d'un tirage de Monte-Carlo du bruit sur les paramètres du DNN pour une estimation de précision égale.

Notamment, la propagation du moment d'ordre deux, représenté par un tenseur de covariance, constitue une part importante de la lourdeur de cette approche basée sur les moments. L'empreinte mémoire de ce dernier grandit quadratiquement avec la taille des sorties intermédiaires du DNN : on peut rapidement observer des erreurs de mémoire insuffisante selon l'architecture de DNN employée. Afin d'y pallier, la bibliothèque MemSE implémente une stratégie d'instanciation paresseuse permettant d'atténuer ce problème. Pour ce faire, nous utilisons une propriété de certaines architectures de DNN populaires : les couches successives tendent à compresser l'information en des sorties intermédiaires de plus en plus petites. La taille du tenseur de covariance étant proportionnelle à la taille de ces sorties intermédiaires, ce problème de mémoire serait donc circonvenu aux premières couches du DNN. De plus, le tenseur de covariance est initialisé à zéro en tous points. En utilisant ces deux propriétés, nous représentons ce tenseur par sa taille jusqu'à ce qu'il soit nécessaire de l'instancier pour y stocker des valeurs de covariance. Ceci retarde l'allocation mémoire problématique au minimum à la deuxième couche du DNN, permettant potentiellement d'éviter les erreurs de mémoire insuffisante. De plus, cela simplifie les calculs de la première couche et bénéficie donc au temps d'exécution. Ces considérations ont permis l'application de l'approche basée sur les moments sur le jeu de données CIFAR-10 [130] avec une architecture de DNN de type VGG [161]. Cette mise en application fut possible par l'obtention d'un facteur d'accélération 10^6 vis-à-vis d'une implémentation initiale naïve sur CPU.

1. En accès libre à l'adresse suivante : `https://github.com/sebastienwood/MemSE`

Néanmoins, en relaxant le critère de précision sur les estimateurs obtenus, un tirage de Monte-Carlo répété du bruit matériel pour calculer ces estimateurs devient plus intéressant d'un point de vue du temps d'exécution tout en évitant les problèmes d'espace mémoire causés par le stockage du tenseur de covariance. De plus, l'approche par tirage de Monte-Carlo permet d'évaluer immédiatement la performance du DNN sur la tâche d'intérêt plutôt que l'erreur quadratique moyenne sur la sortie du DNN tout en évitant certaines contraintes sur la linéarité des opérations imposée par la méthode de propagation des moments.

Cependant, il est nécessaire d'adapter les équations de puissance proposées par [159] pour le contexte de tirages de Monte-Carlo du bruit matériel. Dans ce contexte, on réalise n tirages aléatoires des paramètres bruités du DNN selon le modèle de bruit utilisé. On peut ensuite calculer la performance et la puissance $\tilde{P}$ du DNN pour chacun de ces tirages d'index b en réalisant une passe avant. Comme pour [159], la puissance est estimée en sommant l'apport de chaque couche $i \in \{1, .., \ell\}$ d'un DNN assigné à un crossbar tout en ignorant les couches opérant sur un matériel fiable, soit

$$\tilde{P}_b = \sum_{i=1}^{\ell} \tilde{P}_b^{(i)} \,. \tag{3.39}$$

Enfin, la consommation de puissance P du DNN est obtenue en prenant la moyenne de la puissance mesurée sur chaque passe avant :

$$P = \frac{1}{n} \sum_{b=1}^{n} \tilde{P}_b \,. \tag{3.40}$$

Si la performance dépendra de la tâche d'intérêt (par exemple, la précision pour une tâche de classification), nous devons exprimer le besoin de puissance pour chaque opération ℓ utilisant les crossbar de memristor avec la simulation du bruit : les couches linéaires et de convolution.

Pour ce faire, nous utilisons l'expression de la puissance dissipée par les crossbar de memristor positifs et négatifs d'une couche ℓ pour un tirage du bruit matériel $\tilde{P}^{(\ell)}_{(\text{mem})}$ ainsi que celle des TIA positifs $\tilde{P}^{(+)\,(\ell)}_{(\text{TIA})}$ et négatifs $\tilde{P}^{(-)\,(\ell)}_{(\text{TIA})}$ associés à ces crossbars tel que

$$\tilde{P}^{(\ell)} = \tilde{P}^{(\ell)}_{(\text{mem})} + \tilde{P}^{(+)\,(\ell)}_{(\text{TIA})} + \tilde{P}^{(-)\,(\ell)}_{(\text{TIA})} \,. \tag{3.41}$$

Ces trois composants peuvent être obtenus efficacement sur GPGPU en répétant la passe avant de l'opération f associée à la couche ℓ avec des manipulations de l'entrée $\boldsymbol{\mathcal{X}}$ et des paramètres du DNN transformés en valeur de conductance souhaitée $\boldsymbol{\mathcal{G}}$ avec un tirage du bruit matériel $\boldsymbol{\epsilon}^v$. Le tenseur résultant $\tilde{\boldsymbol{\mathcal{Y}}}$ est réduit par une opération de somme sur tous ses

éléments $\sum$.

On obtient alors

$$\tilde{P}^{(\ell)}_{(\text{mem})} = \sum f(\boldsymbol{\mathcal{X}} \circ \boldsymbol{\mathcal{X}}, |\boldsymbol{\mathcal{G}}|)\,, \tag{3.42}$$

soit la passe avant de la couche ℓ effectuant l'opération f avec le tenseur d'entrée au carré $\boldsymbol{\mathcal{X}} \circ \boldsymbol{\mathcal{X}}$ et la valeur absolue de ses conductances $|\boldsymbol{\mathcal{G}}|$.

On obtient également

$$\tilde{P}^{(+)\,(\ell)}_{(\text{TIA})} = \sum f(\boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{G}}^{(+)} + \boldsymbol{\epsilon}^{v})\,, \tag{3.43}$$

et

$$\tilde{P}^{(-)\,(\ell)}_{(\text{TIA})} = \sum f(\boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{G}}^{(-)} + \boldsymbol{\epsilon}^{v})\,, \tag{3.44}$$

avec les crossbars de valeurs positives $\boldsymbol{\mathcal{G}}^{(+)}$ et négatives $\boldsymbol{\mathcal{G}}^{(-)}$ perturbées par un tirage du bruit matériel $\boldsymbol{\epsilon}^{v}$.

Ainsi, nous pouvons calculer la puissance P et la performance moyenne d'un DNN en utilisant une simulation de Monte-Carlo sur n tirages du bruit matériel. En pratique, chaque tirage sera utilisé lors d'une passe avant avec un mini-lot aléatoire.

Résumé du modèle de memristors

Nous obtenons ainsi les trois éléments composant notre modèle de memristors :

- g_u : le paramètre matériel permettant de réaliser un compromis énergie-fiabilité,
- le degré de fiabilité matériel, indirectement impacté par g_u et simulé par un bruit suivant une loi normale, multiplicative ou additive selon les besoins,
- enfin, la puissance P dissipée par le DNN opérant sur memristor adaptée des contributions de [159].

CHAPITRE 4 ADAPTATION DES RÉSEAUX DE NEURONES PROFONDS POUR IMPLÉMENTATION SUR DU MATÉRIEL NUMÉRIQUE DE FIABILITÉ CONTRÔLÉE

4.1 Introduction

Calculer un réseau de neurones profond sur du matériel bruité ou de fiabilité limitée est un objectif qui a fait l'objet de plusieurs études, comme nous l'avons vu dans le chapitre 2. Ce même matériel peut être doté d'éléments de détection et de correction d'erreurs, et d'autre part, le réseau peut être préparé pour un déploiement dans cet environnement incertain. Dans ce contexte, le NTC est une méthode proposée pour exploiter le compromis possible énergie-fiabilité dans les calculs d'un DNN.

Nous remarquons que les réseaux de neurones modernes sont composés de nombreuses couches avec des caractéristiques différentes dont le nombre de paramètres qui les composent, ou encore le type de calculs effectués. Ainsi, certaines couches d'un DNN sont plus robustes à un environnement bruité, tandis que d'autres, plus critiques pour la bonne performance du DNN, ne peuvent tolérer que très peu de bruit [170]. Ces couches plus robustes peuvent même être réinitialisées aléatoirement sans pour autant réduire à néant la performance du DNN. De même, il existe un spectre de choix entre un environnement certain et un environnement bruité dans lequel chaque couche a son propre compromis bruit-performance, ou de manière similaire, une sensibilité aux pannes matérielles. Cependant, trouver un compromis optimisé par couche n'est pas une mince affaire : les aléas de chaque couche interagissant pour impacter la performance globale du DNN.

Néanmoins, ce phénomène n'est que partiellement utilisé dans l'exploitation du compromis énergie-fiabilité d'un DNN opérant sur une mémoire en NTC. Par exemple, [107] propose une approche par blocs d'un DNN pour fixer le compromis énergie-fiabilité. Ainsi un ensemble de couches partagent un niveau de bruit avec lequel les calculs doivent s'effectuer. On peut cependant noter que l'exploitation d'un niveau de bruit distinct selon un découpage précis de l'élément à calculer est utilisé par exemple pour optimiser l'énergie allouée à chaque bit des paramètres d'un filtre de Kalman [49], ou encore à la lecture des éléments d'une mémoire SRAM en différenciant l'énergie allouée à chaque bit lu [171]. On peut donc supposer que des gains énergétiques supplémentaires sont accessibles en utilisant un découpage du DNN plus précis que par blocs : par exemple par couches.

Notre objectif dans ce chapitre est de développer une méthode systématique pour exploiter

un compromis énergie-fiabilité différencié pour chaque couche d'un DNN. Nous écartons l'utilisation de matériel de détection et de correction d'erreurs pour nous focaliser sur la robustesse naturelle du DNN utilisé. Un objectif secondaire est que cette méthode n'introduise pas de calculs supplémentaires lourds lors de l'entraînement du DNN. Nous considérons comme point de comparaison le même DNN opérant sur une mémoire NTC avec un unique taux de pannes matérielles : l'objectif est d'obtenir une consommation d'énergie moindre à la même performance sur la tâche du DNN, ou de manière équivalente une performance améliorée à consommation d'énergie similaire. Il sera montré plus loin que les gains obtenus avec l'algorithme proposé permettent une réduction de la consommation d'énergie théorique d'un facteur de 2,4 par rapport à un point de comparaison sur une tâche de classification avec un DNN dont les paramètres sont binarisés.

Le reste de ce chapitre est organisé comme suit. Nous commençons par décrire le contexte expérimental section 4.2. Nous poursuivons en présentant deux méthodes alternatives permettant de préparer un DNN au déploiement sur cette mémoire incertaine en simulant les erreurs matérielles lors de l'entraînement du DNN section 4.3 : d'une part, une méthode considérant une mémoire avec un taux d'erreur uniforme pour l'entièreté du DNN section 4.3.1, et d'autre part une méthode allouant un niveau d'énergie distinct à chaque couche du DNN tout en optimisant ce niveau d'énergie lors de l'entraînement du DNN section 4.3.2. Enfin, nous étudions les compromis énergie-précision obtenus par les DNN entraînés avec ces deux méthodes section 4.4 avant de conclure dans la section 4.6 en résumant les principaux résultats de ce chapitre.

4.2 Scénario expérimental

Ce chapitre s'intéresse à la tâche de classification d'image CIFAR-10 [130] en utilisant le modèle de mémoires numériques sous-alimentées présenté à la section 3.3.1. Nous fixons la valeur de $a = 12{,}8$ à l'équation 3.27 tel que suggéré dans [107]. Nous souhaitons évaluer les possibles gains du compromis énergie-fiabilité exploitant la mémoire sous-alimentée avec les méthodes étudiées dans ce chapitre. Ces dernières seraient appliquées en présence de contraintes sur l'énergie consommée par le DNN. On suppose donc que des techniques préalables de réduction de la consommation d'énergie ont été employées et nous les présentons maintenant.

Premièrement, nous évaluons des réseaux de neurones dont les paramètres sont quantifiés. Plus précisément, nous utilisons un schéma de quantification extrême sur 1 bit en nous basant sur les observations de la section 2.2.3 : un DNN aura une meilleure efficacité énergétique à iso-précision avec une quantification plus agressive. Pour ce faire, nous utilisons la méthode *Binary connect* (BC), présentée à la section 3.2.3. À noter que ce choix de quantification

extrême nous rapproche de la pratique de troncature basée sur la magnitude (*clipping*) permettant d'obtenir des réseaux robustes à un bruit sur les paramètres présentée section 3.2.4.

Dans un second temps, nous souhaitons pouvoir comparer les gains énergétiques pour des DNN de taille variée. Pour ce faire, nous utilisons une architecture de DNN convolutionnel de type *WideResNet* [172]. Il s'agit d'une évolution de l'architecture *ResNet* [173] introduisant un paramètre d'inflation permettant de générer des architectures similaires, mais disposant de canaux de convolutions plus ou moins larges. Par ailleurs, l'utilisation conjointe de paramètres binarisés et de l'architecture WideResNet permet d'obtenir de bons résultats expérimentaux [174]. Dans la suite de ce chapitre, nous considérerons un WideResNet standard avec un paramètre de profondeur fixé à 28 selon les bons résultats expérimentaux obtenus avec ce paramètre dans la littérature [172].

Le WideResNet ainsi obtenu suit une structure en trois blocs séquentiels contenant un nombre croissants de filtres de convolution, précédé par une convolution seule et suivi par une couche linéaire avec un biais. Chaque bloc est constitué de quatre sous-blocs : en général, ces sous-blocs réalisent l'enchaînement des opérations de normalisation par lots, d'activation ReLU puis de convolution. Point de divergence, le premier sous-bloc additionne son entrée transformée par une convolution dite de « raccourci » à sa sortie, tandis que le reste des sous-blocs procède à une addition de son entrée à sa sortie sans transformation. Ces convolutions « raccourcies » se démarquent par un nombre de paramètres plus faible comparativement aux convolutions du bloc considéré.

D'autre part, nous fixons un paramètre d'inflation référence à la valeur 10 en nous basant sur les mêmes observations. De manière succincte, nous nous référons à cette configuration par l'appellation *WideResNet28-10*. Le WideResNet28-10 de référence est constitué de 36 millions de paramètres. Afin de réaliser des comparaisons à différents niveaux d'énergie, nous faisons varier le nombre de paramètres des DNN obtenus en ajustant le paramètre d'inflation. Enfin, pour permettre une comparaison facilitée du nombre de paramètres associé au DNN, nous introduisons la notation ρ pour exprimer le nombre de paramètres normalisés d'un DNN en comparaison avec le WideResNet28-10.

Pour rappel, nous considérons que seuls les paramètres $\boldsymbol{\mathcal{W}}$ des couches de convolution et linéaire sont binarisés et stockés sur la mémoire sous-alimentée. Le biais de la couche linéaire finale est ainsi exclu de nos modèles, ainsi que les paramètres des couches de normalisation par mini-lot. Pour le WideResNet28-10, 18 000 paramètres sont donc exclus du schéma de binarisation et d'utilisation de la mémoire sous-alimentée. Ceci représente 0,05% des paramètres totaux du WideResNet28-10. Contre-intuitivement, le WideResNet28-10 dispose de 29 couches avec un tenseur de paramètres binarisé. Le nombre de paramètres de chaque

couche est présenté figure 4.1.

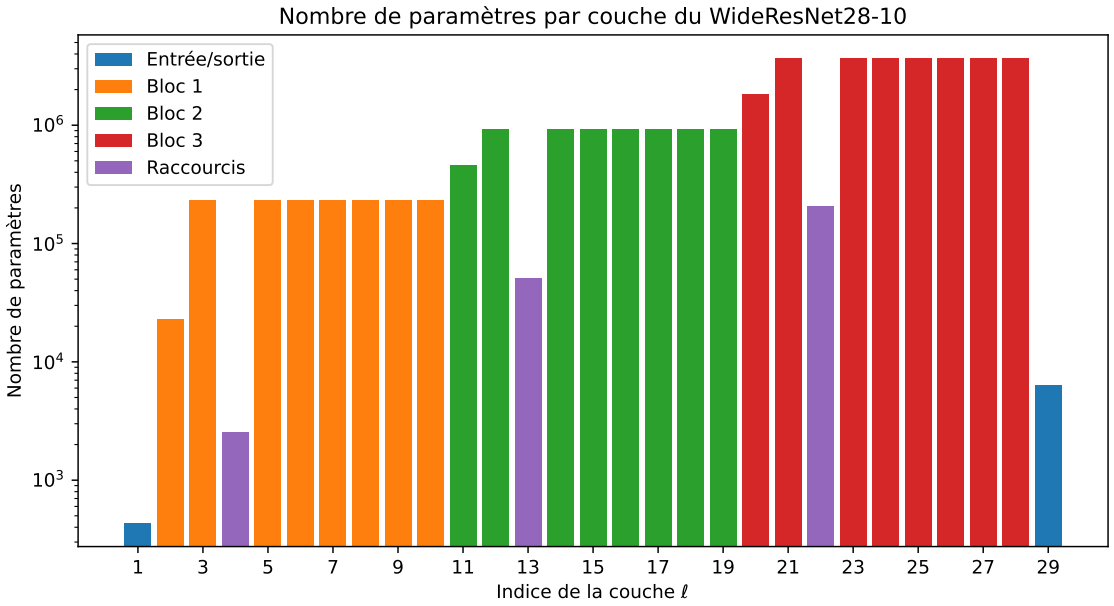


FIGURE 4.1 Nombre de paramètres ($\texttt{count}(\boldsymbol{\mathcal{W}}^{(\ell)})$) par couche du WideResNet28-10 utilisé par défaut dans ce chapitre. La couche de convolution d'entrée et la couche linéaire de sortie comportent relativement peu de paramètres, de même que les convolutions de raccourci.

Pour tous les entraînements de WideResNet réalisés dans ce chapitre, nous utilisons l'algorithme d'optimisation SGD avec un moment de Nesterov et une atténuation des pondérations en utilisant les paramètres suggérés par [172]. L'entraînement est réalisé sur 200 époques pour une taille de mini-lots de 128 et l'ensemble d'entraînement CIFAR-10 utilise des augmentations aléatoires composées de remplissage (*padding*) de 4 pixels dans la dimension spatiale, recadrage aléatoire (*crop*) sur 32 pixels et retournement horizontaux avec une probabilité 50% avant d'être normalisé. La fonction de perte à minimiser $\mathcal{L}$ est la fonction d'entropie croisée. À l'initialisation, le taux d'apprentissage est fixé à la valeur $\iota = 0{,}1$ et les paramètres du DNN sont initialisés en utilisant les recommandations à l'état de l'art [135]. Le paramètre β contrôlant le moment est fixé à 0,9 et le paramètre λ contrôlant l'atténuation des pondérations à $5 * 10^{-4}$. Toutes les 60 époques, ι est divisé par un facteur 10.

4.3 Entraînement de DNN préparés au NTC

L'état de l'art pour la préparation de DNN au déploiement matériel est la simulation de ce dernier lors de l'entraînement du DNN comme détaillé section 3.2.3. Pour le réseau opérant sur une mémoire en NTC, nous avons le cas de figure où p est un scalaire partagé par toutes

les couches du DNN, et le cas où les couches ont un niveau de bruit différencié dans le vecteur $\mathbf{p}$.

4.3.1 NTC uniforme pour tout le DNN

Le cas de figure le plus simple et le plus évident est d'assigner un niveau de bruit unique, synonyme d'une configuration matérielle, pour l'entièreté du DNN. C'est une approche que l'on retrouve dans la littérature [36]. Matériellement, il est également plus simple de gérer une tension d'alimentation unique pour l'ensemble des banques mémoires. Par contre, au prix d'un coût matériel accru, il est faisable de choisir et de configurer une tension d'alimentation propre à chaque banque de mémoire qui permettrait donc de configurer le taux d'erreur accepté dans chaque banque de mémoire. On peut entraîner le DNN en simulant ce bruit d'entraînement p_t : il est même possible de choisir manuellement ce niveau bruit par essai/erreur afin d'obtenir un niveau de précision désiré. On différencie alors le niveau de bruit de déploiement p qui peut être différent de p_t.

La figure 4.2 montre la précision obtenue sur l'ensemble de test CIFAR-10 en suivant cette méthode pour entraîner le WideResNet28-10 avec un niveau de bruit p_t et en le testant à des niveaux de bruits variés p. Pour chaque configuration de p_t, 5 entraînements à partir d'une initialisation aléatoire sont réalisés. La figure présente la précision moyenne de ces 5 DNN distincts évalués au niveau de bruit p, en réalisant 10 évaluations de l'ensemble de test pour chaque DNN. Cette répétition de l'entraînement du DNN et les multiples évaluations sur l'ensemble du jeu de données nous permettent d'apprécier l'impact relatif de l'entraînement avec une simulation de bruit à un niveau p_t sur la perte de performance du DNN avec une précision améliorée. On observe que les DNN sur-apprennent le niveau de bruit p_t lors de leur entraînement, de telle sorte que la meilleure précision obtenue est observée pour $p \approx p_t$. Il est aisé d'observer ce phénomène lorsque p_t devient suffisamment important. Ce fut déjà observé dans des études préalables [36].

Une autre remarque intéressante dans ce cas de bruit uniforme est d'observer que quand un petit $p_t > 0$ est utilisé, une régularisation du DNN prend place ce qui améliore la précision pour tous les choix de p. Dans la figure 4.2, c'est notamment le cas pour la courbe $p_t = 10^{-4}$. Ces deux phénomènes sont à prendre en compte lors du développement d'une solution automatisée pour effectuer le choix de p_t.

En partant d'une solution obtenue de cette manière nous donnant un bon compromis précision-énergie, nous pouvons effectuer une analyse de sensibilité pour déterminer si la fiabilité de certaines couches pourrait être encore diminuée. Pour ce faire, nous testons la performance du WideResNet28-10 entraîné avec $p_t = 1\%$ quand les paramètres de la couche ℓ sont remplacés

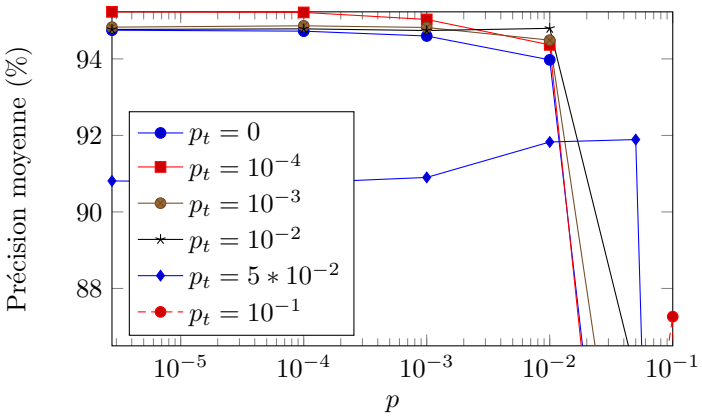


FIGURE 4.2 Précision moyenne de l'architecture WideResNet28-10 dont les paramètres sont binarisés sur l'ensemble de test du jeu de données CIFAR-10 en faisant varier p, un taux d'erreur par bit sur les paramètres du DNN. Plusieurs configurations de DNN sont entraînées en faisant varier le paramètre p_t, un taux d'erreur par bit appliqué aux paramètres lors de l'entraînement du DNN, en partant de 5 initialisations aléatoires pour chaque p_t. On observe un effet de régularisation quand le DNN est entraîné avec un faible p_t, menant à de meilleures performances pour $p \approx 0$. On observe également pour des p_t plus élevés que la performance du DNN est maximisée lorsque testé à $p = p_t$.

par des valeurs aléatoires. Précisément, toutes les couches sauf la couche ℓ sont soumises à un bruit $p = 1\%$. La couche ℓ en revanche est soumise à un bruit $p_\ell = 50\%$. La performance sur l'ensemble de test du jeu de données CIFAR-10 est enfin évaluée. Étant donné la nature fortement stochastique de la sortie du DNN avec l'utilisation du BSC à ce niveau de bruit, cette évaluation est répétée à 10 reprises. Cette répétition abouti à une distribution de performance pour chaque choix de ℓ.

Les résultats de cette expérience sont présentés figure 4.3, avec à l'abscisse l'indice de la couche ℓ et à l'ordonnée la distribution de précision ainsi obtenue par un diagramme en boîtes (*box plot*). On y observe que certaines couches sont critiques pour la performance : la précision du DNN chute à une performance inexploitable quand ces couches sont soumise au bruit augmenté du BSC. Mais on distingue également certaines couches avec des degrés variés de sensibilité à ces paramètres aléatoires, ce qui nous motive à développer un algorithme appelé *LaNMax* afin d'optimiser de manière systématique la fiabilité de chaque couche.

4.3.2 NTC différencié par couche avec LaNMax pour optimiser les niveaux d'énergies

Nous proposons l'algorithme *Layerwise Noise Maximization* (LaNMax) afin d'optimiser le compromis énergie-fiabilité automatiquement pour chaque couche du DNN. Pour ce faire,

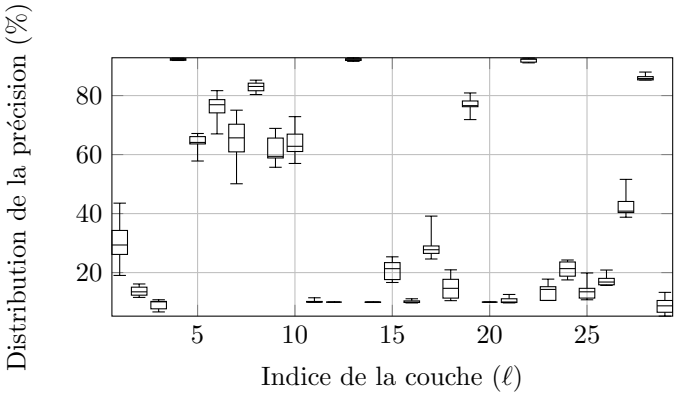


FIGURE 4.3 Analyse de sensibilité d'une architecture WideResNet28-10 entraînée et testée avec bruit uniforme $p = 1\%$ et avec la couche ℓ soumise à un BSC fortement bruité ($p = 50\%$). À l'ordonnée, distribution de précision sur l'ensemble de test CIFAR-10 avec diagramme en boîte pour 10 évaluations de l'ensemble de test. Chaque boîte représente une distribution avec le point minimum et maximum (aux extrêmes des « moustaches » sortant de la boîte), le premier quartile et le troisième quartile (le début et la fin de la boîte) et la médiane (trait à l'intérieur de la boîte). On observe des variations de la sensibilité des couches au bruit du BSC, avec certaines couches affectant de manière critique la performance du DNN et d'autres couches impactant peu la performance du DNN.

nous modélisons le compromis énergie-précision du DNN par une fonction de perte externe $\mathcal{L}_O$ comprenant l'espérance de la fonction de perte du DNN sous le modèle de bruit matériel $\mathbb{E}(\mathcal{L}(\texttt{BSC}(\boldsymbol{\mathcal{W}}, \mathbf{p})))$ et la consommation d'énergie $E(\mathbf{p})$. Par la possibilité de sur-apprentissage du niveau de bruit tel que mentionné dans la section 4.3.1, il est possible lors de l'optimisation de la fonction de perte externe d'aboutir dans un optimum local désavantageux. Une régularisation $\lambda_p \mathbf{p}$, que nous appelons *atténuation du bruit* par analogie à l'atténuation des pondérations (*weight decay*) couramment utilisée pour l'entraînement de DNN, est utilisée pour pallier ce problème. La puissance de cette régularisation est contrôlée par le paramètre λ_p.

Ainsi, nous définissons la fonction de perte externe comme suit

$$\mathcal{L}_O(\boldsymbol{\mathcal{W}}, \mathbf{p}) = \mathbb{E}[\mathcal{L}(\texttt{BSC}(\boldsymbol{\mathcal{W}}, \mathbf{p}))] + \alpha E(\mathbf{p}) + \lambda_p \mathbf{p}\,, \tag{4.1}$$

avec le paramètre α nous permettant de contrôler le compromis entre l'énergie et la précision atteinte par le DNN : un α plus petit nous menant à des solutions avec une meilleure précision au prix d'une plus grande consommation d'énergie de par son poids réduit dans l'optimisation du DNN.

Nous voulons ainsi trouver la meilleure fiabilité pour chaque couche du DNN $\mathbf{p}^\star$ tout en

optimisant les paramètres du DNN $\boldsymbol{\mathcal{W}}$ tel que

$$\mathbf{p}^{\star} = \underset{\mathbf{p}}{\operatorname{argmin}} \min_{\boldsymbol{\mathcal{W}}} \mathcal{L}_O(\boldsymbol{\mathcal{W}}, \mathbf{p}) \,, \tag{4.2}$$

tout en bornant les valeurs de $\mathbf{p}$ selon les observations de la section 3.3.1 et les résultats de la section 4.3.1 tel que

$$0 < p_i \leq \frac{1}{2} \;\; \forall i \in \{1, .., \ell\} \,. \tag{4.3}$$

Les bornes minimum et maximum sont par la suite exprimées avec la notation $p_{\min}$ et $p_{\max}$.

Or, si les paramètres du DNN sont optimisés avec un mini-lot S permettant d'estimer avec une faible variance le gradient des paramètres $\nabla_w \mathcal{L}_O(\boldsymbol{\mathcal{W}}, \mathbf{p}) = \nabla_w \mathcal{L}_S(\texttt{BSC}(\boldsymbol{\mathcal{W}}, \mathbf{p}))$, chaque passe avant permet d'observer un seul tirage du bruit matériel avec le BSC. L'estimation de $\nabla_p \mathcal{L}_O$ serait donc soumise à de fortes variations aléatoires selon l'impact du bruit échantillonné et la variance habituelle de la fonction de perte $\mathcal{L}_S(\texttt{BSC}(\boldsymbol{\mathcal{W}}, \mathbf{p}))$ pour le mini-lot considéré. Il devient ainsi nécessaire d'effectuer plusieurs passes avant pour obtenir un estimé plus précis de $\mathcal{L}_O(\boldsymbol{\mathcal{W}}, \mathbf{p})$, notamment du premier composant comportant l'espérance de la fonction de perte $\mathcal{L}(\texttt{BSC}(\boldsymbol{\mathcal{W}}, \mathbf{p}))$. Ce dernier serait alors estimé sur une moyenne de n mini-lots.

De plus, au vu de la codépendance entre les paramètres du DNN $\boldsymbol{\mathcal{W}}$ et le bruit sur le BSC p observée à la section 4.3.1 il serait peu souhaitable d'optimiser séparément ces deux éléments : un DNN atteindra un maximum de performance si le taux d'erreur de son environnement d'inférence est simulé pendant l'entraînement. Une approche possible serait d'utiliser un algorithme de descente des coordonnées (*Block Coordinate Descent*) [175] avec des mises à jour alternées des paramètres du DNN et de $\mathbf{p}$ basées sur des mesures distinctes de $\mathcal{L}$ avec $\mathbf{p}$ fixe et $\mathcal{L}_O$ avec les paramètres du DNN $\boldsymbol{\mathcal{W}}$ fixes. Au lieu de quoi, afin d'accélérer le processus d'entraînement du DNN, nous utilisons l'information disponible lors de l'optimisation de $\mathcal{L}$ avec un $\mathbf{p}$ fixe pour mesurer $\mathcal{L}_O$ et ainsi estimer le gradient $\nabla_p \mathcal{L}_O$. Ce gradient est ensuite utilisé pour réaliser une descente de gradient standard pour $\mathbf{p}$.

