HAL open science

# Quantization and adversarial robustness of embedded deep neural networks

Thibault Allenet

**To cite this version:**

Thibault Allenet. Quantization and adversarial robustness of embedded deep neural networks. Machine Learning [cs.LG]. Université de Rennes, 2023. English. NNT : 2023URENS003 . tel-04136202

**HAL Id: tel-04136202**

**https://theses.hal.science/tel-04136202v1**

Submitted on 21 Jun 2023

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

COLLEGE DOCTORAL BRETAGNE | MATHS, TELECOMS INFORMATIQUE, SIGNAL SYSTEMES, ELECTRONIQUE

Université de Rennes

# THESE DE DOCTORAT DE

**L'UNIVERSITE DE RENNES**

ECOLE DOCTORALE N° 601
*Mathématiques, Télécommunications, Informatique, Signal, Systèmes, Electronique*
Spécialité : *Informatique*

Par
**Thibault ALLENET**

# Quantization and Adversarial Robustness of Embedded Deep Neural Networks

**Thèse présentée et soutenue à DIGITEO Saclay – CEA LIST, le 24 mars 2023**
**Unité de recherche : IRISA/INRIA & CEA-LIST**

**Rapporteurs avant soutenance** :

| | |
|---|---|
| Fan YANG | Professeur – Université de Bourgogne |
| Kevin BAILLY | Maître de conférences – Sorbonne Université |

**Composition du Jury :**

| | | |
|---|---|---|
| Président : | Alberto BOSIO | Professeur – Ecole Centrale de Lyon, INL |
| Examinateurs : | Pierre-Alain MOELLIC | Ingénieur-chercheur – CEA-LETI |
| | Fan YANG | Professeur – Université de Bourgogne |
| | Kevin BAILLY | Maître de conférences – Sorbonne Université |
| Dir. de thèse : | Olivier SENTIEYS | Professeur - Université de Rennes, IRISA, Rennes, France |
| Co-dir. de thèse : | Olivier BICHLER | Ingénieur Chercheur – CEA-LIST |

**Invité(s) :**

| | |
|---|---|
| David BRIAND | PhotoRoom |

# Remerciements

Je tiens à exprimer ma sincère gratitude à toutes les personnes qui ont contribué à mon parcours de thèse et à son succès.

Tout d'abord, je suis profondément reconnaissant envers mon directeur de thèse, Olivier SENTIEYS, et mes encadrants, Olivier BICHLER et David BRIAND, pour leur mentorat, leur patience, et leur motivation tout au long de ma formation à la recherche. Leur expertise, leurs commentaires et leur critique constructive m'ont poussé à m'améliorer techniquement et à gagner en maturité sur le travail de recherche.

Je tiens également à exprimer mon appréciation aux rapporteurs et membres de mon jury de doctorat, Fan YANG, Kevin BAILLY, Pierre-Alain MOELLIC, Alberto BOSIO pour leur examen critique de ma thèse et leurs suggestions avisées.

Je suis également reconnaissant envers mes collègues de laboratoire. Ces années passées à vos côtés ont cultivé des amitiés que je chéris. Merci à Guillaume pour avoir partagé ses connaissances et son expérience. Merci à Inna pour sa relecture du manucrit et son soutien au moment de la rédaction. Merci à Vincent L. pour ses critiques constructives et ses perspectives diverses. Merci à Clément, Cyril, Gabriel, Guillaume, Inna, Jason, Johannes, Pierre-Guillaume, Quentin, Vincent L., Vincent T. et Thibaut pour les discussions à l'espace café, pour la bonne ambiance quotidienne et pour les soirées.

Merci à mes parents et à mes soeurs, pour leur amour, leur écoute et leur soutien. Enfin, mais surtout, je voudrais exprimer ma profonde gratitude envers ma femme, My, pour son amour, son écoute, sa patience, ses encouragements et son soutien indéfectible tout au long de mon parcours de doctorat.

# Abstract

Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) have been broadly used in many fields such as computer vision, natural language processing and signal processing. Nevertheless, the computational workload and the heavy memory bandwidth involved in deep neural networks inference often prevents their deployment on low-power embedded devices. Moreover, deep neural networks vulnerability towards small input perturbations questions their deployment for applications involving high criticality decisions. This PhD research project objective is twofold. On the one hand, it proposes compression methods to make deep neural networks more suitable for embedded systems with low computing resources and memory requirements. On the other hand, it proposes a new strategy to make deep neural networks more robust towards attacks based on crafted inputs with the perspective to infer on edge.

We begin by introducing common concepts for training neural networks, convolutional neural networks, recurrent neural networks and review the state of the art neural on deep neural networks compression methods. After this literature review we present two main contributions on compressing deep neural networks: an investigation of lottery tickets on RNNs and Disentangled Loss Quantization Aware Training (DL-QAT) on CNNs. The investigation of lottery tickets on RNNs analyze the convergence of RNNs and study its impact when subject to pruning on image classification and language modelling. Then we present a pre-processing method based on data sub-sampling that enables faster convergence of LSTM while preserving application performance. With the Disentangled Loss Quantization Aware Training (DL-QAT) method, we propose to further improve an advanced quantization method with quantization friendly loss functions to reach low bit settings like binary parameters where the application performance is the most impacted. Experiments on ImageNet-1k with DL-QAT show improvements by nearly 1% on the top-1 accuracy of ResNet-18 with binary weights and 2-bit activations, and also show the best profile of memory footprint over accuracy when compared with other state-of-the art methods.

This work then studies neural networks robustness toward adversarial attacks. After introducing the state of the art on adversarial attacks and defense mechanisms, we propose the Ensemble Hash Defense (EHD) defense mechanism. EHD enables better resilience to adversarial attacks based on gradient approximation while preserving application

performance and only requiring a memory overhead at inference time. In the best configuration, our system achieves significant robustness gains compared to baseline models and a loss function-driven approach. Moreover, the principle of EHD makes it complementary to other robust optimization methods that would further enhance the robustness of the final system and compression methods. With the perspective of edge inference, the memory overhead introduced by EHD can be reduced with quantization or weight sharing.

The contributions in this thesis have concerned optimization methods and a defense system to solve an important challenge, that is, how to make deep neural networks more robust towards adversarial attacks and easier to be deployed on the resource limited platforms. This work further reduces the gap between state of the art deep neural networks and their execution on edge devices.

# Résumé substantiel en français

L'attention portée à l'intelligence artificielle et à l'apprentissage automatique s'est amplement accrue au cours de la dernière décennie. L'apprentissage automatique, désigne un ensemble d'algorithmes qui apprennent et s'adaptent sans suivre d'instructions explicites en analysant et en tirant des conclusions à partir de structures dans les données. La capacité à produire et à stocker des données a considérablement augmenté au cours des dernières décennies et, avec elle, le succès de l'apprentissage automatique. Parmis les algorithmes d'apprentissage automatique, les réseaux de neurones profond apportent les meilleures performances de modélisation.

Les réseaux de neurones convolutifs et les réseaux neurones récurrents ont été largement utilisés dans de nombreux domaines tels que la vision par ordinateur, le traitement naturel du langage et le traitement du signal. Néanmoins, la charge de calcul et le besoin en bande passante mémoire impliqués dans l'inférence des réseaux de neurones profonds empêchent souvent leur déploiement sur des cibles embarquées à faible ressources. De plus, la vulnérabilité des réseaux de neurones profonds à de petites perturbations sur les entrées remet en question leur déploiement pour des applications impliquant des décisions de haute criticité. Pour relever ces défis, cette thèse propose deux principales contributions. D'une part, nous proposons des méthodes de compression pour rendre les réseaux de neurones profonds plus adaptés aux systèmes embarqués ayant de faibles ressources. D'autre part, nous proposons une nouvelle stratégie pour rendre les réseaux de neurones profonds plus robustes aux attaques adverses en tenant compte des ressources limitées des systèmes embarqués. Le manuscrit est organisé de la façon suivante.

## Apprentissage profond et état de l'art

Dans un premier temps, nous présentons des principes et des outils de bases de l' apprentissage profond et nous fournissons un état de l'art sur des méthodes de compressions de réseaux de neurones. Nous introduisons des concepts de base pour l'apprentissage des réseaux de neurones comme le principe de généralisation, la rétropropagation, les fonctions de coût, l'apprentissage par transfert, la tendance principale de l'apprentissage profond et certaines limitations. Nous introduisons trois types de réseaux de neurones reconnus

: les perceptrons à plusieurs couches, les réseaux de neurones convolutifs et les réseaux de neurones récurrents. Nous passons en revue quelques méthodes de compression des réseaux de neurones profonds de l'état de l'art.

## Compression de réseaux de neurones

Ensuite, nous présentons deux contributions autour de la compression des réseaux de neurones profonds.

- Une étude de transferabilité d'une technique avancée de pruning, le *lottery ticket* [25], originalement appliqué aux perceptrons à plusieurs couches et aux réseaux de neurones convolutifs dans l'objectif d'être appliqué aux réseaux de neurones récurrents.
- Une méthode de quantification avancée des réseaux de neurones convolutifs, *Disentangled Loss Quantization Aware Training* (DL-QAT).

### Compression de réseaux de neurones récurrents

Cette étude de transferabilité du *lottery ticket* sur les RNN analyse la convergence des RNN et étudie son impact sur l'élagage *(pruning)* pour des taches de classification d'images et de modélisation du langage. Frankle *et al.* [25] ont étudié la corrélation entre l'initialisation et le pruning des réseaux de neurones non récurrent. Plus précisément, l'hypothèse du *lottery ticket* conjecture que les réseaux de neurones contiennent de petits sous-réseaux qui peuvent être réentrainés à partir de la même initialisation et obtenir une précision similaire en un nombre proportionnel d'étapes d'entrainement.

Nous utilisons leur méthode comme outil pour étudier la convergence des réseaux de neurones récurrents. Nous étendons leur cadre expérimental en évaluant leur algorithme sur des Recurrent Neural Networks (RNN) avec des tâches de classification d'images et de modélisation du langage. En comparaison avec les travaux originaux sur MultiLayer Perceptrons (MLP) et Convolutional Neural Networks (CNN), il apparaît que le profil de convergence de RNN est instable. En couplant l'analyse de la convergence avec plusieurs configurations pertinentes de la méthode, nous avons observé que, pour trouver un *lottery ticket* sur une architecture récurrente, le réapprentissage doit être initialisé avec un état de réseau dense à partir duquel le profil de la fonction de coût converge sans instabilités.

De plus, nous proposons une nouvelle méthode de prétraitement basée sur le sous-échantillonnage des données qui permet une convergence plus rapide des LSTM tout en préservant les performances applicatives. La figure 1 montre que le LSTM entrainé avec la méthode proposée converge vers une asymptote à l'itération 20k, soit la moitié du nombre d'itérations nécessaires pour l'apprentissage sans la méthode de sous-échantillonnage, et les performances applicatives sont préservées.

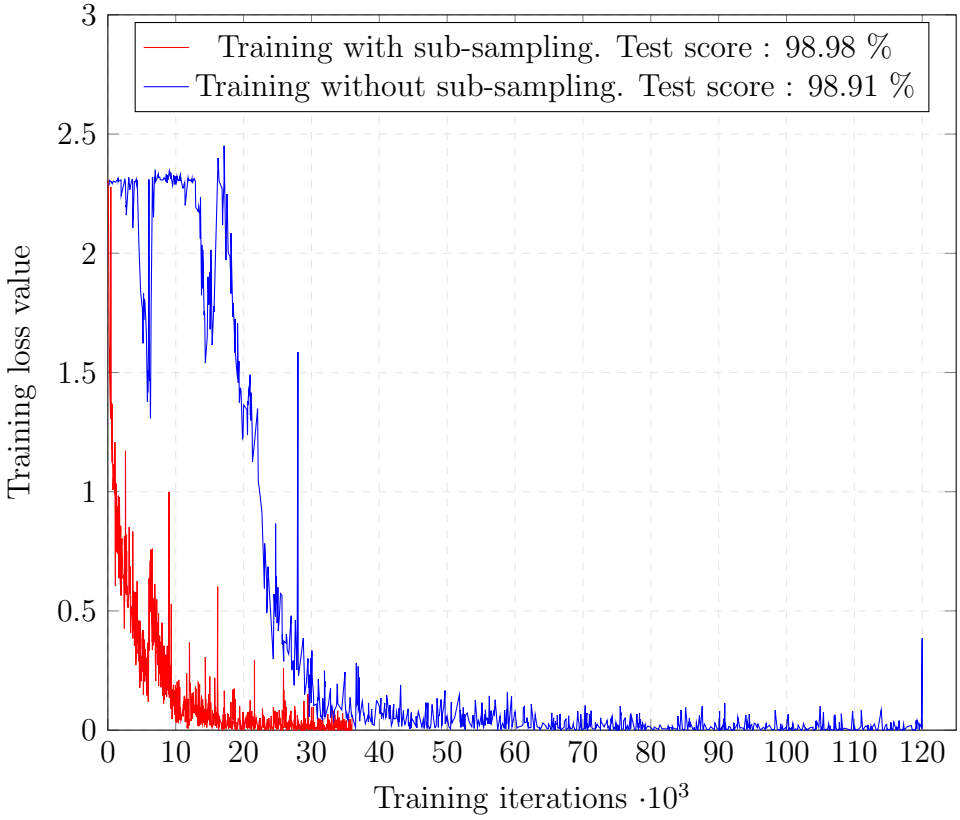



Figure 1: Courbes de convergence des LSTM sur la tache du MNIST séquentiel avec et sans la méthode de sous-échantillonnage proposée. La méthode de sous-échantillonnage proposée permet une convergence plus rapide des LSTM tout en préservant les performances de l'application.

Les méthodes de *pruning* des poids permettent d'obtenir un taux de compression de modèle significatif tout en préservant les performances applicatives. Une telle compression de modèle permet le stockage de nombreux modèles sur du matériel à mémoire limitée. Cependant, l'architecture creuse qui en résulte n'est pas structurée pour être facilement accélérée sur cible. Cela constitue leur principale limitation pour des applications critiques ayant des contraintes en latence et en faible consommation. Au regard de ces critères, la quantification est une solution élégante pour à la fois compresser le stockage mémoire du réseau et permettre une inférence optimisée avec des opérateurs de moindre précision au moment de l'inférence. Ce premier travail a motivé notre choix de poursuivre ce travail de recherche sur des méthodes de quantification de réseaux de neurones.

## Quantification de réseaux de neurones convolutifs

Nous proposons d'améliorer une méthode de l'état de l'art de quantification à l' apprentissage, la méthode *Scaled Adjust Training* (SAT) proposée par Jin *et al.* [48]. Comme toute les autres méthodes de quantification, elle repose sur la fonction de coût *cross-entropy*.

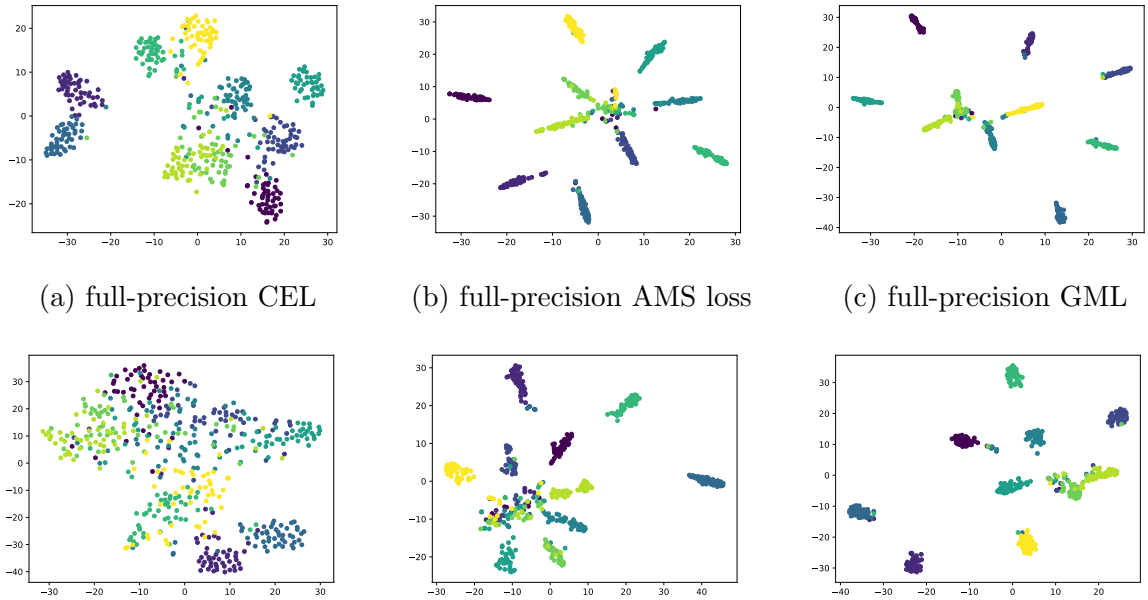
Nous proposons *Disentangled Loss Quantization Aware Training* (DL-QAT), une méthode utilisant des fonctions de coût favorables à la quantification pour quantifier des réseaux de neurones à l'apprentissage. L'algorithme 1 détaille l'implémentation de DL-QAT. Nous posons donc l'hypothèse que l'entraînement de ResNets-18 avec des fonctions de coûts encourageant à produire des caractéristiques discriminantes améliore la résiliance des ces modèles à la quantification.

**Algorithm 1** Disentangled Loss Quantization Aware Training

1: **Inputs:** a neural network $f$ and its FP32 parameters $W_{FP32}$, training data $x$ and its corresponding target $y$, the disentangled Loss $L$. Clamp() is Eq. (5.2), DoReFa is Eq. (5.3) and Eq. (5.4), Scaled-Adjust is Eq. (5.5).
2: **Outputs:** the quantized parameters $Q$ and the activation quantization learned parameters $\alpha$.
3: **DL-QAT**($f$,$W$,$x$,$y$,$L$):
4: Training I. FP32 clamped and scaled weights
5: Learn the network minimizing $L[f(x, Clamp(W_{FP32})), y]$
6: $W_{FP32} \leftarrow$ converged FP32 parameters of the first training
7: Training II. Quantized weight and activation
8: For each Quantization Aware Training iteration using input data $(x, y)$:
9: $W_{clamp} \leftarrow Clamp(W_{FP32})$
10: $Q \leftarrow DoReFa(W_{clamp})$
11: If No Batch Normalization:
12: $Q \leftarrow$ Scaled-Adjust($Q$)
13: $Out \leftarrow f(x, Q)$. *(Propagate and quant. the activations on the fly with PACT($\alpha$))*
14: $Error \leftarrow L(Out, y)$
15: Backpropagate the error. *(The quant. functions are approximated as detailed in [48])*
16: Update $W_{FP32}$ and $\alpha$ with SGD and their respective gradients: $\frac{\partial Error}{\partial W}$, $\frac{\partial Error}{\partial \alpha}$
17: **return** $Q$, $\alpha$

Afin de visualiser la contribution des fonctions de coût utilisées pour quantifier, *Additive Margin Softmax* (AMS) et *Gaussian Mixture Loss* (GML), la figure 2 montre les projections des sorties de ResNets-18 avec l'algorithme t-sne [107] à partir des données de test de **CIFAR-10**. En comparant la fonction de coût cross entropique (a & d) aux fonctions de coût AMS (b & e) et GML(c & f), il apparaît clairement que les caractériques apprises par AMS et GML sont plus discriminatives que les caractériques apprises par la CEL et d'autant plus pour les ResNets-18 quantifiés à faible précision.

Des résultats sur **ImageNet-1k** sont illustrés dans la figure 3 où l'axe $x$ correspond à l'empreinte mémoire du réseau et l'axe $y$ correspond au score de test top-1 sur ImageNet. L'empreinte mémoire d'un réseau neurone quantifié est approximée en additionnant le stockage total des paramètres associé à leur précision respective avec la mémoire nécessaire pour bufferiser l'activation de la plus grande taille. Pour obtenir des réseaux de neurones quantifiés à faible empreinte mémoire, les paramètres de quantification deviennent de plus en plus extrêmes et par conséquent les performances applicatives diminuent. En



(a) full-precision CEL (b) full-precision AMS loss (c) full-precision GML

(d) 2-bit W&A CEL (e) 2-bit W&A AMS loss (f) 2-bit W&A GML

Figure 2: Projection de la sortie d'un resnet-18 avec l'algorithme t-sne à partir des données de test de **CIFAR-10**. Les t-sne ont été exécutés sur 1000 itérations avec une perplexité de 30.

comparant (DL-QAT) à toutes les autres méthodes de l'état de l'art, il apparaît clairement que notre méthode fournit le meilleur profil du compromis entre l'empreinte mémoire et la performance applicative. De plus, notre méthode permet d'atteindre des paramètres binarisés.

Dans l'ensemble, nos expériences confirment notre hypothèse et encouragent l'utilisation et la recherche futures des pertes démêlées pour la formation consciente de la quantification. DL-QAT contribue à rendre l'inférence des réseaux de neurones convolutifs plus accessible sur cibles à faible ressources. Alors que de plus en plus d'applications basées sur les réseaux de neurones sont déployées, les CNN sont de plus en plus exposés aux menaces de sécurité telles que les attaques adverses. Comprendre la vulnérabilité de ces réseaux aux attaques adverses est plus que jamais un problème crucial.

## Robustesse des réseaux de neurones aux attaques adverses

Les réseaux de neurones sont vulnérables à de petites perturbations sur ses entrées. Cette vulnérabilité peut être exploitée par un attaquant pour compromettre l'intégrité du modèle et pour forcer le modèle à faire des prédictions erronées uniquement en perturbant les entrées. Pour y parvenir, un attaquant calcule des exemples adverses, comme introduit

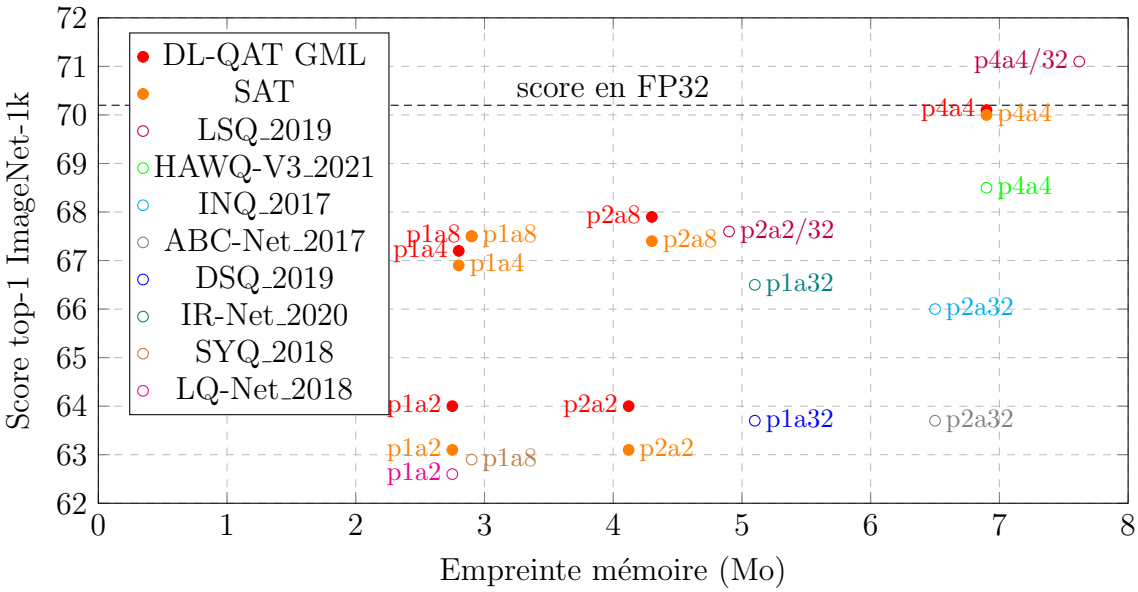



Figure 3: Comparaison de l'empreinte mémoire (en Mo) de réseaux quantifiés à différentes précisions. Chaque point est un réseau quantifié et est associé à la précision de ses poids (p) et de ses activations (a). Les ronds pleins sont les résultats issus de nos expériences tandis que les ronds vides sont les résultats issus de l'état de l'art.

par Szegedy *et al.* [103]. Le but de l'attaque est de trouver une petite variation de l'entrée qui soit imperceptible à la perception humaine mais qui trompe la prédiction du réseau de neurone. Prenons l'exemple de la figure 4, le modèle classe correctement l'image originale comme un panda. Puis en ajoutant une petite perturbation à l'image originale, une image adverse est générée, celle-ci étant identifié de manière érronée comme un gibbon. Bien que les différences entre l'image originale et l'image adverse soient indiscernables à l'œil humain, la petite perturbation est suffisante pour tromper la prédiction du modèle neuronal.