Néanmoins, on remarque que l'opération de BSC, analogue à un XOR (*exclusive-OR*), serait peu pratique à intégrer dans la passe arrière du DNN. On préfère alors l'utilisation de dérivée numérique pour simplifier les opérations. Pour ce faire, un pas $\epsilon \sim \mathcal{U}\{-h, 0, h\}$ est échantillonné aléatoirement pour chaque nouvelle passe avant d'optimisation de $\mathcal{L}$. Le vecteur de bruit sur les BSC de chaque couche effectif est alors

$$\tilde{\mathbf{p}} = \min(\max(p_{\min},\, \mathbf{p} + \epsilon),\, p_{\max}) \,, \tag{4.4}$$

une opération de *clipping* étant utilisée pour s'assurer que le domaine de $\mathbf{p}$ respecte la condi-

tion de l'équation 4.3. Avec ce $\tilde{\mathbf{p}}$, on obtient une valeur de $\mathcal{L}_S(\texttt{BSC}(\boldsymbol{\mathcal{W}}, \tilde{\mathbf{p}}))$ pour le mini-lot S et l'on peut directement calculer le reste des composants de l'équation 4.1. Les mesures successives de $\mathcal{L}_O(\boldsymbol{\mathcal{W}}, \tilde{\mathbf{p}})$ lors de n passe avant consécutives sont utilisées dans une régression linaire $\texttt{OLS}(\{\mathcal{L}_O^{(1)}, .., \mathcal{L}_O^{(n)}\})$: la pente de cette régression est utilisée comme estimateur de $\nabla_p \mathcal{L}_O(\boldsymbol{\mathcal{W}}, \mathbf{p})$. Par simplicité, nous adoptons la notation $\nabla_i \texttt{OLS}$ pour la pente du i-ème élément du vecteur $\mathbf{p}$ et $\nabla\texttt{OLS}$ pour le gradient de l'ensemble du vecteur $\mathbf{p}$. En pratique, l'ensemble des mini-lots d'une époque seront utilisés pour réaliser une estimation de $\nabla_p \mathcal{L}_O(\boldsymbol{\mathcal{W}}, \mathbf{p})$ avec cette procédure : autrement dit, $\mathbf{p}$ est mis à jour à la fin de chaque époque d'entraînement des paramètres $\boldsymbol{\mathcal{W}}$ avec la fonction de perte $\mathcal{L}(\texttt{BSC}(\boldsymbol{\mathcal{W}}, \mathbf{p}))$. Une opération de *clipping* est également appliquée pour s'assurer de la validité de $\mathbf{p}$ après sa mise à jour, comme vu à l'équation 4.4.

On remarque que cette approche de perturbation de $\mathbf{p}$ à chaque mini-lot nous permet d'anticiper le phénomène de sur-apprentissage du niveau de bruit observé précédemment, tout en permettant d'observer plus facilement l'effet de pannes provenant du BSC lorsque $\mathbf{p}$ possède de faibles valeurs. Cependant, dans le but de stabiliser les dernières époques de l'entraînement et de maximiser la performance du DNN avec le $\mathbf{p}$ appris, les mises à jour de $\mathbf{p}$, ainsi que les perturbations avec ϵ sont désactivées à l'époque s.

Par ailleurs, on observe que la fonction de perte $\mathcal{L}_O(\boldsymbol{\mathcal{W}}, \mathbf{p})$ est critiquable : la magnitude du composant $\mathcal{L}$ est vouée à évoluer rapidement lors d'une époque, et la dérivée de la fonction d'énergie $E(\mathbf{p})$ du deuxième composant varie fortement lorsque $p \approx 0$. Ainsi, nous utilisons une fonction de perte substitut (*surrogate*) $\bar{\mathcal{L}}_O(\boldsymbol{\mathcal{W}}, \mathbf{p})$ définie comme suit

$$\bar{\mathcal{L}}_O(\boldsymbol{\mathcal{W}}, \mathbf{p}) = \frac{\mathcal{L}}{\mathcal{L}^\star} + \alpha\sqrt{E(\mathbf{p})} + \lambda_p \mathbf{p}\,, \tag{4.5}$$

où $\mathcal{L}^\star$ est la meilleure valeur de la fonction de perte $\mathcal{L}$ observée sur un mini-lot à ce stade de l'entraînement du DNN. Cette formulation permet de stabiliser les valeurs du premier terme et prendre la racine carrée de l'énergie nous permet de réduire les variations du second terme. De plus, nous utilisons une normalisation du gradient tel que

$$\nabla_p \mathcal{L}_O(\boldsymbol{\mathcal{W}}, \mathbf{p}) = \frac{\nabla_p \texttt{OLS}}{\|\nabla\texttt{OLS}\|}\,, \tag{4.6}$$

cette dernière technique permettant de limiter le pas de chaque mise à jour du vecteur $\mathbf{p}$. Nous observons une amélioration des résultats obtenus avec cette fonction substitut et cette normalisation du gradient.

La procédure d'entraînement d'un DNN en utilisant l'algorithme LaNMax est montrée algorithme 1. Les paramètres du DNN $\boldsymbol{\mathcal{W}}$ sont mis à jour avec SGD pendant K époques

Algorithme 1 Règle de mise-à-jour Layerwise Noise Maximisation (LaNMax)

1: **pour** $k \in \{1, .., K\}$ **faire**
2: **pour** $i \in \{1, 2, \ldots, n\}$ **faire**
3: Échantillonner un mini-lot S
4: **si** $k \leq s$ **alors**
5: $\epsilon_i \sim \mathcal{U}\{-h, 0, h\}\, \forall i \in \{1, .., \ell\}$
6: $\tilde{\mathbf{p}} \leftarrow \min(\max(p_{\min}, \mathbf{p} + \epsilon), p_{\max})$
7: $\mathcal{L}_O^{(i)}, \mathcal{L}^{(i)} \leftarrow \mathcal{L}_O(\boldsymbol{\mathcal{W}}, \tilde{\mathbf{p}})$
8: **sinon**
9: $\mathcal{L}^{(i)} \leftarrow \mathcal{L}_S(\boldsymbol{\mathcal{W}}, \mathbf{p})$
10: **fin si**
11: $\boldsymbol{\mathcal{W}} \leftarrow \boldsymbol{\mathcal{W}} - \iota \nabla_w \mathcal{L}^{(i)}$
12: **fin pour**
13: **si** $k \leq s$ **alors**
14: $\nabla_p \texttt{OLS} \leftarrow \texttt{OLS}(\{\mathcal{L}_O^{(1)}, .., \mathcal{L}_O^{(n)}\})$
15: $\nabla_p \mathcal{L}_O \leftarrow \frac{\nabla_p \texttt{OLS}}{||\nabla \texttt{OLS}||}$
16: $\mathbf{p} \leftarrow \mathbf{p} - \iota_p \nabla_p \mathcal{L}_O$
17: $\mathbf{p} \leftarrow \min(\max(p_{\min}, \mathbf{p}), p_{\max})$
18: **fin si**
19: **fin pour**
20: **retourne** $\mathbf{p}, \boldsymbol{\mathcal{W}}$

(ligne 1). À chaque époque, n mini-lots sont observés (ligne 2). Si les mises à jour de $\mathbf{p}$ sont requises (ligne 4), une expérience aléatoire est conduite dans le voisinage de $\mathbf{p}$ comme suit. Tout d'abord, une perturbation aléatoire ϵ_i est échantillonnée uniformément dans l'ensemble $\{-h, 0, h\}$ pour chaque couche du DNN $i \in \{1, .., \ell\}$ (ligne 5). On obtient ainsi le vecteur de perturbation ϵ, qui est additionné à $\mathbf{p}$ pour obtenir $\tilde{\mathbf{p}}$ afin d'évaluer la fonction de perte dans le voisinage de $\mathbf{p}$. $\tilde{\mathbf{p}}$ est ensuite restreint au domaine $[p_{\min}, p_{\max}]$ (ligne 6). Pour le mini-lot S, la fonction de perte $\mathcal{L}_O$ considérée est évaluée avec $\tilde{\mathbf{p}}$. On évalue simultanément $\mathcal{L}_S$, la fonction de perte du DNN évaluée sur le mini-lot S, puisque $\mathcal{L}_O$ en dépend (ligne 7). La répétition de cette expérience aléatoire sur tous les mini-lots permet d'améliorer incrémentalement la précision du gradient numérique ainsi obtenu.

Si les mises à jour de $\mathbf{p}$ ne sont pas requises (ligne 8), on évalue directement $\mathcal{L}_S$ avec $\mathbf{p}$ (ligne 9). Les mises à jour habituelles des paramètres du DNN pour $\mathcal{L}^{(i)}$ sont appliquées dans tous les cas à la fin des calculs afférents à un mini-lot d'indice i (ligne 11). Quand tous les mini-lots ont été utilisés et si les mises à jour de $\mathbf{p}$ sont requises (ligne 13), on procède à l'estimation de $\nabla_p \mathcal{L}_O$. Dans un premier temps, le gradient numérique est obtenu avec OLS (ligne 14). Deuxièmement, le gradient est normalisé (ligne 15). Pour finir, $\mathbf{p}$ est mis-à-jour (ligne 16) et restreint au domaine valide $[p_{\min}, p_{\max}]$ (ligne 17). Cette mise à jour

de $\mathbf{p}$ avec la règle de descente de gradient nécessite un choix de taux d'apprentissage ι_p. Grâce à la normalisation du gradient appliquée plus tôt, le choix de ce taux d'apprentissage a un impact modéré sur les résultats obtenus. Dans les expériences menées, on fixe ainsi le même taux d'apprentissage pour les paramètres du DNN $\boldsymbol{\mathcal{W}}$ et le vecteur $\mathbf{p}$ soit $\iota_p = \iota$.

Les mises-à-jour de $\mathbf{p}$ cessent avant la fin de l'entraînement des paramètres du DNN $\boldsymbol{\mathcal{W}}$ en utilisant le critère s (lignes 4 et 13).

Enfin, LaNMax nous retourne les paramètres appris du DNN $\boldsymbol{\mathcal{W}}$ ainsi que le vecteur $\mathbf{p}$.

Nous souhaitons maintenant évaluer l'amélioration du compromis énergie-performance du DNN avec LaNMax pour optimiser $\mathbf{p}$, comparativement à l'approche précédente d'un p partagé entre toutes les couches du DNN.

4.4 Résultats expérimentaux avec LaNMax

Pour évaluer les améliorations potentielles du compromis énergie-performance de notre WideResNet, nous comparons les approches proposées dans la section 4.2 de binarisation ou de changement de la taille du DNN (avec le paramètre d'inflation) avec les deux approches pour déterminer le taux d'erreur de la mémoire opérant en NTC. D'une part, nous aurons le p partagé étudié à la section 4.3.1, d'autre part, nous aurons un $\mathbf{p}$ appris pour chaque couche du DNN avec l'algorithme LaNMax.

Nous pourrons ainsi cumuler trois techniques de réduction de la consommation d'énergie pour optimiser le compromis énergie-performance : la taille du DNN, la quantification et l'usage d'une mémoire en NTC. Ces trois facteurs sont pris en compte dans la définition d'énergie à l'équation 3.30, ce qui nous permettra de comparer uniformément les gains de chaque approche. Par ailleurs, quand $p > 0$, les paramètres du DNN deviennent stochastiques lors du calcul de la passe avant du DNN, ce qui implique également une stochasticité sur la sortie du DNN. Afin de mesurer correctement la précision moyenne du DNN sur la tâche de classification, nous évaluons donc le jeu de données de test à multiples reprises. À chaque fois, les paramètres stochastiques sont rééchantillonnés, jusqu'à ce qu'un intervalle de confiance sur l'estimation de la moyenne atteigne une magnitude de 5% à un niveau de confiance de 95%.

4.4.1 Choix d'hyperparamètres

Suivant des essais préliminaires ayant donné de bons résultats, $\mathbf{p}$ est initialisé pour toutes les couches ℓ à $p_\ell = 0{,}01$, la mise à jour de $\mathbf{p}$ par SGD est effectuée avec un moment de Nesterov

paramétré par $\beta_p = 0{,}2$ et une atténuation du bruit $\lambda_p = 5 * 10^{-4}$. Le pas sur $\mathbf{p}$ est paramétré avec $h = 0{,}01$, et les mises à jour de $\mathbf{p}$ sont effectuées jusqu'à l'époque d'entraînement du DNN $s = 160$ (soit 80% de l'entraînement du DNN). Étant donné les résultats utilisant un niveau de bruit uniforme section 4.3.1, où nous avons déterminé que le DNN atteint une meilleure précision quand $p_t = 10^{-4}$ que lorsque $p_t = 0$, nous précisons le domaine de p_i défini à l'équation 4.3 tel que

$$10^{-4} < p_i \leq \frac{1}{2} \;\; \forall i \in \{1, .., \ell\} \, . \tag{4.7}$$

Ces choix d'hyperparamètres n'ont pas de vocation universelle et pourraient être optimisés plus finement avec un budget calculatoire important. On remarque que LaNMax, avec une mise à jour par SGD de $\mathbf{p}$ distinct du SGD utilisé pour les paramètres du DNN $\boldsymbol{\mathcal{W}}$, double approximativement le nombre d'hyperparamètres à optimiser (initialisation et domaine de $\mathbf{p}$, ι_p, β_p, λ_p, h, s).

Nous effectuons plusieurs entraînements du DNN en utilisant LaNMax tout en variant le paramètre α contrôlant le compromis énergie-précision de telle sorte que $\alpha \in \{0{,}1; 0{,}05; 0{,}03; 0{,}02; 0{,}01; 0{,}001; 0\}$.

4.4.2 Résultats

Les valeurs de précision atteinte en fonction de l'énergie mesurée sont rapportées à la figure 4.4. Une courbe significative est celle qui rapporte les résultats obtenus en utilisant une représentation des paramètres à virgule flottante sur 16 bits (FP16). Cette courbe sera utilisée comme point de comparaison. Ce graphique comporte plusieurs courbes significatives et pour chaque courbe, nous avons utilisé une stratégie distincte afin de réduire l'énergie consommée possiblement au prix d'une dégradation de la précision atteinte. Pour les courbes sans bruit, l'énergie est réduite en diminuant le nombre de paramètres du WideResNet avec le facteur d'inflation. Pour la courbe binarisée avec BC simulée avec bruit uniforme, l'énergie est réduite en augmentant l'unique paramètre de bruit p. Enfin, pour les deux courbes LaNMax le paramètre α est modifié. À noter que pour la plupart des points de la courbe avec bruit uniforme, le taux de bruit de la phase de test p est le même que celui de la phase d'entraînement p_t. Cependant, puisque nous avons observé précédemment qu'il est préférable d'entraîner à $p_t = 10^{-4}$ pour $p = 0$ ces valeurs sont utilisées.

Nous pouvons dans un premier temps observer que le réseau entraîné avec un niveau de bruit uniforme pour chaque couche obtient une amélioration limitée en efficacité énergétique. Par exemple, en comparaison de la référence sans bruit, une réduction d'énergie de 28% est

obtenue à un taux de précision 95,2% pour la solution avec bruit uniforme. D'autre part, les optimisations plus précises du niveau de bruit effectuées par LaNMax ont produit des solutions offrant une consommation d'énergie bien plus petite à précision égale. Ainsi, à une précision de 95,3%, les solutions trouvées par LaNMax réduisent l'énergie d'un facteur 2,4× par rapport à une solution avec un niveau de bruit uniforme, et d'un facteur 3,3× par rapport à une solution sans bruit, mais avec des paramètres binaires.

Par ailleurs, il est intéressant de noter que pour le modèle d'énergie-fiabilité considéré, des changements de fiabilité seuls ne permettent pas de couvrir une large gamme de valeurs pour la précision avec une bonne efficacité énergétique. En effet, la figure 4.4 nous montre qu'il est essentiel de modifier également le nombre de paramètres alloués au DNN pour se déplacer sur le front de Pareto.

On relève enfin que ces résultats sont obtenus pour cinq entraînements de DNN à partir d'une initialisation aléatoire pour une seule architecture de DNN et sur un jeu de données. Pour s'assurer de l'intérêt de LaNMax, il serait pertinent de réitérer cette expérience sur des jeux de données tiers, ou encore en variant les architectures de DNN utilisées, ou les représentations numériques des nombres. Cette démarche est cependant limitée par le simple budget calculatoire nécessaire à l'obtention d'un point de données : la simulation du BSC est une procédure lourde lors de l'entraînement du DNN. Ce manque d'efficience motive les travaux du prochain chapitre.

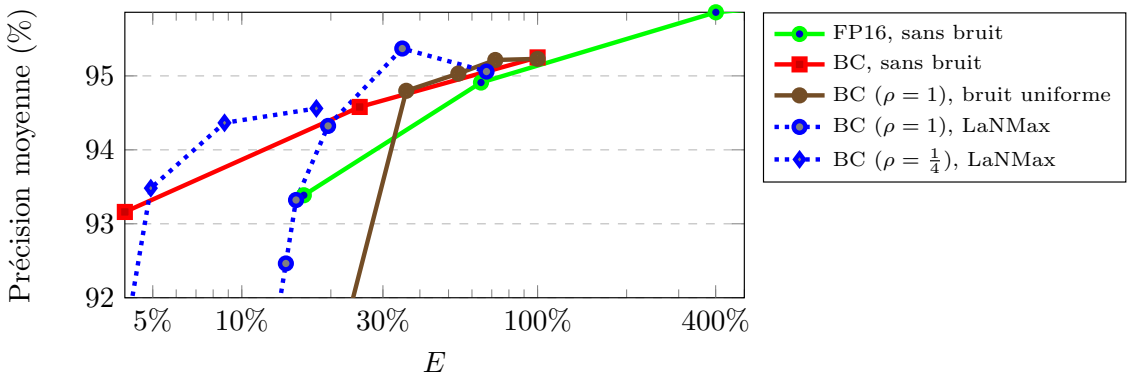


FIGURE 4.4 Précision moyenne sur la tâche de classification en termes de consommation d'énergie normalisée pour un réseau avec des paramètres à virgule flottante sur 16 bits (FP16) et pour des réseaux binarisés avec BC avec différentes configurations de bruit. Pour les courbes FP16 et BC sans bruit, la réduction d'énergie est obtenue en réduisant le nombre de paramètres. Pour les autres courbes, le nombre de paramètres normalisé (ρ) est donné.

Enfin, nous regardons de plus près les niveaux de bruits assignés à chaque couche en utilisant LaNMax figure 4.5, pour $\rho = 1$ et $\alpha = 0{,}001$. Cette solution obtient une précision de

95,4% pour $E = 35\%$. En point de comparaison, nous considérons la solution avec niveau de bruit uniforme à 94,8% de précision pour $E = 36\%$, à $p = 0{,}01$. Afin d'améliorer la précision, la fiabilité des premières couches fut augmentée à $p_\ell = 10^{-4}$, tandis que la fiabilité des dernières couches fut réduite. Cela s'explique au moins en partie par les caractéristiques de l'architecture WideResNet pour laquelle les couches les plus profondes, c'est-à-dire les dernières, disposent de plus de paramètres comme nous pouvons le voir pour le WideResNet28-10 figure 4.1. Ces couches sont donc plus susceptibles de supporter un nombre de pannes supérieur. Dans la prochaine section, nous nous intéressons aux effets de la simulation du BSC sur les propriétés des DNN obtenus.

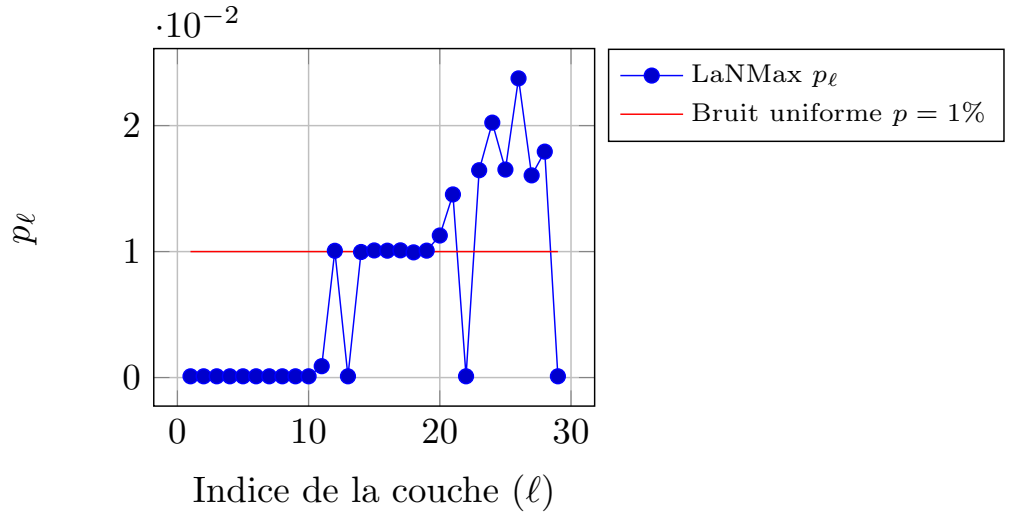


FIGURE 4.5 Le niveau de bruit par couche p_ℓ pour le α obtenant les meilleurs résultats quand $\rho = 1$.

4.5 Analyse spectrale de DNN entraînés avec une simulation de pannes matérielles

Ce chapitre repose sur un entraînement *hardware-aware*. Cela pose plusieurs problématiques. Premièrement, à chaque passe avant, un échantillonnage du BSC est réalisé, ce qui est coûteux en temps et en mémoire. Ce problème observe une relation linéaire avec le nombre de paramètres du DNN, ce qui complique la tâche pour certains modèles plus imposants. Deuxièmement, on peut supposer qu'il est important d'avoir de nombreux échantillons de cette simulation Monte-Carlo pour mieux représenter le matériel cible et y préparer le DNN. Or, certains travaux orthogonaux cherchent à réduire la complexité de l'entraînement du DNN en réduisant le nombre de mini-lots, et donc dans notre formulation le nombre d'échantillons du BSC.

Dans cette section, nous nous penchons sur les propriétés spectrales des paramètres d'un DNN entraîné en simulant le BSC afin de motiver de futurs travaux souhaitant outrepasser cette simulation du BSC. Récemment, les propriétés spectrales d'un DNN ont été utilisées pour guider l'entraînement d'un réseau antagoniste génératif avec la méthode de normalisation spectrale [176]. Concrètement, les tenseurs de paramètres du DNN sont divisés par leurs valeurs propres la plus importante, estimée par la peu coûteuse méthode de la puissance itérée. Les images générées par le réseau génératif entraîné avec cette normalisation ont une meilleure qualité comparativement aux méthodes préalablement utilisées. Par ailleurs, une version spectrale de la populaire méthode Dropout [108] propose d'élaguer et de nullifier aléatoirement les paramètres de couches de convolutions dans le domaine spectral [177]. Il est donc intéressant de chercher à comprendre les caractéristiques d'un DNN au travers d'une analyse spectrale.

Précisément, nous cherchons à savoir si les entraînements avec une simulation Monte-Carlo du BSC pour notre réseau WideResNet28-10 binarisé modifient significativement les valeurs propres du DNN. L'hypothèse nulle de cette étude est que la moyenne des valeurs propres demeure la même avec le bruit du BSC, tandis que l'hypothèse alternative serait une modification de la valeur moyenne des valeurs propres en présence du BSC.

Pour ce faire, nous reprenons la procédure d'entraînement *hardware-aware* avec un niveau de bruit uniforme présenté ci-dessus. Nous réalisons 9 entraînements de WideResNet28-10 binarisés à niveaux de bruit distincts sur la tâche de classification d'images CIFAR-10. La consommation énergétique et la précision sur l'ensemble de test de CIFAR-10 de ces 9 DNN sont rapportées figure 4.6. On y constate une réduction graduelle de la consommation d'énergie estimée, accompagnée d'une réduction de la précision obtenue. Cette dernière, avec un point de données par niveau d'énergie, permet de juger grossièrement du compromis énergie-performance du DNN sans pour autant être une mesure fiable de ce dernier. Nous souhaitons maintenant étudier les différences dans les propriétés spectrales de ces 9 DNN afin d'y découvrir de potentiels changements.

Pour simplifier cette étude, nous nous concentrons sur l'avant-dernière couche du DNN grâce à sa bonne robustesse au bruit, mais également par son nombre élevé de 3 millions de paramètres. Étant donné que ces mêmes paramètres sont binarisés avec BC, les paramètres ont une représentation sur l'ensemble des réels, ce qui nous permet d'appliquer une méthode de décomposition en valeurs singulières (*singular value decomposition*, SVD) pour obtenir les valeurs propres. Préalablement, nous devons transformer le tenseur de paramètres $\boldsymbol{\mathcal{W}}$ en matrice $\mathbf{W}$ par l'usage d'un algorithme de déroulement de tenseur. Ce procédé nous permet de tirer des conclusions sur le tenseur en étudiant les propriétés de la matrice résultante.

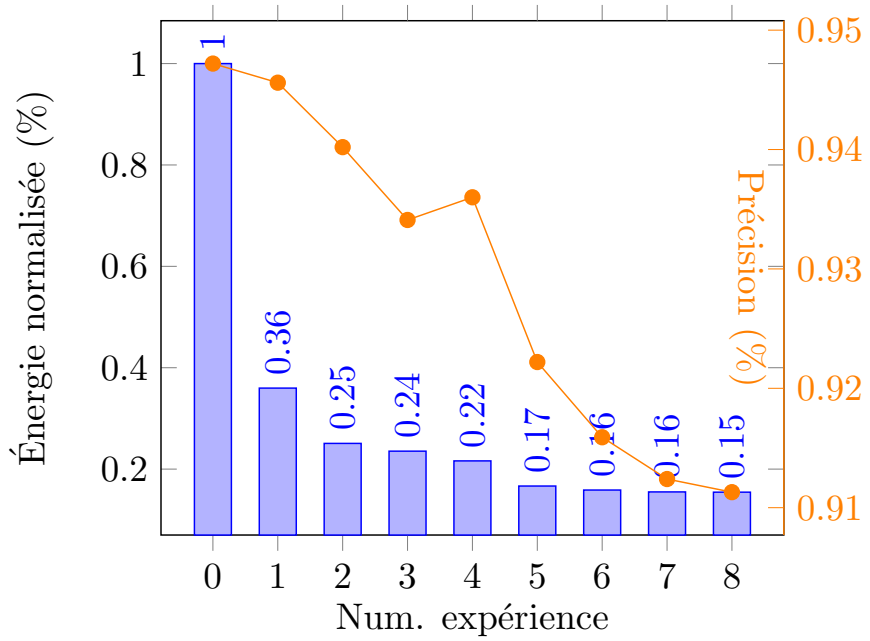


FIGURE 4.6 Consommation d'énergie et précision sur la tâche CIFAR-10 des WideResNet28-10 binarisés analysés spectralement. Les réseaux sont obtenus en faisant varier le taux d'erreur p_t et testés à $p = p_t$.

Or, pour un tenseur $\boldsymbol{\mathcal{W}} \in \mathbb{R}^{n_1,\cdots,n_d}$, il y a plusieurs possibilités d'en construire une matrice $\mathbf{W} \in \mathbb{R}^{N_1,N_2}$ dans laquelle $N_1 N_2 = n_1 * n_2 * \cdots n_d$. Par exemple, on peut utiliser l'algorithme de déroulement en mode-k dans lequel les fibres du tenseur sont assemblées pour produire une matrice de forme n_k par (N/n_k) où $N = n_1 \cdots n_d$. Une fibre est un vecteur obtenu en fixant tous les indices du tenseur sauf un. Avec cet algorithme, il y a 4 façons de dérouler un tenseur composé de 4 dimensions : c'est le cas par exemple des tenseurs de paramètres d'une couche de convolution.

Nous comparons les deux premières valeurs propres des 9 DNN obtenus précédemment dans la figure 4.7. Des résultats similaires sont observés sur les 3 modes restants, qui ne sont donc pas montrés. Nous pouvons voir une évolution importante dans ces valeurs propres avec l'augmentation du niveau de bruit. Pour mieux mesurer la crédibilité statistique de cet effet, nous réalisons un test-t sur la différence des valeurs moyennes de ces valeurs propres. Nous obtenons ainsi les métriques t-stat $= -4{,}5034$ et une p-valeur $= 0{,}002787$, valeurs nous permettant de conclure que le bruit lors de l'entraînement du DNN a un effet significatif sur les valeurs propres des tenseurs de paramètres du DNN et ainsi de pouvoir rejeter l'hypothèse nulle.

Enfin, nous comparons le changement relatif des 10 premières valeurs propres des deux DNN

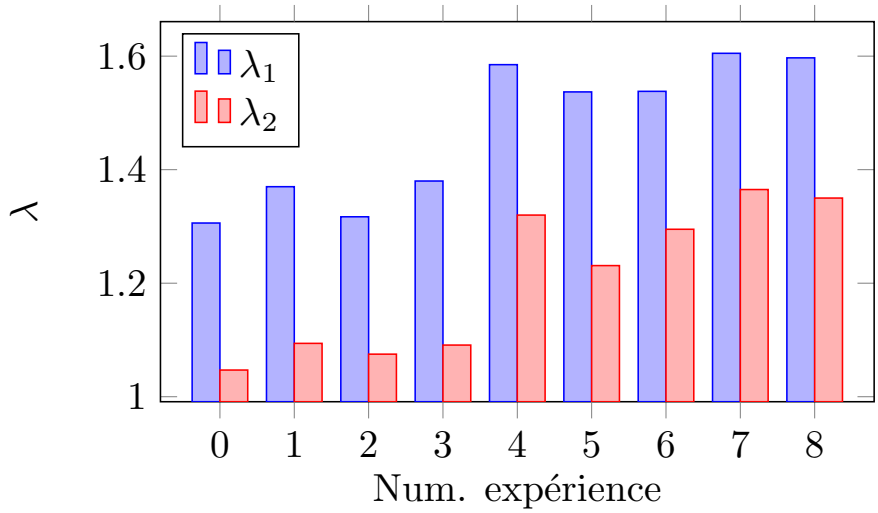


FIGURE 4.7 Deux plus grandes valeurs propres λ_1 et λ_2 pour un déroulement en mode 1 des DNN analysés spectralement

aux extrêmes de la consommation d'énergie figure 4.8. Nous pouvons ainsi avoir un aperçu des différences dans la distribution de ces valeurs propres. On observe une augmentation moyenne de ces 10 valeurs propres d'environ 25%. En particulier, le *eigengap*, ou la différence entre les deux premières valeurs propres, s'est réduit d'environ 5% entre ces deux DNN.

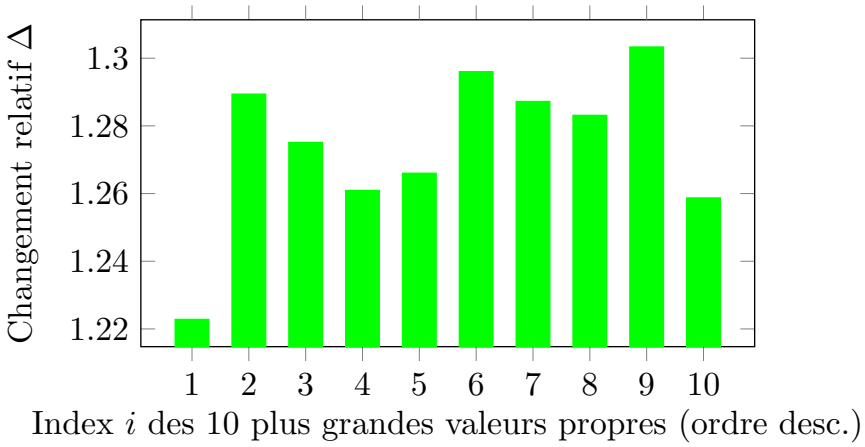


FIGURE 4.8 Changement relatif Δ_i des dix plus grandes valeurs propres entre le scénario le plus bruité (expérience 8) et le scénario le moins bruité (expérience 0) tel que $\Delta_i = (\lambda_i^8 - \lambda_i^1)/\lambda_i^1$ pour la valeur propre d'index i pour l'expérience 1 λ_i^1 et pour l'expérience 8 λ_i^8.

La simulation du BSC impacte donc de manière mesurable les propriétés du DNN obtenu, ce qui pourrait être utilisé dans de futurs travaux pour atteindre un résultat similaire, tout en évitant l'étape de simulation. Il resterait cependant à lier la régularisation spectrale d'un

DNN à la configuration matérielle adéquate, ce qui demeure une question ouverte.

4.6 Synthèse du chapitre

Dans ce chapitre, nous nous sommes intéressés à optimiser le déploiement d'un DNN sur une mémoire NTC avec un niveau de bruit différencié par couche grâce à l'algorithme LaNMax afin d'optimiser l'énergie allouée au stockage des paramètres de ce DNN, mais également à comprendre les impacts d'un entraînement simulant des pannes matérielles sur les paramètres du DNN ainsi obtenus. Cette approche par couche est une évolution de l'approche considérant un taux de pannes uniforme pour l'ensemble du DNN que l'on peut observer dans la littérature [36]. Nous avons considéré que la fiabilité d'une cellule mémoire décroît exponentiellement avec les réductions énergétiques permises par le NTC. Dans ce contexte, l'algorithme LaNMax permet d'automatiser la recherche d'un niveau de fiabilité pour les paramètres de chaque couche d'un DNN, tout en retenant un niveau de performance intéressant sur la tâche de classification CIFAR-10 étudiée. Ce résultat est atteint en utilisant une optimisation par descente de gradient sur une fonction de perte modélisant le compromis énergie-performance d'un DNN, conjoint au traditionnel algorithme d'optimisation d'un DNN et réutilisant les résultats de ce dernier afin d'estimer le gradient numérique de notre mémoire stochastique simulée tout en consommant ainsi peu de ressources de calcul supplémentaires.