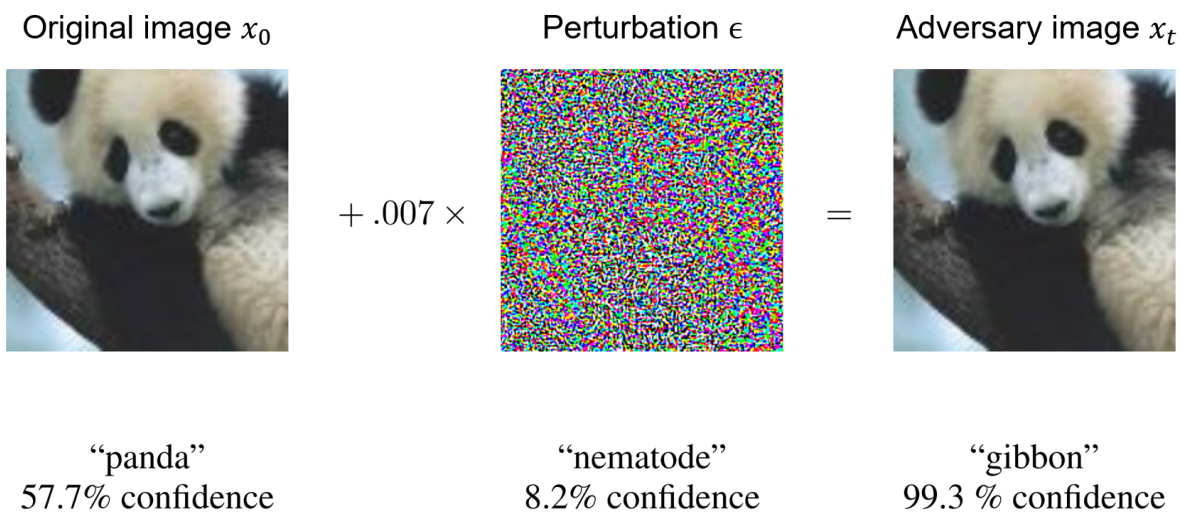



Figure 4: Exemple d'une image adverse. Source de l'illustration : [32]

Nous présentons un état de l'art sur les attaques adverses et les mécanismes de défense. Dans l'ensemble, les mécanismes de défense cherchent soit à augmenter la robustesse des modèles face aux attaques adverses grâce à des mécanismes d'apprentissage spécifiques, soit à détecter les attaques adverses. Ces mécanismes de défense présentent cependant deux limites principales :

- les gains de robustesse se font au détriment des performances applicatives [121] ou d'une surcharge en calcul et en mémoire lors de l'inférence [105, 78, 93, 108, 70]
- les mécanismes de défense restent plus ou moins vulnérables aux attaques par estimation de gradient comme SPSA [77].

Nous proposons alors le système de défense *Ensemble Hash Defense* (EHD) répondant à ces deux limitations. EHD contre les attaques par approximation par gradient avec un mécanisme de défense en deux étapes basé sur un ensemble de modèles et des fonctions de hachage cryptographiques.

**1. Entraînement**

Apprendre $k$ modèles. Chaque modèle doit avoir une performance applicative similaire.

**2. Inférence**

Lors de l'inférence, EHD combine une stratégie d'ensemble de modèles avec un processus de sélection de modèles d'inférence. A partir de l'image d'entrée, le processus de sélection choisit un modèle parmi les $k$ modèles pour inférer l'image d'entrée. La figure 5 illustre le processus d'inférence de EHD pour $k = 3$ modèles.

Nous présentons le sytème de défense EHD selon les points suivants :

- La formulation d'une hypothèse selon laquelle le système EHD bénéficie de la diversité de réponse des $k$ modèles impliqués.
- Les avantages et limitations identifiés d'EHD.
- Le détail du processus de sélection de modèle à l'inférence.
- Un exemple du comportement de la méthode face à une attaque SPSA.

Dans la meilleure configuration, Ensemble Hash Defense (EHD) atteint une résilience de 41% par rapport à une résilience de 10.0% avec une approche d'apprentissage robuste utilisant une fonction de coût Max-Mahalanobolis Center loss (MMC) [77]. Nous pensons que ce résultat n'est que le premier pas vers des systèmes plus robustes basés sur le concept simple introduit par notre méthode EHD. En ayant une approche pratique, lorsqu'il est

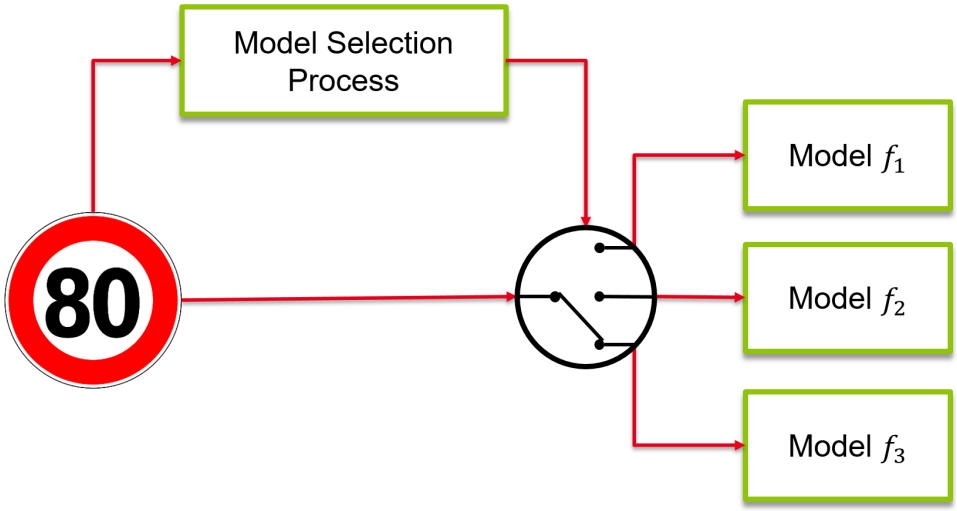



Figure 5: Principe du système de défense *Ensemble Hash Defense* proposé

envisagé d'utiliser des réseaux de neurones pour une application, il est necessaire de prendre en compte les risques d'attaques adverses et d'étudier si une méthode de défense existante permet de réduire la criticité résultante à un niveau acceptable.

En conclusion, les travaux de cette thèse ont proposé des méthodes de compression de réseaux de neurones et un système de défense pour résoudre des défis importants, à savoir comment rendre les réseaux de neurones profonds plus robustes face aux attaques adverses et plus faciles à déployer sur les plateformes à ressources limitées. Ces travaux réduisent davantage l'écart entre l'état de l'art des réseaux neurones profonds et leur exécution sur des cibles embarquées à faible ressources.

# Contents


**1 Introduction** **25**

1.1 Context . . . 25

1.2 Issues related to compression and vulnerability of DNNs . . . 28

1.3 Contributions . . . 28

1.4 Thesis outline . . . 29

**2 Deep Neural Networks** **31**

2.1 Common Concepts for Training Neural Networks . . . 32

2.1.1 Model, data, and generalization . . . 32

2.1.2 Loss function . . . 32

2.1.3 Initialization . . . 33

2.1.4 Training paradigms . . . 34

2.1.5 Data augmentation . . . 35

2.1.6 Supervised training of a neural network . . . 35

2.1.7 Transfer learning . . . 37

2.1.8 Increasing model size . . . 38

2.1.9 Exploding and vanishing gradients phenomena . . . 38

2.2 Multi-Layer Perceptrons . . . 39

2.2.1 Fully-connected layer . . . 39

2.2.2 Non-linear activation function . . . 40

2.2.3 Limitations . . . 41

2.3 Convolutional Neural Network . . . 41

2.3.1 Convolution layer . . . 41

2.3.2 Pooling layer . . . 42

2.3.3 Batch Normalization layer . . . 43

2.3.4 Resnets . . . 43

2.4 Recurrent Neural Networks . . . 46

2.4.1 Concept . . . 46

2.4.2 Backpropagation through time . . . 48

2.4.3 Gated mechanism - Long Short Term Memory . . . 49

2.5 Deep Neural Network compression . . . 51

2.5.1 Compression objective and data availability . . . 51
2.5.2 Pruning . . . 52
2.5.3 Quantization . . . 53
2.6 Conclusion . . . 61

**3 Deep Learning Tools 63**
3.1 Libraries . . . 63
3.1.1 Deep Learning Frameworks . . . 63
3.1.2 Libraries for deploying neural networks on the Edge . . . 65
3.1.3 N2D2 - Neural Network Design & Deployment . . . 66
3.2 Benchmark Datasets . . . 67
3.2.1 MNIST . . . 68
3.2.2 CIFAR-10 & CIFAR-100 . . . 68
3.2.3 ImageNet-1k . . . 69
3.2.4 Wikitext-2 . . . 69

**4 Compressing Recurrent Neural Networks 71**
4.1 Introduction . . . 71
4.2 Sparse and dense models . . . 72
4.3 The Lottery Ticket Hypothesis . . . 73
4.4 Experimental protocol . . . 73
4.5 Handwritten digits recognition . . . 76
4.5.1 Task and training setup. . . . 76
4.5.2 Convergence on Sequential MNIST . . . 76
4.5.3 Lottery ticket experiments . . . 78
4.5.4 Sub-sampling pre-processing . . . 80
4.6 Language Modelling . . . 83
4.6.1 Task and training setup . . . 83
4.6.2 Convergence on Wikitext-2 . . . 84
4.6.3 Lottery ticket experiments . . . 86
4.7 Discussion and Perspectives . . . 88

**5 Disentangled Loss for Low-Bit Quantization Aware Training 89**
5.1 Introduction . . . 89
5.2 Previous Work . . . 91
5.2.1 Quantization Aware Training . . . 91
5.2.2 Disentangled Losses . . . 93
5.3 Disentangled Loss Quantization Aware Training . . . 94
5.4 Experiments . . . 94
5.4.1 Training setups . . . 94

5.4.2 Results and analysis . . . 96
5.4.3 Discussion and Perspectives . . . 101

**6 Adversarial Robustness** **103**
6.1 Introduction . . . 103
6.2 Common concepts for adversarial attacks . . . 104
6.2.1 Distance metrics . . . 104
6.2.2 Attacker goals . . . 105
6.2.3 Attacker knowledge . . . 105
6.3 State of The Art . . . 105
6.3.1 White-box attacks . . . 105
6.3.2 Black-box attacks . . . 107
6.3.3 Defense mechanisms . . . 108
6.3.4 Current limitations . . . 109
6.4 Ensemble Hash Defense (EHD) . . . 110
6.4.1 Concept . . . 110
6.4.2 Diversity hypothesis . . . 110
6.4.3 Model selection process . . . 111
6.4.4 Defense example . . . 113
6.4.5 Advantages and limitations . . . 114
6.5 Experiments . . . 115
6.5.1 Evaluation setup . . . 115
6.5.2 First results . . . 116
6.5.3 EHD with different objective functions . . . 117
6.5.4 Influence of the number of models . . . 118
6.6 Discussion and perspectives . . . 120

**7 Conclusion** **123**
7.1 Summary of our results . . . 123
7.2 Potential improvements . . . 124
7.2.1 Mixed precision . . . 125
7.2.2 Transferability of adversarial examples . . . 125
7.3 Future research directions . . . 125
7.3.1 Neural architecture search for embedded applications . . . 125
7.3.2 Self-supervised learning . . . 126

**A Quantization and Adversarial Robustness** **127**
A.1 Previous work . . . 127
A.2 DL-QAT and Adversarial Robustness . . . 127

**B Simultaneous Perturbation Stochastic Approximation** **130**

**Acronyms** **133**

# List of Figures


1.1 Evolution of the number of publications in the AI field. . . . . 26

2.1 Cross Entropy Loss (CEL) function in a multi-classification task. . . . . 33
2.2 Main training paradigms: supervised learning, self-supervised learning, reinforcement learning. . . . . 34
2.3 Supervised training with the gradient descent method. . . . . 36
2.4 Transfer learning possible advantages: higher asymptote, faster convergence, higher starting performance. . . . . 38
2.5 Fully-connected layer with parameter matrix $\theta \in \mathbb{R}^{3\times n}$ and a non-linear activation function $\sigma$. It takes an input $x \in \mathbb{R}^n$ and returns an output $y \in \mathbb{R}^3$. Several fully-connected layers can be stacked to form a multi-layer perceptron. The last layer will not have an activation function. . . . . 40
2.6 An example of convolution using a $3 \times 3 \times 1$ kernel . . . . . 42
2.7 An example of max pooling and average pooling with kernel of size $2 \times 2$ applied with a stride of 2 . . . . . 43
2.8 Residual blocks proposed by He *et al.* [38]. Left: a residual building block traditionally used for ResNet-34 and shallower ResNets. Right: a "bottleneck" building block traditionally used for ResNets-50/101/152 . . . 44
2.9 The full pre-activation configuration of a residual block proposed by He *et al.* [39] . . . . . 45
2.10 The computational graph of a basic recurrent neural network that maps a sequence of input vectors $X = (x_1, ..., x_{t-1}, x_t, x_{t+1}, ...x_T)$ to a sequence of output probabilities $O = (o_1, ..., o_{t-1}, o_t, o_{t+1}, ...o_T)$. Equations (2.10) and (2.11) defines the forward propagation of this recurrent neural network. 47
2.11 The computational graph of a LSTM layer that maps a sequence of input vectors $X = (x_1, ..., x_t, ...x_T)$ to a sequence of output vectors $(out_1, ..., out_t, ..., out_T)$. Equations (2.12) to (2.17) define the forward propagation of the LSTM layer. Illustration from [28] . . . . . 49
2.12 Concept of pruning a deep neural network . . . . . 52
2.13 IEEE 754 single-precision format on 32 bits . . . . . 54
2.14 Custom fixed-point representation on 8 bits . . . . . 55

2.15 Integer representation on 8 bits . . . . . 56
2.16 Quantization points with uniform scalar quantization (left) and non-uniform scalar quantization in the case of logarithmic quantization (right). Real values in the continuous domain $r$ are mapped into discrete quantized values $q$, which are the red bullets. Note that the distances between the quantized values (quantization levels) are the same in uniform quantization, whereas they can vary in non-uniform quantization . . . . . 58
2.17 Example of one layer propagation and backpropagation during Quantization Aware Training. The Straight-Through-Estimator (STE) estimates the derivative of quantization functions. NQ stands for Non-Quantized values and refers to real values represented with the single-precision floating-point format. FQ stands for Fake Quantized values and refers to the finite set of values representing quantized levels in single-precision floating-point. $\eta$ is the learning rate used to perform the update. . . . . 60

3.1 Overview of the N2D2 Framework . . . . . 67
3.2 MNIST handwritten digits examples . . . . . 68
3.3 CIFAR-10 image examples . . . . . 69

4.1 Performance comparison of sparse and dense models. In (a) the performance of the convolutional neural networks are measured using the top-1 accuracy (the higher the better) on Imagenet. In (b) the performance of the LSTM-based models are measured using the perplexity (the lower the better) on Penn Treebank. Both experiments (a) and (b) show that sparse models outperform comparably-sized dense models. The figures are taken from [125]. 73
4.2 LSTM inference pipeline on Sequential MNIST. . . . . 76
4.3 LSTM training loss on sequential MNIST. The convergence is unstable. . . 77
4.4 Profile of LSTM performance under increasing sparsity, trained on sequential MNIST and following several lottery ticket configurations. Each point corresponds to a test accuracy reported in Table 4.3. . . . . 79
4.5 LSTM training losses on sequential MNIST when trained with and without the proposed sub-sampling method. . . . . 81
4.6 Profile of LSTM performance under increasing sparsity, trained with sub-sampling on sequential MNIST and following several lottery ticket configurations. Each point corresponds to a test accuracy reported in Table 4.6. . 82
4.7 LSTM inference pipeline on Wikitext-2. . . . . 83
4.8 LSTM training loss on Wikitext-2. . . . . 85
4.9 Profile of LSTM performance under increasing sparsity, trained on Wikitext-2 and following several lottery ticket configurations. Each point corresponds to a test accuracy reported in Table 4.3. . . . . 87

5.1 Scaled Adjust Training method. . . . . 92
5.2 Dimension reduction with the t-sne algorithm representing the input features of the linear classifier from **CIFAR-10** test data. The corresponding top-1 test accuracies are reported in Table 5.3. t-sne performed over 1000 iterations and a perplexity of 30. . . . . 97
5.3 Memory footprint (Mega-Bytes) comparison of different precision settings from Table 5.3. Each point is one quantized network associated to the bit precision of its weight and activation. . . . . 98
5.4 Memory footprint (Mega-Bytes) comparison of different precision settings from Table 5.4. Each point is one quantized network associated to the bit precision of its weight and activation. The filled round marks are results from our experiments while the empty round marks are results reported from the State of The Art. . . . . 101

6.1 Example of an adversary image. Illustration from [32] . . . . . 104
6.2 Ensemble Hash Defense principle . . . . . 111
6.3 Model selection process. . . . . 112
6.4 Ensemble Hash Defense targeted by SPSA attack. . . . . 113

B.1 Example of an adversary image. Illustration from [32] . . . . . 130

# List of Tables


4.1 Notations for the winning ticket algorithm. . . . . 75

4.2 LSTM performance on Sequential MNIST. . . . . 77

4.3 LSTM test accuracy on sequential MNIST following several lottery ticket configurations. . . . . 78

4.4 Sparsity per-layer for the WT_30k LSTM with 80% sparsity that achieves 98.96 test top-1 accuracy. . . . . 79

4.5 LSTM test accuracy trained with and without sub-sampling on Sequential MNIST. . . . . 81

4.6 LSTM test accuracy trained with sub-sampling on sequential MNIST following several lottery ticket configurations. . . . . 82

4.7 LSTM performance on Wikitext-2. . . . . 85

4.8 LSTM perplexity on Wikitext-2 with different pruning pipelines . . . . . 86

5.1 2-bit weight and activation (W&A) quantization of Resnet-18 on CIFAR-100 with various GML margins. . . . . 96

5.2 2-bit weight and activation (W&A) quantization of Resnet-18 on CIFAR-100 with various AMS margins. . . . . 96

5.3 CIFAR-10 and CIFAR-100 test top-1 Accuracy for extreme quantization settings of ResNet-18. . . . . 98

5.4 **ImageNet-1k** Top-1 Accuracy for extreme quantization settings of Resnet-18. Disentangled Loss Quantization Aware Training (DL-QAT) and Scaled Adjust Training (SAT) results are obtained from our experiments, all the other results are reported from the original papers. DL-QAT and SAT use original Resnet-18. LSQ and PACT use full pre-activation Resnet-18. LQ-Net use Resnet-18 type-A shortcut. BWN use Resnet-18 type-B shortcut. 100

6.1 ResNet performance on CIFAR-100 for three independent trainings . . . . 116

6.2 SPSA attack targeting the ResNet-32 0 with 72.7% top-1 test accuracy trained on CIFAR-100 with the cross entropy loss. . . . . 116

6.3 SPSA attack targeting the EHD mechanism that achieves 73.4% top-1 test accuracy combining three ResNets-32 trained on CIFAR-100 with different initialisations and the cross entropy loss. . . . . 117
6.4 SPSA attack targeting a ResNet-32 with 73.5% top-1 test accuracy trained on CIFAR-100 with the gaussian mixture loss (GML). . . . . 118
6.5 SPSA attack targeting a ResNet-32 with 71.9% top-1 test accuracy trained on CIFAR-100 with the max mahalanolis center (MMC) loss. . . . . 118
6.6 SPSA attack targeting the EHD mechanism that achieves 73.2% top-1 test accuracy combining three ResNets-32 trained on CIFAR-100 with three different loss functions: CEL, GML and MMC. . . . . 119
6.7 SPSA attack targeting the EHD mechanism that achieves 73.4% top-1 test accuracy combining $k = 4$ ResNets-32 trained on CIFAR-100 with different initialisations and the cross entropy loss. . . . . 119
6.8 SPSA attack targeting the EHD mechanism that achieves 73.1% top-1 test accuracy combining $k = 5$ ResNets-32 trained on CIFAR-100 with different initialisations and the cross entropy loss. . . . . 119

A.1 Top-1 accuracy (%) on the white-box adversarial examples crafted on the test set of CIFAR-100 targeting ResNet-32 quantized with DL-QAT. . . . . 129

# Chapter 1

# Introduction

The attention that Artificial Intelligence (AI) and Machine Learning (ML) get has grown dramatically over the past decade. AI is the science and engineering of creating intelligent machines that have the ability to perform tasks commonly associated with intelligent beings. Traditionally, rule-based AI systems are designed to achieve "intelligence" via a model solely based on predetermined rules. Such systems comprise a set of human-coded rules that result in pre-defined outcomes. In contrast to rule-based programming, machine learning algorithm learn and adapt without following explicit instructions by analysing and drawing inferences from patterns in data. The capacity to produce and store data has drastically increased over the past decades and with it the success of machine learning. One of the best machine learning algorithm is Deep Neural Networks (DNNs) and its use is known as the field of deep learning. DNNs can solve problems in computer vision, such as segmentation for autonomous driving, in natural language processing, and in speech recognition where traditional rule-based programming methods only provide limited solutions. However, there are still many challenges for DNNs. The work in this thesis proposes solutions for two important challenges:

- How to make resource consuming deep neural networks easier to deploy on resource limited platforms.
- How to make deep neural networks more robust towards attacks based on crafted inputs.

## 1.1 Context

Historically, deep learning emerged from a neuroscience perspective in the 1940s in a context of low computation and data resources. The creation of the Internet as the most advanced communication tool of humanity and cloud storage led us in a data abundant era. Moreover, the computing power greatly increased from the 1970s following Moore's law. Especially, it came to light that parallel single instructions multiple data (SIMD)

architectures like graphic processing units (GPUs) are a powerful tool for neural network training. This context led to a significant breakthrough in 2012 when Alexnet [55] was implemented using cuda kernels to accelerate training on graphic processing units and surpassed the human programmed approaches for image classification. Since then, research effort on neural networks and neural network accelerators grew steadily and numerous works have been published to improve those methods and exploit their potential in various applications such as health care, self driving cars, virtual assistants, face recognition, content generation, speech recognition, protein structure prediction, automatic game playing, etc. Figure 1.1 shows the number of publications during the last decade by field of study in the AI domain. The number of publications in the field of machine learning increased from around 5000 in 2010 to almost 40000 in 2021. Notably, this growth took a change in slope between 2015 and 2017.

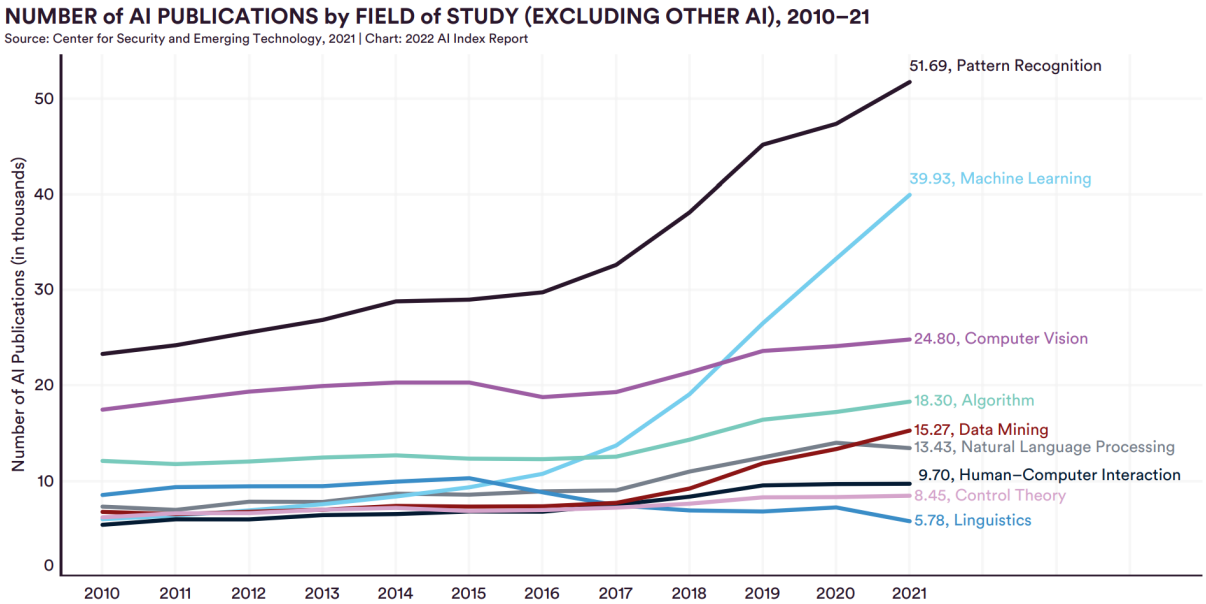



Figure 1.1: Evolution of the number of publications in the AI field.

Several other enabling factors behind this growth include the accessibility of learning neural networks with the release of open-source frameworks, such as Tensorflow in 2015 and Pytorch in 2016, and also the techniques to learn deeper neural networks to achieve better performance, like the approach of residual networks, published in 2015.

Such research effort opens many applications possibilities and in order to support their development and deployment, a tremendous amount of computing power is needed. Cloud-based platforms offer a relevant strategy to centralize those heavy computation needs. However, the demand for deployment on edge is also growing. Indeed, critical applications with real-time constraints such as memory, latency, power consumption, with a resource-scarce hardware target or with privacy issues, cannot be inferred on cloud. Instead, the neural networks are always pre-trained offline, and then implemented in

embedded systems for the inference stage. However, neural networks are still memory and computation intensive during the inference phase. In order to meet the requirements for edge inference, the networks are optimized with compression and/or speed-ups techniques.

Quantization is a class of methods that reduces the number of bits to represent the parameters of the DNN (and sometimes also changes the number representations), while keeping the performance and quality of results as close as possible to the floating-point reference. The compression factor is proportional to the number of bits reduction. For instance, the weights of a Deep Neural Networks (DNN) can be stored as signed integers using 8 bits instead of 32 bits with the single-precision floating-point format without any loss in application performance, in this case the compression factor of the neural network parameters storage is $\frac{32}{8} = 4$. As such, it reduces the memory footprint and once the operators are engineered to take advantage of the quantization scheme, the computations require fewer bits, and thus enable higher throughput and energy savings. Pruning is another class of methods that cut (i.e., set to zero) the redundant neurons in neural networks, and thus reduces its memory storage. Those compression techniques are further discussed in Chapter 2.

In 2013, Szegedy *et al.* [103] first exposed neural networks vulnerability towards small input perturbations. In this aim, they show that introducing a small perturbation into the input, which is invisible to humans, can cause the neural network to dysfunction. Those modified inputs, called adversarial examples, are specifically crafted for a target model using an optimization method. As the scope of neural network-based applications expands, neural networks are facing more and more threats from any attacker who wishes the system to misbehave. This poses strong security concerns with deployment in applications of the deep neural network.

In summary, deep learning is an approach to machine learning that has drawn heavily on our knowledge of the human brain, statistics, and applied mathematics as it developed over the past several decades. In recent years, deep learning has seen tremendous growth in its popularity and usefulness, largely as the result of more computing power, the availability of large datasets, technologies and knowledge to train deeper networks. In order to deploy the neural networks performance on edge, compression and speed-ups techniques and speed-ups techniques are required. Also, neural networks have a vulnerability that poses security concerns for certain applications.

## 1.2 Issues related to compression and vulnerability of DNNs

The research around compression and acceleration of DNNs has been very active in recent years. There are still shortcomings to those methods. Taking quantization methods as an example: while very low-precision settings like binary parameters have the best advantages for compression and acceleration, the resulting drop in performance undermines the application needs. Meanwhile, the proposed defense methods towards adversarial attacks often compromise application performance or significant memory and computational overhead at inference time. To address the limitations of the current compression, acceleration and vulnerability of neural networks, the following questions are proposed:

- Is it possible to transfer existing compression methods to different types of neural networks and to improve quantization methods to find better trade-off points between model compression and application performance?
- To what extent a defense mechanism can help build a neural network robust towards adversarial perturbations and is it possible to propose a new direction to imply robustness towards adversarial attacks, so that it can combine with existing methods and require less resources for inference on edge?

The works in this thesis attempt to answer those questions.

## 1.3 Contributions

In order to compress and accelerate Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), several contributions are proposed. First, we investigate the transferability of an existing iterative pruning method with retraining originally designed on CNNs to compress RNNs. The contributions of this investigation are threefold:

- We investigate practical RNNs learning behaviour.
- We extend the experimental part from the works of Frankle *et al.* [25, 26], an iterative pruning and retraining technique, with the most used recurrent architecture, the Long Short Term Memory (LSTM).
- We introduce a pre-processing method based on data sub-sampling that enables faster convergence of LSTM while preserving application performance.

Then, a new quantization aware training (QAT) method for CNN is proposed by improving an existing advanced quantization method. The contributions are twofold:

- We introduce a new quantization method, called Disentangled Loss Quantization Aware Training (DL-QAT), that improves the Scaled Adjust Training (SAT) method [48] with a type of loss function that we qualify as disentangled losses.
- Experiments on the resnet-18 CNN topology using the ImageNet image classification task highlight that the proposed method provides the best compromise between memory footprint and application performance among existing state of the art.

Finally, the Ensemble Hash Defense is proposed. It enables better resilience to adversarial attacks while preserving application performance and only requiring a memory overhead at inference time. The contributions are the following:

- We survey the state of the art of adversarial attacks and defenses.
- We introduce the Ensemble Hash Defense. The defense system combines several neural networks with comparable application performance and a model selection process. At inference time, the selection process chooses, based on the input, one model for inference. An attack focusing the EHD system will have to deal with multiple targets, *i.e.* with multiple neural networks. The concept of the defense mechanism is compatible with other robust optimization methods and neural network compression methods. With the perspective of edge inference, the memory overhead introduced by EHD can be further reduced with quantization or weight sharing.
- We then provide first experimentation to study the robustness of quantized CNNs with the proposed DL-QAT.

## 1.4 Thesis outline

This thesis is organised into three main parts as follows.

**Background and state of the art.** In Chapter 2, we first introduce common concepts for training neural networks. Then, we present convolutional neural networks and recurrent neural networks and finally we introduce some neural networks compression techniques. Then, Chapter 3 introduces libraries to enable deep neural network training and deployment on edge, along with benchmark datasets used in this work. Those two chapters lay the groundwork for the contributions of this thesis which are divided in two parts : Compression of neural networks and Neural networks robustness toward adversarial attacks.

**Compression of neural networks.** We present our work toward more efficient methods of training neural networks that are easier to deploy on resource limited platforms. On

one hand, the parameters of the neural network are compressed to make it less demanding on memory. On the other hand, the approximation applied to structure, operands, and operators, which makes the calculation faster on real-time platforms with limited computing resources. The primary goal being able to provide the best performance neural networks with the lowest memory and computational usage for inference.

In Chapter 4, we investigate the compression of RNNs with unstructured pruning methods: the parameters are iteratively set to zero during training based on a threshold in order to reduce the parameters memory storage. We extend the experimental scope of an advanced pruning technique to RNNs in order to investigate a recurrent architecture practical learning behaviour, known as LSTM. We analyse the convergence of such models when subject to the pruning techniques on image classification and language modelling. We present a pre-processing method based on data sub-sampling that enables faster convergence of LSTM while preserving application performance.

In Chapter 5, we propose to further improve an advanced quantization method: the number of bits used to represent the parameters and the dataflow at inference is reduced in order to lower the parameters memory storage and unlock possible hardware acceleration. We introduce the context of application and the state of the art methods for quantization. We then present our approach and experiments to compare its performance with state-of-the-art methods on common image classification datasets. In particular our contribution focuses on the challenging low-bit settings like binary parameters where the application performance is the most impacted.

**Neural networks robustness toward adversarial attacks.** We present our work to make neural networks more robust towards Adversarial Attacks based on crafted inputs. In Chapter 6, we introduce the state of the art on adversarial attacks and defense mechanisms. We then present our defense mechanism and evaluate its resilience under several relevant settings using an image classification task. We also discuss its main limitation on memory overhead at inference time. In fact, we put in perspective the EHD system inference on edge thanks to the easy compatibility of EHD with compression methods like quantization or weight sharing. Related to this chapter, Appendix A moreover discusses previous works studying the effect of quantization on adversarial robustness and also presents preliminary attacks results of quantized convolutional neural networks based on our quantization approach presented in Chapter 5.

Finally, we conclude with a discussion of our results, potential improvements on this work and insights for future research directions in Chapter 7.

# Chapter 2

# Deep Neural Networks

Deep neural networks approximate some function $f_*$. Taking a classifier $y = f_*(x)$ that maps an input $x$ to a category $y$, the deep neural network $f$ defines a mapping $y = f(x, \theta)$ and learns network parameters, also called weights $\theta$, that best approximate the function $f_*$. Neural networks are called networks because they compose a succession of functions, often in a chain manner, to form a network. For example, we might have three functions $f_1$, $f_2$, and $f_3$, with the corresponding parameters $\theta_1$, $\theta_2$ and $\theta_3$, to form the model $f(x, \theta) = f_3(f_2(f_1(x, \theta_1), \theta_2), \theta_3)$. In this case, $f_1$ is called the first layer or input layer of the network, $f_2$ is the second layer, and so on. The final layer of a network is called the output layer or last layer. The overall length of the chain gives the depth of the model. The name "deep" in deep neural networks and deep learning arose from this terminology.

How do we optimize the parameters $\theta$? Some deep neural networks may be non-linear systems containing billions of parameters, and a naive random search approach could not get the optimal solution within a reasonable amount of time. In 1986, Rumelhart *et al.* [89] proposed the backpropagation algorithm, that efficiently computes gradients of each parameter with respect to a performance metric, also called loss function, or cost function, or error function. The gradient-based learning algorithm decides how to use each layers to produce an output to best implement an approximation of $f_*$. In the aforementioned classifier example, the loss function is the function assessing the difference between the category $y$ predicted by the network $f$ and the ground truth category $\hat{y}$. In practice, the parameters are iteratively updated to minimize the loss function with gradient descent. To this day, gradient-based optimization strategies remain the most popular methods to learn neural networks.

## 2.1 Common Concepts for Training Neural Networks

### 2.1.1 Model, data, and generalization

Deep learning algorithms approximate some function by analysing and drawing inferences from patterns in data. The model and the data are the two fundamental elements of deep learning. Choosing a model usually depends on the addressed problem. For example, convolutional neural networks (CNN) are powerful to process images, whereas recurrent neural networks (RNN) are used to process temporal signals like audio.

The training dataset should be representative of the function the model tries to approximate. Properties like data quantity and quality and class balance have a critical impact on the training outcome. Usually, the more data the training dataset contains, the better the performance of the model. Data quality refers to the reliability of the data (the presence of duplicate examples, inaccurate values, wrong labels, etc.) and the feature representation. For example, when we want to train a network that can distinguish cat and dog pictures, as the dataset only contains cats and dogs images, the trained network will not recognize pandas or tigers. Also, if the training set only contains images of black dogs and white cats, then the system may learn a bias to recognize colors instead of animals.

The goal of deep neural networks is to adapt properly to new, previously unseen data, not just perform well on training samples. This ability of the learned model to apply to new samples is called generalization. However, neural networks are prone to rely on the specific characteristics of training samples, that is to say, they are easily drowned to learn biases in the training data. This phenomenon is called over-fitting. To overcome this, the quality of the data plays a major role to enable the model to learn general characteristics of samples. Then, there exist regularization techniques to mitigate over-fitting and improve training.

### 2.1.2 Loss function

In this section, we introduce loss functions by providing an example of multi-class classification with the cross entropy loss. The loss function $L$ is used to quantify the quality of the set of trainable parameters. By minimizing the loss function with an iterative optimization process, the parameters are tuned to achieve better performance. The cross entropy loss is the most popular cost function for multi-class classification.

Taking an example where a model has to predict whether the input input contains a dog, a cat, a panda or a tiger, Fig. 2.1 decomposes the steps to compute the cross entropy loss function in the case of one input image that contains a cat. The vector values of

non-normalized predictions that a classification model generates are called logits. The cross-entropy loss function takes this logits vector $\mathbf{y}$ and the ground truth label $\hat{\mathbf{y}}$ as inputs. The cross entropy loss formulation combines the softmax function and the negative log likelihood function as

$$L_{CE}(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_{c=1}^{C} \hat{y}_c \, log \left( \frac{exp(y_c)}{\sum_{i=1}^{C} exp(y_i)} \right) \tag{2.1}$$

where $C$ is the number of classes.

The logits vector is normalized into a vector of probabilities corresponding to each class with the softmax function. As they are probabilities, they are positive and sum to 1. The softmax output probabilities can be interpreted as a vector of dimension $C$. The one-hot vectors encoding the different classes are the orthogonal vectors that construct the canonical basis of $\mathbb{R}^C$. The network tries to map the input image to a vector as close as possible to the orthogonal vector associated with its ground truth class.

Each predicted class probability is compared to the actual class desired output 0 or 1 and the calculated loss penalizes the probability based on how far it is from the actual expected value. Using the negative log likelihood enables a large loss for large differences close to 1 and small loss for small differences tending to 0.

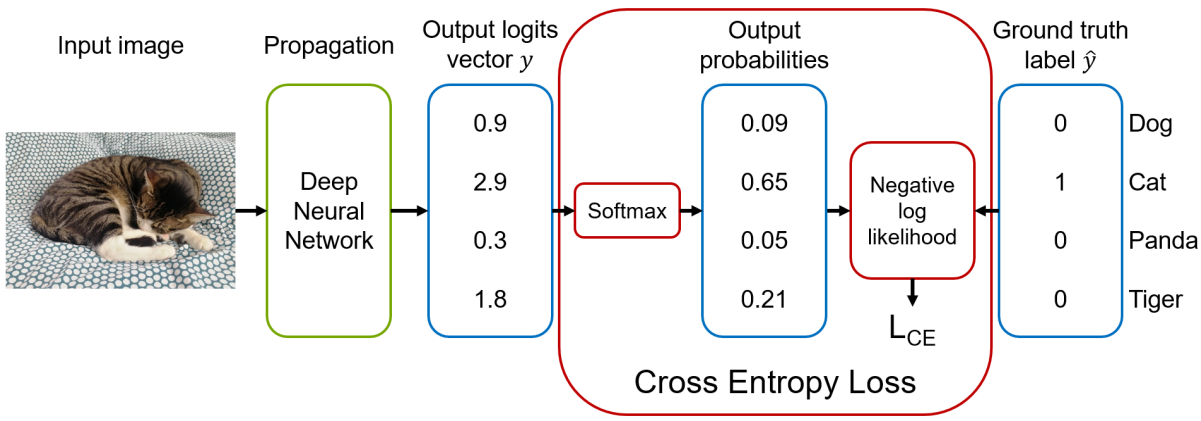



Figure 2.1: Cross Entropy Loss (CEL) function in a multi-classification task.

### 2.1.3 Initialization

Training deep learning models with stochastic gradient descent requires the user to specify some initial point from which to begin the iterations. The choice of the initialization strongly affects the convergence of deep neural networks and can even determine if the stochastic gradient descent converges at all. On one hand, some initial points are so unstable that the algorithm faces numerical difficulties (namely the vanishing and exploding gradients problems) and fails altogether. On the other hand, the initialization can benefit

the convergence, how quickly it converges and whether it converges to a point with high or low cost.

The initialization of deep neural networks gained a lot of maturity in the past decade, improving the performance and convergence speed of deep neural networks. Such maturity is reflected in deep learning frameworks where the initialization is not an primary issue for the user as default initialization methods almost always enable the gradient descent to converge. The Kaiming initialization scheme, proposed by [37], is one of the most popular initializations.

### 2.1.4 Training paradigms

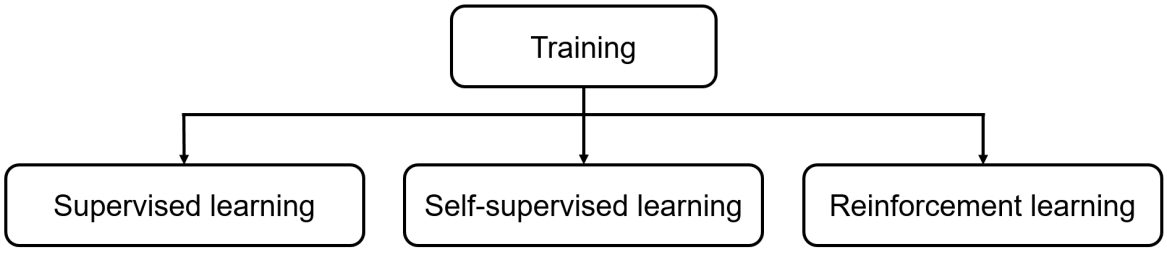



Figure 2.2: Main training paradigms: supervised learning, self-supervised learning, reinforcement learning.

Supervised learning is the machine learning task of learning a function that maps an input to an output based on the example of input-output pairs. A pair consists of an input object and the desired output value (for example an image of a dog and its label "dog"), that is to say, each input is provided with its expected solution. Those input-output pair examples build a labeled dataset. A supervised learning algorithm analyzes the training labeled dataset and produces an inferred function, which can be used for mapping new inputs. In practice, a test dataset, *i.e.*, input-output pairs that are not used during training, is used to evaluate the performance of the learned model. Such a test dataset is essential in order to assess if the network can generalize. The supervised learning paradigm enables learning neural networks with significant guidance and is very efficient to approximate a function related to a problem. However, building large labeled datasets requires a tremendous effort in human labelling, and is thus very expensive. One may not have the data or finance resources to rely on a labeled dataset.

Self-supervised learning is a self-organized learning that finds data patterns without the support of labels and with a minimum of human supervision. In contrast to supervised learning that usually makes use of human-labeled data, the model learns to predict part of its input from other parts of the input. A portion of the input is used as a supervisory

signal to a predictor fed with the remaining portion of the input [49].

Reinforcement learning algorithms enable learning models by evolving in an environment. Software agents take actions in an environment to maximize a cumulative reward in a trial and error fashion. It does not need labeled datasets, instead, the focus is on finding a balance between exploration of uncharted territory and the exploitation of current knowledge [51].

### 2.1.5 Data augmentation

Data augmentation is a technique used in data analysis to increase data volume by adding slightly modified copies of already existing data or newly created synthetic data from existing data. This method is usually used when the training dataset is insufficiently large. It also helps to reduce over-fitting [94]. Classic data augmentations for images are geometric transformations, flipping, color modification, cropping, rotation, noise injection and random erasing.

### 2.1.6 Supervised training of a neural network

This section presents the supervised training iterative method using the Stochastic Gradient Descent (SGD) optimization. The Stochastic Gradient Descent method is an iterative optimization algorithm for finding the minimum value of a cost function. This is probably the most widely used method for optimizing deep neural networks. Figure 2.3 details the basic process flow of one training iteration that can be separated into two phases: propagation and backpropagation. The propagation phase is the prediction process of the model. Until the network converges to a desired performance, the backward propagation process is performed to train the network. From an initial set $\theta_0$, the parameters are iteratively updated to gradually minimize the loss function $L$. It performs two steps iteratively:

1. Compute the gradient $\frac{\partial L}{\partial \theta_i^j}$ that is the first order derivative of the loss function $L$ with respect to $\theta_i^j$, the trainable parameter $j$ at the training iteration $i$. For deep neural networks, the multi-layer can be regarded as a nested compound function which the direct derivation is complicated to calculate. In practice, the chain rule is used to decompose this derivative into simple ones and compute the gradient of the model parameters. It requires that all functions composing the neural network are differentiable.