Les résultats obtenus sur un réseau WideResNet binarisé sur la tâche de classification CIFAR-10 entraîné avec LaNMax obtiennent dans certains cas de figure la même précision qu'un réseau opérant sur une mémoire fiable tout en utilisant trois fois moins d'énergie sur une mémoire NTC paramétrée par ce même algorithme. Comparativement à une approche employant un taux de panne uniforme pour l'ensemble du DNN, l'approche par couche permet de retenir une meilleure performance de DNN à budget énergétique similaire.

Cet algorithme d'entraînement de DNN tel que proposé peut être utilisé pour réaliser des implémentations de solutions d'apprentissage profond sur des dispositifs à très faible consommation d'énergie en augmentant l'attrait de puces CMOS fonctionnant en NTC, mais également sur des technologies émergentes disposant d'un compromis énergie-fiabilité pareillement contrôlable.

Forts de ce premier résultat encourageant, nous cherchons maintenant à proposer une méthode ne nécessitant pas de connaître au préalable les caractéristiques matérielles du compromis énergie-fiabilité pour un matériel analogique.

CHAPITRE 5 PRÉPARATION DE RÉSEAUX DE NEURONES PROFONDS POUR UN MATÉRIEL ANALOGIQUE DE FIABILITÉ RÉDUITE

5.1 Introduction

L'obtention de DNN robustes à un bruit matériel est principalement atteint en simulant les aléas du matériel cible lors de l'entraînement du DNN, comme nous l'avons fait au chapitre précédent. Des degrés variés de précision dans la simulation du matériel cible peuvent être utilisés : d'un simple bruit aléatoire sur les paramètres du DNN à une simulation précise du modèle de pannes. En particulier, coupler un bruit sur les paramètres du DNN à une troncature basée sur la magnitude permet d'améliorer significativement la robustesse d'un DNN en incitant l'entraînement de ce dernier à introduire de la redondance dans les résultats intermédiaires comme présenté section 3.2.4.

Cependant, ces méthodes visant à augmenter la robustesse d'un DNN introduisent un compromis entre la performance du DNN et la robustesse atteinte, réduisant la première pour améliorer la seconde. De plus, ces méthodes reposent sur une connaissance a priori du matériel sur lequel sera déployé le DNN pour sa phase d'utilisation afin de modéliser au mieux les aléas matériels lors d'un entraînement *hardware-aware*. Or, ce besoin de connaissances n'est pas toujours rempli et impose des limites sur le type de matériel cible pour le déploiement du DNN. Enfin, la simulation des aléas matériels impose un surplus de calcul dont le degré varie selon la complexité des simulations.

Ce chapitre examine la possibilité d'améliorer la robustesse d'un DNN opérant sur une technologie de calcul en mémoire, sans pour autant sacrifier la performance du DNN dans un environnement de calcul fiable, et ce, sans information préalable sur le matériel cible.

Pour ce faire, nous étudions le lien entre la métrique d'irrégularité (*sharpness*) de la fonction de perte d'un DNN, présentée section 3.2.5, et la robustesse. Une forte irrégularité caractérise des changements brusques dans la fonction de perte du DNN pour des changements limités dans les paramètres de ce dernier. La réduction de cette irrégularité, et donc l'augmentation de la régularité de la fonction de perte, pourrait ainsi concorder avec nos objectifs. En conséquence, nous formulons l'hypothèse de travail suivante : la régularité de la fonction de perte d'un DNN serait associée à une plus grande robustesse.

Notre objectif spécifique est de promouvoir une robustesse au bruit et variations observés dans le matériel, c'est-à-dire de conserver un maximum de performance pour un DNN dont

les paramètres sont exposés à des variations aléatoires trouvant leur origine dans les aléas matériels. C'est une différence notable avec les modèles d'attaque sujets à un bruit antagoniste [157] pour lesquels les perturbations ont pour objectif de nuire au maximum à la performance du DNN. En dépit des différences, il y a une certaine similitude entre les deux approches dans l'effet d'un bruit appliqué aux paramètres d'un DNN.

Dans le cas de la robustesse matérielle, le bruit injecté sur les paramètres d'un DNN dépend des variations aléatoires du matériel cible sur lequel est déployé le DNN. Ces variations aléatoires trouvent leur origine dans l'utilisation d'une mémoire memristor efficiente en énergie, mais sujette à ces variations. En d'autres termes, après l'entraînement du DNN sur un matériel fiable, nous étudions l'effet du déploiement de ce même DNN sur une mémoire fautive sur la performance atteinte dans sa phase d'utilisation.

Précisément, nous cherchons à améliorer la robustesse du DNN tout en conservant la performance obtenue avec du matériel fiable en l'absence de connaissance ou de simulation des aléas matériels lors de la phase d'entraînement. Ce faisant, nous visons à avoir une approche généraliste ne sacrifiant ni la performance du DNN ni les matériels cibles pour la phase de déploiement.

Pour ce faire, nous proposons dans un premier temps de confirmer notre hypothèse liant la régularité de la fonction de perte et la robustesse du DNN. Nous proposons dans un second temps une méthode SAM appliquée durant l'entraînement du DNN visant à obtenir des DNN performants sur matériel fiable et sur du matériel de fiabilité réduite en exploitant une méthode appelée : *sharpness-aware minimization scaled by outlier normalization* (SAMSON, optimisation de DNN conscient de la régularité de la fonction de perte mis à l'échelle par les valeurs extrêmes). La méthode SAMSON reformule la mesure de la régularité adaptative pour prendre en compte, en sus de la magnitude des paramètres du DNN, l'échelle de la distribution des paramètres du DNN. En proposant une mesure de la régularité tenant compte de ces paramètres aux valeurs extrêmes, nous démontrons que cette mesure observe une forte corrélation avec la robustesse du DNN. Ceci suggère que l'optimisation d'une telle mesure de régularité durant l'entraînement du DNN est une méthode efficace pour promouvoir la robustesse durant la phase d'utilisation du DNN sur un matériel de fiabilité réduite. Ces observations sont conduites pour un modèle de bruit générique sur plusieurs architectures de DNN et jeux de données d'entraînement ainsi que sur des simulations réalistes d'un bruit matériel. Dans l'ensemble, nos résultats démontrent l'utilité de SAMSON pour améliorer la robustesse de DNN mis en œuvre dans un environnement matériel de fiabilité limitée, tout en conservant une performance intéressante dans un environnement fiable.

Les principales contributions de ce chapitre sont les suivantes :

— Nous montrons que la performance d'un DNN sur matériel PIM tel que les memristors est liée à la régularité de la fonction de perte dans le voisinage des paramètres du DNN.
— Nous présentons des résultats expérimentaux sur plusieurs tâches et architectures de DNN populaires de convolution et Transformers confirmant l'amélioration de la performance de DNN soumis à un modèle de bruit de memristor en utilisant des méthodes d'optimisation conscientes de la régularité. En fonction de l'architecture et de la tâche, l'entraînement avec une méthode consciente de la régularité permet de conserver la même précision que des modèles entraînés avec les méthodes de référence et déployés sur matériel fiable mais le réalisant sur des crossbars de memristor avec un rapport signal sur bruit compris entre 14 et 25dB.
— De plus, nous présentons une nouvelle méthode d'entraînement consciente de la régularité (SAMSON) améliorant la performance de généralisation et la robustesse en comparaison avec les méthodes existantes.

Le reste de ce chapitre est organisé comme suit : la méthode SAMSON, que nous proposons pour améliorer la robustesse d'un DNN au bruit matériel, est présentée section 5.2. Nous comparons ensuite les méthodes d'entraînement *hardware-aware*, de *clipping* et d'entraînement de type SAM au travers d'expériences sur la capacité de généralisation des DNN section 5.3 et sur leur robustesse au bruit matériel section 5.4. Nous évaluons également la robustesse des DNN obtenus par ces méthodes à des sources de bruit tierces section 5.5, avant de conclure dans la section 5.6.

5.2 Optimisation de DNN conscient de la régularité de la fonction de perte mis à l'échelle par les valeurs extrêmes (SAMSON)

Dans ce chapitre, nous proposons une mesure de la régularité de la fonction de perte mise à l'échelle par les valeurs extrêmes des paramètres du DNN, ou SAMSON. Succinctement, notre approche est non seulement influencée par la magnitude des paramètres, comme ASAM, mais également par le domaine occupé par les paramètres du DNN afin de calculer la perturbation optimale $\epsilon^\star$ d'un paramètre w. Le conditionnement de la mesure de régularité sur la magnitude et le domaine des paramètres vise à obtenir une taille de voisinage normalisé pour toutes les couches du DNN. Cette caractéristique est particulièrement importante en présence de couches de normalisation, telle que la normalisation par lot (*batch-normalization*), puisque l'échelle de la distribution des paramètres entre chaque couche peut varier grandement, ce faisant créant une divergence dans la distribution des perturbations appliquées dans le DNN selon la couche d'appartenance.

Nous proposons de prendre en compte les valeurs extrêmes d'un tenseur de paramètres, c'est-

à-dire la valeur absolue maximale d'un tenseur de paramètres d'une couche donnée, avec une simple opération de mise à l'échelle du voisinage d'un paramètre $w^{(\ell)}$ selon la norme p de tous les paramètres du tenseur $\boldsymbol{\mathcal{W}}^{(\ell)}$ d'une couche ℓ. Pour ce faire, nous introduisons $\boldsymbol{\mathcal{W}}_n^{(\ell)}$ la valeur normalisée du tenseur de paramètres de la couche ℓ, soit

$$\boldsymbol{\mathcal{W}}_n^{(\ell)} = \frac{\boldsymbol{\mathcal{W}}^{(\ell)}}{\|\boldsymbol{\mathcal{W}}^{(\ell)}\|_p}\,. \tag{5.1}$$

Ainsi, nous considérons le problème d'optimisation *min-max* suivant :

$$\min_{\boldsymbol{\mathcal{W}}} \max_{\|\boldsymbol{\epsilon}|\boldsymbol{\mathcal{W}}_n|^{-1}\|\leq\rho} \mathcal{L}(\boldsymbol{\mathcal{W}}+\boldsymbol{\epsilon}), \tag{5.2}$$

ce qui nous mène à la perturbation suivante pour chaque paramètre au pire des cas :

$$\boldsymbol{\epsilon}^*_{\text{SAMSON}}(\boldsymbol{\mathcal{W}}) = \rho\,\frac{(\boldsymbol{\mathcal{W}}_n \circ \boldsymbol{\mathcal{W}}_n)\,\nabla\mathcal{L}\boldsymbol{\mathcal{W}})}{\||\boldsymbol{\mathcal{W}}_n|\nabla\mathcal{L}(\boldsymbol{\mathcal{W}})\|}\,. \tag{5.3}$$

Nous notons que le choix de la norme p impacte l'effet des paramètres extrêmes dans le calcul de la perturbation optimale. C'est une approche distincte de l'expérience d'ablation de [142] où différentes normes sont utilisées pour définir la taille (non adaptative) du voisinage des paramètres, avec la norme ℓ_2 atteignant les meilleurs résultats. Sans modifier ce choix par défaut de la norme ℓ_2, notre méthode applique une normalisation distincte afin de contrôler l'importance des paramètres extrêmes sur le voisinage de taille adaptative de chaque paramètre. Dans nos expériences, nous utiliserons les normes $p = \{2, \infty\}$. Afin de faciliter la présentation, nous utiliserons par la suite la notation SAMSON_2 pour le choix de $p = 2$ et la notation SAMSON_∞ pour $p = \infty$. Une suggestion d'implémentation de SAMSON avec la librairie Pytorch [169] est visible annexe A.

Nous illustrons les perturbations optimales pour un paramètre w d'une valeur donnée dans le cas de SAMSON, ASAM et SAM figure 5.1, en supposant que $\nabla\mathcal{L}(w) = 1$ pour tout w par simplicité. Afin de démontrer l'adaptabilité de SAMSON à la magnitude et au domaine des tenseurs de paramètres, nous utilisons différents domaines de paramètres symétriques. Pour le besoin de cette illustration, nous générons un tenseur de paramètres échantillonné uniformément dans l'intervalle $[-0{,}3,\ 0{,}3]$ avec un pas de 0,05. Les différents domaines sont obtenus en tronquant les échantillons (dans les intervalles $\pm 0{,}2$, $\pm 0{,}15$, $\pm 0{,}1$, et $\pm 0{,}05$), et sont représentés par des lignes verticales pointillées différenciées par leurs couleurs. Ces mêmes couleurs sont réutilisées pour les courbes des perturbations optimales de SAMSON du domaine correspondant. Nous pouvons y observer que ϵ^*_{SAM} est indépendant de la valeur de w et

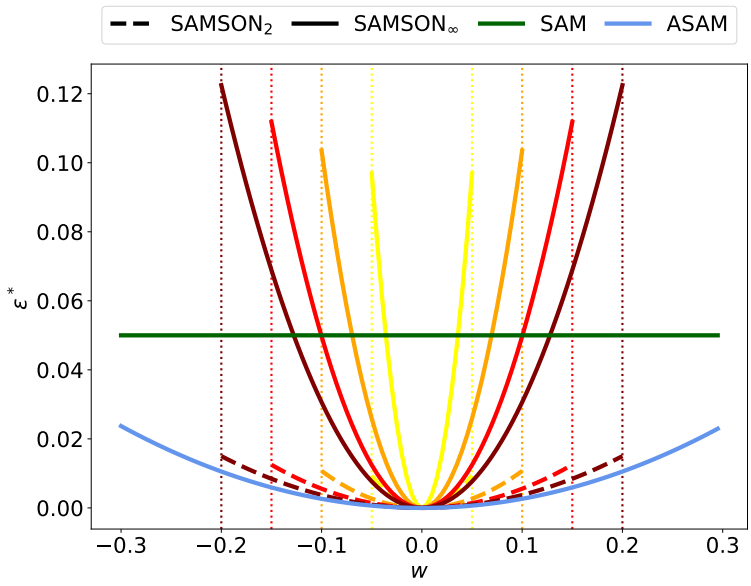


FIGURE 5.1 Perturbations optimales $\epsilon^\star$ de SAMSON, ASAM, et SAM. Chaque ligne verticale pointillée représente différents domaines de paramètres [-c, c], avec $c \in \{0{,}05, 0{,}1, 0{,}15, 0{,}2\}$.

du domaine du tenseur de paramètres, ce qui se traduit par une ligne constante entièrement définie par ρ. Dépendant de ρ et de la valeur de w, la magnitude de ce dernier impacte ϵ^*_{ASAM}. Cependant, puisque ASAM est indépendant du domaine des paramètres, on n'observe pas de changement de ϵ^*_{ASAM} selon les différents domaines de paramètres. Enfin, avec SAMSON, on observe une dépendance de $\epsilon^*_{\text{SAMSON}}$ selon le domaine des paramètres, la magnitude de w et ρ. On obtient ainsi une courbe de SAMSON distincte pour chaque domaine des paramètres, avec SAMSON$_2$ mettant moins d'emphase sur les valeurs extrêmes des paramètres, à l'inverse de SAMSON$_\infty$.

En dépit de son indépendance à toute forme de troncature des paramètres, par construction SAMSON est naturellement adapté à être utilisé en combinaison avec des méthodes permettant de restreindre le domaine des paramètres du DNN. Il est par exemple utile d'entraîner le DNN avec une troncature agressive des paramètres [136] pour améliorer la robustesse du DNN lors de la phase d'utilisation. Lors d'une troncature, le paramètre c est le domaine de troncature symétrique. Avec une troncature agressive, les paramètres sont restreints à un domaine très limité afin de promouvoir la robustesse : par exemple $c \in \mathbb{R} : 0 < c < 1$. Un pseudo-code présentant SAMSON utilisé en combinaison avec une troncature agressive est présenté algorithme 2.

Algorithme 2 SAMSON combiné avec une troncature agressive.

Entrée: paramètres initiaux $\boldsymbol{\mathcal{W}}_0$, domaine de troncature agressif c, taux d'apprentissage ι, taille du voisinage ρ, norme p

$\boldsymbol{\mathcal{W}} \leftarrow \boldsymbol{\mathcal{W}}_0$

tant que non convergence **faire**

Échantillonnage mini-lot S

$\boldsymbol{\epsilon} = \rho \frac{(\boldsymbol{\mathcal{W}}_n \circ \boldsymbol{\mathcal{W}}_n) \nabla \mathcal{L}_S \boldsymbol{\mathcal{W}})}{\| |\boldsymbol{\mathcal{W}}_n| \nabla \mathcal{L}_S(\boldsymbol{\mathcal{W}}) \|}$

$\boldsymbol{\mathcal{W}} \leftarrow \boldsymbol{\mathcal{W}} - \iota \nabla \mathcal{L}_S(\boldsymbol{\mathcal{W}} + \boldsymbol{\epsilon})$ ▷ mise à jour des paramètres

$\boldsymbol{\mathcal{W}} \leftarrow \text{clip}(\boldsymbol{\mathcal{W}}, c)$ ▷ troncature des paramètres (optionnel)

fin tant que

5.3 Performance de généralisation

Avant d'étudier la robustesse de DNN, nous analysons dans un premier temps si l'utilisation de SAMSON impacte négativement la performance de généralisation du DNN dans un environnement fiable. Puisqu'il s'agit de l'environnement le plus répandu, un tel impact serait préjudiciable à l'intérêt de notre méthode. Afin de tester ce scénario, nous étudions la performance de généralisation de SAMSON, ASAM, SAM et le SGD de référence pour des architectures de DNN convolutionnelles et Transformers sur les tâches CIFAR-10, CIFAR-100 [130], ImageNet-1K [32] et IWSLT'14 [178].

5.3.1 Expériences sur les jeux de données CIFAR-10 et CIFAR-100

Scénario expérimental et choix d'hyperparamètres

Pour réaliser ces expériences, nous nous basons sur des architectures convolutionnelles populaires sur ces jeux de données : ResNet-34, ResNet-50 [173], MobileNetV2 [78], VGG-13 [161] et DenseNet-40 [179] [1].

En nous basant sur les hyperparamètres utilisés dans [150], tous les modèles sont entraînés pour 200 époques avec une taille de mini-lots fixée à 128, pour un taux d'apprentissage ι débutant à 0,1 et divisé par 10 toutes les 50 époques. Les ensembles d'entraînement sont soumis à des augmentations aléatoires composées de remplissage (*padding*) de 4 pixels dans la dimension spatiale, recadrage aléatoire (*crop*) sur 32 pixels et retournement horizontaux avec une probabilité 50% avant d'être normalisés. La fonction de perte à minimiser $\mathcal{L}$ est la fonction d'entropie croisée. Le paramètre β contrôlant le moment est fixé à 0,9 et le paramètre λ contrôlant l'atténuation des pondérations à $5 * 10^{-4}$. Les entraînements sont effectués avec un GPU Nvidia RTX8000 et un cœur de CPU.

1. Implémentations disponibles à l'adresse suivante : `https://github.com/kuangliu/pytorch-cifar`

De plus, nous utilisons les tailles de voisinage par défaut pour SAM et ASAM, tel que proposé dans leurs publications originelles : sur CIFAR-10, nous fixons $\rho = 0{,}05$ pour SAM et $\rho = 0{,}5$ pour ASAM ; et pour CIFAR-100 nous fixons $\rho = 0{,}1$ pour SAM et $\rho = 1{,}0$ pour ASAM. Pour permettre une comparaison directe des méthodes, nous donnons les résultats des variantes de SAMSON en utilisant le même ρ qu'ASAM. Une optimisation de l'ensemble de ces hyperparamètres serait possible avec un budget calculatoire plus élevé. Néanmoins, l'utilisation de valeurs préconisées dans les travaux pertinents de la littérature nous assure d'obtenir des résultats avec une performance correcte.

Les résultats présentent la moyenne et l'écart-type de la précision sur l'ensemble de test pour trois entraînements avec un noyau de générateur de nombre aléatoire distinct. Ce nombre de répétitions est similaire aux travaux de référence [150], mais pourrait paraître limité : il convient donc d'apprécier avec une certaine mesure les gains rapportés. Néanmoins, les tests étant effectués sur deux jeux de données et cinq architectures de DNN, on pourra apprécier l'allure générale de la performance des algorithmes d'optimisation employés.

Résultats

Les résultats obtenus sont visibles tableau 5.1 pour CIFAR-10 et tableau 5.2 pour CIFAR-100. Par exemple, au tableau 5.1, on observe pour le modèle ResNet-34 que SAMSON_2 atteint une performance moyenne de 95,96 avec un écart-type $\pm 0{,}34$: cette performance moyenne est supérieure à toutes les autres méthodes, tandis que l'écart-type est supérieur. On peut donc en conclure que SAMSON_2 peut produire des résultats plus performants que le reste des méthodes, avec une précaution toutefois, à cause de la variance relativement élevée des résultats obtenus.

Dans ces tableaux, on observe en général que SAMSON ne mène pas à un amoindrissement de la performance de généralisation : au contraire, dans la plupart des cas, une variante de SAMSON atteint une légère amélioration en comparaison avec SGD, SAM et ASAM. Pour les variantes de SAMSON, il n'y a pas de choix de la norme p supérieur à un autre : c'est donc un hyperparamètre dont la performance dépendra de la tâche et de l'architecture de DNN utilisée. Le seul cas recensé où une variante de SAMSON n'est pas la meilleure méthode est pour l'architecture DenseNet-40 sur la tâche CIFAR-10. Cependant, la performance des deux variantes est dans l'écart-type des résultats d'ASAM pour cette architecture et cette tâche : la différence de performance entre les deux méthodes n'est donc pas statistiquement significative.

TABLEAU 5.1 Performance de généralisation pour 3 initialisations aléatoires (précision moyenne sur l'ensemble de test $\%_{\pm \text{écart-type}}$) des différentes méthodes sur plusieurs architectures de DNN entraînées et testées sur CIFAR-10. La méthode avec la moyenne la plus élevée pour chaque architecture de DNN est indiquée en gras.

Méthode	**ResNet-34**	**ResNet-50**	**MobileNetV2**	**VGG-13**	**DenseNet-40**
SGD	$95{,}84_{\pm 0{,}13}$	$94{,}36_{\pm 0{,}09}$	$94{,}62_{\pm 0{,}06}$	$94{,}19_{\pm 0{,}04}$	$91{,}76_{\pm 0{,}11}$
SAM	$95{,}80_{\pm 0{,}07}$	$94{,}24_{\pm 0{,}13}$	$94{,}91_{\pm 0{,}07}$	$94{,}52_{\pm 0{,}07}$	$92{,}27_{\pm 0{,}30}$
ASAM	$95{,}85_{\pm 0{,}22}$	$94{,}42_{\pm 0{,}57}$	$95{,}37_{\pm 0{,}04}$	$94{,}68_{\pm 0{,}07}$	$\mathbf{92{,}57_{\pm 0{,}06}}$
SAMSON_2	$\mathbf{95{,}96_{\pm 0{,}34}}$	$\mathbf{95{,}09_{\pm 0{,}21}}$	$95{,}29_{\pm 0{,}17}$	$\mathbf{94{,}73_{\pm 0{,}12}}$	$92{,}54_{\pm 0{,}14}$
SAMSON_∞	$95{,}76_{\pm 0{,}29}$	$\mathbf{94{,}94_{\pm 0{,}09}}$	$\mathbf{95{,}41_{\pm 0{,}09}}$	$94{,}66_{\pm 0{,}02}$	$92{,}49_{\pm 0{,}13}$

TABLEAU 5.2 Performance de généralisation (pour 3 initialisations aléatoires précision moyenne sur l'ensemble de test $\%_{\pm \text{écart-type}}$) des différentes méthodes sur plusieurs architectures de DNN entraînées et testées sur CIFAR-100. La méthode avec la moyenne la plus élevée pour chaque architecture de DNN est indiquée en gras.

Méthode	**ResNet-34**	**ResNet-50**	**MobileNetV2**	**VGG-13**	**DenseNet-40**
SGD	$74{,}32_{\pm 1{,}32}$	$74{,}35_{\pm 1{,}23}$	$75{,}44_{\pm 0{,}07}$	$72{,}78_{\pm 0{,}22}$	$68{,}52_{\pm 0{,}25}$
SAM	$75{,}62_{\pm 0{,}33}$	$75{,}36_{\pm 0{,}01}$	$76{,}81_{\pm 0{,}18}$	$73{,}86_{\pm 0{,}40}$	$69{,}14_{\pm 0{,}36}$
ASAM	$76{,}91_{\pm 0{,}44}$	$77{,}88_{\pm 0{,}85}$	$77{,}28_{\pm 0{,}10}$	$74{,}12_{\pm 0{,}01}$	$70{,}21_{\pm 0{,}25}$
SAMSON_2	$\mathbf{77{,}68_{\pm 0{,}57}}$	$\mathbf{78{,}22_{\pm 0{,}67}}$	$77{,}24_{\pm 0{,}13}$	$\mathbf{74{,}77_{\pm 0{,}23}}$	$69{,}94_{\pm 0{,}36}$
SAMSON_∞	$\mathbf{77{,}60_{\pm 0{,}78}}$	$77{,}81_{\pm 1{,}32}$	$\mathbf{77{,}61_{\pm 0{,}23}}$	$\mathbf{74{,}59_{\pm 0{,}15}}$	$\mathbf{70{,}34_{\pm 0{,}37}}$

5.3.2 Expériences sur le jeu de données ImageNet-1K

Pour aller plus loin dans la variété d'architectures, de jeu de données et de protocoles d'entraînement, nous effectuons une expérience d'ajustement sur un modèle de ResNet-18 pré-entraîné sur ImageNet-1K [32] et disponible dans la bibliothèque logicielle Pytorch [169].

Scénario expérimental et choix d'hyperparamètres

Dans un contexte d'ajustement, nous réalisons un faible nombre d'époques de SGD tout en modifiant avec précaution les paramètres du DNN pré-entraîné. Ainsi, cet ajustement est effectué sur 10 époques pour une taille de mini-lot de 400. Ce dernier choix est motivé par la taille importante du jeu de données ImageNet, nous cherchons donc à maximiser la taille de mini-lot utilisée pour accélérer l'ajustement. La fonction de perte à minimiser $\mathcal{L}$ est la

fonction d'entropie croisée. Les paramètres de SGD utilisés sont un taux d'apprentissage $\iota = 0{,}001$, un moment $\beta = 0{,}9$ et une atténuation des pondérations $\lambda = 0{,}0001$. On remarque que le choix de ces derniers permettra des mises à jours de petite ampleur des paramètres du DNN. L'ensemble d'entraînement est soumis à des augmentations aléatoires composées de recadrage aléatoire (*crop*) sur 224 pixels et retournement horizontaux avec une probabilité 50% avant d'être normalisés. L'ensemble de validation est mis à l'échelle de 256 pixels (*resize*), puis recadré en son centre sur 224 pixels (*center-crop*) avant d'être normalisé. Par ailleurs, comme nous partons d'un DNN pré-entraîné un seul point de données sera rapporté. Nous réalisons cependant plusieurs entraînements avec une initialisation aléatoire pour compléter ce résultat, comme rapporté au prochain paragraphe. Le matériel utilisé est un GPU Nvidia A100 40GB et 6 cœurs de CPU.

En l'absence de ρ par défaut pour l'ajustement sur ImageNet-1K dans la publication présentant ASAM, nous expérimentons plusieurs choix de cet hyperparamètre tel que décrit tableau 5.6 et présentons les meilleurs résultats pour SAM, ASAM et SAMSON. Ces derniers sont obtenus en utilisant $\rho = 0{,}05$ pour SAM, $\rho = 0{,}2$ pour SAMSON et $\rho = 0{,}5$ pour ASAM. Avec ce résultat, nous entraînons également un ResNet-18 et un MobileNetV3 [180] en partant d'une initialisation aléatoire pour une durée de 90 époques en utilisant les mêmes hyperparamètres, en changeant toutefois le taux d'apprentissage à $\iota = 0{,}1$ avec une division de ce dernier par 10 toutes les 30 époques. Trois entraînement sont ainsi réalisés : les mêmes précautions qu'aux expériences sur CIFAR-10 et CIFAR-100 sont donc à prendre dans l'analyse des résultats.

Résultats

Les résultats ainsi obtenus sont présentés tableau 5.3. À nouveau, nous observons que les variantes de SAMSON ne dégradent pas la performance de généralisation et, au contraire, l'améliorent légèrement en comparaison avec les méthodes de référence, et ce, dans les deux cas d'ajustement et d'entraînement.

5.3.3 Expériences sur des architectures Transformers de grande taille

Afin de continuer cet effort d'évaluation de l'impact de SAMSON sur la potentielle dégradation des performances de généralisation et de l'étendre à des architectures plus modernes, mais aussi plus grandes, nous présentons maintenant des résultats sur des architectures Transformers [181] pour une tâche de classification d'images sur le jeu de données ImageNet-1K, mais également sur une tâche de traduction automatique. Pour cette tâche, nous considérons la règle de descente de gradient Adam [132] à la place de SGD, étant donné qu'il s'agit de

TABLEAU 5.3 Performance de généralisation (précision moyenne sur l'ensemble de validation $\%_{\pm \text{ écart-type}}$) des différentes méthodes pour les architectures ResNet-18 et MobileNetV3 sur le jeu de données ImageNet-1K pour le scénario d'ajustement et d'entraînement. La méthode avec la moyenne la plus élevée pour chaque architecture de DNN est indiquée en gras. Pour les DNN entraînés, les résultats sont moyennés sur 3 initialisations aléatoires.

Méthode	**Ajustement**		**Entraînement**			
	ResNet-18		**ResNet-18**		**MobileNetV3**	
	top-1	**top-5**	**top-1**	**top-5**	**top-1**	**top-5**
SGD	69,758	89,078	$69{,}91_{\pm,04}$	$89{,}21_{\pm,05}$	$69{,}30_{\pm,01}$	$89{,}01_{\pm,01}$
SAM	70,356	89,480	$70{,}01_{\pm,06}$	$89{,}28_{\pm,06}$	$69{,}32_{\pm,02}$	$88{,}89_{\pm,02}$
ASAM	70,348	89,428	$70{,}15_{\pm,06}$	$89{,}24_{\pm,07}$	$69{,}57_{\pm,08}$	$88{,}90_{\pm,06}$
SAMSON_2	**70,358**	**89,486**	$\mathbf{70{,}16_{\pm,08}}$	$\mathbf{89{,}38_{\pm,10}}$	$\mathbf{69{,}62_{\pm,01}}$	$\mathbf{89{,}14_{\pm,01}}$
SAMSON_∞	**70,366**	**89,504**	$\mathbf{70{,}23_{\pm,06}}$	$\mathbf{89{,}35_{\pm,05}}$	$69{,}57_{\pm,03}$	$88{,}99_{\pm,03}$

la méthode de référence pour les tâches de traitement de texte. En outre, ce choix étend la généralité de notre analyse de SAMSON au-delà de SGD que nous avons utilisé jusqu'ici. Le matériel utilisé est un GPU Nvidia A100 80GB et 6 cœurs de CPU.

Pour la tâche de classification d'images, nous ajustons un modèle Transformer de vision (ViT) [182], en particulier un modèle ViT-Base de 86,6 millions de paramètres utilisant des extraits d'images de taille 16×16 que nous désignerons par ViT-B-16. Ce modèle fut pré-entraîné sur le jeu de données ImageNet-21K [183] et ajusté avec SGD sur ImageNet-1K atteignant une précision sur l'ensemble de validation de 81,79% [2]. Dans nos expériences, nous utilisons les différentes méthodes conscientes de la régularité pour ajuster ce modèle sur 15 époques. Pour ce faire, nous utilisons les mêmes configurations que pour les expériences sur ImageNet-1K, dont le choix de ρ de chaque méthode, excepté la taille de mini-lots que nous fixons ici à 480 pour accélérer l'ajustement. Les résultats ainsi obtenus sont présentés tableau 5.4. Nous observons que les variantes de SAMSON améliorent la performance de généralisation du modèle pré-entraîné. À l'inverse, ASAM fut incapable de retrouver la performance de généralisation du modèle pré-entraîné, tandis que SAM atteint une légère amélioration. De surcroît, les deux variantes de SAMSON atteignent une performance plus intéressante que toutes les autres méthodes.

Pour la tâche de traduction automatique, nous utilisons la même configuration que présentée

2. Disponible à l'adresse suivante : `https://huggingface.co/timm/vit_base_patch16_224.orig_in21k_ft_in1k`. Instructions d'entraînement à l'adresse suivante : `https://github.com/pytorch/vision/tree/main/references/classification#vit_b_16`.

Méthode	Ajustement ViT-B-16 précision de validation
SGD	81,79
SAM	81,80
ASAM	81,77
SAMSON_2	**81,83**
SAMSON_∞	**81,82**

Tableau 5.4 Performance de généralisation (précision sur l'ensemble de validation %) des différentes méthodes pour le modèle ViT-B-16 ajusté sur ImageNet-1K. Les variantes de SAMSON atteignent la meilleure précision et sont visibles en gras.

par [150] pour un modèle de Transformer encodeur-décodeur de 12 couches [3], totalisant 39,4 millions de paramètres et entraîné à partir d'une initialisation aléatoire sur le jeu de données IWSLT'14 [178]. Comme effectué par [150], nous conduisons une recherche de la valeur de ρ pour les choix de valeurs visibles tableau 5.6 pour les variantes de SAMSON en utilisant l'ensemble de validation et obtenons $\rho = 0{,}5$ comme meilleur choix pour les deux variantes. Pour SAM et ASAM, nous utilisons les meilleurs ρ tels que rapporté dans [150]. Les résultats sont répliqués avec trois noyaux de générateur de nombre aléatoire distincts et présentés tableau 5.5. Nous observons que les variantes de SAMSON obtiennent un score BLEU sur l'ensemble de validation supérieur aux méthodes de référence. De plus, SAMSON_∞ obtient un score BLEU sur l'ensemble de test supérieur à toutes les autres méthodes.

Dans l'ensemble, les résultats obtenus sur ces architectures Transformers de grande taille montrent la capacité de SAMSON à atteindre une performance de généralisation à l'état de l'art sur des tâches de classification d'image et de traduction automatique.

Les résultats obtenus dans cette section mettent en relief la capacité de SAMSON à obtenir des DNN dont la performance de généralisation n'est pas dégradée dans de nombreux scénarios d'entraînements, et nous allons maintenant nous intéresser aux gains de robustesse.

5.4 Robustesse de DNN à un bruit sur les paramètres

Nous nous intéressons maintenant à analyser la relation entre la robustesse d'un DNN et son entraînement avec une méthode consciente de la régularité de la fonction de perte en

3. Implémentation disponible à l'adresse suivante : `https://github.com/facebookresearch/fairseq`. Instructions d'entraînement à l'adresse suivante : `https://github.com/facebookresearch/fairseq/blob/main/examples/translation/README.md#training-a-new-model`.

Méthode	Entraînement Transformer	
	BLEU validation	BLEU test
Adam	$35{,}34^{*}_{\pm<,01}$	$34{,}86^{*}_{\pm<,01}$
Adam+SAM	$35{,}52^{*}_{\pm,01}$	$34{,}78^{*}_{\pm,01}$
Adam+ASAM	$35{,}66^{*}_{\pm<,01}$	$35{,}02^{*}_{\pm<,01}$
Adam+SAMSON$_2$	$\mathbf{35{,}70_{\pm<,01}}$	$34{,}94_{\pm<,01}$
Adam+SAMSON$_\infty$	$\mathbf{35{,}73_{\pm<,01}}$	$\mathbf{35{,}06_{\pm<,01}}$

TABLEAU 5.5 Performance de généralisation (pour 3 initialisations aléatoires score BLEU moyen sur l'ensemble de validation et de test) des différentes méthodes pour un modèle Transformer encodeur-décodeur entraîné sur le jeu de données IWSLT'14. Les résultats présentés (*) pour Adam, Adam+SAM, et Adam+ASAM sont extraits de [150].

comparaison avec un entraînement standard par SGD. En particulier, nous nous concentrons sur l'amélioration de la robustesse à un bruit d'origine matérielle sur les paramètres du DNN par l'utilisation d'un accélérateur de réseaux de neurones exploitant le calcul en mémoire tel que décrit section 3.3.2.

Nous testons la robustesse dans une variété de scénarios en suivant les configurations de la section précédente : un modèle VGG-13 entraîné sur CIFAR-10, un modèle MobileNetV2 entraîné sur CIFAR-100, un ResNet-18 ajusté sur ImageNet-1K et un Transformer entraîné sur IWSLT'14. De plus, nous évaluons plusieurs alternatives pour la taille du voisinage de chaque méthode étant donné le contrôle du compromis entre la performance et la robustesse du DNN permis par le choix de ρ. Les choix de ρ considérés sont décrits tableau 5.6. Ainsi, un choix de $\rho = 0{,}5$ ou $\rho = 1{,}0$ atteint le meilleur compromis pour SAMSON et ASAM, tandis que $\rho = 0{,}05$ ou $\rho = 0{,}1$ sont les meilleurs pour SAM. Afin de présenter des résultats clairs, nous retenons le meilleur ρ pour chaque méthode. Enfin, nous notons que $\sigma_c = 0.0$ dans les résultats présentés se réfère au cas particulier où aucun bruit n'est appliqué aux paramètres du DNN, c'est-à-dire à un environnement matériel fiable.

5.4.1 Méthodes de robustesse de référence

En plus d'une référence simple d'entraînement par SGD sans modifications, nous employons deux méthodes de référence additionnelles plébiscitées pour les gains de robustesses qu'elles permettent : l'ajout d'un bruit additif Gaussien sur les paramètres du DNN dans un entraînement *hardware-aware* tel que proposé par [109] et présenté sections 3.2.3 et 3.3.2, et une troncature agressive des paramètres du DNN tel que présenté section 3.2.4. Les modèles auxquels sont appliqués ces méthodes de référence sont entraînés à partir d'une initialisation

Tableau 5.6 Choix d'hyperparamètres pour les différentes méthodes.

Hyperparamètre	**Choix**
SAM ρ	$\{0{,}05\,;\ 0{,}1\,;\ 0{,}2\,;\ 0{,}5\}$
ASAM ρ	$\{0{,}5\,;\ 1{,}0\,;\ 1{,}5\,;\ 2{,}0\}$
SAMSON ρ	$\{0{,}1\,;\ 0{,}2\,;\ 0{,}5\,;\ 1{,}0\}$
α	$\{1{,}5\,;\ 2{,}0\,;\ 2{,}5\}$
c	$\{\pm 0{,}05\,;\ \pm 0{,}10\,;\ \pm 0{,}15\,;\ \pm 0{,}20\}$

aléatoire et utilisent les configurations d'entraînement discutées précédemment.

Le bruit additif Gaussien présenté section 3.3.2 est aléatoirement échantillonné d'une distribution Gaussienne $\mathcal{N}(0, \sigma_n^2)$, où

$$\sigma_n = \frac{\|\boldsymbol{\mathcal{W}}^{(\ell)}\|_\infty \times \sigma_g}{g_u}\,, \tag{5.4}$$

avec σ_g l'écart-type des perturbations matérielles observé en pratique. Les deux paramètres matériels sont fixés à $\sigma_g = 0{,}94$ et $g_u = g_{\max} = 25$ selon les résultats expérimentaux sur 1 million de puces mémoire PIM de type memristor [109]. Étant donné que le montant total de bruit est proportionnel au paramètre de magnitude maximale d'une couche de DNN, nous appliquons une troncature des paramètres après chaque mise à jour de ces derniers : nous utilisons le domaine $\left[-\alpha \times \sigma_{\boldsymbol{\mathcal{W}}^\ell}, \alpha \times \sigma_{\boldsymbol{\mathcal{W}}^\ell}\right]$, où $\sigma_{\boldsymbol{\mathcal{W}}^\ell}$ est l'écart-type des paramètres de la couche ℓ et α est un hyperparamètre fixé à 2,0. Ce choix de α fait suite à une série d'expériences où différents choix de $\alpha \in \{1{,}5; 2{,}0; 2{,}5\}$ furent testés, et la meilleure performance pour les modèles de DNN sur les jeux de données CIFAR-10/100 fut atteint par $\alpha = 2{,}0$. Pour l'ajustement de DNN sur le jeu de données ImageNet-1K, nous utilisons $\alpha = 2{,}5$ tel que suggéré par [109].

Pour la troncature agressive des paramètres, nous testons divers choix pour le domaine de troncature c tel que suggéré par [136] : $\{\pm 0{,}05; \pm 0{,}10; \pm 0{,}15; \pm 0{,}20\}$. En pratique, un domaine plus contraint par un c plus petit mène à des DNN de grande robustesse. Cependant, leur performance de généralisation dans un environnement fiable ou avec un faible bruit est peu intéressante par la perte d'information des paramètres de magnitude supérieure à c. Ainsi, le choix de c permet un compromis entre la performance du DNN et la robustesse de ce dernier. Dans nos expériences, nous observons que le choix $c = 0{,}2$ (et parfois, $c = 0{,}15$) offre le meilleur compromis et nous l'utilisons dans les résultats présentés.

Les choix d'hyperparamètres de ces méthodes avec bruit additif et troncature des paramètres

sont résumés table 5.6. En parallèle de l'application à SGD de ces deux méthodes alternatives, nous présentons des résultats combinants troncature agressive des paramètres et SAM, ASAM et SAMSON pour tenter de maximiser les gains de robustesse. La combinaison de SAM, ASAM et SAMSON avec un bruit additif fut également expérimentée, mais les résultats furent dominés par la combinaison avec la troncature et ne sont donc pas présentés par souci de clarté.

Enfin, afin de réduire l'impact du bruit matériel sur la performance du DNN, [109] propose également une opération de normalisation par mini-lot aux statistiques adaptatives (*adaptive batch-normalization statistics*, AdaBS). Le principe est de mettre à jour les statistiques glissantes de moyenne et d'écart-type des opérations de normalisation par mini-lot du DNN en utilisant un ensemble de calibration pour lequel les passes avant du DNN sont réalisées en simulant le bruit matériel. C'est une opération peu coûteuse, car elle ne nécessite pas de mise à jour du reste des paramètres du DNN ou de rétropropagation de gradient. Ainsi, nous utilisons les hyperparamètres initialement proposés par [109] et appliquons AdaBS à tous les modèles de DNN utilisés dans cette section.

5.4.2 Robustesse à des variations de conductances

La robustesse de modèles de DNN entraînés avec SAMSON, ASAM, SAM et SGD en combinaison avec une troncature agressive des paramètres à plusieurs niveaux de variations de conductance matérielle est illustrée figure 5.2. Nous incluons également les résultats d'entraînement avec SGD et un bruit Gaussien additif tel que discuté précédemment. Les choix d'hyperparamètres retenus pour VGG-13 sur CIFAR-10 sont présentés tableau 5.7, tableau 5.8 pour MobileNetV2 sur CIFAR-100 et tableau 5.9 pour ResNet-18 sur ImageNet-1K. Les résultats présentés sont moyennés sur trois initialisations aléatoires de DNN, sauf pour le ResNet-18 qui est ajusté comme présenté à la section précédente. Les mêmes précautions sont donc à prendre dans l'analyse de ces résultats que précédemment. On pourra cependant apprécier les changements de tendance dans la perte de performance des DNN avec l'augmentation de la variation de conductance.

Nous y observons que les variantes de SAMSON composent en majorité le front de Pareto au travers des modèles et jeux de données. Ainsi, entraîner un modèle de DNN avec SAMSON, en appliquant ou non une troncature agressive des paramètres du DNN, permet d'atteindre un compromis performance/robustesse des plus intéressants pour chaque niveau de bruit. C'est une observation qui s'applique également à un environnement fiable avec $\sigma_c = 0{,}0$, où l'une des variantes de SAMSON atteint le meilleur résultat sur l'ensemble de test ou de validation, comme nous l'avons observé section 5.3. La différence de robustesse entre

Tableau 5.7 Meilleurs hyperparamètres pour l'architecture VGG-13 entraînée sur CIFAR-10.

Méthode	**Meilleur choix**
SGD + bruit	$\alpha = 2{,}0$
SGD + troncature	$c = \pm 0{,}15$
SAM	$\rho = 0{,}1$
SAM + troncature	$\rho = 0{,}1, c = \pm 0{,}2$
ASAM	$\rho = 0{,}5$
ASAM + troncature	$\rho = 0{,}5, c = \pm 0{,}2$
SAMSON_2	$\rho = 0{,}2$
SAMSON_2 + troncature	$\rho = 0{,}5, c = \pm 0{,}2$
SAMSON_∞	$\rho = 1{,}0$
SAMSON_∞ + troncature	$\rho = 0{,}5, c = \pm 0{,}2$

Tableau 5.8 Meilleurs hyperparamètres pour l'architecture MobileNetV2 entraînée sur CIFAR-100.

Méthode	**Meilleur choix**
SGD + bruit	$\alpha = 2{,}0$
SGD + troncature	$c = \pm 0{,}2$
SAM	$\rho = 0{,}2$
SAM + troncature	$\rho = 0{,}2, c = \pm 0{,}2$
ASAM	$\rho = 1{,}0$
ASAM + troncature	$\rho = 1{,}0, c = \pm 0{,}2$
SAMSON_2	$\rho = 1{,}0$
SAMSON_2 + troncature	$\rho = 0{,}5, c = \pm 0{,}2$
SAMSON_∞	$\rho = 1{,}0$
SAMSON_∞ + troncature	$\rho = 1{,}0, c = \pm 0{,}2$

TABLEAU 5.9 Meilleurs hyperparamètres pour l'architecture ResNet-18 entraînée sur ImageNet-1K.

Méthode	**Meilleur choix**
SGD + bruit	$\alpha = 2{,}5$
SGD + troncature	$c = \pm 0{,}2$
SAM	$\rho = 0{,}1$
SAM + troncature	$\rho = 0{,}1, c = \pm 0{,}2$
ASAM	$\rho = 1{,}0$
ASAM + troncature	$\rho = 1{,}0, c = \pm 0{,}2$
SAMSON_2	$\rho = 0{,}2$
SAMSON_2 + troncature	$\rho = 0{,}5, c = \pm 0{,}2$
SAMSON_∞	$\rho = 0{,}5$
SAMSON_∞ + troncature	$\rho = 1{,}0, c = \pm 0{,}2$

les différentes méthodes est plus subtile sur le jeu de données ImageNet-1K, ce que nous attribuons à l'utilisation d'un modèle de DNN pré-entraîné avec un court ajustement de 10 époques. Néanmoins, nous observons que SAMSON est la seule méthode capable de fournir une amélioration significative de la robustesse du DNN pour un niveau de bruit élevé, par exemple pour $\sigma_c = 0{,}4$.

Dans l'ensemble, nous observons que l'utilisation de méthodes d'entraînements conscientes de la régularité de la fonction de perte (SAMSON, ASAM et SAM) propose une large amélioration de la robustesse en comparaison avec le SGD de référence. SAMSON propose le meilleur gain de robustesse, suivi par ASAM et enfin SAM. Cela se traduit non seulement par une meilleure robustesse à différents niveaux de bruit, mais également par les meilleures performances atteintes dans un environnement fiable. De plus, cette amélioration de la robustesse est amplifiée par l'utilisation conjointe de troncature agressive des paramètres. Nous notons cependant que l'utilisation de troncature agressive ou de bruit additif impacte négativement la performance du DNN en environnement fiable, comme attendu. Ce phénomène est visible dans les encarts agrandit sur la zone $\sigma_c = 0{,}0$. Ainsi, l'utilisation de ces méthodes de référence atténue l'aspect positif du gain de robustesse dans un environnement fiable ou faiblement bruité, mais présente de sérieux bénéfices dans un environnement fortement bruité. Les méthodes conscientes de la régularité représentent donc une alternative simple et efficace à la méthode traditionnelle d'entraînement avec simulation du bruit matériel pour ces architectures et ces jeux de données. Comme à la section précédente, nous nous intéressons maintenant à des modèles de DNN Transformers plus modernes et disposant de plus nombreux paramètres sur la tâche de traduction automatique.

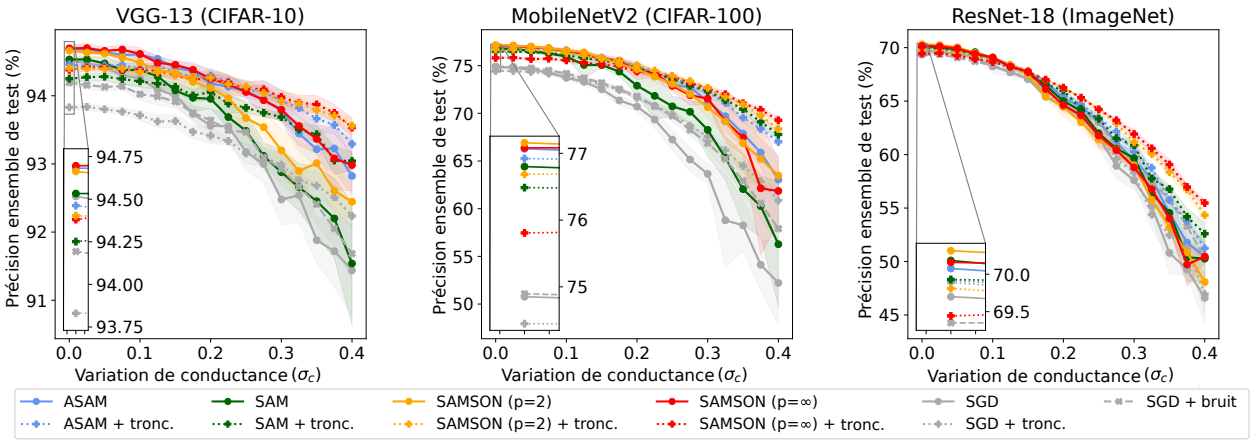


FIGURE 5.2 Performance des différentes méthodes selon la variation de conductance. Nous illustrons la moyenne et l'écart-type sur dix répétitions de l'ensemble de test pour CIFAR-10/100 et sur trois répétitions de l'ensemble de validation pour ImageNet-1K.

5.4.3 Robustesse à des variations de conductances sur architectures Transformers de grande taille

Nous reprenons notre Transformer encodeur-décodeur de la section précédente et testons sa robustesse à différents niveaux de variation de la conductance. Dans le contexte d'une architecture Transformers, il n'y a pas de couches de convolution, mais plutôt des couches linéaires qui sont stockées et calculées sur un crossbar de memristor : ce sont ces couches qui sont soumises au bruit matériel. À nouveau, nous utilisons la règle de descente de gradient Adam plutôt que SGD pour comparer les différentes méthodes en utilisant les mêmes hyperparamètres qu'à la section 5.3.3 pour trois initialisation aléatoire du DNN. Les résultats obtenus en l'absence de troncature agressive des paramètres du DNN sont présentés figure 5.3 (gauche). On peut y remarquer que SAMSON$_\infty$ atteint le meilleur compromis robustesse/performance pour chaque niveau de bruit en comparaison avec les restes des méthodes utilisées. Par ailleurs, SAMSON$_2$ obtient le deuxième meilleur compromis pour un niveau de bruit faible à moyen. Ces niveaux de bruit sont particulièrement intéressants, car ils représentent un cas d'usage plus plausible de par le maintient d'une forte performance du DNN. Enfin, l'encart agrandi confirme la performance de généralisation supérieure des variantes de SAMSON dans un environnement fiable. Nous notons cependant que nous n'avons pas réussi à répliquer les résultats rapportés par [150] pour SAM et ASAM, ce qui explique les différences subtiles entre les scores BLEU présentés ici et ceux extraits de [150] présentés au tableau 5.5.

Nous réalisons également une expérience de troncature agressive des paramètres du DNN

pour ce modèle et ce jeu de données dont les résultats sont présentés figure 5.3 (droite). Dans ce contexte, nous observons que la troncature agressive des paramètres n'améliore pas la robustesse du DNN, mais augmente paradoxalement la performance de généralisation. Ainsi, nous formulons l'hypothèse qu'un plus petit intervalle de troncature puisse être utilisé dans ce cas, au lieu de l'intervalle $c = \pm 0{,}2$ utilisé dans les résultats présentés et dans le reste de ce chapitre pour la tâche de classification d'images. Malgré ce résultat surprenant, SAMSON$_2$ demeure la méthode offrant les résultats avec le meilleur compromis performance/robustesse sur tous les niveaux de bruit. De plus, SAMSON$_\infty$ figure parmi les méthodes offrant les compromis les plus intéressants pour un niveau de bruit faible à moyen. Enfin, nous observons encore une fois que les variantes de SAMSON atteignent la meilleure performance de généralisation dans un environnement fiable, tel que visible dans l'encart agrandit.

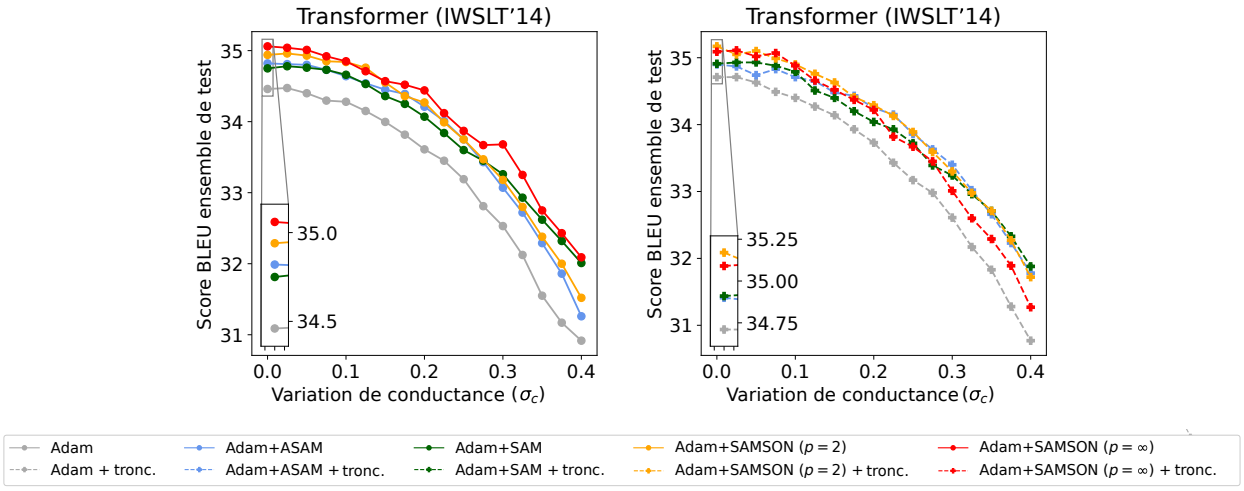


FIGURE 5.3 Performance des différentes méthodes selon la variation de conductance pour l'architecture Transformer encodeur-décodeur avec (gauche) et sans (droite) troncature agressive des paramètres. Nous illustrons la moyenne sur trois répétitions de l'ensemble de test du jeu de données IWLST'14 pour trois entraînements de DNN avec une initialisation aléatoire.

Nous étudions maintenant la corrélation entre la robustesse des DNN obtenus et la mesure de régularité de la fonction de perte.

5.4.4 Corrélation entre robustesse et régularité de la fonction de perte

Afin de mesurer la régularité de la fonction de perte, nous utilisons la métrique de m-irrégularité proposée par [142] et présentée section 3.2.5, et l'étendons à la formulation de SAMSON visible équation 5.2. En pratique, pour un ensemble d'entraînement S_{train} composé de n mini-lots S de taille m, nous calculons la différence de la fonction de perte $\mathcal{L}_S$ entre la

passe avant régulière d'un DNN et la passe avant perturbée de la perturbation optimale $\boldsymbol{\epsilon}^\star$ sur les paramètres $\boldsymbol{\mathcal{W}}$. La métrique de m-irrégularité de SAMSON est calculée comme suit :

$$\frac{1}{n}\sum_{S\in S_{\text{train}}}\frac{1}{m}\max_{\|\boldsymbol{\epsilon}|\boldsymbol{\mathcal{W}}_n|^{-1}\|\leq\rho}\mathcal{L}_S(\boldsymbol{\mathcal{W}}+\boldsymbol{\epsilon})-\mathcal{L}_S(\boldsymbol{\mathcal{W}})\,. \tag{5.5}$$

Dans nos expériences, nous utilisons la taille de mini-lots $m = 400$ pour les modèles de DNN ajustés sur ImageNet-1K et $m = 128$ pour les modèles entraînés sur CIFAR-10/100 dans les modalités d'entraînement normales, avec troncature des paramètres et avec bruit additif sur les paramètres. Les modèles de DNN étudiés sont optimisés avec un entraînement à partir d'une initialisation aléatoire pour chaque méthode d'optimisation (SGD, SAM, etc.).

D'autre part, nous mesurons la robustesse comme le différentiel de performance de généralisation entre les modèles en environnement fiable, c'est-à-dire sans variation de conductance sur les paramètres à $\sigma_c = 0{,}0$ et le même modèle avec une forte variation de conductance $\sigma_c = 0{,}4$. La mesure de ce différentiel de performance étant un processus aléatoire de par l'échantillonnage du modèle de bruit, nous répétons la mesure du différentiel pour chaque modèle de DNN. Ainsi, dix répétitions sont réalisées pour le jeu de données CIFAR-10/100 et trois répétitions pour ImageNet-1K. Nous présentons la relation entre la régularité et la robustesse des modèles VGG-13 sur CIFAR-10, MobileNetV2 sur CIFAR-100 et ResNet-18 sur ImageNet-1K en utilisant la métrique de m-irrégularité de SAMSON_2 figure 5.4. Nous pouvons y observer une forte corrélation pour chaque configuration, c'est-à-dire avec ou sans bruit additif ou troncature agressive des paramètres, sur toutes les architectures de DNN et jeux de données.

Pour confirmer formellement ces résultats visuels, nous calculons la corrélation de Pearson et la p-valeur associée pour les métriques de m-irrégularité de SAMSON, mais également de SAM et d'ASAM dans la table 5.10. Dans l'ensemble, les coefficients de corrélation sont élevés et accompagnés de p-valeur faibles, ce qui suggère une corrélation statistiquement significative entre la robustesse d'un DNN et la métrique de m-irrégularité des différentes méthodes. Parmi ces coefficients de corrélation, ceux obtenus par SAMSON sont en moyenne plus élevés que ceux de SAM et ASAM.

Ces résultats suggèrent qu'un entraînement avec SAMSON, notamment en conjonction avec des méthodes d'amélioration de la robustesse éprouvées telles que la troncature agressive des paramètres, est une méthode efficace pour obtenir des DNN robustes lors du déploiement sur un matériel fautif analogique avec un modèle d'erreurs proche de celui utilisé dans ces expériences. Nous nous intéressons enfin à la robustesse de DNN entraîné avec une méthode consciente de la régularité de la fonction de perte en présence de bruit sur les paramètres en

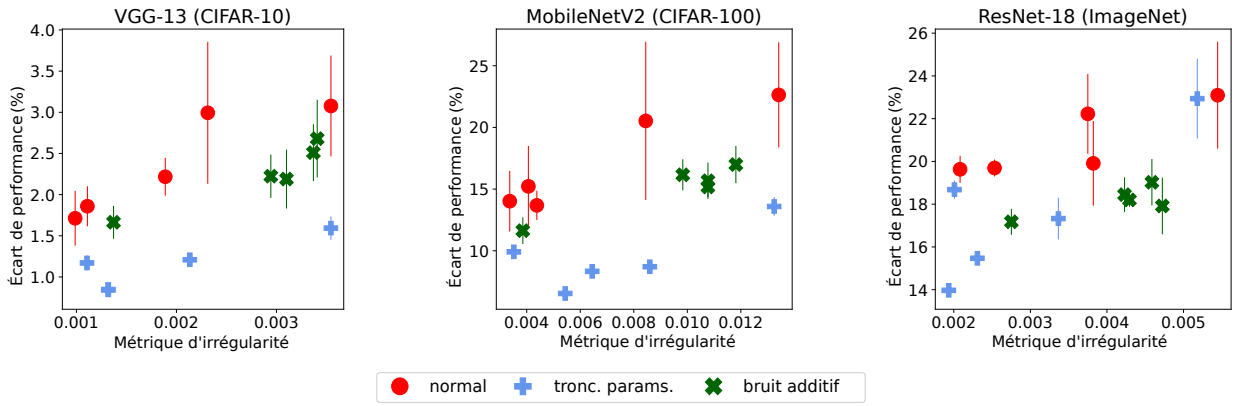


FIGURE 5.4 Corrélation entre la métrique de m-irrégularité de SAMSON_2 (équation 5.5, $\rho = 0{,}5$, $p = 2$) et robustesse d'un DNN mesurée par le différentiel de performance en environnement fiable $\sigma_c = 0{,}0$ et un environnement bruité $\sigma_c = 0{,}4$. Nous illustrons la moyenne et l'écart-type sur dix répétitions de l'ensemble de test pour CIFAR-10/100 et sur trois répétitions sur l'ensemble de validation de ImageNet-1K.

TABLEAU 5.10 Coefficients de corrélation de Pearson et p-valeurs entre les différentes mesures de m-irrégularité et la robustesse sur différentes architectures, jeux de données et modalités d'entraînement : normal (N), bruit additif (BA), et troncature des paramètres (TP).

m-sharpness	CIFAR-10 VGG-13			CIFAR-100 MobileNetV2			ImageNet ResNet-18		
	N	**BA**	**TP**	**N**	**BA**	**TP**	**N**	**BA**	**TP**
SAM	$,59_{p=,28}$	$,93_{p=,02}$	$,89_{p=,04}$	$,87_{p=,05}$	$,78_{p=,11}$	$\mathbf{,90_{p=,03}}$	$\mathbf{,94_{p=,01}}$	$,88_{p=,04}$	$\mathbf{,94_{p=,01}}$
ASAM	$\mathbf{,92_{p=,02}}$	$,93_{p=,01}$	$,87_{p=,05}$	$\mathbf{,98_{p<,01}}$	$,92_{p=,02}$	$,73_{p=,15}$	$,85_{p=,06}$	$,76_{p=,13}$	$,84_{p=,07}$
SAMSON_2	$\mathbf{,92_{p=,02}}$	$,93_{p=,01}$	$,86_{p=,05}$	$,95_{p=,01}$	$\mathbf{,95_{p=,01}}$	$,72_{p=,16}$	$,84_{p=,07}$	$,76_{p=,13}$	$,84_{p=,07}$
SAMSON_∞	$,91_{p=,02}$	$\mathbf{,96_{p<,01}}$	$\mathbf{,89_{p=,03}}$	$,97_{p<,01}$	$,80_{p=,09}$	$,83_{p=,07}$	$,86_{p=,05}$	$\mathbf{,91_{p=,02}}$	$\mathbf{,94_{p=,01}}$

conditions réalistes, c'est-à-dire mesurés sur un matériel concret.

5.4.5 Robustesse au bruit mesuré sur matériel

Précédemment, nous avons mené des expériences visant à mesurer la robustesse à un modèle de bruit générique aisément simulé. Nous souhaitons maintenant mesurer comment ces gains de robustesses théoriques se traduisent sur un matériel existant. Pour ce faire, nous réalisons des expériences avec la librairie *Analog Hardware Acceleration Kit* (AIHwKit) d'IBM [29]. Cette dernière propose une simulation de la passe avant de DNN en utilisant des perturbations tirées d'un matériel existant. En détail, cette simulation repose sur des mesures empiriques réalisées sur un million de memristors implémentés physiquement [184] afin de simuler

précisément l'effet du bruit matériel sur les paramètres du DNN [109]. Nous rapportons la performance de généralisation sur l'ensemble de validation de ImageNet-1K des différentes méthodes d'entraînement conscient de la régularité et SGD en combinaison avec une troncature agressive des paramètres du DNN sur le ResNet-18 ajusté sur ImageNet-1K obtenu aux sections précédentes. Cette performance est mesurée après une année de déploiement du DNN sur le matériel cible en simulant le bruit de programmation et de lecture des memristors. Nous rappelons qu'il y a un phénomène de dégradation des conductances programmées sur les memristors, d'où ce délai. Par ailleurs, pour mieux mesurer l'impact du bruit matériel aléatoire, la performance du DNN est évaluée à dix reprises et nous présentons la moyenne de précision obtenue. Ces résultats sont présentés figure 5.5 et confirment la tendance observée sur les expériences précédentes. Notamment, la robustesse de SGD est largement dépassée par les méthodes conscientes de la régularité, avec les variantes de SAMSON conservant un maximum de performance à l'échéance d'un an. En pratique, ce scénario apparaît quand la reprogrammation régulière des paramètres du DNN sur les crossbars n'est pas envisageable.