2. Update the parameters from the current point to the direction of the gradient descent. The basic gradient descent updating process of $\theta_i^j$, the trainable parameter $j$ at the

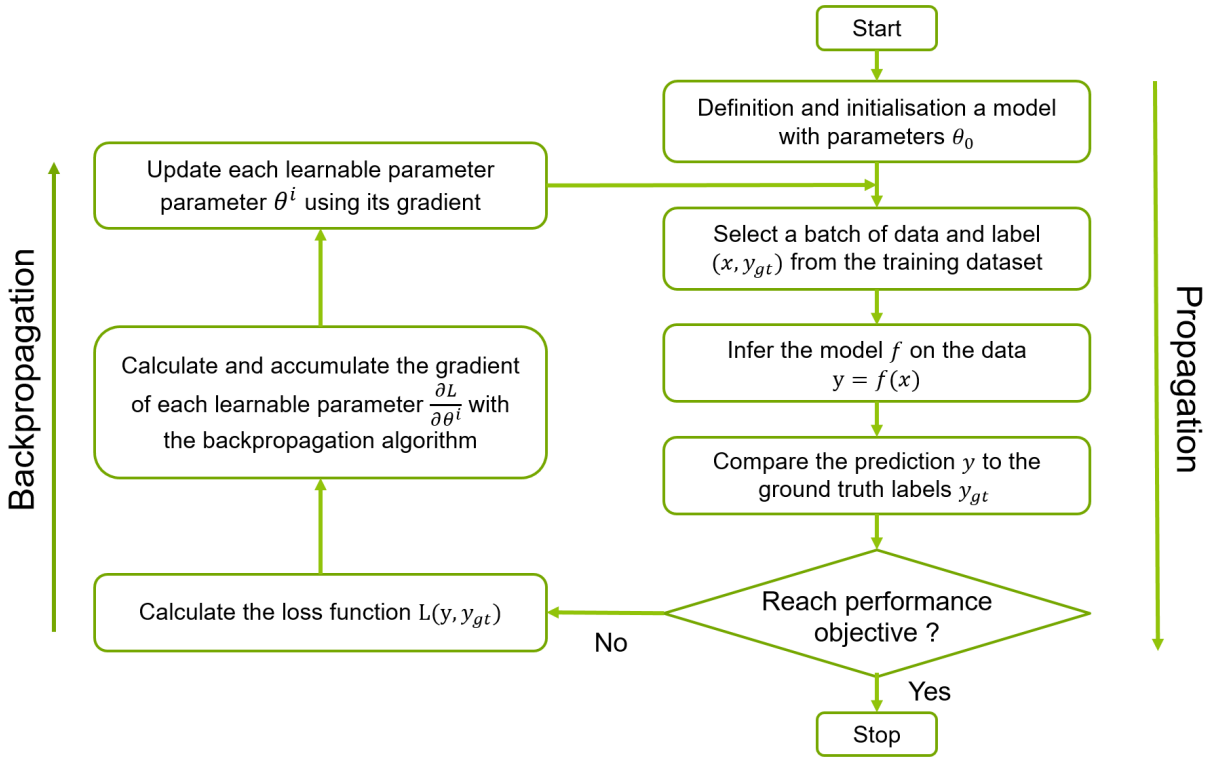



Figure 2.3: Supervised training with the gradient descent method.

training iteration $i$ is expressed as:

$$\theta_{i+1}^j = \theta_i^j - \eta \frac{\partial L}{\partial \theta_i^j} \tag{2.2}$$

where $\eta$ is the learning rate, a scalar that directly scales the contribution of the gradient in the update process. This formulation is often upgraded with momentum to help accelerate the convergence of the model. More advanced update methods like the Adam optimizer [53] further accelerate the convergence of neural networks.

Moreover, the way the data is fed to the model during the training process impacts its convergence:

- Random sampling without replacement of input data helps to avoid local minimums. This technique is the reason the gradient descent optimization is denominated as "stochastic". When conducting experiments, the random sampling without replacement is emulated by shuffling the training dataset at each epoch, *i.e.* one complete pass through the training data.

- Perform one update based on a batch of data. In practice, all gradients corresponding to each input in the batch are accumulated. This accumulation of gradients carries more accurate information than a single gradient. Updating using this accumulation at each iteration of the training contributes to obtain a better convergence profile and

a better asymptote. When conducting experiments, one generally tries to maximize the size of the batch. Training on GPUs, the computation overhead induced by a bigger batch can easily be parallelized. The practical bottleneck is the memory overhead induced to enable this parallelized computing. Distributed learning enables splitting a big batch into smaller ones in order to, first, calculate all gradients on different devices, then accumulate the gradients and perform an overall update.

### 2.1.7 Transfer learning

Transfer learning is a machine learning technique where a model trained on one task is re-purposed on a second related task, also called downstream task. It is based on the fact that the features learned on a task are useful for a related task. For example, it is known that the first layers in convolutional neural networks learn general features such as edges and those features are generic to almost any task involving image processing. In practice, two tools are widely used to adapt a pre-trained model to a new task:

- **Learning new heads.** For example, a supervised pre-trained model on ImageNet-1k can be adapted to a detection task where the classification head would be replaced by a detection head.

- **Fine-tuning.** The parameters of the pre-trained models are trained again during the training of the new task. Their training is usually performed on a subset of parameters in the deep layers with a small learning rate to adapt the features to the new task and avoid relearning from scratch.

There are three potential benefits in using transfer learning, as shown in Fig. 2.4:

- **Higher asymptote.** The convergence of the trained model based on the pre-trained model is better than training the model from scratch.

- **Faster convergence.** The slope of convergence of the trained model based on the pre-trained model is better than training the model from scratch.

- **Higher starting performance.** The initial performance using the pre-trained model on the new task allows to have some initial performance.

Those benefits are especially relevant for niche applications where the amount of labeled data is very limited and/or the data are expensive to label. It is often the case for edge applications, and thus transfer learning significantly improves the performance of those niche applications compared to models learned from scratch. Transfer learning is one key method to embed deep neural networks on edge devices with state-of-the-art performance.

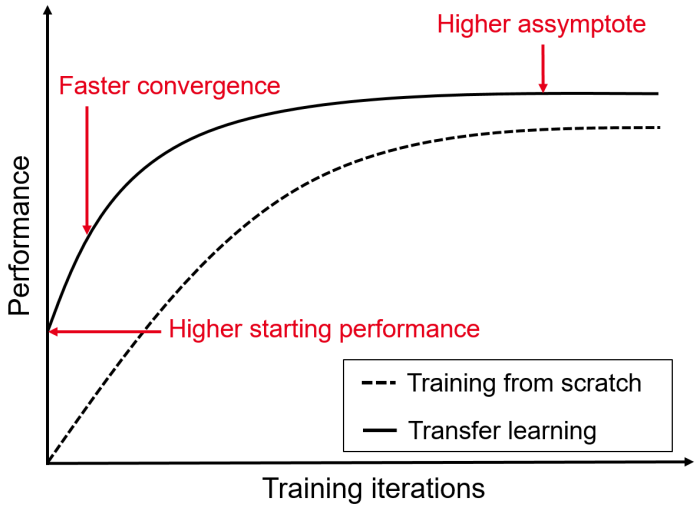



Figure 2.4: Transfer learning possible advantages: higher asymptote, faster convergence, higher starting performance.

### 2.1.8 Increasing model size

Following Moore's law, the computing resources have grown significantly over the past decades and make it possible to run much larger models. Larger models are able to achieve higher accuracy on more complex tasks. Since neural networks are modeled with neurons, the major research axis consists in increasing the number of neurons in the model. This growth is driven faster by the technological breakthrough and by the availability of larger datasets. Technological breakthroughs like the single instruction multiple data hardware Graphic Processing Unit (GPU), more efficient memory and better software infrastructure for distributed computing are among the most important trends in the history of deep learning. Today, self-supervised learning unlocks training on much bigger datasets than the supervised learning. Despite many works pursuing this legacy, a recent research direction, causal learning, focuses on the reasoning ability of the model instead of the performance only.

### 2.1.9 Exploding and vanishing gradients phenomena

The phenomena of vanishing and exploding gradients were first discussed by Hochreiter *et al.* [40] in 1991, and then demonstrated by Bengio *et al.* [11] to arise when training deep neural networks with gradient-based strategies. When backpropagating errors, the gradients are subject to numerical instabilities, known as the exploding gradient phenomenon and the vanishing gradients phenomenon. In the case of the exploding gradient, large error gradients accumulate and result in very large updates on the neural network parameters during training. The resulting optimization step is not reliable and causes instability for the next optimization steps. The network is unable to learn from the training data,

the objective function is also likely to diverge. In the case of the vanishing gradient, the backpropagation algorithm is unable to backpropagate useful gradient information from the output end of the model back to the layers near the input end of the model. As the gradients do not carry useful information, the resulting optimization step is not reliable, and the model is unable to converge.

As discussed in previous Section 2.1.8, increasing the model size, and so the depth of deep networks, is the main research axis to achieve better performance. The exploding and vanishing gradient phenomena are more likely to arise as the depth of the network increases. Many works were proposed to mitigate those phenomena and unlock the convergence of deeper networks.

## 2.2 Multi-Layer Perceptrons

There are different types of deep neural networks such as Multi-Layer Perceptrons, Convolutional Neural Networks, Recurrent Neural Networks, Transformers, Graph Neural Networks. In this work, we limit our scope to compressing and accelerating CNNs and RNNs. Multi-layer perceptrons (MLPs) are the quintessential of deep learning models. In this section, we give a brief introduction to MLPs and its basic components.

MLPs and convolutional neural networks (presented in Section 2.3) are feedforward networks, that is to say the information flows through the model being evaluated from an input $x$ to the output $y$. There are no feedback connections in which outputs of the model are fed back into itself. When neural networks are extended to include feedback connections, they are called recurrent neural networks, as presented in Section 2.4.

### 2.2.1 Fully-connected layer

A multi-layer perceptron is a neural network with an input and an output layers and at least one hidden layers with many densely connected neurons that are stacked together. Figure 2.5 illustrates one densely connected layer, also called fully-connected layer that can be stacked to form a multi-layer perceptron. The last layer will not have an activation function. The matrix-vector multiplication highlighted by the blue dotted box is a linear operation. In the general case, the layer input vector $x \in \mathbb{R}^n$ represents the input data or the previous layer output in the neural network. $\theta \in \mathbb{R}^{m \times n}$ is the trainable parameter matrix associated to the fully connected layer. The result of the matrix-vector multiplication, symbolized by $\times$, is the vector $y \in \mathbb{R}^m$ defined as

$$y = \theta \times x. \tag{2.3}$$

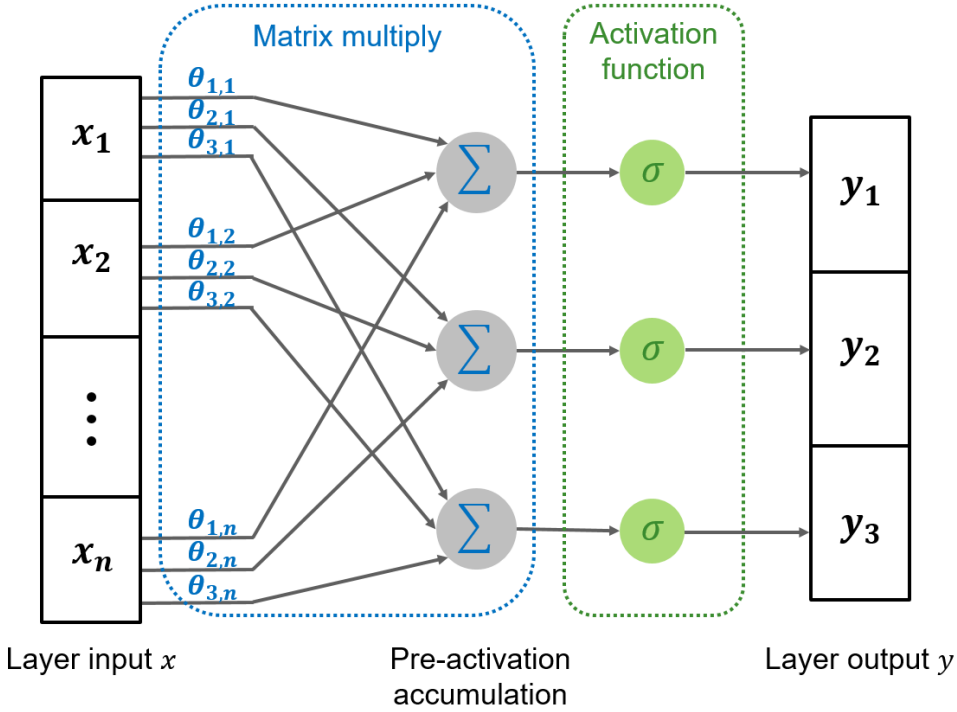



Figure 2.5: Fully-connected layer with parameter matrix $\theta \in \mathbb{R}^{3\times n}$ and a non-linear activation function $\sigma$. It takes an input $x \in \mathbb{R}^n$ and returns an output $y \in \mathbb{R}^3$. Several fully-connected layers can be stacked to form a multi-layer perceptron. The last layer will not have an activation function.

Usually, a vector of trainable biases $b$ is added to the multiplication result to form the multiplication accumulate operator, which gives

$$y = \theta \times x + b. \tag{2.4}$$

### 2.2.2 Non-linear activation function

In order to model complex data patterns, a neural network needs to establish non-linear relation between inputs and outputs. Non-linear activation functions are used in each layer to introduce non-linearity. The activation function $\sigma$, highlighted by the green dotted box in Fig. 2.5, is applied element-wise on the propagated signal at each layer. The most popular activation functions are the following:

- Rectified Linear Unit (ReLU)

$$ReLU(x) = max(0, x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{otherwise} \end{cases} \tag{2.5}$$

- Sigmoid

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.6}$$

- Hyperbolic tangent

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.7}$$

### 2.2.3 Limitations

While MLPs or fully connected models can learn handwritten digit recognition on the MNIST dataset, it does not scale up well to more complicated tasks. Two main limitations arise. Firstly, as the size of the input data increases, the densely connected paradigm implies many parameters. For example, processing a 3-channels colored image from the ImageNet dataset that represents $224 * 224 * 3 \sim 150k$ data values by a fully connected layer with merely 100 output units would already contain several million parameters. This over-parameterization can easily lead to overfitting. Secondly, the fully connected architectures entirely ignore the structure of the input. On the one hand the input can be presented in any fixed order without affecting the outcome of the training. On the other hand it is difficult to learn local correlations, for example on images or spectral representations that have strong 2D local correlations. Today, fully connected layers are used as heads rather than as standalone models.

## 2.3 Convolutional Neural Network

To overcome the shortcomings of fully connected layers, Yan LeCun introduced convolutional neural networks in 1989 [60]. Convolutional Neural Networks (CNNs) have been widely used in 2D image and video tasks and 1D signals. In this section, we give a brief introduction to CNNs.

### 2.3.1 Convolution layer

The discrete spatial convolution applied to a two-dimensional input image $I \in \mathbb{R}^{width \times height}$ using a smaller kernel $K \in \mathbb{R}^{k_w \times k_h}$, and symbolized as $\otimes$, is formulated as

$$(I \otimes K)(w, h) = \sum_{n_1=1}^{f_w} \sum_{n_2=1}^{f_h} K(n_1, n_2) I(w + n_1, h + n_2). \tag{2.8}$$

The convolution of a pixel corresponds to the weighting and accumulation of its surrounding pixels. The calculation in Fig. 2.6 shows an example that calculates the convolution of one pixel with a filter with size $3 \times 3 \times 1$.

To use this convolution operation over the whole image, the filter is applied to each pixel location, scanning the image one kernel area at a time. The convolution of the image strides in the width and height dimensions. The output of the convolution is a two-dimension tensor recording the positions of receptive and non-receptive areas, called feature-map.

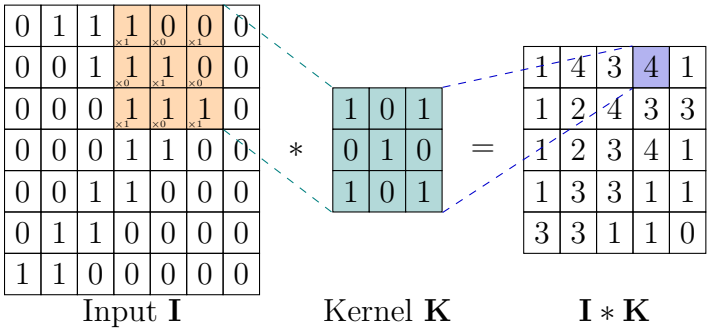



Figure 2.6: An example of convolution using a $3 \times 3 \times 1$ kernel

Each filter can extract one feature-map and more filters can be added to extract more feature-maps. Those feature-maps are stacked in a channel dimension to produce an output.

A convolution layer contains a set of trainable kernels, also called filters $\theta \in \mathbb{R}^{k_w \times k_h \times k_c}$ and trainable bias $b \in \mathbb{R}^{k_c}$. The computation of a convolution layer from an input $X \in \mathbb{R}^{width \times height \times channel}$ can be expressed as

$$Conv(X)(w, h, c) = \theta(w, h, c) \otimes X(w, h, c) + b. \tag{2.9}$$

It is a common vision that the filters in the first convolutional layers learn visual features like edges, and also higher level features such as textures in deeper convolutional layers. Compared to the fully connected layers, convolutional layers yield several advantages.

- Striding filters on the image instead of densely connecting all pixels allows to reduce significantly the number of trainable parameters. This weight sharing also helps mitigate the overfitting problems of large fully-connected layers.
- Convolutional layers force the extraction of local features by restricting the receptive fields of hidden units to be local. Moreover, when multiple convolutional layers are stacked, their receptive fields expand.
- By forcing the replication of weight configurations across space, convolutional layers, and by extension CNNs, are shift invariant, a convenient property for the model to generalize better to unseen data. For example, shift invariance means that the classifier is not affected by the position of the object (*e.g.* cat) in the image.

### 2.3.2 Pooling layer

Pooling is a tool to reduce the size of a 2D input. Pooling layers are used in CNNs to cut down the size of feature maps and so reduce the number of parameters and computation in the network. Doing so, pooling layers contributes to mitigate over-fitting. Similarly to convolution kernels, pooling kernels strides the input on both width and height. They resize

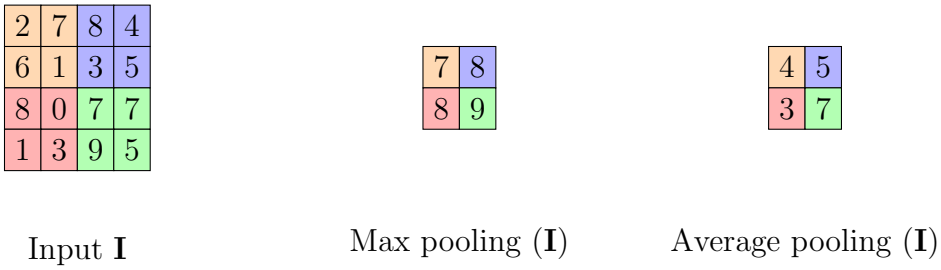



Figure 2.7: An example of max pooling and average pooling with kernel of size $2 \times 2$ applied with a stride of 2

those two dimensions, whereas the channel dimension remains unchanged. Figure 2.7 illustrates the most commonly used pooling layer with filters of size $2 \times 2$ applied with a stride of 2. This pooling setting down-samples both width and height discarding by 2 and so discard 75% of the feature-maps. Also, the pooling units can perform different functions, such as the maximum operation, the average operation, or the L2-norm.

### 2.3.3 Batch Normalization layer

When learning the parameters of a neural network, the distribution of each layer input changes during training. This can cause the gradient descent trajectory to oscillate, thus taking more steps to reach the minimum. Before the use of batch normalization, the learning rate was set to a low value and the initialization was carefully designed for the network to converge, hence a long training time. To enable faster and more stable convergence, Ioffe *et al.* [46] proposed to normalize layer inputs with a learnable normalization layer called batch normalization. In practice, batch normalization applies a transformation that maintains the mean output close to 0, and the output standard deviation close to 1.

### 2.3.4 Resnets

Increasing the depth of a deep neural network, *i.e.*, increasing the number of stacked layers, generally enables to achieve higher performance on more complex tasks. However training deeper models comes with challenges like vanishing and exploding gradients phenomena, as well as overfitting. He *et al.* [38] conducted preliminary experiments to study how the depth influences the training of CNNs. They learned several CNNs with an increasing number of convolutional layers on image classification datasets CIFAR-10 and ImageNet. Results on both datasets showed two major points. First, they confirmed that deeper networks achieve better performance. Secondly, there is a limit on which the depth of the network leads to higher performance. Indeed, the training error, and consequently the testing error, of a very deep CNN is worse than a shallower CNN. These experiments

showed that deeper networks are harder to train as they face the aforementioned challenges.

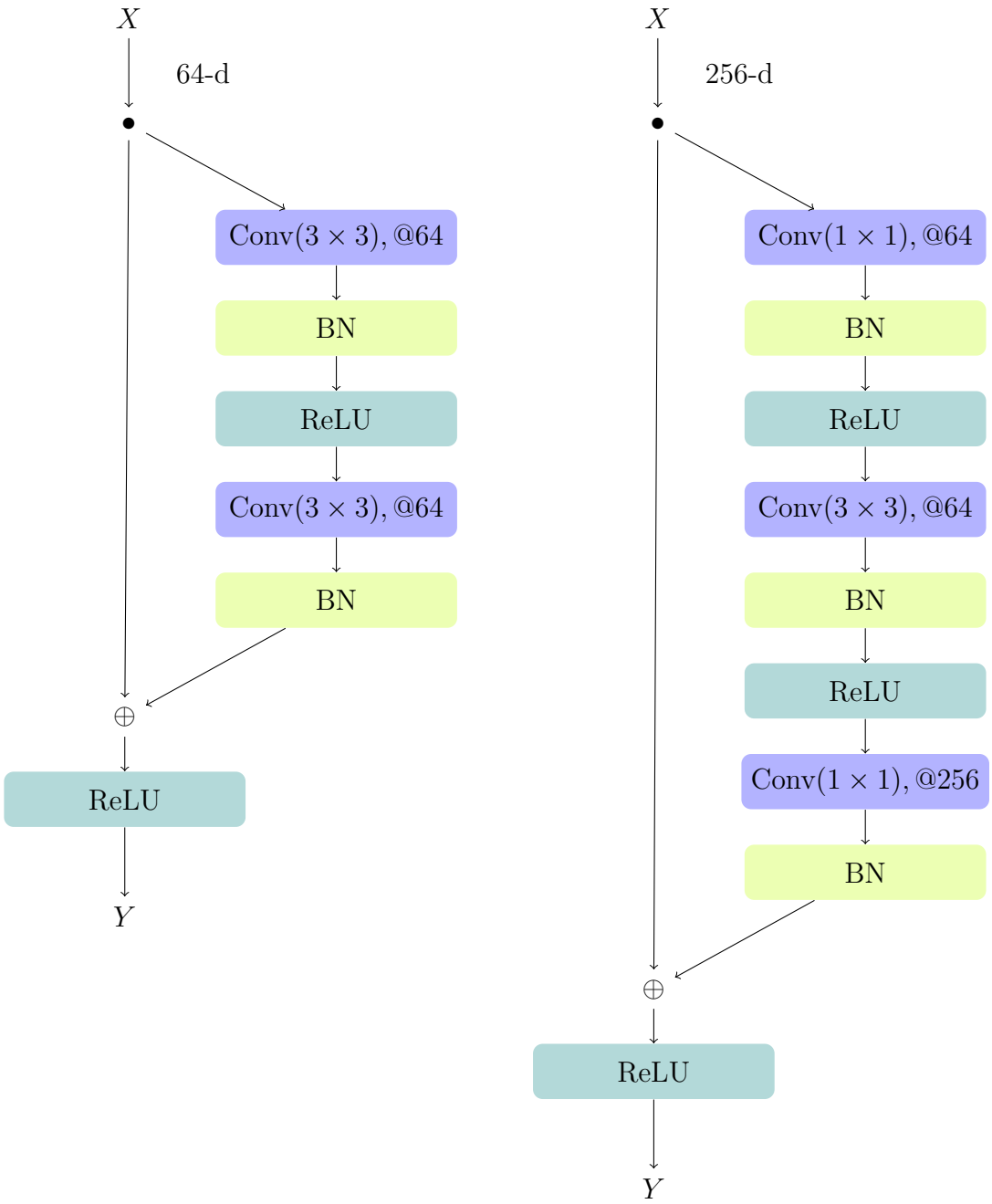



Figure 2.8: Residual blocks proposed by He *et al.* [38]. Left: a residual building block traditionally used for ResNet-34 and shallower ResNets. Right: a "bottleneck" building block traditionally used for ResNets-50/101/152

To overcome these challenges, He *et al.* [38] proposed a new convolutional topology called ResNets that is capable of scaling up the depth of the model. To build ResNets, the authors introduced two basic blocks illustrated in Fig. 2.8. The Resnet topology contribution can be summarized with two main features:

- **Residual connections.** A residual connection is an identity mapping that bypasses

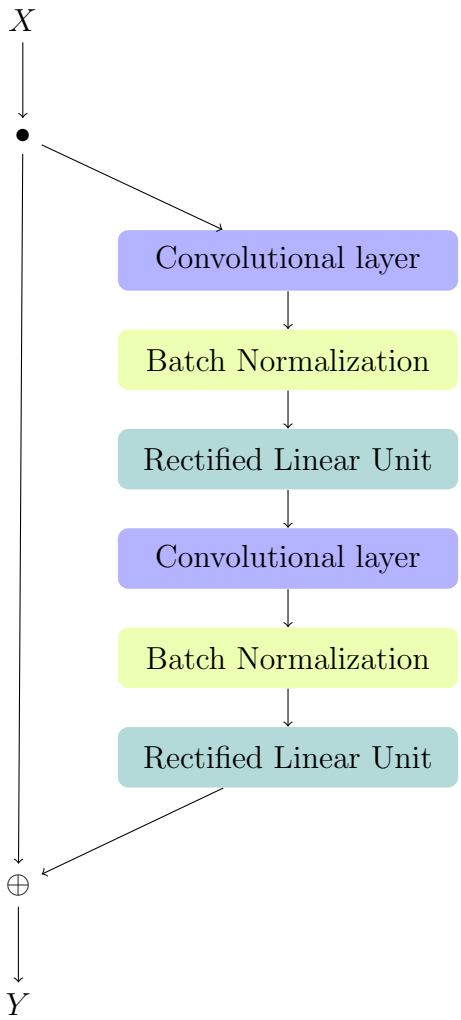



Figure 2.9: The full pre-activation configuration of a residual block proposed by He *et al.* [39]

layers and later adds to the output of the bypassed layers. He *et al.* [39] investigated different arrangements of convolutions, batch normalization and ReLU to implement the residual connections and proposed the full pre-activation configuration illustrated in Fig. 2.9. It is important to notice that the choice of arrangement leads to significant differences when paired with neural network quantization. This will be discussed in Section 5.4.2.

- **Backpropagation friendly activations.** As the depth of the network increases, the more non-linear activations are stacked across the layers, the more backpropagated signals have to be derived from the activation. In order to provide relevant parameter gradients throughout the model, two main elements fluidify the activation backpropagation. First, the derivative of the ReLU activation function is the Heavyside function. Thanks to this derivative, the information backpropagated through each ReLU is kept entirely. Then, batch normalization layers are used after each convolutional layer and contribute to keep gradients in the same scale. This allows

faster convergence for very deep neural networks that have many parameters and are longer to train.

Residual networks showed significant improvements over all other approaches in ILSVRC 2015 and revealed to have the most significant impact in the domain during the past decade. They provide a new solution for training deeper networks and are an important step towards more complex tasks such as object detection and advanced semantic feature extraction.

## 2.4 Recurrent Neural Networks

### 2.4.1 Concept

Historically, recurrent neural networks (RNNs) arose from the motivation of modelling time dependencies within data, e.g., video, audio chunks, and words as part of sentences. Indeed, traditional machine learning algorithms such as Support Vector Machines and logistic regression cannot model time dependencies well. Throughout the 20th century, approaches based on Markov chains [18] like Hidden Markov Models (HMMs) [100, 110] were widely studied. When modelling complex time dependency, the approach of the Markov chain paradigm to represent each possible event is fundamentally limited. Indeed, when the time dependency becomes more complex, more events in the HMM are required. The model representativeness being exponentially correlated to computational needs directly limits their capacity to model complex time dependencies.

In the 1980-90s, early recurrent neural networks approaches were presented by Hopfield *et al.*, Jordan *et al.*, and by Elman *et al.* with a neuroscience perspective. Motivated by practical results rather than biological plausibility, a formulation of a RNN is described in Eqs. (2.10) and (2.11) and illustrated in Fig. 2.10. A recurrent layer is specialized for processing a sequence of input vectors $x_1, ..., x_t, ..., x_T$ with $t \in \{1, T\}$. While each input vector $x_t$ must have the same size $input_size$, the recurrent layer can process sequences with a variable number of vectors, that corresponds to the length $T$ of the sequence. We refer to the entire input sequence as $X = (x_1, ..., x_t, ...x_T)$ and the hidden size of the layer as $hidden_size$. Given the input to hidden weight matrix $U$ of size $(input_size, hidden_size)$ and the hidden to hidden weight matrix $V$ of size $(hidden_size, hidden_size)$,the hidden bias vector $b$ of size $hidden_size$, the hidden to output weight matrix $W$ and the output bias the basic RNN is formulated as

$$h_t = \sigma(Ux_t + Vh_{t-1} + b) \tag{2.10}$$

$$o_t = softmax(Wh_t + b_o) \tag{2.11}$$

where $h_t$ is the hidden state at step $t$ and $o_t$ the output probabilities at step $t$. Depending on the application, $h_0$ can be the initial zero state $h_0 = (0, .., 0)$ or it can be the initial state containing information from a previous sequence. The $\sigma$ function is a non-linear activation function such as hyperbolic tangent or Rectified Linear Unit. In Fig. 2.10 the implicit operator between a matrix and a vector is the matrix-vector multiplication. The green square boxes are the trainable parameters and the blue circle is the non-linear activation function. Thanks to those recurrent connections, the hidden state retains the information

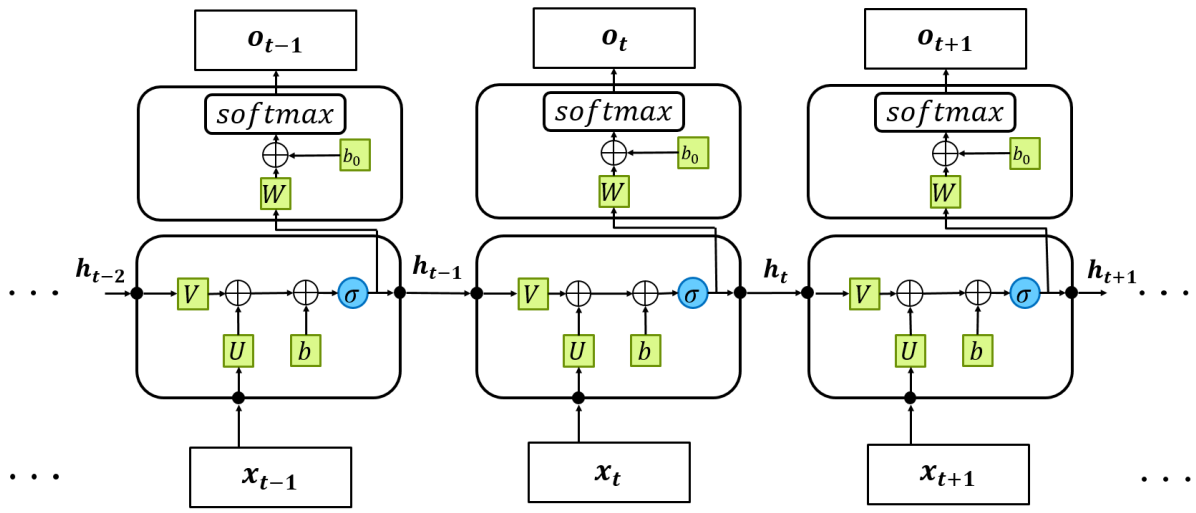



Figure 2.10: The computational graph of a basic recurrent neural network that maps a sequence of input vectors $X = (x_1, ..., x_{t-1}, x_t, x_{t+1}, ...x_T)$ to a sequence of output probabilities $O = (o_1, ..., o_{t-1}, o_t, o_{t+1}, ...o_T)$. Equations (2.10) and (2.11) defines the forward propagation of this recurrent neural network.

from previous inputs. The key value of RNN lies in the expressive power of the hidden state, indeed, the ensemble of representable states grows exponentially with the number of neurons inside a layer, i.e., the dimension of the hidden state [67]. To summarize, the recurrent connections coupled with the modelization capacity of neural networks enable RNNs to model patterns whose origins could pertain down to the beginning of the input sequence.

A first noticeable aspect of the recurrent architecture is its capacity to learn the context needed to complete a task without any human knowledge input. As the RNN is trained with data and an objective function, it models the data patterns guided by the optimization algorithm without any input on the time interval needed to capture those patterns. To illustrate this, let's consider a Structure Health Monitoring task where a recurrent model needs to predict the maintenance of mechanical parts of a plane based on multiple sensors recording their vibration. Even with an expertise in the domain, the context needed for

such a task is not intuitive. Throughout the learning phase, the RNN would learn the relevant patterns to predict maintenance without any prior information on the overall pattern time window.

Another noticeable aspect of the recurrent architecture is that the input sequence length is not constrained. In fact, the input sequence is an arbitrary long sequence of fixed length elements. During the processing of the sequence $X$, the same input weight matrix $U$ is applied on each element of the input sequence $x_t$ and the same recurrent weight matrix $V$ is applied on each hidden state $h_{t-1}$ to compute the next hidden state $h_t$. Thus, input sequences fed to an RNN can have different lengths without the need for more model parameters.

### 2.4.2 Backpropagation through time

Applying the backpropagation strategy to RNNs, the contribution of each parameter to the error is affected by the recurrence paradigm. Indeed, the contribution of each RNN parameter depends on each time step. A derivative of the backpropagation algorithm is then employed, the Backpropagation Through Time (BPTT) algorithm [115]. The BPTT algorithm relies on the product rule to exhaustively compute the contribution of the parameters across all the timesteps. For an in-depth explanation of this algorithm, the reader can refer to the works of Werbos *et al.* [115] and Goodfellow *et al.* [31]. Instead, this section focuses on the practical limitations of RNNs and the different solutions explored in the literature.

In the case of RNNs, backpropagating errors with BPTT not only retraces each layer, but also every time steps, making recurrent architectures significantly more sensitive to numerical instabilities and to vanishing and exploding gradients. Indeed, as the length of the input sequence grows, the shrinking or expanding contribution of each time step can grow exponentially and lead to vanishing or exploding gradients. In practice, they directly affect the capacity of RNNs to learn complex time dependencies, as presented by Hochreiter *et al.* [41], to the point that the basic recurrent architecture presented in Eq. (2.10) can have trouble converging on many tasks. Motivated by the hope to unlock the potential of the recurrent approach, different approaches were studied to mitigate those phenomena. To enable smooth convergence, Quoc Le *et al.* [59] initialized recurrent weights as identity matrix and used Rectified Linear Units instead of hyperbolic tangent. Arjovsky *et al.* [3] proposed to tackle the loss of information through backpropagation with unitary matrices but their approach does not scale well on more complicated task. To this day, the main approach is the gated mechanism approach proposed by Sepp Hochreiter and Jürgen Schmidhuber in 1997.

### 2.4.3 Gated mechanism - Long Short Term Memory

In 1997, the RNN field reached a turning point with the LSTM architecture proposed by Sepp Hochreiter and Jürgen Schmidhuber. In order to tackle Exploding and Vanishing gradients, they introduced a variation of the recurrent architecture based on a gated mechanism. This groundbreaking contribution is the beacon of the field as LSTM is the first to achieve practical results on many tasks using the BPTT algorithm. LSTM provided significant improvements in language modelling and unraveled many applications such as machine translation [7, 102], image captioning [109], handwriting recognition [33], question answering [112], speech recognition [34, 90], and more. The most popular formulation of LSTM is described in Eqs. (2.12) to (2.17) and illustrated in Fig. 2.11.

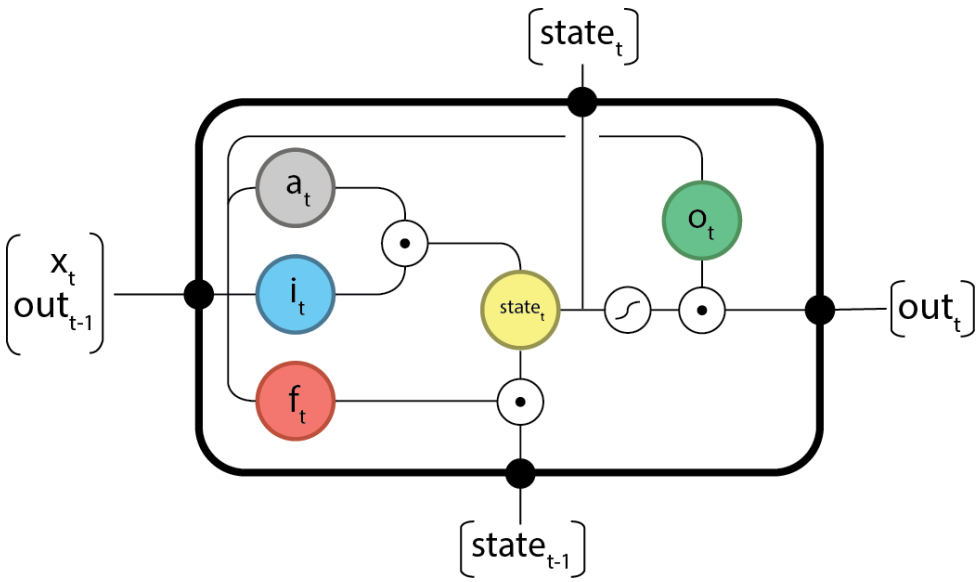



Figure 2.11: The computational graph of a LSTM layer that maps a sequence of input vectors $X = (x_1, ..., x_t, ...x_T)$ to a sequence of output vectors $(out_1, ..., out_t, ..., out_T)$. Equations (2.12) to (2.17) define the forward propagation of the LSTM layer. Illustration from [28]

An LSTM layer maps a sequence of input vectors $X = (x_1, ..., x_t, ...x_T)$ with $T$ elements to a sequence of output vectors $(out_1, ..., out_t, ..., out_T)$ with $T$ elements . The proposed gated mechanism is articulated between an internal cell state $state_t$ that embed temporal information and gates that regulate the information flow. In the following formulas, the basic matrix operators are the matrix-vector multiplication, which is an implicit operation, and the element-wise matrix product or Hadamard product, which is noted with a "$\cdot$". The output sequence $Y$ is computed in a recurrent manner by the set of following equations.

- Input activation:

$$a_t = \tanh(U_a x_t + V_a out_{t-1} + b_a) \tag{2.12}$$

- Input gate:

$$i_t = \sigma(U_i x_t + V_i out_{t-1} + b_i) \tag{2.13}$$

- Forget gate:

$$f_t = \sigma(U_f x_t + V_f out_{t-1} + b_f) \tag{2.14}$$

- Output gate:

$$o_t = \sigma(U_o x_t + V_o out_{t-1} + b_o) \tag{2.15}$$

- Cell state:

$$state_t = a_t \cdot i_t + f_t \cdot state_{t-1} \tag{2.16}$$

- Output:

$$out_t = \tanh(state_t) \cdot o_t \tag{2.17}$$

in which, $U_a$, $U_i$, $U_f$, $U_o$ are the weights matrices applied to the input $x_t$ that belong to the input activation, the input gate, the forget gate and the output gate, respectively. $V_a$, $V_i$, $V_f$, $V_o$ are weights matrices applied to the recurrent input $out_{t-1}$ that belong to the input activation, the input gate, the forget gate and the output gate, respectively. $b_a$, $b_i$, $b_f$, $b_o$ are the bias vectors that belong to the input activation, the input gate, the forget gate and the output gate, respectively. Similarly to the RNN Eq. (2.10), the same weights matrix and biases are applied on each time step. The notation $\sigma$ corresponds to the sigmoid function and $a_t$, $i_t$, $f_t$, $o_t$, $state_t$ respectively denote the input activation, the input gate, the forget gate, the output gate and the cell state at the time step $t$. The output $out_t$ has a double purpose: it is propagated throughout the sequence as the recurrent input, and it stacks across the time steps to build the output sequence $Y$ that is then propagated to the next neural layers.

The main contribution of the LSTM is the gated mechanism that articulates the internal cell state $state_t$ and the gates. As the gates are activated by a sigmoid function, they filter the information flow from the input sequence elements $x_t$ and the recurrent inputs $out_{t-1}$. Simply put, the scalars from the gates vectors range between 0 and 1 allowing the LSTM to characterize relevant information to retain when the scalar is close to 1, and, on the contrary, to cut irrelevant information when the scalar is close or equal to 0. By controlling the information flow with this mechanism, the LSTM focuses the gradient calculations on relevant information throughout the sequence. The gradients can carry information through longer time dependencies, and thus the LSTM mitigates the effects of the vanishing and exploding gradients.

Looking at the gates and input activation, each one has its own set of weights and biases that are optimised towards specific roles that structures the LSTM architecture:

- The input gate $i_t$ (Eq. (2.13)) and input activation $a_t$ (Eq. (2.12)) compose the first part of the internal cell state $state_t$ (Eq. (2.16)). The input gate filters the input activation information to be integrated by the internal cell state.

- The forget gate $f_t$ is responsible for filtering information from the recurrent input of the cell state $state_{t-1}$ to the cell state $state_t$. This gate is involved in the unrolling of gradients computation with the product rule. As such, the forget gate plays a major role in mitigating vanishing and exploding gradients.

- The output gate $o_t$ filters the information from the cell state $state_t$ to the output $out_t$.

However, this improvement comes with a higher number of parameters making the LSTM one of the most memory and computation expensive within the recurrent architectures. With the objective of reducing memory and computational needs while preserving performance, other works built on this approach and proposed memory and computation efficient recurrent architectures variations, such as the works in [20, 58].

## 2.5 Deep Neural Network compression

In order to make deep neural network viable to infer and sometimes even learn on edge, many compression and acceleration methods have been proposed in the past decade. The work of Han *et al.* [36] on CNN compression with pruning, quantization and Huffman encoding showed significant compression ratios on AlexNet and VGG-16 and motivated many other works. There are many different methods to compress neural networks such as pruning, weight sharing, matrix low rank factorization, structured matrices, quantization, knowledge distillation, and others. In this work, we limit our scope to pruning and quantization methods for RNNs and CNNs. This section introduces the compression objectives, the impact of the training data for compressing neural network and pruning and quantization methods.

### 2.5.1 Compression objective and data availability

When compressing neural networks, there is a trade-off between the compression ratio and the application performance. The higher compression ratio, the bigger the application performance drop. Traditionally, the objective of neural network compression methods is to find the best operating point among this trade-off, often being the highest compression ratio without "significant" application performance loss. Recently, several approaches propose a more hardware-oriented objective, with new hardware agnostic metric [122] or directly optimizing a target metric like latency [64].

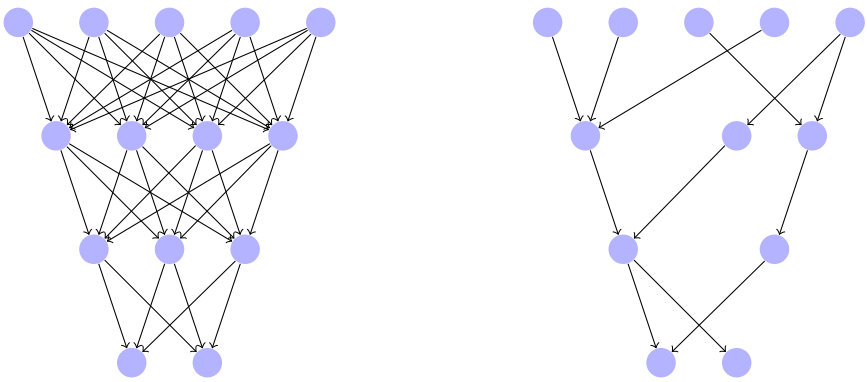
The quality of the operating point highly depends on the usage of the training data. The application training data are a valuable resource for compressing neural networks. Each compression method has its data requirements, ranging from data-free to the entire

training data. In practice, choosing a method highly depends on the availability of data. Usually, the training data are used for calibration or fine-tuning or retraining from scratch.

### 2.5.2 Pruning

As the size of neural network increases, the number of parameters increases too. However, some parameters have little to no contribution to the final result, in other words, they are redundant. Pruning is the process of removing redundant connections in a DNN, which is equivalent to setting some weights to zero.

An example of a neural network is shown on the left-hand side of Fig. 2.12. In this example, every neuron in the lower layer is connected to the neuron in the upper layer, which means that the model is heavy to store and there are a lot of operations to perform. A network is qualified as dense when there are little to no parameters equal to zero. When pruning the neural network, the redundant connections are removed, that is to say the redundant parameters are set to zero. This results in a sparse neural network, as exemplified in the right-hand side of Fig. 2.12. The memory storage of the resulting sparse model is reduced and there is a potential for reducing the amount of calculations. The sparsity of a network is the metric that measures the percentage of zero parameters with respect to the total number of parameters.



Dense neural network  Sparse neural network

Figure 2.12: Concept of pruning a deep neural network

Generally, there are two strategies for pruning a neural network: the structured pruning and the magnitude pruning (or weight pruning). In the case of convolutional neural networks, structured pruning aims at removing entire filters to gain model memory storage and directly reduce the amount of computation by bypassing the pruned filter. The magnitude pruning removes redundant parameters at the granularity of single parameters to reduce the model memory storage. Weight pruning methods often rely on a pruning

threshold to determine which parameters are replaced with zeros. Srinivas *et al.* [98] proposed a data-free method for removing redundant parameters in trained CNNs. Using training data to prune while training achieves better compressing ratios and reduces performance loss [125]. Moreover, the lottery ticket hypothesis proposed by Frankle *et al.* [25] found that sparse CNNs can be retrained from scratch. In this work, we limit our scope to magnitude pruning on RNNs.

### 2.5.3 Quantization

Quantization for deep learning is the process of approximating a neural network that uses floating-point numbers by a neural network that uses lower bit-width numbers, and potentially other number representations such as fixed-point or integer arithmetic. In general, neural networks are learned using the 32-bit, also called single-precision, or 64-bit, also called double-precision, floating-point format from the IEEE-754 standard. Quantization takes advantage of DNN resilience to small errors to relax the need for fully precise operations. This dramatically reduces both the memory bandwidth and computational cost of deep neural networks training and inference. In practice, the desired objective is to maximize the throughput of the network.