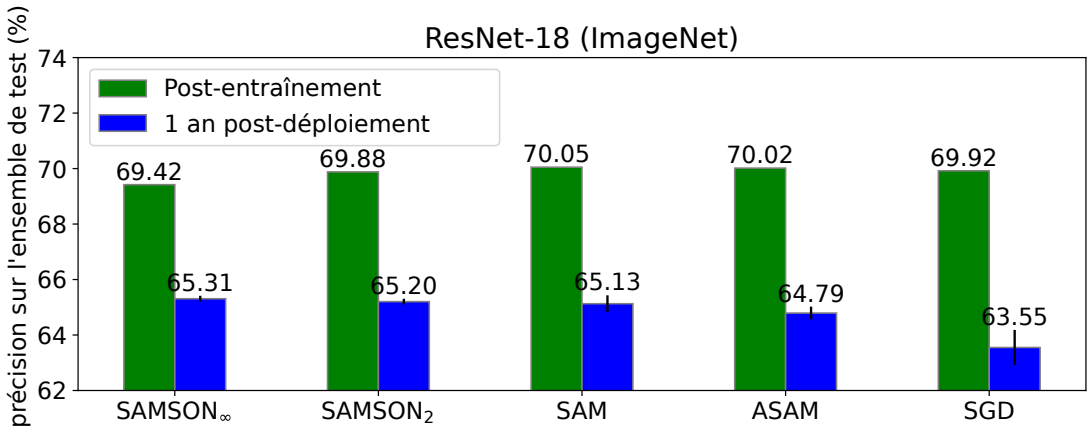


FIGURE 5.5 Performance des différentes méthodes avec une troncature agressive des paramètres du DNN pour le modèle ResNet-18 ajusté sur ImageNet-1K à la fin de l'ajustement et un an après son déploiement (les paramètres programmés sur memristors sont soumis à un phénomène de dégradation dans le temps) sur le matériel cible. Nous illustrons la moyenne sur dix répétitions de l'ensemble de validation du jeu de données ImageNet-1K.

5.5 Robustesse de DNN à des sources de bruit tierces

Les variantes de SAMSON et les méthodes d'entraînement conscientes de la régularité proposant une performance améliorée de DNN en présence d'un bruit matériel simulé ou tel que mesuré sur matériel, nous cherchons maintenant à savoir si ce gain de robustesse dépasse le simple cadre de la robustesse matérielle que nous avons visé jusqu'à présent dans ce

chapitre. Les expériences menées dans cette section considèrent donc que le matériel utilisé est fiable, et réutilisent les hyperparamètres ayant donné la meilleure performance en environnement fiable de la section 5.3. En particulier, nous cherchons à évaluer la robustesse de DNN entraîné avec ces méthodes à des sources de bruit tierces : le bruit de quantification des paramètres d'un DNN, et à des données d'entrées en dehors de la distribution d'entraînement (*Out-of-distribution*, OOD).

5.5.1 Robustesse à un bruit de quantification

Comme abordé dans la revue de littérature, la quantification d'un DNN est une étape importante de sa préparation, permettant d'optimiser la vitesse et l'énergie consommée lors de la phase d'inférence. Traditionnellement, un DNN subit une phase d'ajustement avec une simulation de la quantification matérielle pour s'assurer d'une performance correcte une fois déployé, comme lors d'un entraînement *hardware-aware* présenté section 3.2.3. Ici, nous cherchons à évaluer un DNN entraîné avec les méthodes conscientes de la régularité et SAMSON en le soumettant à une quantification linéaire pour différents niveaux de bits alloués lors de la phase d'évaluation du DNN. La première couche du DNN est exclue de cette quantification. Pour ce faire, nous utilisons une architecture MobileNetV2 dont les paramètres sont quantifiés dans l'intervalle 4-8 bits pour le jeu de données CIFAR-100 et une architecture ResNet-18 dont les paramètres sont quantifiés dans l'intervalle 5-8 bits pour le jeu de données ImageNet-1K. L'expérience est répétée pour trois DNN initialisés aléatoirement et entraînés avec les méthodes conscientes de la régularité et la performance moyenne est présentée figure 5.6. On observe une robustesse à la quantification similaire pour l'ensemble des méthodes jusqu'à un seuil de 5 bits pour l'architecture MobileNetV2 et 6 bits pour l'architecture ResNet-18. Passé ce seuil, la perte de performance est moins marquée pour SAMSON_∞ qui conserve une performance supérieure au reste des méthodes. Dans l'ensemble, les méthodes d'entraînement conscientes de la régularité ont une perte de performance moins marquée que pour SGD.

5.5.2 Robustesse à des données d'entrée hors distribution d'entraînement

Lors de l'entraînement d'un DNN, les données d'entraînement suivent une certaine distribution. Lors de la phase d'utilisation du DNN, des données d'entrée OOD peuvent naturellement être présentées au DNN : c'est un cas de figure critique, car les prédictions obtenues par le DNN sur ces données OOD sont peu fiables étant donné que ce dernier n'a pas été préparé pour leur traitement. Pour évaluer la robustesse d'un DNN à des données OOD, il est possible d'appliquer des déformations sur le jeu de donnée de test [185, 186] : rotations, zoom et cisaillement sous divers paramètres. Nous appliquons ces transformations aux jeux de données

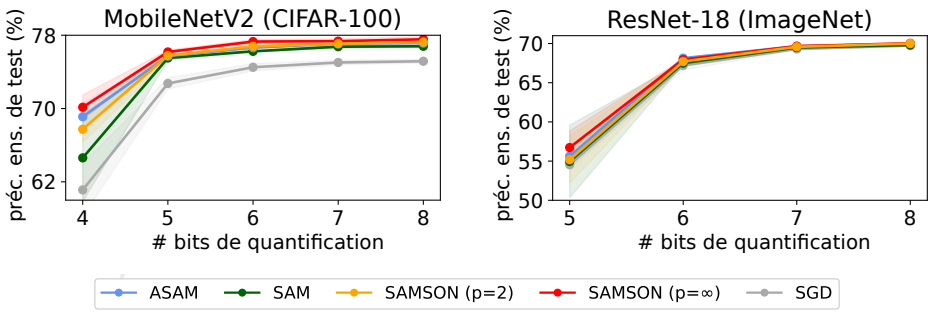


FIGURE 5.6 Performance des différentes méthodes conscientes de la régularité en soumettant les paramètres du DNN à une quantification linéaire pour différents choix de bits pour la quantification sans ajustement *hardware-aware* pour l'architecture MobileNetV2 sur le jeu de données CIFAR-100 et l'architecture ResNet-18 sur le jeu de données ImageNet-1K. Nous illustrons la moyenne pour 3 entraînements de DNN distincts.

CIFAR-10 et CIFAR-100 pour les DNN obtenus section 5.3 sans ajustement, ce qui résulte en 40 scénarios par jeu de données. Les résultats sont présentés synthétiquement à la figure 5.7 en comptant le nombre d'occurrences où une méthode d'entraînement atteint la meilleure performance sur trois répétitions de l'ensemble de test. On y observe une excellente performance de SAMSON, notamment de SAMSON_2 dont la performance se démarque sur le jeu de données CIFAR-100. Les performances sur l'ensemble de test pour chaque scénario OOD sont présentées en détails Annexe B pour le jeu de données CIFAR-10 dans les tables B.1 à B.5 et pour le jeu de données CIFAR-100 dans les tables B.6 à B.10.

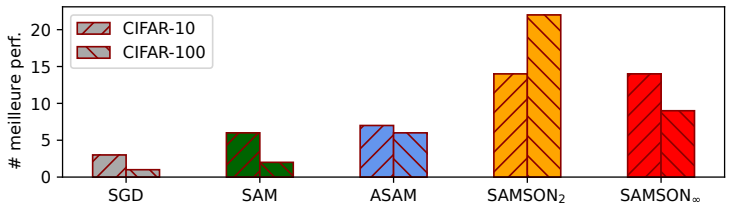


FIGURE 5.7 Robustesse à des données d'entrée OOD sur 40 scénarios par jeu de données en comptant le nombre de meilleures performances sur l'ensemble de test parmi l'ensemble des méthodes. Les méthodes SAMSON observent une performance supérieure dans une majorité de scénarios.

5.6 Synthèse du chapitre

Dans ce chapitre, nous avons proposé d'analyser le lien entre la robustesse de divers DNN opérant sur un matériel de calcul en mémoire de type memristor et les méthodes d'entraînement consciente de la régularité de la fonction de perte. Nous avons proposé SAMSON, une variante des méthodes existantes proposant un calcul de la régularité conditionné non seulement par la magnitude des paramètres du DNN, comme ASAM, mais également sur le domaine des paramètres d'une couche donnée du DNN.

Nous avons testé ces méthodes et SAMSON sur une variété d'architectures de DNN, jeux de données, règles de descente de gradient et niveaux de bruit matériel pour démontrer l'intérêt d'appliquer une méthode d'entraînement consciente de la régularité de la fonction de perte pour améliorer la robustesse d'un DNN dans un environnement bruité sans pour autant sacrifier sa performance dans un environnement fiable. Ces observations s'appliquent à un bruit théorique générique, mais également à des simulations réalistes du bruit matériel. En particulier, la méthode SAMSON proposée produit une amélioration de la performance dans la plupart de ces scénarios en comparaison avec les méthodes SAM [142] et ASAM [150]. Ces méthodes conscientes de la régularité de la fonction de perte améliorent significativement la robustesse d'un DNN en comparaison avec la méthode de référence d'entraînement simulant les aléas matériels, tout en bonifiant la performance en environnement fiable. Enfin, l'utilisation de la troncature basée sur la magnitude des paramètres du DNN tel que suggéré par [136] permet de bonifier à nouveau la robustesse d'un DNN tout en obtenant des résultats plus intéressants avec SAMSON qu'avec le reste des méthodes étudiées. Dans ces expériences, nous avons pu observer une performance similaire à un DNN de référence opérant sur un matériel fiable pour un DNN entraîné avec les méthodes conscientes de la régularité opérant sur un crossbar de memristor avec un rapport signal sur bruit compris entre 14 et 25dB, en fonction du cas de figure.

Or, les méthodes d'entraînement conscientes de la régularité, dont SAMSON, souffrent d'un alourdissement de la phase d'entraînement du DNN en requérant deux rétropropagation du gradient pour une mise à jour des paramètres. Cette particularité peut les rendre moins attractives dans un contexte de puissance de calcul limitée. Néanmoins, cette problématique n'est pas sans solution : des alternatives de SAM efficientes [144, 146] permettent d'atténuer ce surplus de calculs.

Dans le prochain chapitre, nous nous intéressons à la recherche d'architectures de DNN tolérant une plus grande proportion de bruit provenant des memristors en optimisant conjointement le paramètre d'échelle matérielle g_u dans le but de minimiser l'énergie requise par

le DNN tout en maximisant la performance de ce dernier. Les réseaux ainsi obtenus seront propices à un ajustement avec SAMSON pour obtenir des DNN dont les paramètres et l'architecture seront performants en utilisant des crossbars de memristors configurés pour ce même DNN.

CHAPITRE 6 CO-OPTIMIZATION DES PARAMÈTRES MATÉRIELS ET DE L'ARCHITECTURE DU DNN

6.1 Introduction

Tel que décrit à la section 3.3.2, le calcul sur memristor implique une mise à l'échelle des paramètres d'un DNN afin de représenter les paramètres obtenus à l'issue de l'entraînement de ce dernier dans une plage de conductance admise par la technologie de memristor utilisée. Or, il est également possible d'utiliser une plage de conductance utile réduite : ce faisant, la puissance allouée aux calculs est réduite, et donc la consommation énergétique du système.

Néanmoins, la réduction de la plage de conductance augmente l'impact du bruit matériel sur les calculs effectués. Nous cherchons donc à trouver un compromis entre le niveau de bruit pour conserver la performance du DNN et la puissance allouée aux memristors pour réduire les besoins énergétiques. Dans cette optique, nous souhaitons trouver une valeur de $g_u \leq g_{\max}$ satisfaisant les besoins utilisateurs.

Cette approche est abordée dans [159], avec une expérience menée pour optimiser la valeur de g_u en utilisant un estimateur basé sur les moments comme présenté section 3.3.2. Notamment, [159] démontre expérimentalement que l'implémentation de convolution UR présenté section 3.3.2 permet de minimiser l'impact du bruit matériel sur les calculs du DNN ; mais également qu'une assignation de g_u par couche du DNN permet d'obtenir un compromis énergie-performance intéressant. Cependant, cette expérience est limitée au jeu de données CIFAR-10 [130] sur une architecture VGG [161] de taille réduite, dans un processus lourd en calculs.

Or, le choix de cette architecture en particulier nous conduit à la question suivante : quelle est la relation entre l'architecture d'un DNN et les paramètres d'échelle g_u du matériel cible ? En particulier, est-il souhaitable d'optimiser conjointement l'architecture d'un DNN et les paramètres d'échelle des memristors du matériel cible ? Le cas échéant, nous obtiendrions des DNN dont l'architecture même est robuste au bruit matériel tout en configurant simultanément les paramètres matériel. Ce faisant, le processus d'optimisation dédié à une architecture tel qu'appliqué par [159] serait évité. On remarque que l'optimisation conjointe de ces deux dimensions, l'architecture du DNN et les paramètres matériels des memristors, dans l'objectif d'améliorer le compromis énergie-performance atteint est une approche absente de la littérature, notamment des travaux les plus proches [125]. Cela se reflète dans la littérature section 2.3.3.

Nous étudions cette question en utilisant les outils de la recherche d'architecture neuronale (NAS) sur une tâche plus complexe pour améliorer la généralité de notre étude : ImageNet-1K [32]. Comme observé dans la revue de littérature section 2.3.3, le NAS vise à optimiser les architectures de DNN pour améliorer des métriques variées : par exemple, la performance brute du DNN sur une tâche d'intérêt, ou encore la latence et la consommation d'énergie sur matériel cible pour une variété de plateformes matérielles.

Dans ce chapitre, nous nous concentrons sur l'approche de NAS par *super-réseau*. Cela consiste à pré-entraîner un super-réseau duquel des sous-configurations prêtes à l'emploi peuvent être échantillonnées à la volée, modifiant ainsi la profondeur, la largeur et autres paramètres architecturaux du DNN. C'est une approche peu gourmande en calcul pour la phase d'entraînement du DNN car pour obtenir l'ensemble des configurations possibles un seul entraînement est nécessaire, et ce, sans ajustement (*finetuning*) des paramètres du DNN, hormis la mise à jour des moyennes et écart-type glissant des couches de normalisation. Le DNN ainsi échantillonné n'est donc pas préparé pour le bruit matériel : ceci constitue un axe de recherche distinct. On note cependant que la génération des sous-configurations emploie un procédé d'élagage structuré, ce qui renforce la nature généraliste du compromis énergie-performance utilisé. Ainsi, ce chapitre utilise le super-réseau ResNet50 Once-for-all [121] pré-entraîné sur le jeu de données ImageNet-1K [1] sans préparation au bruit matériel des memristors.

Dans un second temps, il est possible d'appliquer une métaheuristique tel qu'un algorithme génétique afin d'optimiser la configuration selon des métriques d'intérêt : dans notre cas, il s'agit de la puissance allouée aux memristors et de la précision atteinte sur la tâche de classification. L'optimisation métaheuristique requiert l'évaluation de ces métriques pour chaque configuration échantillonnée. Cela représente une charge de calcul conséquente, notamment pour le jeu de données ImageNet-1K. Dans une certaine mesure, ce problème est atténué en entraînant un estimateur rapide des métriques basé sur une population restreinte d'échantillons. Lors de l'optimisation, cet estimateur se substitue à la coûteuse évaluation des métriques.

Pour évaluer la puissance et la précision d'un DNN sur memristor, nous nous basons sur une simulation de Monte-Carlo de ces métriques tel que présenté section 3.3.2, qui permet d'obtenir une accélération importante par rapport aux travaux de [159] utilisant un estimateur basé sur les moments, mais pourtant encore trop lent pour nous permettre d'étudier le jeu de données ImageNet-1K au complet. Nous décrirons à la section 6.3 une procédure de Monte-Carlo avec un nombre d'échantillons optimisé afin de nous assurer de la cohérence des

1. Disponible à l'adresse suivante : `https://github.com/mit-han-lab/once-for-all`

résultats tout en réduisant encore la charge de calculs.

Le chapitre actuel est structuré de la manière suivante : dans un premier temps, la méthode de NAS est détaillée section 6.2. Dans cette section sont abordés la fonction objectif et l'espace d'état sur lequel l'optimisation est réalisée ; les différentes méthodes d'optimisation appliquées, dont le cas de référence auquel nous nous comparons ; et enfin les opérations spécifiques à l'algorithme génétique utilisé. Dans un deuxième temps, le processus de NAS est présenté section 6.3 avec une attention particulière sur la procédure de Monte-Carlo optimisée mentionnée au paragraphe précédent. Enfin, les résultats expérimentaux appliquant cette optimisation sont présentés section 6.4, avant de conclure section 6.5.

6.2 Méthode du NAS

6.2.1 Objectif et espace d'états

Ce travail vise à optimiser le front de Pareto puissance-précision d'un DNN déployé sur memristors en optimisant les paramètres d'échelle $\mathbf{g}_u$ et l'architecture du DNN m. Notre tâche est la classification d'image sur le jeu de données ImageNet-1K [32] avec le réseau pré-entraîné Once-for-all (OFA) [121] ResNet50. Sur cette tâche, la performance du DNN est mesurée par la précision A qu'il faut maximiser.

Selon les résultats présentés dans [159], nous utilisons les convolutions UR et un g_u par couche du DNN. Notre espace d'états est divisé en deux composants : premièrement, l'architecture usuelle du DNN m et deuxièmement les paramètres d'échelle $\mathbf{g}_u$ du memristor servant à stocker les paramètres du DNN organisé en vecteur, avec un paramètre d'échelle pour chaque couche du super-réseau. Pour la première partie m, le OFA-ResNet50 définit la profondeur D, largeur et le rapport d'expansion des blocs constituant le DNN, ainsi que la taille des images présentées en entrée.

D'autre part, nous avons les paramètres d'échelle $\mathbf{g}_u$. En ciblant une technologie matérielle spécifique, les paramètres $g_{\min}$, $g_{\max}$ et ϵ^v sont connus. Ici, nous n'avons pas de technologie spécifique en main : nous utilisons les valeurs $g_{\min} = 0$ et $\epsilon^v = 0{,}001$ par simplicité. Ainsi, la tolérance au bruit des crossbars peut être contrôlée en ajustant les paramètres $\mathbf{g}_u$.

Par ailleurs, nous définissons une configuration par défaut où l'impact du bruit matériel sur la performance du DNN est négligeable. Pour la valeur de ϵ^v définie ci-dessus, cette configuration par défaut est obtenue pour chaque couche programmée sur un crossbar en prenant $g_u \triangleq g_u^d = \|\boldsymbol{\mathcal{W}}\|_\infty$, avec $\|\boldsymbol{\mathcal{W}}\|_\infty$ la valeur maximale du tenseur de paramètres $\boldsymbol{\mathcal{W}}$ de

la couche considérée. En outre, ces dispositions simplifient l'équation 3.33 tel que

$$g = |w| \,. \tag{6.1}$$

Nous devons maintenant circonscrire le domaine de g_u pour simplifier et guider l'optimisation. Il est préférable d'envisager un domaine suffisamment grand pour ne pas manquer une configuration intéressante. Par construction, ce domaine est dans le voisinage de notre initialisation par défaut. En outre, en l'absence de connaissance a priori sur le domaine des paramètres du DNN, notamment leur valeur maximale $\|\boldsymbol{\mathcal{W}}\|_\infty$, il est préférable de considérer un domaine de g_u en dessous de la valeur par défaut, pour viser à réduire la puissance du DNN, et au-dessus de la valeur par défaut, pour les couches du DNN ayant un faible domaine pour lequel le ϵ^v supposé petit demeurerait nocif à la performance du DNN. En se basant sur des tests préliminaires, un domaine suffisant serait

$$g_u \in [0{,}5g_u^d,\ 1{,}5g_u^d] \,, \tag{6.2}$$

seuil en dessous duquel la précision du DNN se dégrade rapidement, et limite supérieure à laquelle on n'observe aucune amélioration de la précision du DNN.

Chaque élément de la configuration architecturale du DNN m et $\mathbf{g}_u$ est représenté par un vecteur de taille fixe, et tous sont concaténés pour former un vecteur d'état $s \in \Omega$. Les éléments de m sont représentés identiquement à [121]. Nous représentons par Ω l'ensemble des configurations possibles de ce vecteur d'état. Ce vecteur d'état s est la représentation sur laquelle la métaheuristique est appliquée. Nous sommes intéressés par deux objectifs concurrents : maximiser la précision et minimiser la puissance dissipée par le DNN. Cela peut nous mener à considérer une formulation de problème multi-objectif, ou encore simple objectif, en transformant le deuxième objectif en une contrainte d'optimisation. Nous retenons la formulation multi-objectif, qui a l'avantage de générer plusieurs configurations Pareto-optimales en une seule passe d'optimisation. Nous considérons donc le problème de minimiser la fonction multi-objectif $\mathcal{F}$ couvrant la précision A et la puissance P des DNN représentés par le vecteur d'état s tel que

$$\min_{s\in\Omega} \mathcal{F}(s) = (-A(s), P(s)) \,. \tag{6.3}$$

Nous résolvons ce problème en utilisant l'algorithme génétique d'optimisation multi-objectif NGAS-II [187, 188].

6.2.2 Méthodes d'optimisation

Nous considérons trois méthodes d'optimisation notées M1, M2 et M3 que nous allons maintenant décrire.

Nous effectuons une optimisation M1 comme point de référence, afin de faciliter la comparaison aux approches de NAS tierces n'incluant pas les paramètres d'échelle des memristors tel que [125]. Dans ce scénario M1, seules les variables d'architecture du DNN m sont optimisées, tandis que les paramètres d'échelles des memristors $\mathbf{g}_u$ sont initialisés à la valeur par défaut g_u^d.

En parallèle, nous réalisons une seconde optimisation M2 où m et $\mathbf{g}_u$ sont tous deux optimisés.

Enfin, pour chaque configuration d'architecture de DNN obtenue par l'optimisation M1, nous générons dix configurations dérivées en multipliant le vecteur $\mathbf{g}_u$ avec un ensemble de coefficients pris dans l'intervalle $[0.9, 1.1]$. Ces configurations dérivées sont par la suite appelées configurations mises à l'échelle et constituent l'optimisation M3. Les configurations obtenues par M3 ont donc un effet contrôlé sur la puissance et un effet inconnu sur la précision du DNN. Comme nous le verrons dans la section 6.4, les configurations de M3 produisent un front de Pareto similaire aux configurations obtenues par l'optimisation M2, incluant les paramètres de mise à l'échelle $\mathbf{g}_u$ comme variable d'optimisation. Le procédé suivi par M3 est coûteux en temps de calculs, chaque solution ainsi générée est évaluée pour ensuite n'en retenir que le front de Pareto, mais est une solution d'optimisation alternative basée sur une recherche procédurale simple.

6.2.3 Algorithme génétique

L'algorithme génétique NGAS-II est réalisé en utilisant la librairie Pymoo [188] et opère un ensemble d'opération pour optimiser le vecteur d'état s. Pour ce faire, l'algorithme fait évoluer une population d'individus, chacun représentant un vecteur d'état, au travers des mécanismes inspirés de la sélection naturelle. Nous considérons les trois opérations suivantes :

- échantillonnage de vecteur d'états aléatoires $s \in \Omega$;
- mutation d'un état s_α à un état voisin $s_{\alpha'}$;
- et croisement s_c entre deux états s_1 et s_2.

Ces opérations sont directement proposées par la librairie utilisée, et nous utilisons les paramètres par défaut suggérés. Pour un nombre donné de générations d'évolution, ces opérations sont répétées afin d'améliorer la population. L'estimateur de la précision du DNN et de sa puissance selon le vecteur d'état est utilisée pour pondérer la probabilité à ce qu'un individu et les composantes de son vecteur d'état demeure dans la population. Ainsi, un vecteur

d'état atteignant un meilleur compromis puissance-précision comparativement au reste de la population aura une plus grande probabilité de survivre jusqu'à la prochaine génération et d'effectuer des croisements avec d'autres individus.

Il y a cependant une dépendance entre l'architecture du DNN et le paramètre d'échelle des memristors, tous deux encapsulés dans le vecteur d'état. En l'occurrence, si le paramètre de profondeur de l'architecture du DNN D est changé, certains éléments du vecteur de paramètres d'échelle $\mathbf{g}_u$ peuvent devenir caducs ou nécessiter une initialisation. Cette caractéristique suppose de prendre des précautions lors des opérations d'échantillonnage, de mutation et de croisement de l'algorithme génétique. Nous utilisons des opérations de *réparation* pour pallier ce problème, que nous allons maintenant présenter.

Réparations de l'échantillonnage et des mutations Cette procédure de réparation prend en entrée le vecteur de profondeur des blocs du DNN D et le vecteur des paramètres d'échelle des memristors $\mathbf{g}_u$ et produit un vecteur $\mathbf{g}_u$ *réparé*. Nous définissons la fonction $\texttt{mask}(D)$ retournant un vecteur booléen de la même taille que le vecteur $\mathbf{g}_u$, c'est-à-dire le nombre de couches dans le super-réseau, spécifiant pour chaque couche si le sous-modèle partiellement décrit par D la comporte (valeur 1) ou non (valeur 0). La procédure de réparation nous assure de la validité des valeurs de $\mathbf{g}_u$: quand le masque est de valeur 1, un g_u respectant les contraintes de l'équation 6.2 doit être présent, et quand le masque est de valeur 0, nous devons mettre la valeur $g_u = 0$. En outre, quand le masque est de valeur 1, nous devons prendre en compte s'il vient de prendre cette valeur par mutation (détectable si un $g_u = 0$ est présent), ou s'il était déjà de valeur 1 avant la mutation (détectable si un $g_u > 0$ est présent). Pour respecter ces conditions, une opération de réparation $g_u \leftarrow R(g_u, \texttt{mask}(D))$ est effectuée, avec $R(\cdot)$ définie comme suit :

$$R(g_u, \texttt{mask}(D)) = \begin{cases} g_u^d & \text{si } \texttt{mask}(D) = 1 \text{ et } g_u = 0 \\ g_u & \text{si } \texttt{mask}(D) = 1 \text{ et } g_u > 0 \\ 0 & \text{sinon.} \end{cases} \tag{6.4}$$

Réparation des croisements Nous avons maintenant le cas où deux vecteurs d'états s_1 et s_2 sont mélangés aléatoirement dans une opération de croisement pour aboutir à un nouveau vecteur d'état s_c. Chacun dispose d'un masque distinct $\texttt{mask}(D_1)$ et $\texttt{mask}(D_2)$, ainsi que d'un vecteur de paramètre d'échelle distinct $\mathbf{g}_{u1}$ et $\mathbf{g}_{u2}$. Le croisement construit un nouveau vecteur de profondeur des blocs du DNN D_c à partir de D_1 et D_2 en sélectionnant

aléatoirement chaque élément i du vecteur tel que

$$D_c[i] \begin{cases} D_1[i] & \text{avec probabilité } \frac{1}{2}\,, \\ D_2[i] & \text{avec probabilité } \frac{1}{2}\,. \end{cases} \tag{6.5}$$

Avec D_c en main, nous calculons également $\texttt{mask}(D_c)$. Pour le croisement des paramètres d'échelle $\mathbf{g}_{u1}$ et $\mathbf{g}_{u2}$ afin d'obtenir $\mathbf{g}_{uc}$, il est nécessaire de prendre en compte les possibles valeurs nulles de $\mathbf{g}_u$ dans les parents. Les résultats sont présentés dans la table de vérité suivante, où k est échantillonné aléatoirement uniformément dans l'ensemble $\{0, 1\}$ et $\mathbf{m}_x = \texttt{mask}(D_x), x \in \{1, 2, c\}$:

k	$\mathbf{m}_1$	$\mathbf{m}_2$	$\mathbf{m}_c$	$\boldsymbol{g_{uc}}$
X	0	0	X	0
X	0	1	0	0
X	0	1	1	$\boldsymbol{g_{u2}}$
X	1	0	0	0
X	1	0	1	$\boldsymbol{g_{u1}}$
0	1	1	X	$\boldsymbol{g_{u1}}$
1	1	1	X	$\boldsymbol{g_{u2}}$

En d'autres termes, si $\mathbf{m}_1 \cdot \mathbf{m}_2 = 1$ alors $\mathbf{g}_{uc}$ est pris aléatoirement dans les deux parents, sinon si $\mathbf{m}_c = 1$ alors $\mathbf{g}_{uc}$ prend la valeur du seul $\mathbf{g}_u$ des parents qui n'est pas zéro, sinon $\mathbf{g}_{uc}$ est zéro.

Avec ces opérations élémentaires d'algorithme génétique, nous présentons maintenant le processus de NAS optimisé que nous allons employer.

6.3 Processus de NAS

Notre processus de NAS est basé sur l'usage d'un estimateur entraîné à simuler la précision et la puissance dissipée par les architectures de DNN échantillonnées du super-réseau. Afin d'entraîner cet estimateur, nous récoltons un jeu de données associant des vecteurs de configuration d'architectures échantillonnées aléatoirement à des estimations de précision et de puissance dissipée avec le modèle de panne matériel tel que décrit section 3.3.2. Cet estimateur est enfin utilisé pour réaliser rapidement l'algorithme génétique tel que décrit Section 6.2. Étant donné que l'entraînement de l'estimateur est la tâche la plus exigeante en calculs, nous procédons à certaines optimisations pour réduire cette charge, qui sont maintenant décrites.

6.3.1 Entraînement de l'estimateur

Notre estimateur est un réseau de neurones entraîné à minimiser l'erreur quadratique moyenne entre ses prédictions de puissance et de précision et la cible pour une configuration s donnée. Pour ce faire, nous récoltons un jeu de données contenant des estimations de puissance et de précision pour K vecteurs d'états sur la tâche d'intérêt. Ces estimations seront ensuite les cibles utilisées pour entraîner l'estimateur. En utilisant l'approche du super-réseau proposée par OFA [121], le temps de calcul alloué à l'entraînement des DNN chute de manière drastique de $\mathcal{O}(K)$ à $\mathcal{O}(1)$ et offre donc une accélération conséquente. Dans ce travail, nous ne considérons pas d'ajustement des paramètres du DNN échantillonnés depuis le super-réseau au bruit matériel des memristors. Néanmoins, évaluer la précision et la puissance dissipée d'un DNN échantillonné depuis le super-réseau OFA-ResNet50 sur la tâche ImageNet1K demeure une tâche lourde, et nécessitant d'être répétée pour chaque vecteur d'états échantillonné.

Afin de réduire le temps d'exécution de cette étape, l'évaluation de la précision peut se faire sur un sous-ensemble de mini-lots de taille n_a pris dans l'ensemble d'évaluation complet de N mini-lots. Ce faisant, un facteur d'accélération N/n_a est observé. Cette méthode nous procure un compromis entre l'erreur d'évaluation de la précision et le temps d'exécution. Or, l'objectif poursuivi est d'ordonner les configurations de DNN selon leur précision atteignable, plutôt que d'obtenir une estimation précise de la précision atteinte par chaque configuration échantillonnée. De manière similaire, la puissance dissipée est calculée sur un sous-ensemble de mini-lots de taille n_p.

Lors de la génération du jeu de données de K vecteurs d'états, il est nécessaire de s'assurer de la diversité des vecteurs d'états. Une technique comme l'échantillonnage par hypercube latin (*Latin hypercube sampling*, LHS) [189] peut être utilisée pour s'en assurer [121, 125]. Concrètement, les K vecteurs d'états sont composés de k architectures de DNN m, chacune avec k_g échantillonnages des paramètres d'échelles $\mathbf{g}_u$, avec la relation $K = k \times k_g$. Les k vecteurs d'états aléatoires sont initialisés avec des architectures de DNN m échantillonnées aléatoirement, et pour chacun de ces k vecteurs aléatoires, on procède à l'échantillonnage de k_g vecteurs de paramètres d'échelle $\mathbf{g}_u$ tel que :

$$\mathbf{g}_{ui} = \mathbf{g}_u^d \circ \texttt{LHS}_i \, \forall i \in 1, ..., k_g, \tag{6.6}$$

avec $\texttt{LHS}_i$ un échantillonage par hypercube latin, $\mathbf{g}_u^d$ un vecteur comprenant les g_u^d de chaque couche du super-réseau et $\circ$ le produit matriciel d'Hadamard. En pratique, le domaine des g_u est obtenu en nous basant sur l'équation 6.2.

Nous avons ainsi quatre variables contrôlant la vitesse et la variété du jeu de données généré

(n_a, n_p, k, k_g). Afin d'évaluer le temps de calcul de la procédure de NAS, nous définissons le temps requis pour évaluer la puissance et la précision pour un mini-lot par t_p et t_a respectivement. Ainsi, le calcul de la puissance et la précision pour un mini-lot a une charge de calcul de $T = t_p + t_a$, ce qui nous donne la charge de calcul par vecteur d'état TN. Avec l'usage d'un sous-ensemble de taille réduite pour la précision n_a et la puissance n_p, la charge de calcul par vecteur d'état devient $t_a n_a + t_p n_p$. Sachant la taille du jeu de données récolté $K = kk_g$ et la charge de calcul pour évaluer la précision et la puissance pour un vecteur d'état de $t_a n_a + t_p n_p$, la charge de calcul totale est donc définie par $kk_g(t_a n_a + t_p n_p)$. Nous allons maintenant décrire comment le choix de n_a, n_p, k et k_g peut être fait afin d'optimiser la charge de calcul de la phase d'entraînement de l'estimateur.