#### What to quantize?

One can quantize three main elements in a deep neural network: weights, activations and gradients. The motivation behind quantizing the weights is the fact that it can decrease the memory storage of the model, the memory bandwidth associated to the loading of the parameters, as well as the computations. For instance, the weights of a DNN can be stored as signed integers using 8 bits, instead of 32 bits with the single-precision floating-point format without any loss in application performance. In this case, the compression factor of the neural network parameter storage is $\frac{32}{8} = 4$.

The choice of which elements {weight, activation, gradient} are subject to quantization is motivated by two main objectives:

- **Accelerate inference with weight and activation quantization.** Quantizing the weights decreases the memory storage of the model as well as the memory bandwidth involved in loading the weight. Quantizing the activations further reduces the memory bandwidth involved in the inference of the neural network. When both weights and activations are quantized with a matching format, each computation uses the corresponding format. In the end, this enables higher throughput and energy savings.

- **Accelerate training with weight, activation and gradient quantization.** The weight, activation and gradient floating-point precision can be relaxed to lower

bit-width to train faster and reduce the memory needs at training time. This approach is mainly focused on the context of the data-parallel training framework of DNNs. Quantizing gradients directly reduces their representation potential and hinders their capacity to backpropagate information. That is to say, relaxing the precision of the gradients impacts the neural network capacity to learn from the data and consequently its application performance. The main existing library to enable quantization of gradients is developed by NVIDIA for GPUs and is called Apex. They propose to automatically choose the precision of weight, activation and gradient to accelerate training without impacting the application performance. So far, the quantization mainly remains in 16-bit floating point format, also called half precision. While using half precision can lead to important savings for training and inferring on Cloud services, the memory bandwidth and computing size gains are still often not sufficient to deploy neural networks on low-power Edge devices.

In this work, we focus on accelerating neural network inference with weight and activation quantization.

### Reducing precision

Reducing the number of bits used to represent the data and computations can lead to some changes in the representation of a number. Several representations can be used to represent numbers with low precision which are integers, fixed point and small floats. Here, we present the IEEE754 single precision floating-point, the fixed-point and the integer representations.

**Floating-Point** The floating-point representation regroups three sequences of bits, namely, the sign bit, the exponent and the fraction. Figure 2.13 illustrates the single-precision format using 32 bits. The representation of the floating-point number can also be encoded with 64 bits, also called double-precision, or 16 bits, also called half-precision. Small floats are customized floating-point formats with fewer bits of exponent and fraction than the half precision.

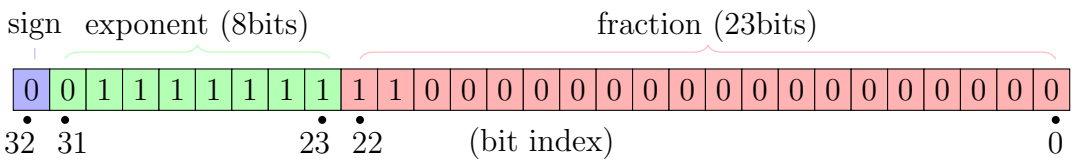



Figure 2.13: IEEE 754 single-precision format on 32 bits

The single-precision floating-point numbers are expressed from binary values following

$$number = (-1)^{sign} \times 2^{E-127} \times \left(1 + \sum_{i=1}^{23} b_{23-i} 2^{-i}\right) \tag{2.18}$$

As an example, the number shown in Fig. 2.13 is calculated as:

- $sign = b_{32} = 0$
- $E = \sum_{i=0}^{7} b_{23+i} 2^i = 2^0 + 2^1 + 2^2 + 2^3 + 2^4 + 2^5 + 2^6 = 127$
- $fraction = 1 + \sum_{i=1}^{23} b_{23-i} 2^{-i} = 1 + 2^{-1} + 2^{-2} = 1.75$
- $number = (-1)^0 \times 2^{127-127} \times 1.75 = 1.75.$

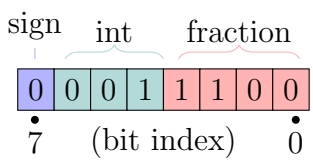



Figure 2.14: Custom fixed-point representation on 8 bits

**Fixed-Point** When fewer bits are used to express parameters, numbers can be represented using the fixed-point format. The fixed point representation regroups three sequences of bits, namely, the sign bit, the integer and the fraction parts. Figure 2.14 illustrates a custom fixed-point format to represent the same number as the floating point in Fig. 2.13 but using 8-bit fixed-point with 3 bits of integer part. The fixed point number is calculated as:

- $sign = b_7 = 0$
- $integer = 2^0 = 1$
- $fraction = 2^{-1} + 2^{-2} = 0.75$
- $number = (-1)^0 \times (1 + 0.75) = 1.75$

**Integers** The numbers can also be scaled directly to integers without fraction parts. Floating-point parameters can then be represented as integers. Figure 2.15 illustrates the 8-bit integer representation and uses the same number of bits as the previous custom fixed-point representation.

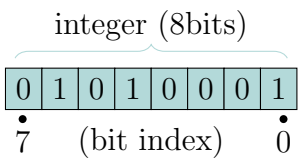



Figure 2.15: Integer representation on 8 bits

The most basic way to encode unsigned integers with binary values follows

$$number = \sum_{i=0}^{7} b_i 2^i. \tag{2.19}$$

Of course, the potential to represent a single-precision floating-point number is much higher than for an 8-bit integer. In most cases, the scaled floating-point number will not directly correspond to the one that can be represented with the integer format. A quantization function $Q$, also called quantizer, maps the non-quantized value $r$ to the quantized value $q$. The range of parameters is accounted for with two floating-point numbers to store the maximum real value $max$ and minimum real value $min$. In this example, we scale the floating point number $r$ and use the popular rounding operator resulting in uniform quantization, such as

$$Q(r) = round(\frac{r}{S}) \tag{2.20}$$

with

$$S = \frac{max - min}{2^B - 1} \tag{2.21}$$

where $B$ is the bit-width of the integer representation. For example, taking the value $r = 1.75$ in a weight distribution clipped in $[min, max]$ with $min = -2.5$ and $max = 3$, the quantized value $q$ using the 8-bit integer format is calculated as

$$q = Q(1.75) = round(1.75 \times \frac{2^8 - 1}{3 + 2.5}) = 81 \tag{2.22}$$

The floating point number $r = 1.75$ represented in Fig. 2.13 in a distribution clipped in $[-2.5, 3]$ is then represented by the 8-bit integer format as shown in Fig. 2.15 and following

$$number = 2^6 + 2^4 + 2^0 = 81. \tag{2.23}$$

In the end, the choice of the representation all comes down to the operator implementation on a target device. An implementation of an operator computes inputs with a determined number representation. For example the multiply accumulate operator has a different implementation for 8-bit integers than for single-precision floating-point inputs,

with much less hardware and energy consumption. According to the hardware design, specific implementations of operators can be created to exploit different input representations. In practice, integer implementations are more commonly integrated on general-purpose chips than fixed-point or small-float implementations. Therefore, the quantization using the integer representation is often preferred over the fixed-point representation or small floats to deploy deep neural networks on common targets.

In order to represent a single-precision floating-point number with an integer, a rounding quantization function is used. The next paragraphs further introduce quantization functions.

### Deterministic and stochastic

A quantization function is called deterministic when there is a one-to-one mapping that exists between the non-quantized value and the quantized value. There is no randomness involved in the deterministic quantization function. Given one non-quantized value, the quantization function will always output the same quantized value. The most basic deterministic quantization function is rounding. Well known contributions like Binary Connect [21], Dorefa-Net [124] and Xnor-Net [86] use the rounding function. For example, Binary Connect uses the *sign* function to quantize a real values $r$ into a one-bit quantized value $q \in \{-1, 1\}$ such as

$$q = sign(r) = \begin{cases} 1 & r \geq 0 \\ -1 & \text{otherwise.} \end{cases} \tag{2.24}$$

A quantization function is called stochastic quantization when the quantized value is sampled from the discrete distribution of its non-quantized counterpart. There is randomness involved in the stochastic quantization function. Given one non quantized value, the quantization function can output different quantized values. The most basic stochastic quantization function is random rounding. Binary Connect [21] also formulates a stochastic quantization function $Q_s$ to quantize a real values $r$ into a one-bit quantized value $q \in \{-1, 1\}$ such as

$$q = Q_s(r) = \begin{cases} 1 & \text{with probability } p = max(0, min(1, \frac{r+1}{2}))) \\ -1 & \text{with probability } 1 - p. \end{cases} \tag{2.25}$$

### Vector quantization and non-uniform quantizers

The two quantization functions in Section 2.5.3 map one real value to one quantized value and the quantization levels are uniformly spaced. This type of quantizer is referred to as uniform scalar quantization. The quantization process induces a loss of information when relaxing the number of bits, and several works proposed to reduce this loss of information.

Using non-uniform spaced quantization levels like logarithmic quantization can cover wide ranges using fewer bits than uniform quantization. In a base-two logarithmic representation, parameters are quantized into powers of two with a scaling factor. Lee *et al.* [62] quantized CNNs with weights encoded in a four-bit logarithmic format and proposed an inference engine based on bitshift-add convolutions.

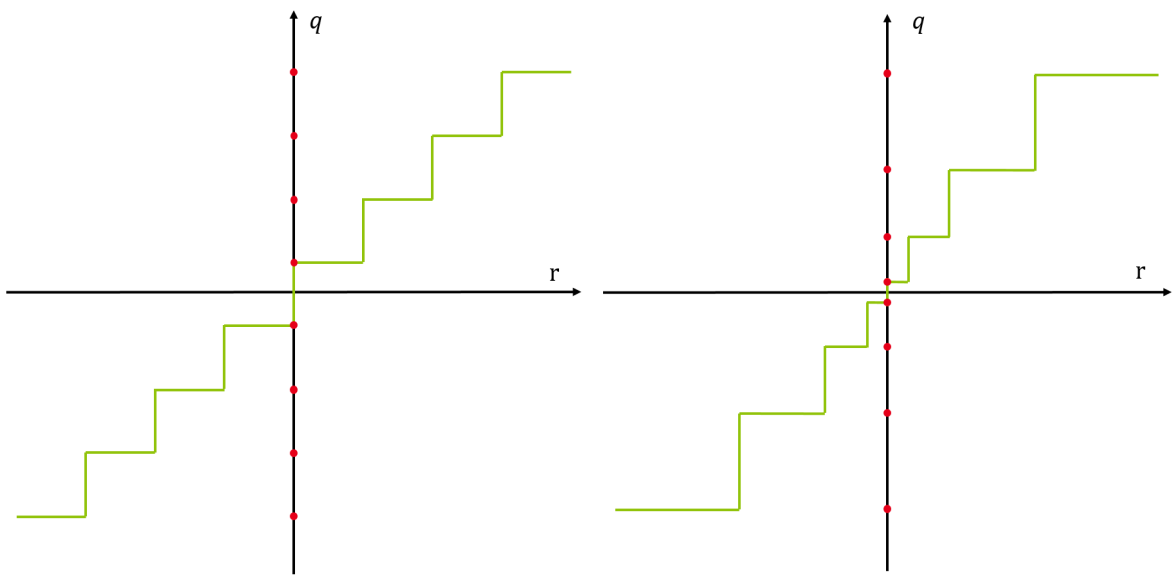



Figure 2.16: Quantization points with uniform scalar quantization (left) and non-uniform scalar quantization in the case of logarithmic quantization (right). Real values in the continuous domain $r$ are mapped into discrete quantized values $q$, which are the red bullets. Note that the distances between the quantized values (quantization levels) are the same in uniform quantization, whereas they can vary in non-uniform quantization

Instead of applying the quantization function in an element-wise manner, vector quantization applies the quantization function on clusters of parameters. Gong *et al.* [30] replaced weights by the centroids of the k-mean clusters during inference. Stock *et al.* [99] introduced a vector quantization method to compress weights using codebooks that are optimized to minimize the layer output reconstruction error. Such methods lead to more flexible quantization sets and often achieve better compression ratios when compared to scalar quantization.

### Quantization methods

According to their hypothesis on the training data availability, the quantization methods can be classified into two main categories, Post Training Quantization (PTQ) and Quantization Aware Training (QAT).

Post Training Quantization methods do not rely on training data or use a limited quantity of training data. Nagel *et al.* [75] proposed a data-free quantization method relying on weight equalization and bias correction which does not require any fine-tuning. Banner *et al.* [8] used per-channel quantization to allow more flexible quantization and used a batch of data to calibrate activation quantization thresholds. Then, Nagel *et al.* [74] proposed to use limited data to fine-tune one layer at a time in order to improve quantization locally.

Quantization Aware Training methods rely on the full training data to retrain the model from scratch with stochastic gradient descent. Given a network $f\colon \mathbb{R}^n \Rightarrow \mathbb{R}$ with its parameters $W$, an input $x \in \mathbb{R}^n$ and its corresponding label $y$, we refer to QAT for classification as finding the non-differentiable quantization function $q$ with the loss function $L$ as

$$\min_W L[f(x, Q(W)), y]. \tag{2.26}$$

In order to enable Quantization Aware Training as illustrated in Fig. 2.17, two tools are most commonly used:

- Straight-Through Estimator (STE). Bengio *et al.* proposed the Straight-Through Estimator (STE) to enable training with backpropagation [10]. The STE method estimates the gradients of the quantized parameters assuming that the derivative of the quantization function $Q$ is the identity function.

- Fake quantization. Instead of using a different format like integers to represent the quantized values, the quantized values are represented in a finite set of values using the floating-point format that is called "fake" quantized values. All weights and activations subjected to quantization have both a non-quantized value represented with single-precision floating-point and a "fake" quantized value, a value taken from a finite set represented in floating-point. The target quantization bit-width determines the number of quantization levels, i.e., the number of values in this finite set. For example, fake quantized values on two bits would have a finite set of $2^2 = 4$ values. At each inference, the quantization function is applied on the non-quantized value to obtain the fake quantized values. These fake quantized values are then used to infer the network and obtain a prediction. During backpropagation, the gradients are calculated using the fake quantized values of weights and activations

subjected to quantization and accumulated in single-precision floating-point. Finally, the parameters are updated using the non-quantized value.

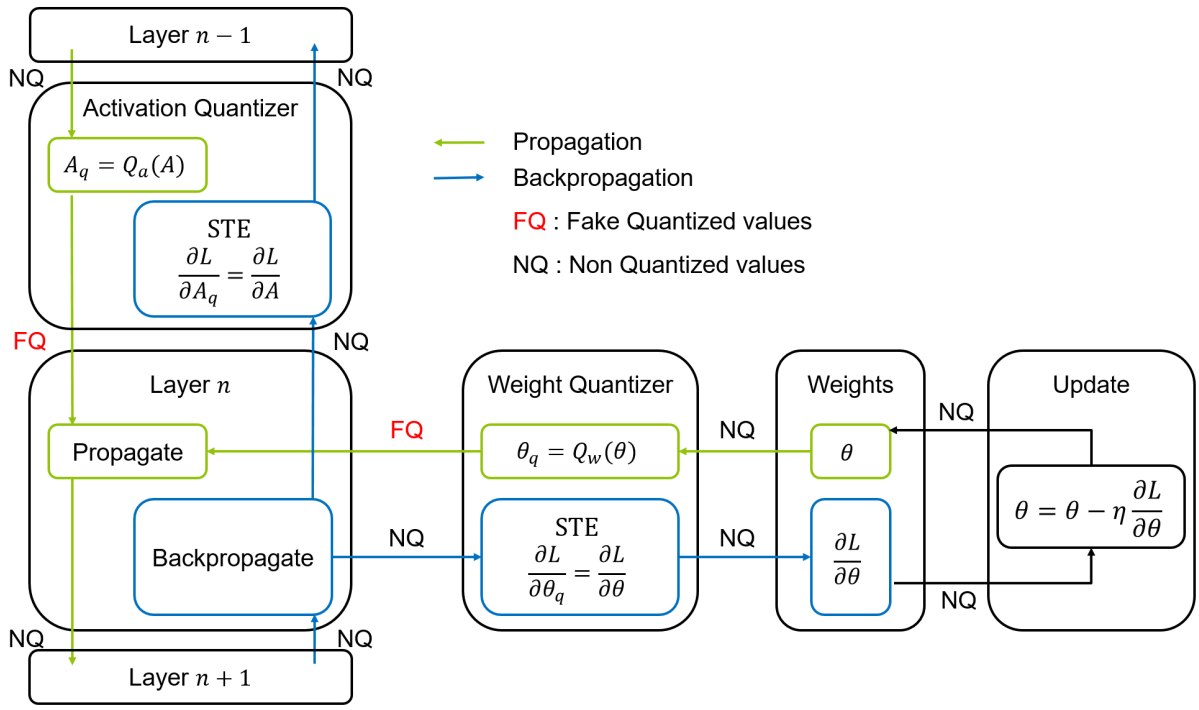



Figure 2.17: Example of one layer propagation and backpropagation during Quantization Aware Training. The Straight-Through-Estimator (STE) estimates the derivative of quantization functions. NQ stands for Non-Quantized values and refers to real values represented with the single-precision floating-point format. FQ stands for Fake Quantized values and refers to the finite set of values representing quantized levels in single-precision floating-point. $\eta$ is the learning rate used to perform the update.

Quantization Aware Training methods better exploit the domain information and normally achieve better compression over performance ratios. However, the quantized models still suffer from significant application performance reduction. The question arises whether the application performance reduction is due to the reduced capacity of quantized models or to the non-suitable quantization procedure. Following the latter, the approximation error made by STE grows bigger as the bit-width goes smaller, hence decreasing the performance for low-bit settings. Esser *et al.* tackled this issue by scaling dynamically the gradients with a learnable step [23]. Following their method, the gradient landscape is shaped to encourage the full precision parameters towards the quantized points. Doing so, the proposed Learned Step Size Quantization (LSQ) method implicitly reduces the approximation error introduced by the STE and shows substantially better results over the previous quantization techniques. Alternatively, the Scaled Adjust Training (SAT) method introduced by Jin *et al.* directly scales the weights instead of the gradients to control the training dynamics, which yields state-of-the-art results [48].

## 2.6 Conclusion

In this chapter, we first introduce several deep learning concepts. Section 2.1 explains the importance of data quality and its influence on a model generalization or over-fitting. Different bricks that enable training neural networks such as initialization, cost functions, data augmentation, backpropagation and stochastic gradient descent are presented and Section 2.1.6 combines those into a typical pipeline of a supervised training. In many niche applications, data quantity and/or quality are the bottlenecks to a machine learning approach. Section 2.1.7 introduces transfer learning and how it can cope with this issue by transferring the knowledge of a given DNN to others. Section 2.1.8 discusses the most popular research direction consisting in scaling up models for better performance. One of the main lever to scale up models is to increase their depth, that is to say to increase the number of consecutive layers. Training deeper neural network faces numerical instabilities that are introduced in Section 2.1.9. Despite many works pursuing this legacy, another research direction, causal learning, focus on the reasoning ability of the model instead of the performance only.

We then present different types of neural network. Section 2.2 introduces the multi-layer perceptron composed of multiple fully connected layers and non-linear activation functions. The densely connected layers imply many parameters. Section 2.3 presents convolutional neural networks composed of convolutional layers, pooling layers and batch normalization layers. As one ground-breaking innovation for training deeper networks, the Resnet architecture is detailed. Then, Section 2.4 introduces Recurrent Neural Networks, their historical context, the adaptation of the backpropagation algorithm for the recurrent paradigm and its impact on the numerical instabilities during training. The LSTM is then detailed as it is the recurrent architecture that allows to achieve practical results on many tasks.

Despite the success of deep neural networks in many applications, they remain difficult to run on embedded devices with challenges such as memory bandwidth, computing resources, energy efficiency and latency. Therefore, we finally present the two most popular state-of-the-art solutions for compression of neural networks, pruning and quantization. The pruning method removes connections of neural networks based on an evaluation of each connection importance. Pruning methods are separated between two strategies: structured pruning and magnitude pruning (or weight pruning). While both allow significant compression of model parameters, the structured pruning strategy is known to allow better acceleration. Meanwhile, in [25], Frankle *et al.* observed that sparse CNN can be trained from scratch using a method based on magnitude pruning. Doing so, it opened new perspectives for CNN optimization. A similar study is yet to be conducted on RNN

and we propose to extend their experimentation scope on RNN in Chapter 4.

Quantization is the process of approximating a neural network, from values expressed with a high number of bits to values expressed with fewer bits. Sometimes this process involves changing the values representation and we introduced three common representations used for DNN: floating-point, fixed-point and integers. Quantization allows to compress the model storage and reduces both the memory bandwidth and computational cost of deep neural networks. Different purposes such as accelerating inference or accelerating training determine which components to quantize, *e.g.*, relax the precision of weights and activation for accelerating inference. The methods that yield state-of-the-art results for CNN quantization down to 2 bits involve training the models with fake quantized values and the whole database. The approximation error made to train those quantized values grows bigger as the target precision is lower, and thus, hinders the optimization of quantized networks. In Chapter 5, we propose to improve those methods with quantization friendly loss functions to enable the quantization of CNN down to binary parameters.

# Chapter 3

# Deep Learning Tools

## 3.1 Libraries

Advances in deep learning have also depended heavily on advances in software infrastructure. Software libraries such as Scikit Learn [83], Theano [12, 9], Caffe [47] TensorFlow [1], Pytorch [82, 52] and N2D2 [14] have all supported important research projects or commercial products. We first present deep learning frameworks, deployment libraries and N2D2 then we introduce the benchmark datasets used in this work.

### 3.1.1 Deep Learning Frameworks

Most of the operations in neural networks are modular, and they are composed of computational primitives, such as matrix multiplications or convolutions. Machine learning frameworks encapsulate those basic operators to build the backbone for neural network learning. These tools are often open-source, meaning that the source code is freely available for possible modifications. Such a quality facilitates research developments and new implementations for novel ideas. Frameworks provide different sets of features to further improve the accessibility to deep learning tools:

- Accelerated operators on the single instruction multiple data architectures like Graphic Processing Units. Those operators are either coded from scratch in Cuda or used from an existing library like Cuda Basic Linear Algebra Subprograms (CuBLAS) or Cuda Deep Neural Network (CuDNN).

- Automatic differentiation of basic differentiable operators that automate the computation of gradients in order to train using the gradient descent optimization algorithm. From a user perspective, this feature greatly unravels the complexity of deep neural network training.

- Distributed training across multi-node and/or multi-GPU systems. With this feature, the training can be performed with larger batches thanks to the stacked memories

of each device. When it is done correctly, training with bigger batches enables a smoother convergence and can lead to better results. Also, scaling up this process to many nodes can drastically reduce the training time.

- Tools for loading, pre-processing and augmenting different type of data.

In this section, the two main frameworks, Pytorch and TensorFlow, are introduced. It should be noted that the frameworks of neural networks are difficult to compare with each other, which means that there is no best or worst framework, but one needs to make choices based on the purpose of tasks. Meanwhile, the framework of the neural network is a commercial technology, they are also developing rapidly, or are gradually eliminated by users. This is just an overview of the framework as of the time of writing.