6.3.2 Optimisations de n_a et n_p

Nous utilisons la notation $A_k(n)$ pour la précision moyenne d'un DNN k sur n mini-lots. Notre objectif est de distinguer pour une paire de DNN lequel atteint la meilleure précision moyenne, pour une tolérance à l'erreur donnée. Précisément, pour deux DNN j et ℓ, nous voulons trouver le plus petit N tel que

$$\Pr(A_j(n) > A_\ell(n)) > 1 - \epsilon\,, \tag{6.7}$$

sachant que $A_j(N) > A_\ell(N) + \delta$, où $\delta > 0$ est une tolérance sur l'erreur de la précision obtenue pour distinguer clairement la précision des deux DNN, et ϵ est une probabilité d'échec prédéfinie.

Pour déterminer n, nous effectuons une expérience sur 100 architectures de DNN échantillonnées aléatoirement du OFA-ResNet50. La précision moyenne de ces architectures est mesurée sur n mini-lots, pour $n \in \{1, \ldots, N\}$. Ces mesures sont exprimées avec $a_{n,j}$ pour $n \in 1, ..., N$ et $j \in \{1, ..., 100\}$ les indices des architectures de DNN. Pour un n donné, nous pouvons évaluer l'équation 6.7 en utilisant une simulation de Monte-Carlo : on tire aléatoirement deux architectures distinctes j et ℓ satisfaisant $j \neq \ell$ et $A_j(N) > A_\ell(N) + \delta$.

Nous nous intéressons maintenant à résoudre numériquement l'équation 6.7 avec n la variable à résoudre. En pratique, nous calculons cette valeur avec une recherche dichotomique sur les résultats de l'expérience précédente : à chaque étape de la recherche, n est augmenté ou réduit par des valeurs de plus en plus petites, tout en nous assurant de respecter l'équation 6.7. Enfin, nous voulons nous assurer que les résultats de cette recherche dichotomique soient monotones selon la variable δ afin d'avoir un critère de décision cohérent. Ce dernier point est obtenu en utilisant le front de Pareto des résultats de la recherche dichotomique, et en interpolant

linéairement ces points. Les résultats de cette procédure sont rapportés à la figure 6.1. Sur cette base, nous utilisons $n_a = 7$: un point pivot, car le δ y est relativement faible tout en coïncidant avec la région plate de la courbe précédant une augmentation marquée de n.

Cette expérience est répétée pour l'estimation de la puissance sur n_p mini-lots. Le résultat est d'intérêt : la métrique de la puissance observe une variance bien moindre et le rang de deux architectures distinctes peut être estimé de manière fiable dès le premier mini-lot. Ce résultat expérimental est également dépeint figure 6.1 par une ligne constante pour chaque choix de δ. Nous utilisons ainsi $n_p = 1$.

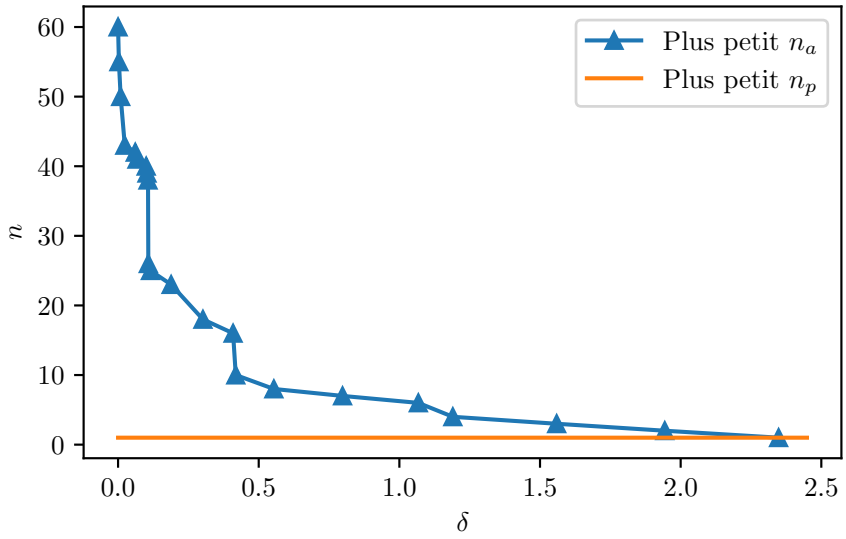


FIGURE 6.1 Plus petit n_a et n_p satisfaisant l'équation 6.7, pour $\epsilon = 0{,}05$

6.3.3 Choix de k et k_g

Enfin, nous choisissons $k = 64\,000$ en nous basant sur les travaux de référence de [121]. Il reste alors à décider de la valeur pour la variable k_g. Selon les gains de temps observés avec les n_a et n_p optimisés, nous utilisons $k_g = 50$ pour obtenir un temps de récolte du jeu de donnée comparable aux travaux de référence.

6.4 Résultats expérimentaux NAS

Avec ces optimisations, nous effectuons le processus de NAS décrit section 6.3.1. Comme attendu, la récolte du jeu de données est une opération lourde : avec 1,86 seconde de temps de calcul GPU en moyenne pour estimer la précision et la puissance d'un DNN échantillonné,

le temps total est de 69 jours de calculs GPU. De plus, nous observons que $t_a = 2t_p$. Nous entraînons ensuite le DNN estimateur de la précision et de la puissance tel que décrit annexe C. Enfin, nous réalisons plusieurs optimisations avec l'algorithme génétique tel que décrit à la section 6.2.2. Selon des résultats préliminaires ayant donné de bons résultats, la taille de population est fixée à 1024 et 500 générations d'optimisation sont réalisées. L'utilisation de l'estimateur nous permet de compléter l'optimisation avec un algorithme génétique en quelques heures de calculs.

Pour chaque optimisation M1, M2 et M3, une simulation complète de la précision et de la puissance dissipée par le DNN est calculée sur le jeu de données de validation de ImageNet-1K pour les 1024 individus obtenus à la 500ème génération et les résultats sont compilés dans la figure 6.2. Nous montrons également le front de Pareto présent dans le jeu de données récolté au début du processus de NAS courbe *« Jeu de données »* : pour chaque point une simulation complète de la précision et de la puissance dissipée est réalisée. On observe pour cette courbe une faible précision en plus d'une forte variance dans les résultats obtenus. Ces deux éléments démontrent les fortes erreurs d'estimation lors de la récolte du jeu de données.

D'autre part, la courbe formant le front de Pareto de l'optimisation M1 atteint une bonne précision dans les cas de forte puissance, comme attendu, mais la perte de précision est marquée dès lors que la puissance allouée diminue, nous menant à des solutions peu intéressantes. Le front de Pareto des configurations mises à l'échelle M3 est illustré avec une courbe pointillée et prodigue un très bon compromis puissance-précision, au niveau du front de Pareto de l'optimisation M2 incluant les paramètres de mise à l'échelle des memristors $\mathbf{g}_u$.

Dans l'ensemble, les configurations obtenues par l'optimisation M2 incluant les paramètres de mise à l'échelle des memristors améliorent le front de Pareto puissance-précision, et les configurations mises à l'échelle M3 s'en approchent au prix d'une recherche plus coûteuse en calculs.

Nous notons que la recherche de configurations mises à l'échelle n'est pas compétitif pour les points issus de l'optimisation M2, car cela génère un front de Pareto similaire avec des calculs supplémentaires. En revanche, les configurations de M1 sont clairement améliorées avec l'approche de mise à l'échelle de M3.

Il est donc d'intérêt d'optimiser les paramètres de mise à l'échelle des memristors $\mathbf{g}_u$ utilisés par les architectures de DNN pour atteindre le meilleur compromis puissance-précision, que ce soit lors d'une optimisation conjointe de ces deux éléments (M2) ou en acceptant un surplus de calculs lors d'une recherche procédurale à l'issue de l'algorithme génétique (M3).

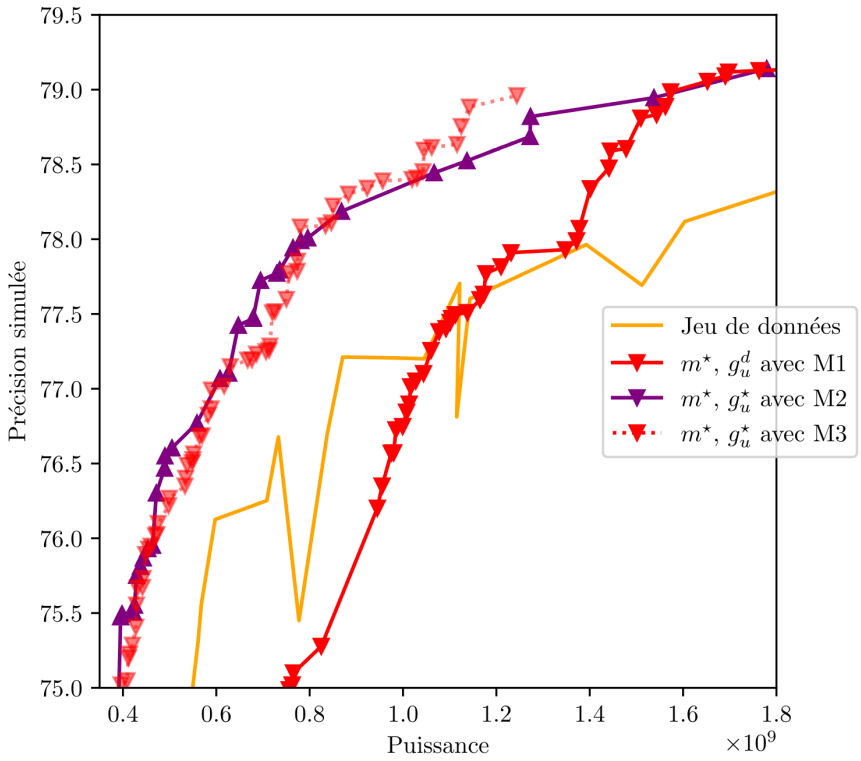


FIGURE 6.2 Front de Pareto puissance-précision sur ImageNet-1K des solutions obtenues avec les algorithmes génétiques M1, M2 et M3. La courbe « Jeu de données » est la simulation complète des meilleurs points présents dans le jeu de données (estimés avec les sous-ensembles de taille n_a et n_p) et est montrée pour permettre de juger de la forte erreur d'estimation autorisée par la procédure de NAS proposée. La courbe pointillée M3 est le front de Pareto des configurations mises à l'échelle de l'optimisation M1.

6.5 Synthèse du chapitre

Dans ce chapitre, nous avons mené une expérience de NAS pour évaluer l'opportunité d'optimiser conjointement les paramètres de mise à l'échelle des memristors et les paramètres architecturaux du DNN. Cette expérience de NAS en organisée en deux étapes : entraînement d'un estimateur et optimisation de l'architecture du DNN. Nous nous sommes intéressés à l'optimisation en termes de temps d'exécution de la phase d'entraînement de l'estimateur en introduisant un compromis sur l'erreur des données récoltées dans une simulation de Monte-Carlo du comportement des memristors. Ces optimisations nous permettent de proposer une charge de calculs similaire aux approches de NAS antérieures étudiant un espace d'état plus simple, c'est-à-dire excluant les paramètres de mise à l'échelle des memristors. Enfin, nous avons démontré que cette approche peut améliorer le front de Pareto puissance-précision sur la tâche de classification ImageNet-1K, notamment dans un domaine de faible puissance.

Comparativement, les approches de NAS antérieures [125] excluent de l'optimisation les paramètres d'échelle des memristors et parviennent à un compromis énergie-performance moins intéressant. Nous retrouvons ces approches antérieures dans les expériences menées avec la stratégie M1.

D'autre part, nous avons obtenu des résultats compétitifs en utilisant une solution procédurale simple consistant à multiplier une valeur par défaut des paramètres de mise à l'échelle des memristors par des coefficients. Cette solution alternative augmente le coût de la recherche du front de Pareto en introduisant une quantité supplémentaire d'architectures de DNN à évaluer. Une étude plus poussée sur ce résultat serait souhaitable afin de mesurer plus précisément les gains en termes de temps d'exécution que cette approche pourrait amener.

CHAPITRE 7 CONCLUSION

Dans cette conclusion, nous ferons une brève synthèse des travaux effectués dans cette thèse. Nous formulerons ensuite les limites identifiées dans les travaux proposés, avant d'ouvrir la discussion aux potentielles améliorations futures.

7.1 Synthèse et discussion des travaux

Cette thèse a eu pour objet l'étude de DNN dont les paramètres sont stockés sur des mémoires de fiabilité limitée tout en proposant une performance énergétique accrue. Dans un premier temps, nous avons ainsi étudié une mémoire SRAM opérant à une tension proche du seuil de fonctionnement, et la technologie émergente de calcul en mémoire avec des memristors dans un second temps. Dans les deux cas, nous avons identifié un paramètre matériel permettant d'instaurer le compromis énergie fiabilité, la tension d'alimentation pour les SRAM et la plage de conductance utile pour les memristors. Nous avons ensuite proposé un entraînement de DNN permettant de le préparer aux bruits matériels, en simulant les pannes pour la SRAM et en exploitant une technique de *sharpness-aware minimization* pour les memristors. Nous avons également proposé l'algorithme SAMSON permettant de bonifier la robustesse obtenue avec ce type d'entraînement. Nous obtenons ainsi des DNN robustes aux bruits matériels dans ces deux cas de figure. Pour l'approche employée pour les memristors par algorithmes *sharpness-aware*, l'utilisation de ces derniers a permis une amélioration importante de la robustesse du DNN dans une variété de scénarios (architectures, jeux de données, algorithmes d'optimisation) en comparaison avec les algorithmes de référence d'entraînement *hardware-aware* [109], ou d'entraînement avec troncature des paramètres du DNN [136]. En particulier nous avons observés dans les expériences menées que l'algorithme SAMSON obtient avec une régularité marquée un résultat plus intéressant en environnement fiable, et retient une meilleure portion de performance en environnement simulant le bruit matériel des memristors en comparaison avec les algorithmes *sharpness-aware* de référence SAM [142] et ASAM [150]. Qui plus est, une bonne complémentarité a été démontrée entre SAMSON et la troncature des paramètres du DNN [136] pour améliorer la robustesse du DNN en environnement fortement bruité. Selon le cas de figure, ces gains permettent de conserver une performance similaire à un DNN opérant sur matériel fiable tout en opérant sur un crossbar de memristor avec un rapport signal sur bruit compris entre 14 et 25dB.

Enfin, nous avons tiré parti de ces réseaux robustes en optimisant les paramètres matériels d'énergie-fiabilité pour réduire la consommation d'énergie des DNN avec l'algorithme LaN-

Max pour un DNN opérant sur SRAM et l'algorithme de recherche d'architecture neuronale pour un DNN opérant sur memristors, optimisant ainsi conjointement la taille du DNN et les paramètres matériels.

Il est impossible d'évaluer chaque solution intermédiaire lors de ces optimisations : pour un DNN sur SRAM, il est nécessaire de procéder à un ajustement des paramètres du DNN selon le taux de pannes présentes dans le matériel ; tandis qu'un algorithme de recherche d'architecture neuronale observe une explosion combinatoire du nombre d'architectures possibles. Ces facteurs nous ont conduit à proposer des algorithmes permettant d'effectuer l'optimisation avec une charge de calcul réduite. Pour les SRAM, LaNMax permet d'ajuster les paramètres matériels lors de l'entraînement initial du DNN ; tandis que le processus de recherche d'architecture neuronale adapté aux memristors disposait d'hyperparamètres que nous avons optimisés pour pouvoir augmenter le nombre de solutions évaluées tout en maintenant un temps de calcul similaire. Ceci eu pour effet d'explorer de manière plus exhaustive les combinaisons possibles d'architectures et de paramètres matériels.

Dans les deux cas, nous avons effectué des expériences pour démontrer les gains de robustesse des DNN lors d'une augmentation du bruit matériel. Pour les SRAM, nous avons observé une réduction de l'énergie consommée dans le meilleur des cas au tiers de la consommation initiale avant que la performance du DNN ne commence à se dégrader. En comparaison, la méthode d'un taux d'erreur uniforme pour tout le DNN [36] obtient une plus faible robustesse aux pannes de SRAM dans les expériences menées. L'utilisation de l'approche par couche nous a ainsi permis de tolérer un plus grand taux de pannes matérielles dans l'ensemble, ce qui se traduit par une consommation d'énergie plus faible avec le modèle matériel utilisé. Pour l'approche de NAS sur memristor, l'optimisation des paramètres matériels proposée permet une amélioration du front de Pareto puissance-précision d'un DNN de 40% dans un régime de faible puissance relativement à une approche excluant ces paramètres [125].

Dans l'ensemble, nous avons proposé des méthodes permettant de préparer les DNN à une plateforme matérielle de fiabilité limitée, mais également de configurer rapidement le niveau de fiabilité de ces plateformes. Ces méthodes sont proposées comme un outil possible pour l'utilisateur souhaitant réduire la consommation d'énergie d'un DNN. Cet outil est à considérer parmi l'éventail de solutions existantes (quantification, compression, élagage, etc.) comme une nouvelle possibilité combinatoire. Par exemple, nous avons démontré l'intérêt de combiner la quantification et la réduction de la taille de DNN avec l'utilisation d'un matériel numérique de fiabilité limitée au chapitre 4, ou encore d'agir sur l'architecture même de DNN utilisée pour un calcul sur memristor chapitre 6. Le chapitre 5 se veut comme une approche généraliste pour augmenter la robustesse d'un DNN et serait propice à l'application de toutes

autres méthodes permettant d'améliorer l'efficacité énergétique d'un DNN. Ainsi, en composant les gains des approches proposées et des approches existantes, on pourrait espérer réduire l'empreinte énergétique d'un DNN.

7.2 Limitations de la solution proposée

Les méthodes et les algorithmes proposés dans cette thèse ont pour objectif d'être génériques, afin de pouvoir être rapidement appliqués à une diversité de DNN et tâches d'intérêts. Les modèles de fiabilité-énergie sont également facilement remplaçables selon les évolutions technologiques et les besoins de qualification des DNN obtenus. Il est cependant à noter qu'une plus grande précision dans les modèles d'erreur et leur simulation peut rendre difficile l'évaluation des solutions intermédiaires en l'absence de ressources de calculs adéquates. Ainsi, lors des expériences menées dans cette thèse, nous n'avons pu étudier qu'un sous-ensemble des architectures de DNN populaires et nous sommes concentrés sur les tâches de classification d'images en privilégiant des modèles de fiabilité-énergie de complexité maîtrisée.

Ce besoin de ressources de calcul est d'autant plus aiguë que l'entraînement d'un DNN doit être adapté pour préparer le réseau à un modèle de pannes matérielles réalistes pour réaliser un entraînement *hardware-aware*. Le surplus de calculs est conséquent : quantification du réseau à la volée, échantillonnage du modèle de pannes et perturbation du réseau. Si ces opérations sont inclues dans les graphes de calcul du DNN comme composante à part entière de la « profondeur » du réseau, on assiste à une multiplication d'un facteur 4 de la profondeur observée. En d'autres termes, le léger ResNet18 devient le ResNet72-perturbé lors de l'entraînement du DNN sans pour autant atteindre la précision sur matériel fiable d'un ResNet72. Ce constat s'applique notamment aux travaux du chapitre 4 et motive les travaux du chapitre 5.

Dans une moindre mesure, cette observation s'applique également à la phase d'évaluation du DNN. Cette phase d'évaluation peut croître en complexité pour certaines tâches spécifiques : nous l'avons notamment observé chapitre 6 pour la récolte d'un jeu de données afin de conduire un algorithme NAS. Pour les memristors, nous disposons d'un outil de calcul de l'erreur théorique [159], mais en pratique une simulation de Monte-Carlo demeure plus intéressante du point de vue de la charge de calculs en relaxant les critères de précision sur l'erreur obtenue.

Ainsi, à l'aune de l'augmentation de la taille des DNN, de la complexité des tâches et des jeux de données, les surplus de calculs des approches proposées peuvent limiter la portée des travaux de cette thèse. Ce phénomène se heurte également aux limitations du matériel

contemporain : les memristors étudiés aux chapitres 5 et 6 dont les prototypes comportent quelques millions d'emplacements de stockage. Ceci contraindrait très fortement les DNN utilisés. Les utilisateurs seraient alors dans l'incapacité d'utiliser les architectures de DNN les plus populaires.

7.3 Améliorations futures

Afin de répondre à ces limitations, nous proposons deux axes pour améliorer les contributions de cette thèse.

Un premier axe porte sur l'efficience de la préparation de DNN robustes. Cet axe comprend une analyse de l'effet des apprentissages effectués aux chapitres 4 et 5. L'objectif serait de proposer une méthode d'ajustement des paramètres d'un DNN pour répliquer les effets d'un entraînement complet avec LaNMax ou SAMSON. Dans le meilleur des cas, cet ajustement s'effectuerait pour chaque couche du DNN prise indépendamment des autres. Par exemple, cela pourrait s'effectuer en cachant les sorties intermédiaires du DNN opérant en environnement fiable, puis en considérant chaque couche comme un DNN à part entière opérant sur un matériel de fiabilité limitée devant répliquer la sortie attendue et d'y appliquer les algorithmes de robustesse. Un écueil à éviter est de tomber dans un minimum local du DNN prit au complet, qu'une co-adaptation des couches aurait évité.

D'autre part, cet axe peut s'enrichir en étudiant les super-réseaux utilisés au chapitre 6. Ces super-réseaux sont utilisés pour générer une grande quantité de sous-réseaux lors d'une démarche NAS. Ainsi, il serait souhaitable d'étudier la robustesse de ces sous-réseaux et de potentielles méthodes pour la bonifier. Le front de Pareto obtenu au chapitre 6 en serait alors amélioré.

Un second axe s'intéresse à la capacité de déploiement des DNN sur des environnements fortement limités en capacité mémoire, comme c'est le cas par exemple pour les memristors. Ainsi, en s'inspirant d'une méthode de partage des paramètres telle que ThriftyNet [76], l'architecture d'un DNN voué à être déployé sur crossbar ou autre dispositif mémoire pourrait être conçue en amont pour refléter les caractéristiques matérielles. L'emphase serait mise sur la réutilisation des dispositifs mémoires, de sorte à en limiter l'utilisation. De manière orthogonale, un effort sur la populaire architecture Transformers [1] serait de s'assurer que les lourdes opérations matricielles puissent être effectuées sur memristors. En effet, le mécanisme d'attention [181] sollicite deux opérations de ce type avec deux matrices dynamiques et donc non programmables en avance sur le crossbar. On remarque également que les interactions

1. Ou autres DNN à la mode quand vous lirez ces lignes.

du crossbar avec le système numérique l'accueillant pourront mener à un compromis énergie-fiabilité mixte analogique/numérique, pour lequel une solution d'optimisation des paramètres matériels et de robustesse serait à proposer.

Finalement, ces deux axes participeront d'une part à faciliter l'obtention de DNN robustes, et d'autre part à s'assurer que les dispositifs mémoires permettant un compromis énergie-fiabilité puissent être utilisés pour les réseaux à l'état de l'art.

RÉFÉRENCES

[1] V. Sze, Y.-H. Chen, T.-J. Yang *et al.*, *Efficient Processing of Deep Neural Networks*, ser. Synthesis Lectures on Computer Architecture. Cham : Springer International Publishing, 2020.

[2] A. Gholami, Z. Yao, S. Kim *et al.*, "AI and memory wall," 2021.

[3] "Portrait des ressources énergétiques d'hydro-québec," nov. 2019. [En ligne]. Disponible : https://www.hydroquebec.com/data/achats-electricite-quebec/pdf/portrait-ressources-energetiques.pdf

[4] O. Mutlu, S. Ghose, J. Gómez-Luna *et al.*, "Processing data where it makes sense : Enabling in-memory computation," *Microprocessors and Microsystems*, vol. 67, p. 28–41, juin 2019.

[5] A. Pedram, S. Richardson, M. Horowitz *et al.*, "Dark memory and accelerator-rich system optimization in the dark silicon era," *IEEE Design & Test*, vol. 34, nº. 2, p. 39–50, 2017.

[6] S. Kim, P. Howe, T. Moreau *et al.*, "Energy-efficient neural network acceleration in the presence of bit-level memory errors," *IEEE Transactions on Circuits and Systems I : Regular Papers*, vol. 65, nº. 12, p. 4285–4298, 2018.

[7] M. Horowitz, "1.1 Computing's energy problem (and what we can do about it)," dans *IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, 2014, p. 10–14.

[8] R. G. Dreslinski, M. Wieckowski, D. Blaauw *et al.*, "Near-threshold computing : Reclaiming moore's law through energy efficient integrated circuits," *Proceedings of the IEEE*, vol. 98, nº. 2, p. 253–266, 2010.

[9] A. Sebastian, M. Le Gallo, R. Khaddam-Aljameh *et al.*, "Memory devices and applications for in-memory computing," *Nature Nanotechnology*, vol. 15, nº. 7, p. 529–544, juill. 2020.

[10] R. Ross, V. Pillitteri, R. Graubart *et al.*, "Developing cyber-resilient systems : a systems security engineering approach," National Institute of Standards and Technology (U.S.), Gaithersburg, MD, Rapport technique NIST SP 800-160v2r1, déc. 2021.

[11] A. D. Mauro, F. Conti, P. D. Schiavone *et al.*, "Always-on 674µW4GOP/s error resilient binary neural networks with aggressive SRAM voltage scaling on a 22-nm IoT end-node," *IEEE Transactions on Circuits and Systems I : Regular Papers*, vol. 67, nº. 11, p. 3905–3918, 2020.

[12] K. Asifuzzaman, N. R. Miniskar, A. R. Young *et al.*, “A survey on processing-in-memory techniques : Advances and challenges,” *Memories - Materials, Devices, Circuits and Systems*, vol. 4, p. 100022, juill. 2023.

[13] L. Chua, “Memristor-The missing circuit element,” *IEEE Transactions on Circuit Theory*, vol. 18, n°. 5, p. 507–519, 1971.

[14] A. Beck, J. G. Bednorz, C. Gerber *et al.*, “Reproducible switching effect in thin oxide films for memory applications,” *Applied Physics Letters*, vol. 77, n°. 1, p. 139–141, juill. 2000.

[15] R. Waser et M. Aono, “Nanoionics-based resistive switching memories,” *Nature Materials*, vol. 6, n°. 11, p. 833–840, nov. 2007.

[16] D. B. Strukov, G. S. Snider, D. R. Stewart *et al.*, “The missing memristor found,” *Nature*, vol. 453, n°. 7191, p. 80–83, mai 2008.

[17] R. Khaddam-Aljameh, M. Stanisavljevic, J. Fornt Mas *et al.*, “HERMES core – a 14nm CMOS and PCM-based in-memory compute core using an array of 300ps/LSB linearized CCO-based ADCs and local digital processing,” dans *Symposium on VLSI Circuits*, 2021, p. 1–2.

[18] S. Liu, Y. Wang, M. Fardad *et al.*, “A memristor-based optimization framework for artificial intelligence applications,” *IEEE Circuits and Systems Magazine*, vol. 18, n°. 1, p. 29–44, 2018.

[19] S. Kvatinsky, D. Belousov, S. Liman *et al.*, “MAGIC–Memristor-Aided logic,” *IEEE Transactions on Circuits and Systems II : Express Briefs*, vol. 61, n°. 11, p. 895–899, 2014.

[20] S. Kvatinsky, G. Satat, N. Wald *et al.*, “Memristor-based material implication (IMPLY) logic : Design principles and methodologies,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, n°. 10, p. 2054–2066, 2014.

[21] K. Alhaj Ali, M. Rizk, A. Baghdadi *et al.*, “Memristive computational memory using memristor overwrite logic (MOL),” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, n°. 11, p. 2370–2382, 2020.

[22] Y. Zhang, X. Wang et E. G. Friedman, “Memristor-based circuit design for multilayer neural networks,” *IEEE Transactions on Circuits and Systems I : Regular Papers*, vol. 65, n°. 2, p. 677–686, 2018.

[23] R. Hasan, T. M. Taha et C. Yakopcic, “On-chip training of memristor crossbar based multi-layer neural networks,” *Microelectronics Journal*, vol. 66, p. 31–40, août 2017.

[24] M. Hu, C. E. Graves, C. Li, Y. Li, N. Ge, E. Montgomery, N. Davila, H. Jiang, R. S. Williams, J. J. Yang *et al.*, “Memristor-based analog computation and neural network

classification with a dot product engine," *Advanced Materials*, vol. 30, nº. 9, p. 1705914, 2018.

[25] P. Yao, H. Wu, B. Gao *et al.*, "Fully hardware-implemented memristor convolutional neural network," *Nature*, vol. 577, nº. 7792, p. 641–646, janv. 2020.

[26] K. Smagulova, O. Krestinskaya et A. P. James, "A memristor-based long short term memory circuit," *Analog Integrated Circuits and Signal Processing*, vol. 95, nº. 3, p. 467–472, juin 2018.

[27] M. S. Alam, B. R. Fernando, Y. Jaoudi, C. Yakopcic, R. Hasan, T. M. Taha et G. Subramanyam, "Memristor based autoencoder for unsupervised real-time network intrusion and anomaly detection," dans *International Conference on Neuromorphic Systems (ICONS)*. New York, NY, USA : Association for Computing Machinery, 2019.

[28] O. Krestinskaya, K. N. Salama et A. P. James, "Learning in memristive neural network architectures using analog backpropagation circuits," *IEEE Transactions on Circuits and Systems I : Regular Papers*, vol. 66, nº. 2, p. 719–732, 2019.

[29] M. J. Rasch, D. Moreda, T. Gokmen *et al.*, "A flexible and fast PyTorch toolkit for simulating training and inference on analog crossbar arrays," dans *IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*. IEEE, 2021, p. 1–4.

[30] X. Peng, S. Huang, H. Jiang *et al.*, "DNN+NeuroSim V2.0 : An End-to-End Benchmarking Framework for Compute-in-Memory Accelerators for On-Chip Training," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 40, nº. 11, p. 2306–2319, nov. 2021.

[31] S. Jain, A. Sengupta, K. Roy *et al.*, "RxNN : A framework for evaluating deep neural networks on resistive crossbars," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 40, nº. 2, p. 326–338, 2021.

[32] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li et L. Fei-Fei, "ImageNet : A large-scale hierarchical image database," dans *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE Computer Society, 2009, p. 248–255.

[33] K. Pham, S. Tran, T. Nguyen *et al.*, "Asymmetrical Training Scheme of Binary-Memristor-Crossbar-Based Neural Networks for Energy-Efficient Edge-Computing Nanoscale Systems," *Micromachines*, vol. 10, nº. 2, p. 141, févr. 2019.

[34] G. L. Zhang, B. Li, X. Huang *et al.*, "An efficient programming framework for memristor-based neuromorphic computing," dans *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2021, p. 1068–1073.

[35] J. Park, M. Kwak, K. Moon *et al.*, "TiOx-Based RRAM synapse with 64-Levels of conductance and symmetric conductance change by adopting a hybrid pulse scheme for neuromorphic computing," *IEEE Electron Device Letters*, vol. 37, n°. 12, p. 1559–1562, 2016.

[36] T. Hirtzlin, M. Bocquet, J. Klein, E. Nowak, E. Vianello, J. Portal et D. Querlioz, "Outstanding bit error tolerance of resistive RAM-based binarized neural networks," dans *IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*. IEEE, 2019, p. 288–292.

[37] K. Alhaj Ali, A. Baghdadi, E. Dupraz *et al.*, "MOL-Based in-memory computing of binary neural networks," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 30, n°. 7, p. 869–880, 2022.

[38] Y. Zhang, M. Cui, L. Shen *et al.*, "Memristive quantized neural networks : A novel approach to accelerate deep learning on-chip," *IEEE Transactions on Cybernetics*, vol. 51, n°. 4, p. 1875–1887, 2021.

[39] S. Gi, I. Yeo, M. Chu *et al.*, "Fundamental issues of implementing hardware neural networks using memristor," dans *International SoC Design Conference (ISOCC)*, 2015, p. 215–216.