### Pytorch - a framework for research

PyTorch is an open-source deep learning framework mainly based on Python, C++ and Cuda. It was created by Facebook and is currently widely used in research and industry. Here are some of its features:

- Pytorch provides accelerated operators on GPU.
- With the automatic differentiation module "autograd", neural networks design is done dynamically without having to pre-define a static network diagram to perform calculations.
- Pytorch provides the "DistributedDataParallel" library for training across multi-node and multi-GPU systems.
- Several Pytorch libraries are available to load, pre-process and augment well-known datasets and custom data of different types like torchvision for images, torchtext for text, torchaudio for audio signals.
- Pytorch is also compatible with Python libraries such as NumPy and SciPy.

However, Pytorch features to compress and deploy neural networks on edge like QPytorch only propose limited solutions compared to other efforts. Pytorch's reputation is to be easy to use, the many tutorials proposed on the official website and the support provided on both git and Pytorch forums make it attractive for new users. Consequently, PyTorch has quickly become the mainstream deep learning framework in academia.

### TensorFlow - a framework for cloud deployment

Often put in competition with Pytorch, TensorFlow is one of the most popular deep learning frameworks today. This is an open-source framework developed and maintained

by Google. Many famous groups such as Gmail, Uber, Airbnb, Nvidia are using it. Here are some of its features:

- TensorFlow provides accelerated operators on GPU and Tensor Processing Units and an automatic differentiation mechanism.
- TensorFlow provides the "distribute" library for training across multi-node and multi-GPU systems.
- TensorFlow has visualization tools like TensorBoard that help to see the structure of neural network graphs and the distribution of data, which facilitates neural networks design.
- TensorFlow not only can deploy on powerful computing clusters (with the TensorFlow Serving library), but also on mobile platforms such as iOS, Android and Raspberry Pi with the TensorFlow Lite library.

TensorFlow recently transitioned to a new version TensorFlow 2.0 to improve the user experience and the developer productivity. To that end, the high-level API Keras was added to build and train models more easily. However, this transition resulted in major compatibility issues and many open-source contributions based on the previous version on github are hardly still exploitable.

Instead of GPUs, training and inference using Tensorflow can be achieved on Tensor Processing Units, a hardware architecture developed by Google that is specialized into computing the Matrix Accumulate operator.

TensorFlow is widely used in the industrial field and, as Google continues to optimize and improve the framework, it will become more powerful and easier to use.

### 3.1.2 Libraries for deploying neural networks on the Edge

There is a growing need to execute neural networks on edge devices to reduce latency, preserve privacy, and enable new interactive use cases. Designing and deploying a neural network on the Edge is a complex task that requires expertise from both software and hardware fields. To achieve this type of deployment, the most mainstream approach is a two-step process. First, compression and acceleration techniques are applied on a neural network to make it viable in terms of memory and computing size for embedded devices. Then, the code for deep neural networks execution on one or multiple edge targets is generated.

Several libraries have been developed and propose technical blocks to help automate this process:

- Apache Tensor Virtual Machine (TVM) is an open-source deep learning compiler for CPUs, GPUs, and machine learning accelerators. It aims to enable machine learning

engineers to optimize and run computations efficiently on any hardware backend. The currently supported targets are Android, Jetson Nano, Raspberry Pi, CPU and microcontrollers like the STM32.

- TensorFlow Lite is a library for easily deploying TensorFlow models on mobile, microcontroller and Raspberry Pi targets.
- TensorRT is a software development kit that facilitates high-performance machine learning inference. It focuses specifically on deploying and running an already-trained network quickly and efficiently on NVIDIA hardware such as the Jetson Nano.
- Xilinx FINN is a dataflow compiler for quantized neural network inference on FPGAs.
- Pytorch Mobile (beta version) is a library for easily deploying Pytorch models on mobile and microcontroller targets.

Another notable approach for neural network execution on the Edge is the design of a specific neural network accelerator ASIC like PNeuro [16]. In other words, there are many different efforts to help reduce the gap between neural network design and neural network on edge.

### 3.1.3 N2D2 - Neural Network Design & Deployment

Neural Network Design & Deployment (N2D2) [14] is an open-source deep learning framework that focuses on building full DNN-based applications on embedded platforms. It is developed along with industrial and academic partners in the Embedded Artificial Intelligence Laboratory at the french research institute CEA-LIST. Its objective is to become a hub that centralizes and normalizes processes for deep neural network deployment on the Edge. As such, it is the first European deep learning hub for developing and deploying DNN. The code is accessible on github, together with its documentation.

The N2D2 framework tackles this complex task with an approach combining many tools from the data conditioning to the code generation for embedded inference on different targets, as illustrated in Fig. 3.1.

The main features of N2D2 are:

- Several data drivers to load, pre-process and augment well-known datasets (such as ImageNet, KITTI, Cityscapes, DOTA) and custom data.
- Accelerated operators for training on GPU.
- ONNX imports and exports to provide an intuitive interface with other libraries and frameworks.

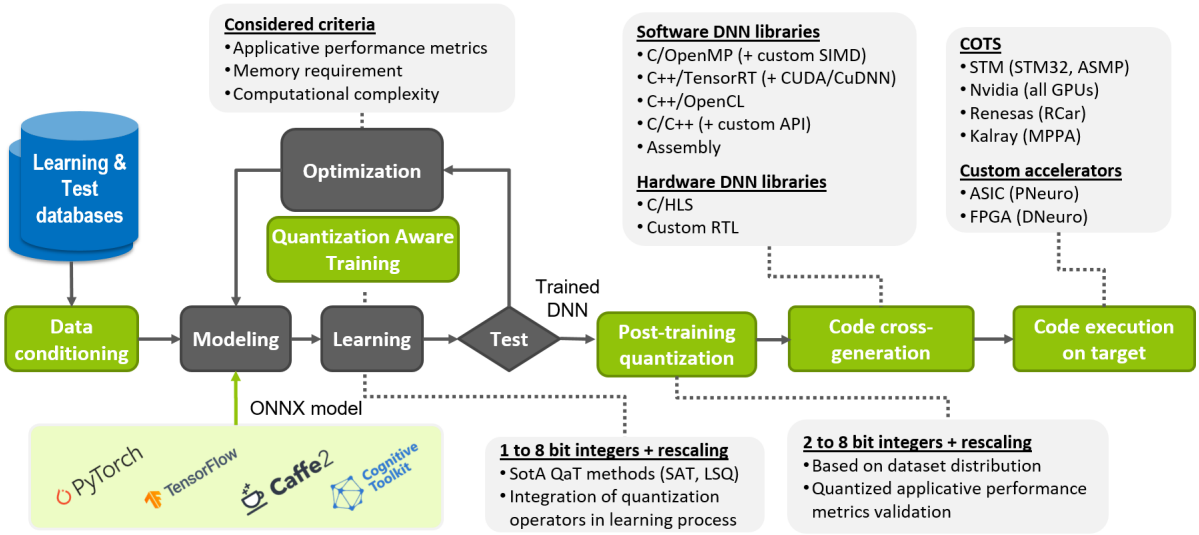



Figure 3.1: Overview of the N2D2 Framework

- A Python API that provides a user-friendly interface for neural network design with the efficiency of C++ and Cuda for training.
- The continuous integration of state-of-the-art compression and acceleration tools like the advanced quantization aware training method Scaled Adjust Training [48].
- A strong diversity of hardware targets and associated code generation tools that enable flexibility for neural network deployment.

Thanks to this unique workflow proposition, N2D2 is able to reach competitive solutions and is sure to grow as a major actor to the edge AI industrial ecosystem. In this work, we use N2D2 to support the experimentation of Chapter 5 as it allows to implement quantization tools with precise control over computation and memory overhead while training on GPU.

## 3.2 Benchmark Datasets

This section presents the tasks and datasets used throughout this research project. In this work, two tasks are considered: image classification and language modelling.

Most of the experiments are carried out on the image classification task. It is generally believed that image classification is an easier task in the field of computer vision. Given an input image, the aim of image classification is to determine the category of the image.

On the other hand, language modeling is the task of predicting the next word or character in a document. This technique can be used to train language models that can further be applied to a wide range of natural language tasks like text generation, text classification, and question answering.

### 3.2.1 MNIST

The Modified National Institute of Standards and Technology (MNIST) handwritten digits database was introduced by LeCun *et al.* [61]. The dataset includes 10 classes for handwritten digits ranging from 0 to 9. It contains a training set of $60,000$ labeled examples and a test set of $10,000$ labeled examples for a total of 6000 handwritten digits per class. The digits have been size-normalized and centered in a fixed-size image of $28 \times 28$ pixels. Figure 3.2 displays some examples. It is the most basic dataset to perform image classification.

Figure 3.2: MNIST handwritten digits examples

In Chapter 4, we use a variation called Sequential MNIST that considers each $28 \times 28$ handwritten digit image as a sequence of 784 pixels. It is used to measure how well recurrent architectures can learn long-term dependencies.

### 3.2.2 CIFAR-10 & CIFAR-100

The CIFAR-10 and CIFAR-100 datasets are labeled subsets of the 80 millions tiny image dataset. They were collected by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton [54]. The CIFAR-10 and CIFAR-100 datasets [54] are used for image classification tasks containing RGB images of $32 \times 32$ pixels with respectively 10 and 100 classes. Each dataset contains $50,000$ training labeled images and $10,000$ test labeled images for a total of 6000 images per class for CIFAR-10 and 600 images per class for CIFAR-100. Figure 3.3 displays some image examples from CIFAR-10.

In Chapter 5, we use **CIFAR-10** and **CIFAR-100** for preliminary experiments. We then use **CIFAR-100** to evaluate neural networks robustness towards adversarial attacks in Chapter 6.

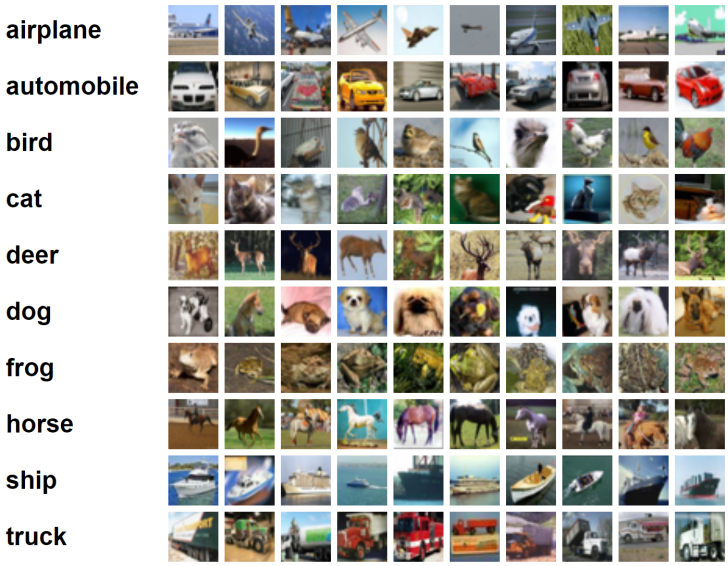



Figure 3.3: CIFAR-10 image examples

### 3.2.3 ImageNet-1k

The database **ImageNet-1k** refers to the database used for the ImageNet Large Scale Visual Recognition Challenge 2012 (ILSVRC 2012). The training data was extracted from the large hand-labeled ImageNet dataset and contains 1000 classes and 1.2 million images.

In this work, we use **ImageNet-1k** to benchmark our method for quantization aware training and compare it to other state of the art methods.

### 3.2.4 Wikitext-2

The WikiText-2 is a subset of the WikiText database introduced by Merity *et al.* [72]. The data was extracted from the set of verified Good and Featured articles on Wikipedia. It contains a training set with more than 2 millions of tokens, a validation set with about 218k tokens and a test set with about 246k tokens. The validation set is used to evaluate the model performance during training while the test set is used to assess the performance of the converged model. The WikiText-2 database features a vocabulary size of around 33k and retains the original case, punctuation, and numbers.

In Chapter 4, we use the WikiText-2 database for a language modelling task. An LSTM model is used to predict the next word based on the previous set of words.

# Chapter 4

# Compressing Recurrent Neural Networks

## 4.1 Introduction

The basic concept of RNNs presented in Section 2.4.1 is an intuitive approach to model complex time dependencies: the temporal information is recurrently updated within a state. Training recurrent architectures to learn complex time dependencies is however very difficult due to vanishing and exploding gradients. At first, it is not easy to understand the learning dynamics of the recurrent architectures. For instance, it took several years of research to propose dedicated initialization [50], normalization [6] and regularization techniques [119, 56, 126]. Also, the update mechanism RMSProp and Adam are generally found to allow smoother and faster convergence of RNNs than the basic SGD.

The Long Short Term Memory marked a turning point in the field as it provided ground breaking performance on many different applications such as machine translation [7, 102], image captioning [109], hand writing recognition [33], question answering [112], speech recognition [34, 90] and more. However, the gated mechanism in the LSTM introduces a lot of parameters and makes the recurrent architecture a lot more computation expensive. A first approach to tackle this drawback is to propose lighter variations of recurrent architecture built on the gated mechanism like the Gated Recurrent Unit [20] and the Fast Gated Recurrent Neural Network [58]. Another approach aims at compressing RNN by reducing the total number of parameters, using pruning methods for instance.

The contribution of this chapter is threefold:

- We investigate practical RNN learning behaviour.
- We extend the experimental part from the works of Frankle *et al.* [25, 26], an iterative pruning and retraining technique, with LSTM.

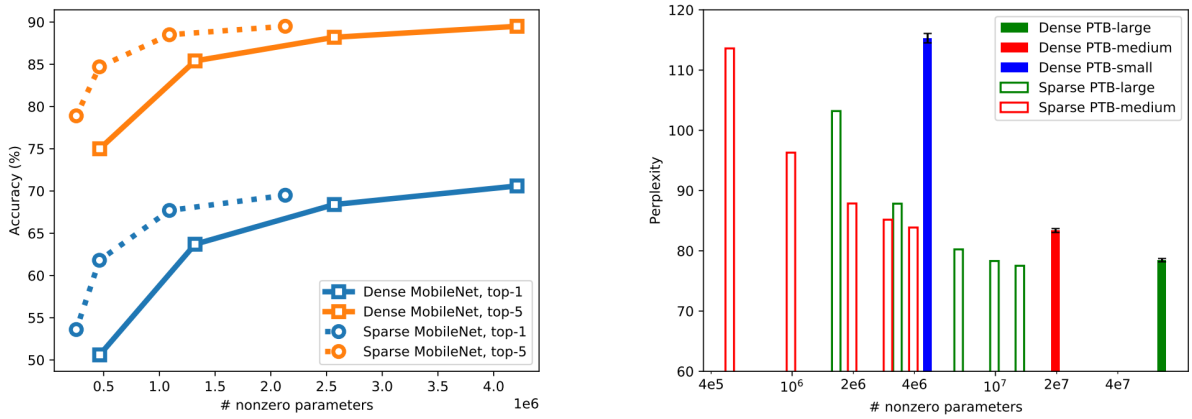
- We introduce a pre-processing method based on data sub-sampling that enables faster convergence of LSTM, while preserving application performance on Sequential MNIST.

This chapter is organized as follows. Section 4.2 introduces a weight pruning method and the main empirical observation in the domain. Section 4.3 presents the lottery ticket hypothesis and motivates its investigation for RNN. Section 4.4 details different configurations of the finding winning ticket algorithm that are used in the experimentation. The experiments on Sequential MNIST are presented in Section 4.5 and motivate the pre-processing method introduced in Section 4.5.4. Section 4.6 then studies RNN using the wikitext-2 language modelling task. Each dataset experiment is developed through three parts. First, we explain the task and training setup, then we analyse a dense LSTM convergence profile, and we finally apply relevant pruning algorithm configurations to provide insight on RNN learning behaviour. Finally, some perspectives are discussed in Section 4.7.

## 4.2 Sparse and dense models

In this section, we present a weight pruning method and the main empirical observation in the domain. Pruning DNNs is the process that removes redundant connections. This method showed impressive compression ratios without performance loss on Deep Convolutional Neural Networks [36]. Similar results were found by Narang *et al.* [76] while pruning RNNs.

Moreover, Zhu and Gupta [125] proposed a method to gradually prune the network parameters in a element-wise fashion during training to achieve a defined pruning objective. Doing so, the network is trained to gradually increase the sparsity of the network while allowing the network training steps to recover from any pruning-induced loss in performance. Their experiments covers both convolutional and recurrent architectures on three different tasks : image classification, language modelling and machine translation. Figure 4.1 reports their results on image classification with a convolutional neural network and language modelling with a recurrent neural network. Dense models refers to models that are not pruned, *i.e.* all parameters are non-zero parameters. Sparse models refers to models that are pruned, *i.e.* part of the parameters are zeroes. First, they confirmed the trade-off between model accuracy and model size encountered while pruning models. Secondly, both experiments (a) and (b) show that sparse models outperform comparably-sized dense models.




(a) (b)

Figure 4.1: Performance comparison of sparse and dense models. In (a) the performance of the convolutional neural networks are measured using the top-1 accuracy (the higher the better) on Imagenet. In (b) the performance of the LSTM-based models are measured using the perplexity (the lower the better) on Penn Treebank. Both experiments (a) and (b) show that sparse models outperform comparably-sized dense models. The figures are taken from [125].

## 4.3 The Lottery Ticket Hypothesis

In 2018 and 2019, Frankle *et al.* [25, 26] studied the correlation between initialization and pruning of Feed Forward Neural Networks that provides insight on their learning behaviour. Specifically, their Lottery Ticket Hypothesis conjectures that typical neural networks contain small sub-networks that can train to similar accuracy in a commensurate number of steps. They proposed a method to find Winning Tickets (WT), *i.e.*, sparse network initialization that can converge to similar or even better performance than the dense network.

Inspired by the works of Frankle *et al.* [25, 26], this chapter conveys insight on RNN learning behaviour rather than showing opportunities to directly improve performance or the above-mentioned trade-off. Towards that objective, we extend the experimental scope of their works with RNN in order to better understand their learning behaviour.

## 4.4 Experimental protocol

The notations used for the Algorithm 2 are listed in Table 4.1.

In order to find winning tickets for Feed Forward Neural networks, Frankle *et al.* proposed an iterative pruning and retraining algorithm that can be implemented following Algorithm 2. Algorithm 3 details the implementation of the magnitude pruning algorithm.

Those algorithms are applied on an LSTM for Sequential MNIST classification task in Section 4.5 and for Word prediction on Wikitext-2 in Section 4.6. The following method

**Algorithm 2** Finding Winning Tickets

1: **Inputs:** a neural network $f$ and its parameters $\theta$, a list of $K$ pruning objectives $P_{obj} = (p_1, .., p_k, .., p_K)$, the parameters save iteration $t_{save}$ that is used as initialization for each retraining and the total number of training iterations $T$.
2: **Outputs:** a list of $K$ masks $M = (m_1, ..m_k, .., m_K)$ corresponding to each pruning objective. Each mask $m_k$ unravels the sparse parameters $m_k \circ \theta$ corresponding to the pruning objective $p_k$.
3: **Winning tickets**($f$,$\theta$,$P_{obj}$,$t_{save}$,$T$):
4: $m_0 \leftarrow$mask filled with ones.
5: Randomly initialize the neural network parameters as $\theta^0$.
6: Train the network for t iterations and save its parameters as $\theta^{t_{save}}$.
7: For $k$ in range(1..$K$):
8: From $f(m_{k-1} \circ \theta^t)$ train for $T - t_{save}$ iterations.
9: Obtain the converged parameters $\theta_k^T$.
10: $m_k \leftarrow$ Pruning($\theta_k^T$,$p_k$)
11: **return** $M = (m_1, .., m_k, .., m_K)$

**Algorithm 3** Magnitude Pruning method

1: **Inputs:** a set of parameters $\theta$ to prune, a pruning objective $p \in [0, 1]$ and the threshold step $\epsilon$.
2: **Outputs:** a mask $m$ filtering pruned parameters with element-wise product: $m \circ \theta$.
3: **Pruning**($\theta$,$p$,$\epsilon$):
4: $threshold \leftarrow 0$
5: $s \leftarrow 0$
6: While $s < p$:
7: $m \leftarrow \theta \geqslant threshold$
8: $s \leftarrow \frac{Number\ of\ zeros\ in\ m}{Total\ number\ of\ parameters\ in\ \theta}$
9: $threshold \leftarrow threshold + \epsilon$
10: **return** $m$

| Notation | Description |
|---|---|
| f | neural network model |
| $\theta$ | parameters of the model f |
| $P_{obj} = (p_1, .., p_k, .., p_K)$ | a list of K pruning objective defining the iterative pruning steps |
| $M = (m_1, .., m_k, .., m_K)$ | a list of K binary masks corresponding to each pruning objective. |
| $T$ | total number of training iterations |
| $t_{save}$ | save the parameters of the network $f$ at iteration $t_{save}$ to be used as initialization at each retraining |
| $\theta^{t_{save}}$ | saved parameters at training iteration $t_{save}$ from the first training |
| $\theta_k^T$ | converged parameters $k$ |
| $\theta_{rand}$ | random initialization of parameters |

Table 4.1: Notations for the winning ticket algorithm.

configurations are compared:

**Winning Tickets.** Iterative pruning and retraining from scratch using the same initialization. In this case, the save iteration $t_{save} = 0$, that is to say each retraining starts from the same initialization $\theta^0$. This configuration is the original method proposed in their first work [25].

**Winning Tickets with save.** Iterative pruning and retraining from scratch using as initialization an early save of the parameters $\theta^{t_{save}}$ from the first training. This configuration is the upgraded method proposed in follow-up work [26] in order to stabilize their algorithm.

**Rand_init.** Iterative pruning and retraining from scratch using a new random initialization. In this case, line 6 of Algorithm 2 is ignored and line 8 is replaced by:

From $f(m_{k-1} \circ \theta_{rand})$ train for $T$ iterations.

with $\theta_{rand}$ a new random initialization at each retraining. Similarly to Frankle *et al.* [25], this configuration is used to assess the importance of a winning ticket's initialization.

**Iterative Magnitude Pruning (IMP).** Iterative pruning during learning. The *Finding Winning Tickets* algorithm is adapted to become a naive Iterative Magnitude Pruning. Instead of retraining from scratch with a re-initialisation, the network continues learning with its current parameters. In this case, line 6 of Algorithm 2 is ignored and line 8 is replaced by:

From $f(m_{k-1} \circ \theta_{k-1}^T)$ train for $T$ iterations.

with $\theta_{k-1}^T$ the parameters from the previous converged training and $\theta_{k-1}^T = \theta^0$ when $k = 1$. This configuration can be interpreted as the edge case between the *Finding Winning Tickets* algorithm and *Iterative Magnitude Pruning* methods. It gives a first idea of the

performance that could be obtained with more elaborated IMP methods. Note that from all the configurations, it is the only one that does not retrain from an earlier state of the network.