[40] Y. Li, Z. Wang, R. Midya, Q. Xia et J. J. Yang, "Review of memristor devices in neuromorphic computing : materials sciences and device challenges," *Journal of Physics D : Applied Physics*, vol. 51, n°. 50, p. 503002, 2018.

[41] C. Baeumer, R. Valenta, C. Schmitz *et al.*, "Subfilamentary Networks Cause Cycle-to-Cycle Variability in Memristive Devices," *ACS Nano*, vol. 11, n°. 7, p. 6921–6929, juill. 2017.

[42] H. Lv, X. Xu, H. Liu *et al.*, "Evolution of conductive filament and its impact on reliability issues in oxide-electrolyte based resistive random access memory," *Scientific Reports*, vol. 5, n°. 1, p. 7764, janv. 2015.

[43] M. A. Zidan, H. A. H. Fahmy, M. M. Hussain *et al.*, "Memristor-based memory : The sneak paths problem and solutions," *Microelectronics Journal*, vol. 44, n°. 2, p. 176–183, févr. 2013.

[44] X. Zhao, J. Ma, X. Xiao *et al.*, "Breaking the Current-Retention Dilemma in Cation-Based Resistive Switching Devices Utilizing Graphene with Controlled Defects," *Advanced Materials*, vol. 30, n°. 14, p. 1705193, avr. 2018.

[45] K. Zhou, X. Xue, J. Yang *et al.*, "Nonvolatile crossbar 2D2R TCAM with cell size of 16.3 F2 and K-means clustering for power reduction," dans *IEEE Asian Solid-State Circuits Conference (a-SSCC)*, 2018, p. 135–138.

[46] H. Li, S. Wang, X. Zhang *et al.*, "Memristive Crossbar Arrays for Storage and Computing Applications," *Advanced Intelligent Systems*, vol. 3, n°. 9, p. 2100017, sept. 2021.

[47] Z. Chen, C. Schoeny et L. Dolecek, "Hamming distance computation in unreliable resistive memory," *IEEE Transactions on Communications*, vol. 66, n°. 11, p. 5013–5027, 2018.

[48] M.-A. Cantin, Y. Savaria et P. Lavoie, "A comparison of automatic word length optimization procedures," dans *IEEE International Symposium on Circuits and Systems (ISCAS)*, vol. 2, 2002, p. II–II.

[49] J. Kern, E. Dupraz, A. Aïssa-El-Bey *et al.*, "Optimizing the Energy Efficiency of Unreliable Memories for Quantized Kalman Filtering," *Sensors*, vol. 22, n°. 3, p. 853, janv. 2022.

[50] R. E. Kalman, "A New Approach to Linear Filtering and Prediction Problems," *Journal of Basic Engineering*, vol. 82, n°. 1, p. 35–45, mars 1960.

[51] D. D. Kalamkar, D. Mudigere, N. Mellempudi *et al.*, "A study of BFLOAT16 for deep learning training," *CoRR*, vol. abs/1905.12322, 2019.

[52] X. Sun, N. Wang, C.-y. Chen, J.-m. Ni, A. Agrawal, X. Cui, S. Venkataramani, K. El Maghraoui, V. Srinivasan et K. Gopalakrishnan, "Ultra-low precision 4-bit training of deep neural networks," dans *International Conference on Neural Information Processing Systems (NIPS)*. Red Hook, NY, USA : Curran Associates Inc., 2020.

[53] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv et Y. Bengio, "Quantized neural networks : training neural networks with low precision weights and activations," *J. Mach. Learn. Res.*, vol. 18, n°. 1, p. 6869–6898, janv. 2017.

[54] C. Zhu, S. Han, H. Mao *et al.*, "Trained ternary quantization," dans *International Conference on Learning Representations (ICLR)*, 2017.

[55] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv et Y. Bengio, "Binarized neural networks," dans *International Conference on Neural Information Processing Systems (NIPS)*. Red Hook, NY, USA : Curran Associates Inc., 2016, p. 4114–4122.

[56] M. Rastegari, V. Ordonez, J. Redmon et A. Farhadi, "XNOR-Net : ImageNet classification using binary convolutional neural networks," dans *Proceedings of the European Conference on Computer Vision (ECCV)*. Springer, 2016, p. 525–542.

[57] S. Zhou, Z. Ni, X. Zhou *et al.*, "DoReFa-Net : Training low bitwidth convolutional neural networks with low bitwidth gradients," *CoRR*, vol. abs/1606.06160, 2016.

[58] M. AskariHemmat, R. A. Hemmat, A. Hoffman, I. Lazarevich, E. Saboori, O. Mastropietro, Y. Savaria et J. David, "QReg : On regularization effects of quantization," *CoRR*, vol. abs/2206.12372, 2022.

[59] B. Moons, K. Goetschalckx, N. V. Berckelaer et M. Verhelst, “Minimum energy quantized neural networks,” dans *Asilomar Conference on Signals, Systems, and Computers (ACSSC)*, M. B. Matthews, édit. IEEE, 2017, p. 1921–1925.

[60] Y. Le Cun, J. S. Denker et S. A. Solla, “Optimal brain damage,” dans *International Conference on Neural Information Processing Systems (NIPS)*. Cambridge, MA, USA : MIT Press, 1989, p. 598–605.

[61] S. Han, J. Pool, J. Tran et W. J. Dally, “Learning both weights and connections for efficient neural networks,” dans *International Conference on Neural Information Processing Systems (NIPS)*. Cambridge, MA, USA : MIT Press, 2015, p. 1135–1143.

[62] A. Krizhevsky, I. Sutskever et G. E. Hinton, “ImageNet classification with deep convolutional neural networks,” dans *International Conference on Neural Information Processing Systems (NIPS)*. Red Hook, NY, USA : Curran Associates Inc., 2012, p. 1097–1105.

[63] Y.-H. Chen, T. Krishna, J. S. Emer *et al.*, “Eyeriss : An energy-efficient reconfigurable accelerator for deep convolutional neural networks,” *IEEE Journal of Solid-State Circuits*, vol. 52, nº. 1, p. 127–138, 2017.

[64] T. Yang, Y. Chen et V. Sze, “Designing energy-efficient convolutional neural networks using energy-aware pruning,” dans *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE Computer Society, 2017, p. 6071–6079.

[65] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke et A. Rabinovich, “Going deeper with convolutions,” dans *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE Computer Society, 2015, p. 1–9.

[66] J. Frankle et M. Carbin, “The lottery ticket hypothesis : Finding sparse, trainable neural networks,” dans *International Conference on Learning Representations (ICLR)*, 2019.

[67] Y. He et L. Xiao, “Structured pruning for deep convolutional neural networks : A survey,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 46, nº. 5, p. 2900–2919, 2024.

[68] S. Han, H. Mao et W. J. Dally, “Deep compression : Compressing deep neural networks with pruning, trained quantization and huffman coding,” dans *International Conference on Learning Representations (ICLR)*, 2016.

[69] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz et W. J. Dally, “EIE : Efficient Inference Engine on compressed deep neural network,” dans *ACM/IEEE Annual International Symposium on Computer Architecture (ISCA)*. IEEE Computer Society, 2016, p. 243–254.

[70] J. Albericio, P. Judd, T. Hetherington *et al.*, “Cnvlutin : Ineffectual-neuron-free deep neural network computing,” dans *ACM/IEEE Annual International Symposium on Computer Architecture (ISCA)*, 2016, p. 1–13.

[71] B. Reagen, P. Whatmough, R. Adolf *et al.*, “Minerva : Enabling low-power, highly-accurate deep neural network accelerators,” dans *ACM/IEEE Annual International Symposium on Computer Architecture (ISCA)*, 2016, p. 267–278.

[72] C. Buciluǎ, R. Caruana et A. Niculescu-Mizil, “Model compression,” dans *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2006, p. 535–541.

[73] G. E. Hinton, O. Vinyals et J. Dean, “Distilling the knowledge in a neural network,” *CoRR*, vol. abs/1503.02531, 2015.

[74] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta et Y. Bengio, “Fitnets : Hints for thin deep nets,” dans *International Conference on Learning Representations (ICLR)*, Y. Bengio et Y. LeCun, édit., 2015.

[75] L. J. Ba et R. Caruana, “Do deep nets really need to be deep ?” dans *International Conference on Neural Information Processing Systems (NIPS)*. Cambridge, MA, USA : MIT Press, 2014, p. 2654–2662.

[76] G. Coiffier, G. Boukli Hacene et V. Gripon, “ThriftyNets : Convolutional Neural Networks with Tiny Parameter Budget,” *IoT*, vol. 2, n°. 2, p. 222–235, mars 2021.

[77] X. Tang, Y. Wang, T. Cao, L. L. Zhang, Q. Chen, D. Cai, Y. Liu et M. Yang, “LUT-NN : empower efficient neural network inference with centroid learning and table lookup,” dans *Annual International Conference on Mobile Computing and Networking (MobiCom)*, X. Costa-Pérez, J. Widmer, D. Perino, D. Giustiniano, H. Al-Hassanieh, A. Asadi et L. P. Cox, édit. ACM, 2023, p. 70 :1–70 :15.

[78] M. Sandler, A. G. Howard, M. Zhu, A. Zhmoginov et L. Chen, “MobileNetV2 : Inverted residuals and linear bottlenecks,” dans *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Computer Vision Foundation / IEEE Computer Society, 2018, p. 4510–4520.

[79] J. Kossaifi, A. Bulat, G. Tzimiropoulos et M. Pantic, “T-Net : Parametrizing fully convolutional nets with a single high-order tensor,” dans *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Computer Vision Foundation / IEEE, 2019, p. 7822–7831.

[80] V. Lebedev, Y. Ganin, M. Rakhuba, I. V. Oseledets et V. S. Lempitsky, “Speeding-up convolutional neural networks using fine-tuned CP-Decomposition,” dans *International Conference on Learning Representations (ICLR)*, Y. Bengio et Y. LeCun, édit., 2015.

[81] J. Han et M. Orshansky, “Approximate computing : An emerging paradigm for energy-efficient design,” dans *IEEE European Test Symposium (ETS)*, 2013, p. 1–6.

[82] P. Knag, W. Lu et Z. Zhang, “A native stochastic computing architecture enabled by memristors,” *IEEE Transactions on Nanotechnology*, vol. 13, n°. 2, p. 283–293, 2014.

[83] L. N. Smith et N. Topin, “Super-convergence : Very fast training of residual networks using large learning rates,” *CoRR*, vol. abs/1708.07120, 2017.

[84] Y. Song, T. Wang, P. Cai, S. K. Mondal et J. P. Sahoo, “A comprehensive survey of few-shot learning : Evolution, applications, challenges, and opportunities,” *ACM Comput. Surv.*, vol. 55, n°. 13s, juill. 2023. [En ligne]. Disponible : https://doi.org/10.1145/3582688

[85] A. Ardakani, F. Leduc-Primeau, N. Onizawa *et al.*, “VLSI implementation of deep neural network using integral stochastic computing,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, n°. 10, p. 2688–2699, 2017.

[86] M. G. Taylor, “Reliable information storage in memories designed from unreliable components,” *The Bell System Technical Journal*, vol. 47, n°. 10, p. 2299–2337, 1968.

[87] C. Hadjicostis et G. Verghese, “Coding approaches to fault tolerance in linear dynamic systems,” *IEEE Transactions on Information Theory*, vol. 51, n°. 1, p. 210–228, 2005.

[88] R. Gallager, “Low-density parity-check codes,” *IRE Transactions on Information Theory*, vol. 8, n°. 1, p. 21–28, 1962.

[89] C.-H. Huang, Y. Li et L. Dolecek, “Gallager B LDPC decoder with transient and permanent errors,” dans *IEEE International Symposium on Information Theory (ISIT)*, 2013, p. 3010–3014.

[90] F. Leduc-Primeau, F. R. Kschischang et W. J. Gross, “Modeling and energy optimization of LDPC decoder circuits with timing violations,” *IEEE Transactions on Communications*, vol. 66, n°. 3, p. 932–946, 2018.

[91] J. Nadal, M. Yaoumi, E. Dupraz *et al.*, “Energy optimization of faulty quantized min-sum LDPC decoders,” dans *International Symposium on Topics in Coding (ISTC)*, 2023, p. 1–5.

[92] T. Liu, W. Wen, L. Jiang *et al.*, “A fault-tolerant neural network architecture,” dans *ACM/IEEE Design Automation Conference (DAC)*, 2019, p. 1–6.

[93] R. Hegde et N. Shanbhag, “Energy-efficient signal processing via algorithmic noise-tolerance,” dans *International Symposium on Low Power Electronics and Design (ISLPED)*, 1999, p. 30–35.

[94] ——, “Soft digital signal processing,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 9, n°. 6, p. 813–823, 2001.

[95] Y. Lin, S. Zhang et N. R. Shanbhag, "Variation-tolerant architectures for convolutional neural networks in the near threshold voltage regime," dans *IEEE International Workshop on Signal Processing Systems (SiPS)*. IEEE, 2016, p. 17–22.

[96] J. Choi, E. P. Kim, R. A. Rutenbar et N. R. Shanbhag, "Error resilient MRF message passing architecture for stereo matching," dans *IEEE International Workshop on Signal Processing Systems (SiPS)*. IEEE, 2013, p. 348–353.

[97] R. A. Abdallah et N. R. Shanbhag, "Error-resilient low-power Viterbi decoders," dans *International Symposium on Low Power Electronics and Design (ISLPED)*, 2008, p. 111–116.

[98] Y. Lin et J. R. Cavallaro, "Energy-efficient convolutional neural networks via statistical error compensated near threshold computing," dans *IEEE International Symposium on Circuits and Systems (ISCAS)*, 2018, p. 1–5.

[99] S. Lloyd, "Least squares quantization in PCM," *IEEE Transactions on Information Theory*, vol. 28, n°. 2, p. 129–137, 1982.

[100] J. Wang, C. Wang, Q. Lin *et al.*, "Adversarial attacks and defenses in deep learning for image recognition : A survey," *Neurocomputing*, vol. 514, p. 162–181, déc. 2022.

[101] J.-C. Vialatte et F. Leduc-Primeau, "A study of deep learning robustness against computation failures," *CoRR*, vol. abs/1704.05396, 2017.

[102] L. Yang, D. Bankman, B. Moons *et al.*, "Bit error tolerance of a CIFAR-10 binarized convolutional neural network processor," dans *IEEE international Symposium on Circuits and Systems (ISCAS)*, 2018, p. 1–5.

[103] X. Sun, R. Liu, Y.-J. Chen *et al.*, "Low-VDD operation of SRAM synaptic array for implementing ternary neural network," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, n°. 10, p. 2962–2965, 2017.

[104] J. Choi, S. Venkataramani, V. V. Srinivasan *et al.*, "Accurate and efficient 2-bit quantized neural networks," dans *Proceedings of Machine Learning and Systems (MLSys)*, A. Talwalkar, V. Smith et M. Zaharia, édit., vol. 1, 2019, p. 348–359.

[105] K. Chitsaz, G. Mordido, J.-P. David *et al.*, "Training DNNs resilient to adversarial and random bit-flips by learning quantization ranges," *Transactions on Machine Learning Research (TMLR)*, 2023.

[106] A. Murray et P. Edwards, "Enhanced MLP performance and fault tolerance resulting from synaptic weight noise during training," *IEEE Transactions on Neural Networks*, vol. 5, n°. 5, p. 792–802, 1994.

[107] G. B. Hacene, F. Leduc-Primeau, A. B. Soussia, V. Gripon et F. Gagnon, "Training modern deep neural networks for memory-fault robustness," dans *IEEE International*

Symposium on Circuits and Systems (ISCAS). IEEE, 2019, p. 1–5.

[108] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever et R. Salakhutdinov, "Dropout : a simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, n°. 1, p. 1929–1958, janv. 2014.

[109] V. Joshi, M. Le Gallo, S. Haefeli *et al.*, "Accurate deep neural network inference using computational phase-change memory," *Nature Communications*, vol. 11, n°. 1, p. 2473, mai 2020.

[110] M. J. Rasch, C. Mackin, M. Le Gallo *et al.*, "Hardware-aware training for large-scale and diverse deep learning inference workloads using in-memory computing-based accelerators," *Nature Communications*, vol. 14, n°. 1, p. 5282, août 2023.

[111] S. Buschjäger, J.-J. Chen, K.-H. Chen *et al.*, "Margin-maximization in binarized neural networks for optimizing bit error tolerance," dans *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2021, p. 673–678.

[112] P. Dey, K. Nag, T. Pal *et al.*, "Regularizing multilayer perceptron for robustness," *IEEE Transactions on Systems, Man, and Cybernetics : Systems*, vol. 48, n°. 8, p. 1255–1266, 2018.

[113] X. He, K. Zhao et X. Chu, "AutoML : A survey of the state-of-the-art," *Knowledge-Based Systems*, vol. 212, p. 106622, janv. 2021.

[114] T. Elsken, J. H. Metzen et F. Hutter, "Neural architecture search : A survey," *Journal of Machine Learning Research*, vol. 20, n°. 1, p. 1997–2017, janv. 2019.

[115] Z. Liu, H. Hu, Y. Lin, Z. Yao, Z. Xie, Y. Wei, J. Ning, Y. Cao, Z. Zhang, L. Dong, F. Wei et B. Guo, "Swin transformer V2 : scaling up capacity and resolution," dans *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2022, p. 11 999–12 009.

[116] B. Wu, K. Keutzer, X. Dai, P. Zhang, Y. Wang, F. Sun, Y. Wu, Y. Tian, P. Vajda et Y. Jia, "FBNet : Hardware-aware efficient ConvNet design via differentiable neural architecture search," dans *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. Los Alamitos, CA, USA : IEEE Computer Society, juin 2019, p. 10 726–10 734.

[117] M. Chen, K. Wu, B. Ni, H. Peng, B. Liu, J. Fu, H. Chao et H. Ling, "Searching the search space of vision transformer," dans *International Conference on Neural Information Processing Systems (NIPS)*. Red Hook, NY, USA : Curran Associates Inc., 2024.

[118] C. Gong, Z. Jiang, D. Wang, Y. Lin, Q. Liu et D. Z. Pan, "Mixed precision neural architecture search for energy efficient deep learning," dans *IEEE/ACM International*

Conference on Computer-Aided Design (ICCAD), 2019, p. 1–7.

[119] T.-J. Yang, A. Howard, B. Chen, X. Zhang, A. Go, M. Sandler, V. Sze et H. Adam, "Netadapt : Platform-aware neural network adaptation for mobile applications," dans *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, p. 285–300.

[120] H. Cai, L. Zhu et S. Han, "ProxylessNAS : Direct neural architecture search on target task and hardware," dans *International Conference on Learning Representations (ICLR)*. OpenReview.net, 2019.

[121] H. Cai, C. Gan, T. Wang, Z. Zhang et S. Han, "Once-for-All : Train one network and specialize it for efficient deployment," dans *International Conference on Learning Representations (ICLR)*. OpenReview.net, 2020.

[122] K. T. Chitty-Venkata et A. K. Somani, "Neural architecture search survey : A hardware perspective," *ACM Comput. Surv.*, vol. 55, n°. 4, nov. 2022. [En ligne]. Disponible : https://doi.org/10.1145/3524500

[123] H. Benmeziane, K. El Maghraoui, H. Ouarnoughi *et al.*, "Hardware-Aware Neural Architecture Search : Survey and Taxonomy," dans *International Joint Conference on Artificial Intelligence (IJCAI)*. Montreal, Canada : International Joint Conferences on Artificial Intelligence Organization, août 2021, p. 4322–4329.

[124] C. Li, Z. Yu, Y. Fu, Y. Zhang, Y. Zhao, H. You, Q. Yu, Y. Wang, C. Hao et Y. Lin, "HW-NAS-Bench : Hardware-aware neural architecture search benchmark," dans *International Conference on Learning Representations (ICLR)*. OpenReview.net, 2021.

[125] H. Benmeziane, C. Lammie, I. Boybat *et al.*, "AnalogNAS : A neural network design framework for accurate inference with analog in-memory computing," dans *IEEE International Conference on Edge Computing and Communications (EDGE)*, 2023, p. 233–244.

[126] W. Li, X. Ning, G. Ge *et al.*, "FTT-NAS : Discovering fault-tolerant neural architecture," dans *Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2020, p. 211–216.

[127] L. Yang, Z. Yan, M. Li, H. Kwon, L. Lai, T. Krishna, V. Chandra, W. Jiang et Y. Shi, "Co-exploration of neural architectures and heterogeneous ASIC accelerator designs targeting multiple tasks," dans *ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2020, p. 1–6.

[128] W. Jiang, Q. Lou, Z. Yan *et al.*, "Device-circuit-architecture co-exploration for computing-in-memory neural accelerators," *IEEE Transactions on Computers*, vol. 70, n°. 4, p. 595–605, 2021.

[129] E. Dupraz, L. R. Varshney et F. Leduc-Primeau, “Power-efficient deep neural networks with noisy memristor implementation,” dans *IEEE Information Theory Workshop (ITW)*, 2021, p. 1–5.

[130] A. Krizhevsky et G. Hinton, “Learning multiple layers of features from tiny images,” Technical report, University of Toronto / University of Toronto, Toronto, Ontario, Rapport technique 0, 2009.

[131] I. Sutskever, J. Martens, G. E. Dahl et G. E. Hinton, “On the importance of initialization and momentum in deep learning,” dans *International Conference on Machine Learning (ICML)*, ser. JMLR Workshop and Conference Proceedings, vol. 28. JMLR.org, 2013, p. 1139–1147.

[132] D. Kingma et J. Ba, “Adam : A method for stochastic optimization,” dans *International Conference on Learning Representations (ICLR)*, San Diega, CA, USA, déc. 2015.

[133] M. Courbariaux, Y. Bengio et J.-P. David, “Binaryconnect : training deep neural networks with binary weights during propagations,” dans *International Conference on Neural Information Processing Systems (NIPS)*. Cambridge, MA, USA : MIT Press, 2015, p. 3123–3131.

[134] Y. Bengio, N. Léonard et A. C. Courville, “Estimating or propagating gradients through stochastic neurons for conditional computation,” *CoRR*, vol. abs/1308.3432, 2013.

[135] K. He, X. Zhang, S. Ren et J. Sun, “Delving deep into rectifiers : Surpassing human-level performance on ImageNet classification,” dans *IEEE International Conference on Computer Vision (ICCV)*. IEEE Computer Society, 2015, p. 1026–1034.

[136] D. Stutz, N. Chandramoorthy, M. Hein et B. Schiele, “Bit error robustness for energy-efficient DNN accelerators,” dans *Proceedings of Machine Learning and Systems (MLSys)*, A. Smola, A. Dimakis et I. Stoica, édit., vol. 3, 2021, p. 569–598.

[137] D. Stutz, N. Chandramoorthy, M. Hein *et al.*, “Random and adversarial bit error robustness : Energy-efficient and secure DNN accelerators,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, n°. 3, p. 3632–3647, 2023.

[138] S. Hochreiter et J. Schmidhuber, “Simplifying neural nets by discovering flat minima,” dans *International Conference on Neural Information Processing Systems (NIPS)*. Cambridge, MA, USA : MIT Press, 1994, p. 529–536.

[139] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy et P. T. P. Tang, “On large-batch training for deep learning : Generalization gap and sharp minima,” dans *International Conference on Learning Representations (ICLR)*, 2017.

[140] B. Neyshabur, S. Bhojanapalli, D. McAllester et N. Srebro, “Exploring generalization in deep learning,” dans *International Conference on Neural Information Processing*

Systems (NIPS). Red Hook, NY, USA : Curran Associates Inc., 2017, p. 5949–5958.

[141] P. Chaudhari, A. Choromanska, S. Soatto, Y. LeCun, C. Baldassi, C. Borgs, J. T. Chayes, L. Sagun et R. Zecchina, "Entropy-SGD : Biasing gradient descent into wide valleys," dans *International Conference on Learning Representations (ICLR)*. OpenReview.net, 2017.

[142] P. Foret, A. Kleiner, H. Mobahi et B. Neyshabur, "Sharpness-aware minimization for efficiently improving generalization," dans *International Conference on Learning Representations (ICLR)*. OpenReview.net, 2021.

[143] J. Du, D. Zhou, J. Feng, V. Y. F. Tan et J. T. Zhou, "Sharpness-aware training for free," dans *International Conference on Neural Information Processing Systems (NIPS)*. Red Hook, NY, USA : Curran Associates Inc., 2024.

[144] J. Du, H. Yan, J. Feng, J. T. Zhou, L. Zhen, R. S. M. Goh et V. Y. F. Tan, "Efficient sharpness-aware minimization for improved training of neural networks," dans *International Conference on Learning Representations (ICLR)*. OpenReview.net, 2022.

[145] W. Zhou, F. Liu, H. Zhang et M. Chen, "Sharpness-aware minimization with dynamic reweighting," dans *Empirical Methods in Natural Language Processing (EMNLP)*, Y. Goldberg, Z. Kozareva et Y. Zhang, édit. Association for Computational Linguistics, 2022, p. 5686–5699.

[146] Y. Liu, S. Mai, X. Chen, C. Hsieh et Y. You, "Towards efficient and scalable sharpness-aware minimization," dans *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2022, p. 12 350–12 360.

[147] Y. Zhao, H. Zhang et X. Hu, "SS-SAM : Stochastic scheduled sharpness-aware minimization for efficiently training deep neural networks," *CoRR*, vol. abs/2203.09962, 2022.

[148] J. Zhuang, B. Gong, L. Yuan, Y. Cui, H. Adam, N. C. Dvornek, S. Tatikonda, J. S. Duncan et T. Liu, "Surrogate gap minimization improves sharpness-aware training," dans *International Conference on Learning Representations (ICLR)*. OpenReview.net, 2022.

[149] M. Kim, D. Li, S. X. Hu et T. Hospedales, "Fisher SAM : Information geometry and sharpness aware minimisation," dans *International Conference on Machine Learning (ICML)*, K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvari, G. Niu et S. Sabato, édit., vol. 162. PMLR, juill. 2022, p. 11 148–11 161.

[150] J. Kwon, J. Kim, H. Park et I. K. Choi, "ASAM : Adaptive sharpness-aware minimization for scale-invariant learning of deep neural networks," dans *International Conference on Machine Learning (ICML)*, M. Meila et T. Zhang, édit., vol. 139. PMLR, juill. 2021, p. 5905–5914.

[151] M. Andriushchenko et N. Flammarion, “Towards understanding sharpness-aware minimization,” dans *International Conference on Machine Learning (ICML)*, K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvári, G. Niu et S. Sabato, édit., vol. 162. PMLR, 2022, p. 639–668.

[152] J. Liu, J. Cai et B. Zhuang, “Sharpness-aware quantization for deep neural networks,” *CoRR*, vol. abs/2111.12273, 2021.

[153] H. Liu, J. Z. HaoChen, A. Gaidon et T. Ma, “Self-supervised learning is more robust to dataset imbalance,” dans *International Conference on Learning Representations (ICLR)*. OpenReview.net, 2022.

[154] L. Dinh, R. Pascanu, S. Bengio et Y. Bengio, “Sharp minima can generalize for deep nets,” dans *International Conference on Machine Learning (ICML)*, D. Precup et Y. W. Teh, édit., vol. 70. PMLR, 2017, p. 1019–1028.

[155] D. Stutz, M. Hein et B. Schiele, “Relating adversarially robust generalization to flat minima,” dans *IEEE/CVF International Conference on Computer Vision (ICCV)*. IEEE, 2021, p. 7787–7797.

[156] X. Sun, Z. Zhang, X. Ren, R. Luo et L. Li, “Exploring the vulnerability of deep neural networks : A study of parameter corruption,” dans *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, n°. 13, 2021, p. 11 648–11 656.

[157] A. Madry, A. Makelov, L. Schmidt, D. Tsipras et A. Vladu, “Towards deep learning models resistant to adversarial attacks,” dans *International Conference on Learning Representations (ICLR)*. OpenReview.net, 2018.

[158] W. He, J. Wei, X. Chen, N. Carlini et D. Song, “Adversarial example defenses : ensembles of weak defenses are not strong,” dans *Proceedings of the 11th USENIX Conference on Offensive Technologies*, ser. WOOT’17. USA : USENIX Association, 2017, p. 15.

[159] J. Kern, S. Henwood, G. Mordido *et al.*, “Fast and accurate output error estimation for memristor-based deep neural networks,” *IEEE Transactions on Signal Processing*, p. 1–13, 2024.

[160] T. Gokmen, M. Onen et W. Haensch, “Training Deep Convolutional Neural Networks with Resistive Cross-Point Devices,” *Frontiers in Neuroscience*, vol. 11, p. 538, oct. 2017.

[161] K. Simonyan et A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” dans *International Conference on Learning Representations (ICLR)*, Y. Bengio et Y. LeCun, édit., 2015.

[162] Z. Wang, C. Li, P. Lin *et al.*, “In situ training of feed-forward and recurrent convolutional memristor networks,” *Nature Machine Intelligence*, vol. 1, n°. 9, p. 434–442, sept. 2019.

[163] M. Hu, J. P. Strachan, Z. Li *et al.*, “Dot-product engine for neuromorphic computing : Programming 1T1M crossbar to accelerate matrix-vector multiplication,” dans *ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2016, p. 1–6.

[164] E. Dupraz, F. Leduc-Primeau, K. Cai *et al.*, “Turning to information theory to bring in-memory computing into practice,” *IEEE BITS the Information Theory Magazine*, p. 1–13, 2023.

[165] H. Tsai, S. Ambrogio, C. Mackin, P. Narayanan, R. M. Shelby, K. Rocki, A. Chen et G. W. Burr, “Inference of long-short term memory networks at software-equivalent accuracy using 2.5m analog phase change memory devices,” dans *Symposium on VLSI Technology*, 2019, p. T82–T83.

[166] A. J. Pérez-Ávila, G. González-Cordero, E. Pérez *et al.*, “Behavioral modeling of multilevel HfO2-based memristors for neuromorphic circuit simulation,” dans *Conference on Design of Circuits and Integrated Systems (DCIS)*, 2020, p. 1–6.

[167] V. Milo, C. Zambelli, P. Olivo *et al.*, “Multilevel HfO2-based RRAM devices for low-power neuromorphic networks,” *APL Materials*, vol. 7, n°. 8, p. 081120, août 2019.

[168] S. Ambrogio, M. Gallot, K. Spoon, H. Tsai, C. Mackin, M. Wesson, S. Kariyappa, P. Narayanan, C.-C. Liu, A. Kumar, A. Chen et G. W. Burr, “Reducing the impact of phase-change memory conductance drift on the inference of large-scale hardware neural networks,” dans *IEEE International Electron Devices Meeting (IEDM)*, 2019, p. 6.1.1–6.1.4.

[169] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai et S. Chintala, “PyTorch : an imperative style, high-performance deep learning library,” dans *International Conference on Neural Information Processing Systems (NIPS)*. Red Hook, NY, USA : Curran Associates Inc., 2019.

[170] C. Zhang, S. Bengio et Y. Singer, “Are all layers created equal ?” *Journal of Machine Learning Research*, vol. 23, n°. 1, janv. 2022.

[171] Y. Kim, M. Kang, L. R. Varshney *et al.*, “Generalized water-filling for source-aware energy-efficient SRAMs,” *IEEE Transactions on Communications*, vol. 66, n°. 10, p. 4826–4841, 2018.

[172] S. Zagoruyko et N. Komodakis, “Wide residual networks,” dans *British Machine Vision Conference 2016 (BMVC)*, sept. 2016, p. 1–87.

[173] K. He, X. Zhang, S. Ren et J. Sun, “Deep residual learning for image recognition,” dans *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE Computer Society, 2016, p. 770–778.

[174] M. D. McDonnell, “Training wide residual networks for deployment using a single bit for each weight,” dans *International Conference on Learning Representations (ICLR)*, 2018.

[175] S. J. Wright, “Coordinate descent algorithms,” *Mathematical Programming*, vol. 151, n°. 1, p. 3–34, juin 2015.

[176] T. Miyato, T. Kataoka, M. Koyama et Y. Yoshida, “Spectral normalization for generative adversarial networks,” dans *International Conference on Learning Representations (ICLR)*. OpenReview.net, 2018.

[177] S. H. Khan, M. Hayat et F. Porikli, “Regularization of deep neural networks with spectral dropout,” *Neural Networks*, vol. 110, p. 82–90, févr. 2019.

[178] M. Cettolo, J. Niehues, S. Stüker *et al.*, “Report on the 11th IWSLT evaluation campaign,” dans *Proceedings of the 11th International Workshop on Spoken Language Translation (IWSLT) : Evaluation Campaign*, M. Federico, S. Stüker et F. Yvon, édit., Lake Tahoe, California, déc. 2014, p. 2–17.

[179] G. Huang, Z. Liu, L. van der Maaten et K. Q. Weinberger, “Densely connected convolutional networks,” dans *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE Computer Society, 2017, p. 2261–2269.

[180] A. Howard, R. Pang, H. Adam, Q. V. Le, M. Sandler, B. Chen, W. Wang, L. Chen, M. Tan, G. Chu, V. Vasudevan et Y. Zhu, “Searching for MobileNetV3,” dans *IEEE/CVF International Conference on Computer Vision (ICCV)*. IEEE, 2019, p. 1314–1324.

[181] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser et I. Polosukhin, “Attention is all you need,” dans *International Conference on Neural Information Processing Systems (NIPS)*. Red Hook, NY, USA : Curran Associates Inc., 2017, p. 6000–6010.

[182] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit et N. Houlsby, “An image is worth 16x16 words : Transformers for image recognition at scale,” dans *International Conference on Learning Representations (ICLR)*. OpenReview.net, 2021.

[183] T. Ridnik, E. Ben-Baruch, A. Noy et L. Zelnik, “ImageNet-21K pretraining for the masses,” dans *Proceedings of the Neural Information Processing Systems (NIPS) Track on Datasets and Benchmarks*, J. Vanschoren et S. Yeung, édit., vol. 1. Curran Associates Inc., 2021.

[184] S. R. Nandakumar, I. Boybat, V. Joshi, C. Piveteau, M. L. Gallo, B. Rajendran, A. Sebastian et E. Eleftheriou, "Phase-change memory models for deep learning training and inference," dans *IEEE International Conference on Electronics, Circuits and Systems (ICECS)*. IEEE, 2019, p. 727–730.

[185] V. Verma, A. Lamb, C. Beckham *et al.*, "Manifold mixup : Better representations by interpolating hidden states," dans *International Conference on Machine Learning (ICML)*, K. Chaudhuri et R. Salakhutdinov, édit., vol. 97. PMLR, juin 2019, p. 6438–6447.

[186] M. Faramarzi, M. Amini, A. Badrinaaraayanan, V. Verma et S. Chandar, "PatchUp : A feature-space block-level regularization technique for convolutional neural networks," dans *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, nº. 1, juin 2022, p. 589–597.

[187] K. Deb, A. Pratap, S. Agarwal *et al.*, "A fast and elitist multiobjective genetic algorithm : NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, nº. 2, p. 182–197, avr. 2002.

[188] J. Blank et K. Deb, "Pymoo : Multi-objective optimization in python," *IEEE Access*, vol. 8, p. 89 497–89 509, 2020.

[189] M. D. McKay, "Latin hypercube sampling as a tool in uncertainty analysis of computer models," dans *Winter Simulation Conference (WSC)*, R. C. Crain, édit. ACM Press, 1992, p. 557–564.

ANNEXE A IMPLÉMENTATION DE SAMSON AVEC LA LIBRAIRIE PYTORCH

En nous basant sur l'implémentation de référence de la méthode ASAM [150] [1], nous proposons une implémentation de SAMSON visible ci-dessous :

```python
class SAMSON(ASAM):
  def __init__(self, optimizer, model, rho=0.5, eta=0.01, norm="inf"):
      self.optimizer = optimizer
      self.model = model
      self.rho = rho
      self.eta = eta
      self.state = defaultdict(dict)
      self.norm = float(norm)

  @torch.no_grad()
  def ascent_step(self):
      wgrads = []
      for n, p in self.model.named_parameters():
          if p.grad is None:
              continue
          t_w = self.state[p].get("eps")
          if t_w is None:
              t_w = torch.clone(p).detach()
              self.state[p]["eps"] = t_w
          if 'weight' in n:
              t_w[...] = p[...]
               t_w.abs_().add_(self.eta) # ASAM !
              t_w.abs_().add_(self.eta).div_(torch.norm(p, p=self.norm)) # SAMSON !
              p.grad.mul_(t_w)
          wgrads.append(torch.norm(p.grad, p=2))
      wgrad_norm = torch.norm(torch.stack(wgrads), p=2) + 1.e-16
      for n, p in self.model.named_parameters():
```

1. Disponible à l'adresse suivante : `https://github.com/SamsungLabs/ASAM/blob/master/asam.py`

```
28            if p.grad is None:
29                continue
30            t_w = self.state[p].get("eps")
31            if 'weight' in n:
32                p.grad.mul_(t_w)
33            eps = t_w
34            eps[...] = p.grad[...]
35            eps.mul_(self.rho / wgrad_norm)
36            p.add_(eps)
37        self.optimizer.zero_grad()
```

Un exemple d'utilisation de la classe ASAM est proposé dans le répertoire Github de l'implémentation de référence [2]. Un objet de la classe SAMSON proposée ci-dessus peut être substitué à un objet de la classe ASAM sans autre adaptation du script.

2. Disponible à l'adresse suivante : `https://github.com/SamsungLabs/ASAM/blob/master/example_cifar.py`

ANNEXE B RÉSULTATS DÉTAILLÉS DES EXPÉRIENCES OOD POUR LES MÉTHODES D'ENTRAÎNEMENT CONSCIENTES DE LA RÉGULARITÉ

Sont présentées à chaque tableau de cette annexe pour une architecture de DNN donnée (DenseNet-40, MobileNetV2, ResNet-34, ResNet-50, VGG-13) sur un jeu de données (CIFAR-10 puis CIFAR-100) la précision moyenne et l'écart-type pour trois initialisation aléatoires de DNN, chacun évalué à trois reprises sur l'ensemble de test pour chaque transformation OOD du jeu de données pour chaque méthode d'entraînement considérée (SGD, SAM, ASAM, SAMSON_2, SAMSON_∞). La ou les meilleures méthodes pour chaque ligne de chaque tableau de cette annexe participe au compte permettant de produire la figure 5.7.

Tableau B.1 Expériences OOD pour l'architecture DenseNet-40 sur CIFAR-10.

Transformation	**SGD**	**SAM**	**ASAM**	**SAMSON_2**	**SAMSON_∞**
rotation_{20}	$78{,}81_{\pm 0{,}62}$	$78{,}38_{\pm 0{,}53}$	$79{,}49_{\pm 0{,}93}$	$79{,}25_{\pm 0{,}38}$	$\mathbf{80{,}33_{\pm 0{,}82}}$
rotation_{40}	$56{,}86_{\pm 0{,}77}$	$56{,}51_{\pm 0{,}69}$	$57{,}51_{\pm 1{,}53}$	$57{,}33_{\pm 1{,}73}$	$\mathbf{58{,}73_{\pm 0{,}94}}$
$\text{cisaillement}_{28{,}6}$	$79{,}47_{\pm 0{,}49}$	$80{,}22_{\pm 0{,}69}$	$81{,}41_{\pm 0{,}56}$	$80{,}31_{\pm 0{,}85}$	$\mathbf{81{,}65_{\pm 0{,}64}}$
$\text{cisaillement}_{57{,}3}$	$54{,}49_{\pm 0{,}24}$	$54{,}05_{\pm 0{,}69}$	$56{,}08_{\pm 1{,}33}$	$54{,}89_{\pm 1{,}82}$	$\mathbf{56{,}69_{\pm 0{,}34}}$
zoom_{120}	$69{,}95_{\pm 1{,}88}$	$70{,}08_{\pm 1{,}79}$	$70{,}08_{\pm 1{,}65}$	$\mathbf{71{,}49_{\pm 1{,}97}}$	$70{,}64_{\pm 1{,}40}$
zoom_{140}	$39{,}18_{\pm 2{,}63}$	$39{,}23_{\pm 3{,}93}$	$38{,}65_{\pm 1{,}61}$	$\mathbf{39{,}47_{\pm 0{,}34}}$	$39{,}17_{\pm 1{,}13}$
zoom_{60}	$69{,}91_{\pm 0{,}53}$	$71{,}26_{\pm 0{,}37}$	$71{,}84_{\pm 1{,}43}$	$\mathbf{72{,}72_{\pm 0{,}47}}$	$72{,}60_{\pm 0{,}74}$
zoom_{80}	$85{,}53_{\pm 0{,}39}$	$86{,}01_{\pm 0{,}27}$	$86{,}48_{\pm 0{,}24}$	$86{,}75_{\pm 0{,}39}$	$\mathbf{87{,}08_{\pm 0{,}47}}$

TABLEAU B.2 Expériences OOD pour l'architecture MobileNetV2 sur CIFAR-10.

Transformation	**SGD**	**SAM**	**ASAM**	**SAMSON$_2$**	**SAMSON$_\infty$**
rotation$_{20}$	86,89$_{\pm 0,45}$	87,02$_{\pm 0,04}$	**88,21$_{\pm 0,06}$**	87,47$_{\pm 0,42}$	87,43$_{\pm 0,48}$
rotation$_{40}$	64,25$_{\pm 0,04}$	66,16$_{\pm 1,41}$	**68,05$_{\pm 0,14}$**	66,24$_{\pm 0,04}$	67,17$_{\pm 0,37}$
cisaillement$_{28,6}$	85,49$_{\pm 0,04}$	86,82$_{\pm 0,18}$	**87,70$_{\pm 0,16}$**	87,40$_{\pm 0,57}$	87,30$_{\pm 0,30}$
cisaillement$_{57,3}$	59,93$_{\pm 0,18}$	**63,09$_{\pm 1,64}$**	62,56$_{\pm 1,51}$	62,10$_{\pm 0,64}$	62,56$_{\pm 0,70}$
zoom$_{120}$	77,56$_{\pm 1,13}$	**81,46$_{\pm 1,87}$**	79,80$_{\pm 1,02}$	80,97$_{\pm 1,41}$	80,97$_{\pm 1,35}$
zoom$_{140}$	47,20$_{\pm 2,41}$	**50,24$_{\pm 3,09}$**	49,28$_{\pm 0,18}$	48,62$_{\pm 2,03}$	48,62$_{\pm 1,92}$
zoom$_{60}$	76,46$_{\pm 1,68}$	75,59$_{\pm 0,12}$	77,81$_{\pm 1,33}$	**78,61$_{\pm 1,16}$**	**78,61$_{\pm 0,30}$**
zoom$_{80}$	90,11$_{\pm 0,45}$	90,69$_{\pm 0,22}$	**91,53$_{\pm 0,21}$**	91,16$_{\pm 0,10}$	91,16$_{\pm 0,04}$

TABLEAU B.3 Expériences OOD pour l'architecture ResNet-34 sur CIFAR-10.

Transformation	**SGD**	**SAM**	**ASAM**	**SAMSON$_2$**	**SAMSON$_\infty$**
rotation$_{20}$	88,15$_{\pm 0,37}$	89,11$_{\pm 0,60}$	88,18$_{\pm 0,94}$	**90,07$_{\pm 0,66}$**	89,29$_{\pm 0,58}$
rotation$_{40}$	66,29$_{\pm 1,92}$	66,19$_{\pm 1,21}$	65,95$_{\pm 1,70}$	**68,02$_{\pm 1,75}$**	66,81$_{\pm 0,40}$
cisaillement$_{28,6}$	86,76$_{\pm 0,57}$	88,46$_{\pm 0,58}$	86,88$_{\pm 0,94}$	**89,05$_{\pm 1,19}$**	87,92$_{\pm 0,16}$
cisaillement$_{57,3}$	60,01$_{\pm 1,61}$	62,00$_{\pm 0,69}$	59,10$_{\pm 2,89}$	**63,63$_{\pm 1,68}$**	62,12$_{\pm 0,60}$
zoom$_{120}$	77,92$_{\pm 1,39}$	**84,09$_{\pm 2,03}$**	78,98$_{\pm 2,07}$	77,14$_{\pm 1,23}$	80,40$_{\pm 0,61}$
zoom$_{140}$	45,08$_{\pm 2,50}$	**51,81$_{\pm 3,21}$**	46,74$_{\pm 2,57}$	44,22$_{\pm 1,59}$	48,41$_{\pm 3,74}$
zoom$_{60}$	76,24$_{\pm 0,23}$	78,16$_{\pm 0,68}$	75,94$_{\pm 1,42}$	78,64$_{\pm 1,17}$	**78,65$_{\pm 0,21}$**
zoom$_{80}$	90,78$_{\pm 0,11}$	91,65$_{\pm 0,74}$	91,05$_{\pm 0,39}$	**92,44$_{\pm 0,74}$**	92,09$_{\pm 0,43}$

TABLEAU B.4 Expériences OOD pour l'architecture ResNet-50 sur CIFAR-10.

Transformation	**SGD**	**SAM**	**ASAM**	**SAMSON$_2$**	**SAMSON$_\infty$**
rotation$_{20}$	85,46$_{\pm 0,06}$	84,79$_{\pm 0,52}$	84,40$_{\pm 1,60}$	**86,30$_{\pm 0,94}$**	85,78$_{\pm 0,83}$
rotation$_{40}$	**65,45$_{\pm 1,47}$**	61,05$_{\pm 1,20}$	61,83$_{\pm 2,18}$	64,15$_{\pm 1,18}$	64,83$_{\pm 1,30}$
cisaillement$_{28,6}$	**86,54$_{\pm 0,54}$**	84,95$_{\pm 0,47}$	84,67$_{\pm 1,06}$	86,40$_{\pm 0,80}$	86,44$_{\pm 0,58}$
cisaillement$_{57,3}$	61,63$_{\pm 1,20}$	58,37$_{\pm 1,94}$	59,97$_{\pm 0,90}$	**62,33$_{\pm 0,54}$**	62,24$_{\pm 0,68}$
zoom$_{120}$	77,05$_{\pm 4,03}$	70,94$_{\pm 1,78}$	74,80$_{\pm 1,47}$	77,17$_{\pm 2,81}$	**80,09$_{\pm 4,54}$**
zoom$_{140}$	**51,10$_{\pm 7,88}$**	38,27$_{\pm 0,99}$	42,32$_{\pm 2,46}$	43,47$_{\pm 4,56}$	48,46$_{\pm 6,04}$
zoom$_{60}$	73,57$_{\pm 0,81}$	74,70$_{\pm 0,57}$	70,48$_{\pm 2,99}$	74,92$_{\pm 1,37}$	**76,30$_{\pm 0,23}$**
zoom$_{80}$	88,94$_{\pm 0,28}$	89,29$_{\pm 0,28}$	88,84$_{\pm 0,74}$	89,62$_{\pm 0,80}$	**90,56$_{\pm 0,14}$**

Tableau B.5 Expériences OOD pour l'architecture VGG-13 sur CIFAR-10.

Transformation	**SGD**	**SAM**	**ASAM**	**SAMSON$_2$**	**SAMSON$_\infty$**
rotation$_{20}$	$85{,}00_{\pm 0{,}69}$	$86{,}56_{\pm 0{,}28}$	$86{,}81_{\pm 0{,}08}$	$86{,}81_{\pm 0{,}21}$	**$87{,}19_{\pm 0{,}36}$**
rotation$_{40}$	$64{,}46_{\pm 1{,}71}$	$67{,}05_{\pm 0{,}52}$	**$67{,}06_{\pm 0{,}90}$**	$66{,}83_{\pm 0{,}93}$	$66{,}41_{\pm 0{,}73}$
cisaillement$_{28,6}$	$84{,}52_{\pm 0{,}40}$	$86{,}49_{\pm 0{,}46}$	**$87{,}14_{\pm 0{,}85}$**	$86{,}66_{\pm 0{,}32}$	$86{,}61_{\pm 0{,}34}$
cisaillement$_{57,3}$	$58{,}56_{\pm 0{,}43}$	$59{,}18_{\pm 0{,}36}$	**$61{,}14_{\pm 1{,}25}$**	$58{,}76_{\pm 0{,}30}$	$59{,}33_{\pm 0{,}84}$
zoom$_{120}$	$84{,}12_{\pm 0{,}09}$	$85{,}73_{\pm 1{,}46}$	$84{,}58_{\pm 0{,}69}$	**$86{,}46_{\pm 1{,}48}$**	**$86{,}46_{\pm 1{,}53}$**
zoom$_{140}$	$58{,}97_{\pm 0{,}49}$	$61{,}31_{\pm 2{,}20}$	$59{,}15_{\pm 1{,}62}$	**$61{,}58_{\pm 1{,}83}$**	**$61{,}58_{\pm 2{,}35}$**
zoom$_{60}$	$72{,}11_{\pm 0{,}61}$	**$74{,}34_{\pm 2{,}18}$**	$72{,}65_{\pm 0{,}80}$	$72{,}92_{\pm 0{,}56}$	$72{,}92_{\pm 0{,}91}$
zoom$_{80}$	$88{,}37_{\pm 0{,}37}$	$89{,}66_{\pm 0{,}03}$	$89{,}66_{\pm 0{,}35}$	**$90{,}06_{\pm 0{,}23}$**	**$90{,}06_{\pm 0{,}15}$**

Tableau B.6 Expériences OOD pour l'architecture DenseNet-40 sur CIFAR-100.

Transformation	**SGD**	**SAM**	**ASAM**	**SAMSON$_2$**	**SAMSON$_\infty$**
rotation$_{20}$	$49{,}09_{\pm 0{,}29}$	$49{,}74_{\pm 0{,}55}$	$50{,}19_{\pm 0{,}52}$	**$50{,}51_{\pm 1{,}40}$**	$50{,}11_{\pm 0{,}28}$
rotation$_{40}$	$31{,}89_{\pm 0{,}18}$	$33{,}08_{\pm 0{,}27}$	$32{,}74_{\pm 0{,}15}$	**$33{,}23_{\pm 1{,}46}$**	$32{,}63_{\pm 0{,}84}$
cisaillement$_{28,6}$	$52{,}59_{\pm 0{,}27}$	$53{,}94_{\pm 0{,}35}$	$54{,}19_{\pm 0{,}30}$	**$55{,}36_{\pm 0{,}98}$**	$54{,}22_{\pm 0{,}53}$
cisaillement$_{57,3}$	$34{,}60_{\pm 1{,}22}$	$35{,}91_{\pm 0{,}26}$	$36{,}00_{\pm 0{,}33}$	$36{,}00_{\pm 1{,}03}$	**$36{,}50_{\pm 0{,}42}$**
zoom$_{120}$	$41{,}99_{\pm 1{,}71}$	$41{,}52_{\pm 0{,}93}$	**$42{,}55_{\pm 1{,}90}$**	$41{,}91_{\pm 0{,}66}$	$42{,}11_{\pm 0{,}63}$
zoom$_{140}$	**$16{,}93_{\pm 0{,}36}$**	$15{,}24_{\pm 0{,}17}$	$15{,}82_{\pm 0{,}70}$	$15{,}82_{\pm 0{,}69}$	$15{,}96_{\pm 1{,}04}$
zoom$_{60}$	$37{,}85_{\pm 2{,}06}$	$38{,}46_{\pm 2{,}64}$	$39{,}28_{\pm 1{,}06}$	**$40{,}67_{\pm 0{,}68}$**	$39{,}24_{\pm 0{,}91}$
zoom$_{80}$	$58{,}49_{\pm 0{,}55}$	$59{,}35_{\pm 0{,}22}$	$60{,}01_{\pm 0{,}14}$	**$60{,}66_{\pm 0{,}37}$**	$60{,}40_{\pm 0{,}40}$

Tableau B.7 Expériences OOD pour l'architecture MobileNetV2 sur CIFAR-100.

Transformation	**SGD**	**SAM**	**ASAM**	**SAMSON$_2$**	**SAMSON$_\infty$**
rotation$_{20}$	$62{,}28_{\pm 0{,}63}$	$62{,}76_{\pm 0{,}08}$	$63{,}47_{\pm 0{,}40}$	**$64{,}59_{\pm 0{,}52}$**	$64{,}09_{\pm 0{,}32}$
rotation$_{40}$	$42{,}97_{\pm 0{,}36}$	$42{,}95_{\pm 0{,}18}$	$44{,}09_{\pm 0{,}46}$	**$44{,}31_{\pm 0{,}11}$**	$44{,}28_{\pm 0{,}37}$
cisaillement$_{28,6}$	$61{,}50_{\pm 0{,}43}$	$64{,}42_{\pm 0{,}59}$	$64{,}88_{\pm 0{,}29}$	**$65{,}01_{\pm 0{,}37}$**	$64{,}72_{\pm 0{,}09}$
cisaillement$_{57,3}$	$42{,}13_{\pm 0{,}88}$	$42{,}80_{\pm 1{,}70}$	$42{,}62_{\pm 0{,}48}$	**$43{,}89_{\pm 0{,}76}$**	$42{,}98_{\pm 0{,}09}$
zoom$_{120}$	$54{,}60_{\pm 0{,}99}$	$53{,}32_{\pm 3{,}61}$	**$60{,}53_{\pm 1{,}21}$**	$56{,}40_{\pm 1{,}53}$	$55{,}13_{\pm 2{,}88}$
zoom$_{140}$	$26{,}69_{\pm 1{,}36}$	$24{,}88_{\pm 2{,}40}$	**$34{,}06_{\pm 2{,}05}$**	$27{,}29_{\pm 2{,}28}$	$27{,}46_{\pm 2{,}83}$
zoom$_{60}$	$45{,}51_{\pm 2{,}21}$	$43{,}48_{\pm 0{,}59}$	$42{,}58_{\pm 2{,}23}$	$42{,}89_{\pm 2{,}65}$	**$46{,}33_{\pm 0{,}23}$**
zoom$_{80}$	$66{,}62_{\pm 0{,}36}$	$67{,}85_{\pm 0{,}18}$	$69{,}30_{\pm 0{,}39}$	$68{,}93_{\pm 0{,}55}$	**$69{,}65_{\pm 0{,}29}$**

TABLEAU B.8 Expériences OOD pour l'architecture ResNet-34 sur CIFAR-100.

Transformation	**SGD**	**SAM**	**ASAM**	**SAMSON$_2$**	**SAMSON$_\infty$**
rotation$_{20}$	$62{,}62_{\pm 0{,}63}$	$64{,}40_{\pm 0{,}22}$	$65{,}34_{\pm 0{,}45}$	$65{,}17_{\pm 0{,}50}$	$\mathbf{66{,}01_{\pm 0{,}57}}$
rotation$_{40}$	$43{,}59_{\pm 0{,}14}$	$44{,}03_{\pm 1{,}09}$	$45{,}49_{\pm 0{,}24}$	$45{,}32_{\pm 0{,}30}$	$\mathbf{45{,}77_{\pm 0{,}64}}$
cisaillement$_{28,6}$	$62{,}63_{\pm 0{,}70}$	$65{,}03_{\pm 0{,}53}$	$65{,}44_{\pm 1{,}03}$	$66{,}06_{\pm 0{,}29}$	$\mathbf{66{,}20_{\pm 0{,}38}}$
cisaillement$_{57,3}$	$40{,}27_{\pm 1{,}15}$	$40{,}81_{\pm 0{,}74}$	$43{,}63_{\pm 0{,}57}$	$\mathbf{44{,}79_{\pm 0{,}59}}$	$43{,}56_{\pm 0{,}39}$
zoom$_{120}$	$59{,}78_{\pm 0{,}92}$	$61{,}12_{\pm 2{,}42}$	$62{,}42_{\pm 0{,}44}$	$\mathbf{64{,}59_{\pm 2{,}41}}$	$60{,}50_{\pm 1{,}19}$
zoom$_{140}$	$37{,}58_{\pm 2{,}85}$	$37{,}17_{\pm 3{,}68}$	$38{,}50_{\pm 0{,}80}$	$\mathbf{42{,}03_{\pm 4{,}63}}$	$35{,}37_{\pm 2{,}76}$
zoom$_{60}$	$43{,}14_{\pm 1{,}07}$	$45{,}32_{\pm 0{,}97}$	$46{,}27_{\pm 0{,}42}$	$\mathbf{47{,}24_{\pm 2{,}00}}$	$46{,}00_{\pm 1{,}62}$
zoom$_{80}$	$66{,}09_{\pm 0{,}71}$	$68{,}34_{\pm 0{,}67}$	$\mathbf{70{,}22_{\pm 0{,}56}}$	$70{,}10_{\pm 0{,}84}$	$69{,}57_{\pm 0{,}35}$

TABLEAU B.9 Expériences OOD pour l'architecture ResNet-50 sur CIFAR-100.

Transformation	**SGD**	**SAM**	**ASAM**	**SAMSON$_2$**	**SAMSON$_\infty$**
rotation$_{20}$	$59{,}44_{\pm 0{,}80}$	$60{,}36_{\pm 0{,}70}$	$60{,}37_{\pm 1{,}46}$	$\mathbf{64{,}53_{\pm 1{,}58}}$	$61{,}84_{\pm 0{,}57}$
rotation$_{40}$	$40{,}10_{\pm 1{,}53}$	$40{,}47_{\pm 1{,}21}$	$40{,}45_{\pm 1{,}61}$	$\mathbf{44{,}73_{\pm 1{,}44}}$	$42{,}80_{\pm 0{,}50}$
cisaillement$_{28,6}$	$59{,}70_{\pm 1{,}87}$	$62{,}47_{\pm 0{,}68}$	$62{,}71_{\pm 1{,}01}$	$\mathbf{65{,}63_{\pm 0{,}27}}$	$63{,}30_{\pm 1{,}05}$
cisaillement$_{57,3}$	$38{,}96_{\pm 1{,}05}$	$40{,}56_{\pm 0{,}79}$	$42{,}45_{\pm 0{,}27}$	$\mathbf{45{,}57_{\pm 1{,}82}}$	$41{,}57_{\pm 1{,}72}$
zoom$_{120}$	$53{,}62_{\pm 2{,}09}$	$57{,}13_{\pm 1{,}49}$	$54{,}54_{\pm 5{,}03}$	$\mathbf{57{,}97_{\pm 0{,}00}}$	$57{,}05_{\pm 1{,}07}$
zoom$_{140}$	$27{,}24_{\pm 3{,}47}$	$\mathbf{29{,}51_{\pm 0{,}75}}$	$26{,}04_{\pm 6{,}95}$	$29{,}35_{\pm 0{,}95}$	$29{,}21_{\pm 0{,}86}$
zoom$_{60}$	$40{,}36_{\pm 3{,}12}$	$45{,}47_{\pm 1{,}60}$	$45{,}19_{\pm 2{,}65}$	$\mathbf{47{,}96_{\pm 1{,}16}}$	$44{,}71_{\pm 1{,}99}$
zoom$_{80}$	$63{,}45_{\pm 2{,}57}$	$66{,}28_{\pm 0{,}32}$	$67{,}63_{\pm 1{,}65}$	$\mathbf{69{,}77_{\pm 0{,}33}}$	$67{,}78_{\pm 1{,}91}$

TABLEAU B.10 Expériences OOD pour l'architecture VGG-13 sur CIFAR-100.

Transformation	**SGD**	**SAM**	**ASAM**	**SAMSON$_2$**	**SAMSON$_\infty$**
rotation$_{20}$	$56{,}80_{\pm 0{,}45}$	$58{,}52_{\pm 0{,}86}$	$59{,}77_{\pm 0{,}36}$	$\mathbf{59{,}87_{\pm 0{,}30}}$	$59{,}85_{\pm 0{,}43}$
rotation$_{40}$	$38{,}53_{\pm 0{,}71}$	$40{,}14_{\pm 0{,}25}$	$\mathbf{40{,}68_{\pm 0{,}40}}$	$40{,}45_{\pm 0{,}43}$	$40{,}30_{\pm 0{,}47}$
cisaillement$_{28,6}$	$58{,}88_{\pm 0{,}08}$	$60{,}45_{\pm 0{,}45}$	$60{,}68_{\pm 0{,}15}$	$\mathbf{60{,}97_{\pm 0{,}48}}$	$60{,}70_{\pm 0{,}21}$
cisaillement$_{57,3}$	$36{,}70_{\pm 0{,}24}$	$38{,}53_{\pm 0{,}47}$	$38{,}29_{\pm 0{,}39}$	$38{,}58_{\pm 0{,}27}$	$\mathbf{39{,}85_{\pm 1{,}23}}$
zoom$_{120}$	$56{,}94_{\pm 0{,}76}$	$60{,}30_{\pm 1{,}63}$	$\mathbf{61{,}03_{\pm 0{,}12}}$	$59{,}71_{\pm 0{,}49}$	$59{,}06_{\pm 1{,}36}$
zoom$_{140}$	$33{,}18_{\pm 1{,}00}$	$\mathbf{37{,}00_{\pm 2{,}70}}$	$35{,}98_{\pm 1{,}00}$	$35{,}09_{\pm 0{,}23}$	$34{,}04_{\pm 1{,}31}$
zoom$_{60}$	$35{,}52_{\pm 0{,}42}$	$38{,}30_{\pm 0{,}15}$	$39{,}10_{\pm 1{,}14}$	$40{,}72_{\pm 0{,}71}$	$\mathbf{41{,}33_{\pm 1{,}23}}$
zoom$_{80}$	$64{,}11_{\pm 0{,}85}$	$65{,}33_{\pm 0{,}47}$	$66{,}28_{\pm 0{,}14}$	$66{,}03_{\pm 0{,}12}$	$\mathbf{66{,}48_{\pm 0{,}30}}$

ANNEXE C DÉTAILS SUR L'ENTRAÎNEMENT DE L'ESTIMATEUR POUR LE NAS

Prétraitement du jeu de données récolté

Le jeu de données récolté est présenté de façon tabulaire et contient le tuple $(\boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{O}})$ avec $\boldsymbol{\mathcal{X}} = \{s_1, .., s_{3\,200\,000}\}$ les configurations aléatoires du super-réseau et $\boldsymbol{\mathcal{O}}$ le tuple précision, puissance $(\tilde{A}(s, n_a), \tilde{P}(s, n_p))$ correspondant obtenu avec une simulation de Monte-Carlo sur respectivement n_a et n_p mini-lots.

Un pré-traitement est appliqué sur les estimés de puissance $\tilde{P}$ de $\boldsymbol{\mathcal{O}}$ pour mettre à l'échelle les valeurs observées dans l'intervalle $[0, 1]$, le domaine naturel des estimés de précision $\tilde{A}$. Ainsi, ces deux éléments ont le même poids lors de l'entraînement du DNN. La formule appliquée suit la forme

$$\tilde{P}_{[0,1]} = \frac{\tilde{P} - \tilde{P}_{\min}}{\tilde{P}_{\max} - \tilde{P}_{\min}}\,, \tag{C.1}$$

avec $\tilde{P}$ la valeur initiale, $\tilde{P}_{\min}$ et $\tilde{P}_{\max}$ les valeurs minimales et maximales observées dans le jeu de données. Le jeu de données est présenté sous n mini-lots S de taille $m = 256$ au DNN estimateur. 20% du jeu de donnée est réservé pour estimer la performance de généralisation lors de l'entraînement de l'estimateur.

Architecture du DNN estimateur

Le DNN estimateur est composé de trois couches linéaires avec biais produisant une sortie de 400 éléments, avec une opération ReLU entre chaque, ainsi que une ultime couche linéaire avec biais effectuant la prédiction pour la précision et pour la puissance simultanément. En entrée, ce réseau traite les architectures représentées par le vecteur s. Ce petit réseau nous permet d'évaluer un grand nombre d'architectures s en parallèle en utilisant une taille de lot appropriée (soit 1024 pendant les itérations de l'algorithme génétique). Nous représentons par la notation $f(s)_A$ et $f(s)_P$ la prédiction du DNN estimateur pour la précision et la puissance.

Paramètres de SGD

L'entraînement utilise la fonction de perte d'erreur quadratique moyenne $\mathcal{L}_{\text{MSE}}(\tilde{A}(S, n_a), f(S)_A, \tilde{P}_{[0,1]}(S, n_p), f(S)_P)$ définie comme suit :

$$\frac{1}{n} \sum_{S \in S_{\text{train}}} \frac{1}{m} (\tilde{A}(S, n_a) - f(S)_A)^2 + (\tilde{P}_{[0,1]}(S, n_p) - f(S)_P)^2 \,, \tag{C.2}$$

avec $\tilde{A}(S, n_a)$ et $\tilde{P}(S, n_p)$ les estimations de précision et de puissance avec la simulation de Monte-Carlo sur n_a et n_p pour le mini-lot S.

La descente de gradient est effectuée avec SGD pendant 200 époques, avec le taux d'apprentissage $\iota = 0{,}01$ et le moment $\beta = 0{,}9$. Le taux d'apprentissage est réduit d'un facteur 10 à la détection d'un plateau sur la fonction de perte, avec les paramètres par défaut de la librairie Pytorch [1]. L'utilité du DNN obtenu est jugée sur pièce selon les résultats de l'algorithme génétique.

1. Documentation disponible : `https://pytorch.org/docs/stable/generated/torch.optim.lr_scheduler.ReduceLROnPlateau.html`.