## 4.5 Handwritten digits recognition

### 4.5.1 Task and training setup.

The LSTM architecture [42] is trained on the **MNIST**[61] handwritten digits classification task. Each $28 \times 28$ 2D image is flattened into a sequence of 784 pixels. Figure 4.2 shows the inference pipeline on an LSTM, each pixel is sequentially fed into the LSTM and only the last output is used to make a prediction with a linear classifier. The network is composed

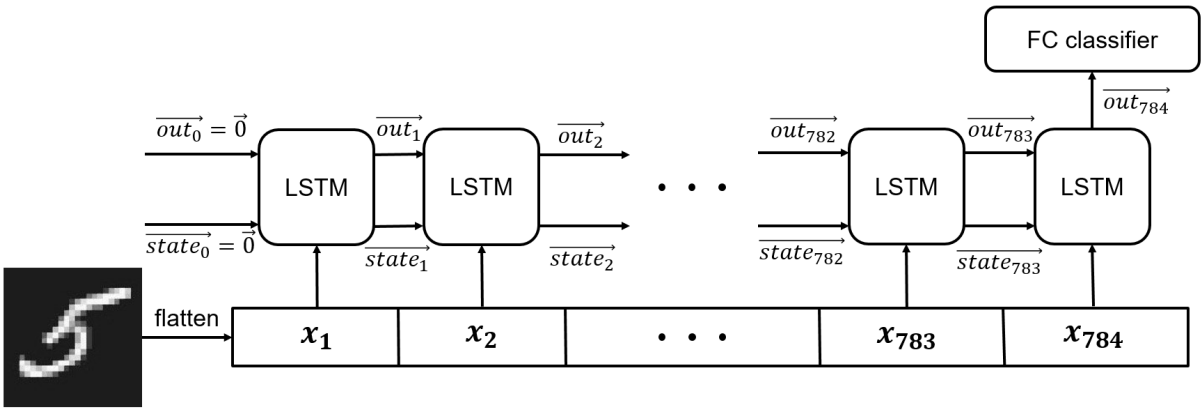



Figure 4.2: LSTM inference pipeline on Sequential MNIST.

of a single LSTM layer with a hidden dimension of 128 and a fully connected layer $128 \times 10$ for a total of 68k parameters. Standardization is applied on each image. Each training uses the Cross Entropy Loss (CEL) objective function, the RMSProp optimizer [104] with a learning rate of $10^{-3}$, $120k$ iterations with a batch size of 100 (*i.e.* 200 epochs). No Regularization is applied.

The performance of each network is evaluated by their top-1 accuracy on MNIST and the compression of the network is assessed by the sparsity percentage of the parameters, *i.e.* the percentage of zero parameters in the network.

### 4.5.2 Convergence on Sequential MNIST

In this section, we investigate the convergence of one dense LSTM model on Sequential MNIST. Our model reproduces a similar top-1 accuracy of the LSTM from the work of Arjovsky *et al.* [3], and both scores are reported in Table 4.2. We then use this model as the baseline performance to compare to sparse models.

| Model | Top-1 Accuracy |
|---|---|
| LSTM [3] | 98.2 |
| LSTM (Ours) | 98.9 |

Table 4.2: LSTM performance on Sequential MNIST.

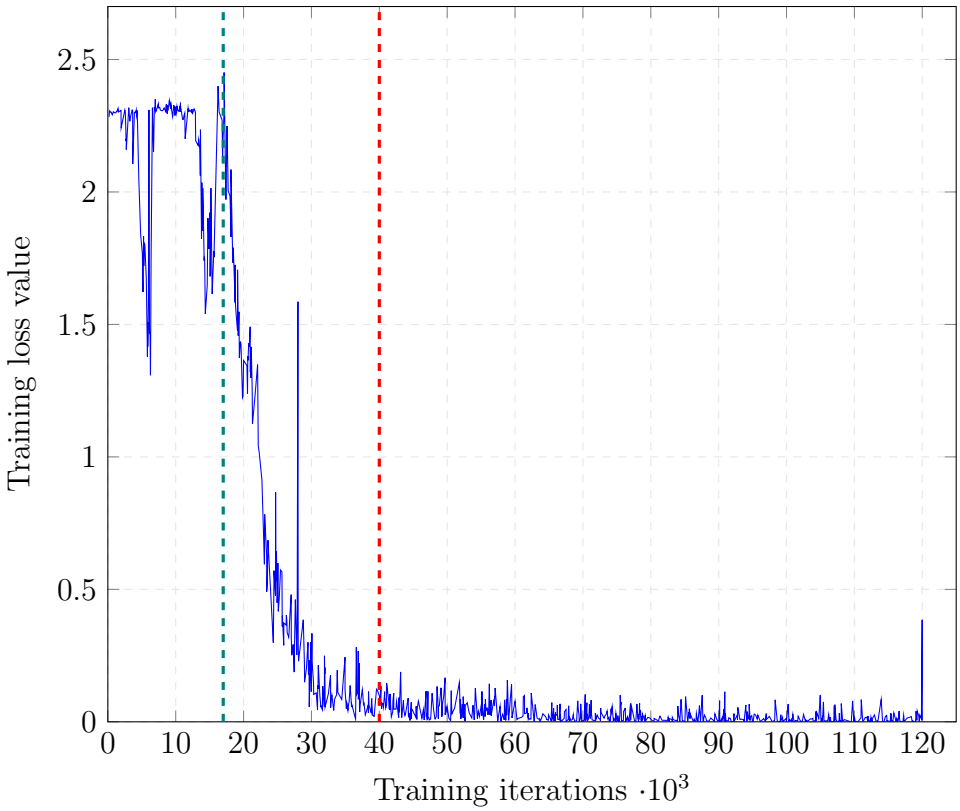



Figure 4.3: LSTM training loss on sequential MNIST. The convergence is unstable.

The training loss convergence associated with our LSTM score is plotted on Fig. 4.3. The loss convergence profile is divided into three parts.

1. From 0 to around 17k iterations (highlighted by the green dashline on Fig. 4.3), the loss does not converge. On two occasions around 5k and 15k iterations, the network starts to converge but then quickly setback.

2. Then, from around 17k iterations to 40k iterations (highlighted by the red dashline on Fig. 4.3), the loss profile shows a clear convergence. Despite this converging tendency, the loss is very unstable. In particular, one can notice the loss pick around 27k iterations.

3. In the last 80k iterations, the loss reaches an asymptote.

This training instability is explained by the Vanishing and Exploding gradients as the Backpropagation Through Time algorithm has to back propagate through the long

784-pixel sequence. Note that the gated mechanism introduced by the LSTM structure only mitigates the recurrent architecture sensitivity to those phenomenons.

### 4.5.3 Lottery ticket experiments

In this section, sparse LSTM models are learned with the following configurations: Rand_init, Winning Tickets (WT), WT with save at iteration 1200 (*i.e.* the second epoch), WT with save at iteration 30k (*i.e.* epoch 50) and the Iterative Magnitude Pruning. The two save iterations were chosen according to the convergence of the LSTM from Fig. 4.3. At iteration 1200 the LSTM is not converging yet while it is converging at iteration 30k. All pruning experiments follow the same iterative pruning objectives $P_{obj} = (0.3,\ 0.5,\ 0.7,\ 0.8,\ 0.9,\ 0.95)$. We found that using a threshold step $\epsilon = 3 \cdot 10^{-4}$ for the magnitude pruning Algorithm 3 to accurately achieve the pruning objective.

The results for each configuration are reported in Table 4.3 and plotted in Fig. 4.4. The dense LSTM score of 98.91 is considered as the baseline and all sparse networks outperforming it are highlighted in bold.

| Sparsity (%) | Params (k) | Rand_init | WT | WT_1200 | WT_30k | IMP |
|---|---|---|---|---|---|---|
| 0 | 68 | **98.91** | **98.91** | **98.91** | 98.91 | 98.91 |
| 30 | 48 | 98.86 | 98.58 | 98.87 | 98.91 | 98.9 |
| 50 | 34 | 98.55 | 98.3 | 98.82 | **98.94** | **98.94** |
| 70 | 21 | 98.55 | 98.23 | 98.48 | **98.94** | 37.37 |
| 80 | 14 | 98.48 | 98.49 | 98.47 | **98.96** | 14.77 |
| 90 | 6.8 | 76.21 | 71.53 | 98.46 | 98.43 | 13.68 |
| 95 | 3.4 | 93.16 | 68.12 | 63.32 | 96.78 | 20.72 |

Table 4.3: LSTM test accuracy on sequential MNIST following several lottery ticket configurations.

As we read Table 4.3 from top to bottom, the networks are more and more sparse. Considering the three configurations Rand_init, WT and WT_1200 (respectively in teal, blue and orange on Fig. 4.4), the scores only drops compared to the dense model. With those configurations, the pruned networks have to retrain through the first no convergence part showed on Fig. 4.3. Only the WT_30k retrains from a save in the converging part of the first learning and this configuration finds pruned networks with similar performance to the dense model. This observation confirms the importance of the initialization to find winning tickets emphasized by the original works. It also leads to the hypothesis that the training instability of recurrent architectures conditions the finding of winning tickets.

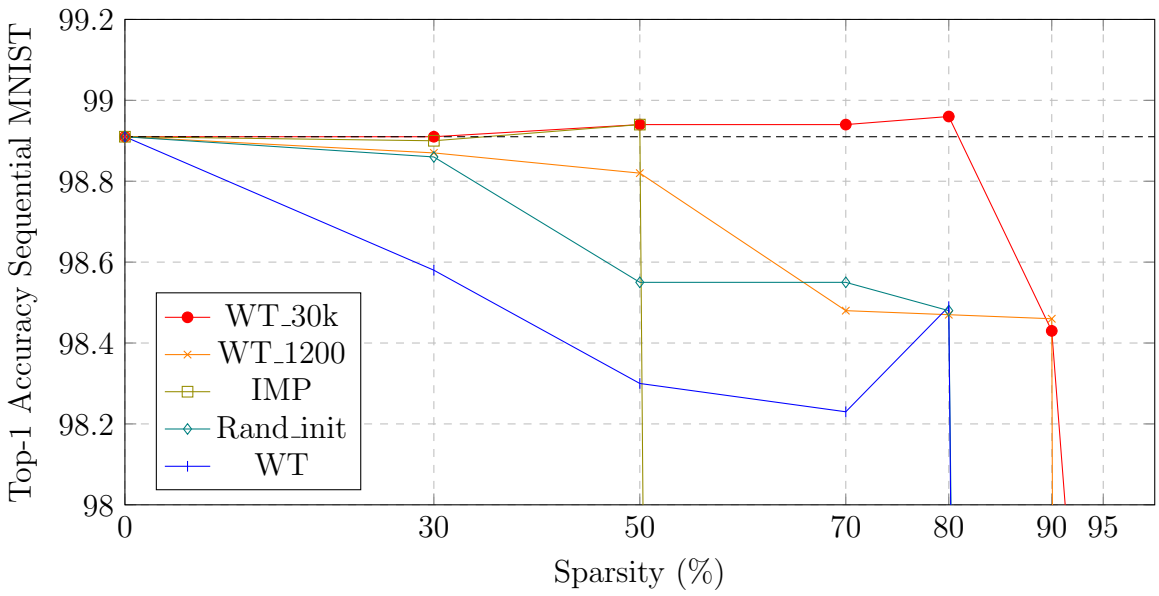



Figure 4.4: Profile of LSTM performance under increasing sparsity, trained on sequential MNIST and following several lottery ticket configurations. Each point corresponds to a test accuracy reported in Table 4.3.

One noticeable result is the winning ticket at 80% sparsity with a score of 98.96%. To better understand how the network allocates its parameters to learn, the sparsity percentages of each layer are summarized in Table 4.4. Input LSTM (InpLSTM) and Recurrent LSTM (RecLSTM) respectively regroups all input and recurrent weight matrices from each LSTM gate, Bias LSTM regroups all input and recurrent biases from each LSTM gate, FC regroups the weights and biases from the linear layer. Note that the great majority of parameters are the recurrent weight matrices from each LSTM gate. For this winning ticket, the network prunes proportionally less parameters in the InpLSTM and the linear layer compared to RecLSTM and BiasLSTM. We interpret such partitioning to avoid a drop in performance as InpLSTM conditions the information input, and the linear layer directly outputs the prediction.

| Layer | Params | Sparsity (%) |
|---|---|---|
| Input LSTM | 512 | 52 |
| Recurrent LSTM | 65536 | 81 |
| Bias LSTM | 1024 | 84 |
| FC | 1290 | 48 |

Table 4.4: Sparsity per-layer for the WT_30k LSTM with 80% sparsity that achieves 98.96 test top-1 accuracy.

Considering the IMP method, the method achieves reliable performance for 30% and

50% pruned models, but drops starting 70% of pruned parameters. The learning diverged during the training iterations with 70% sparsity and did not manage to converge again with less parameters.

Motivated by our observations on the convergence analysis and the pruning experimentation, we propose a simple method to enable early convergence of the LSTM for this task by sub-sampling the input data.

### 4.5.4 Sub-sampling pre-processing

In order for the loss to converge in the first training iterations, we propose to introduce the task complexity progressively with a data pre-processing method relying on sub-sampling. The implementation of this sub-sampling method is detailed in Algorithm 4. The input data is sub-sampled into a smaller sequence at the beginning of the training for the LSTM backpropagation to be less sensitive to Vanishing and Exploding gradients. We use a naive sub-sampling process that extracts the sequence elements based on a dilation rate. For instance, sub-sampling the sequence of pixels from an MNIST digit $X = (x_1, .., x_{784})$, with a dilation rate $r = 2$, produces the sub-sampled sequence $X_s = (x_1, x_3, .., x_{2l+1}, .., x_{783})$. During the beginning of the training, the input data progressively carries all of its original information by decreasing the sub-sampling dilation rate with a step strategy.

**Algorithm 4** Sub-sampling method

1: **Inputs:** one train data sequence $X = (x_1, .., x_L)$ with $L$ the sequence length, the training iteration $i$ and a list of $J$ sub-sampling dilation rates $R = (r_1, ..r_J)$ correlated to the step strategy target iterations $S = (s_1, .., s_J)$ (and $s_0 = 0$).
2: **Outputs:** the sub-sampled sequence data $X_s$.
3: **Sub-sampling**($X$,$i$,$R$,$S$):
4:  If $\exists\ j$ such as $s_{j-1} \leqslant i < s_j$ then
5:  $X_s \leftarrow (x_{(r_j l+1)})$ for each $l \in [0, (L//r_j) - 1]$
6:  else
7:  $X_s \leftarrow X$
8:  **return** $X_s$

#### Convergence with sub-sampling

We apply this method to learn the same LSTM over 36k iterations with the same batch size of 100 (*i.e.* 60 epochs). The method uses the sub-sampling dilation rates $R = (8, 6, 4, 2)$ correlated to the step strategy target iterations $S = (3k, 6k, 9k, 12k)$ resulting into input sequences of length $L = (98, 130, 196, 392)$. After 12k iterations, the input sequence length is $L = 784$. With a top-1 accuracy of $98,98\%$, the LSTM trained with the sub-sampling method achieves similar performance with both LSTM models reported in Table 4.5.

| Model | Top-1 Accuracy |
|---|---|
| LSTM trained without sub-sampling | 98.91 |
| LSTM trained with sub-sampling | 98.98 |

Table 4.5: LSTM test accuracy trained with and without sub-sampling on Sequential MNIST.

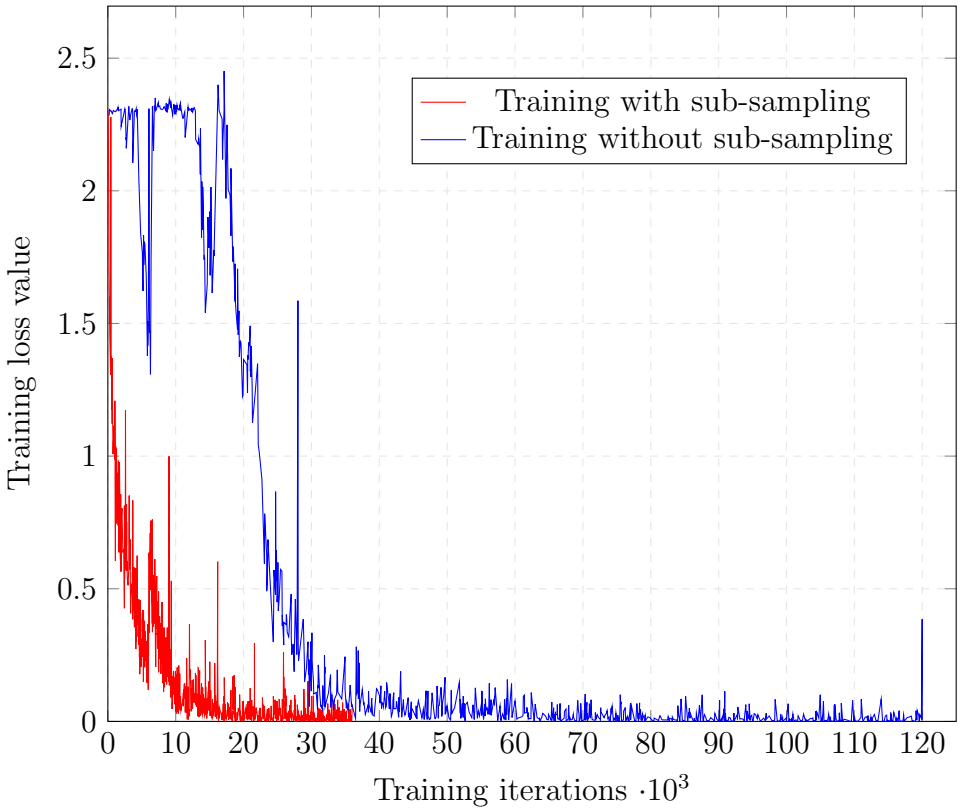



Figure 4.5: LSTM training losses on sequential MNIST when trained with and without the proposed sub-sampling method.

Figure 4.5 compares the LSTM training losses on sequential MNIST when trained with and without the proposed sub-sampling method. The plot shows clearly that the loss convergence is smoother and faster when the sub-sampling method is used. Each transition from one sub-sampling dilation to another triggers a little loss pick, but the network quickly adapts to the increasing complexity. As a result, the network converges to an asymptote by the iteration 20k that is half the number iterations needed when training without the sub-sampling method, and the application performance is preserved.

### Lottery ticket experiments with sub-sampling

We conduct lottery ticket experiments using the sub-sampling method with the following configurations: Rand_init, WT, WT with save at iteration 13.2k (*i.e.* epoch 22) and the IMP. The save on iteration 13.2k was chosen to be after the first 12k iterations where

the sub-sampling method dilates the input sequence. The data sub-sampling is applied during each training with the exception of the WT with save at iteration 13.2k (*i.e.* epoch 22) and the IMP. Indeed, for those two configurations, the state of the network at the beginning of re-trainings already learned through the sub-sampling method. The results for each configuration are reported in Table 4.6 and plotted in Fig. 4.6. The dense LSTM score of 98.98 is considered as the baseline, and all sparse networks outperforming it are highlighted in bold.

| Sparsity (%) | Params (k) | Rand_init | WT | WT_13.2k | IMP |
|---|---|---|---|---|---|
| 0 | 68 | **98.98** | **98.98** | 98.98 | 98.98 |
| 30 | 48 | 98.92 | 98.13 | **99.08** | **99.14** |
| 50 | 34 | 98.66 | 98.75 | **99.15** | **99.19** |
| 70 | 21 | 98.44 | 98.64 | **99.03** | **99.08** |
| 80 | 14 | 98.47 | 98.62 | 98.82 | 98.98 |
| 90 | 6.8 | 97.44 | 97.12 | 97.96 | 97.63 |
| 95 | 3.4 | 90.52 | 92.49 | 88.51 | 71.03 |

Table 4.6: LSTM test accuracy trained with sub-sampling on sequential MNIST following several lottery ticket configurations.

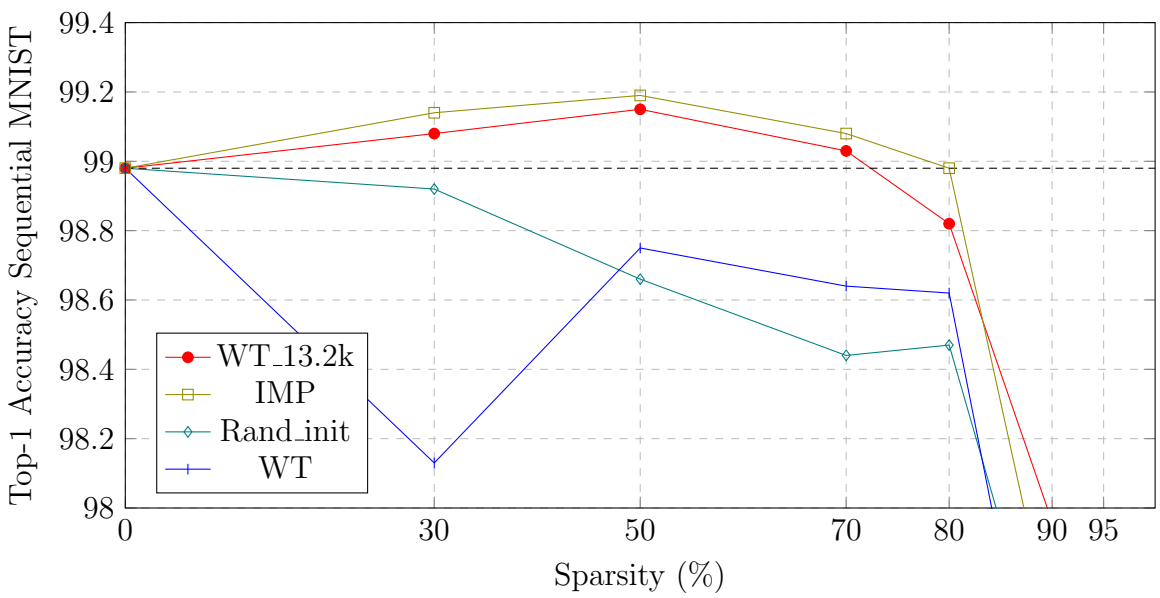



Figure 4.6: Profile of LSTM performance under increasing sparsity, trained with sub-sampling on sequential MNIST and following several lottery ticket configurations. Each point corresponds to a test accuracy reported in Table 4.6.

Comparing the results without the sub-sampling method in Table 4.3 to the results using the data sub-sampling method in Table 4.6, we find that similar tendencies for Rand_init, WT and WT with save. The WT with save is able to find pruned networks

with similar performance to the dense model. The results of the IMP configuration show more consistency when the network is learned with the sub-sampling method.

In the end, the sub-sampling method enables faster convergence while maintaining the performance for the Sequential MNIST task. However, training with the sub-sampling method failed to change the tendency of the WT configurations.

To broaden the scope of the study, the next section performs similar experiments on a text prediction task.

## 4.6 Language Modelling

### 4.6.1 Task and training setup

In this section, LSTM models are trained on the **Wikitext-2** [72] Language Modelling task. We chose this dataset because it is a reference for benchmarking models on Language Modelling. The dataset is introduced in Section 3.2.4.

The text is partitioned into sequences of words and for each input word, the model tries to predict the next. Figure 4.2 shows the inference pipeline on an LSTM, each word is first encoded into a word embedding that is sequentially fed to the recurrent architecture. Each recurrent output is then projected into the vocabulary space with a fully connected layer and compared to the target word. We build our experiments based on the standard

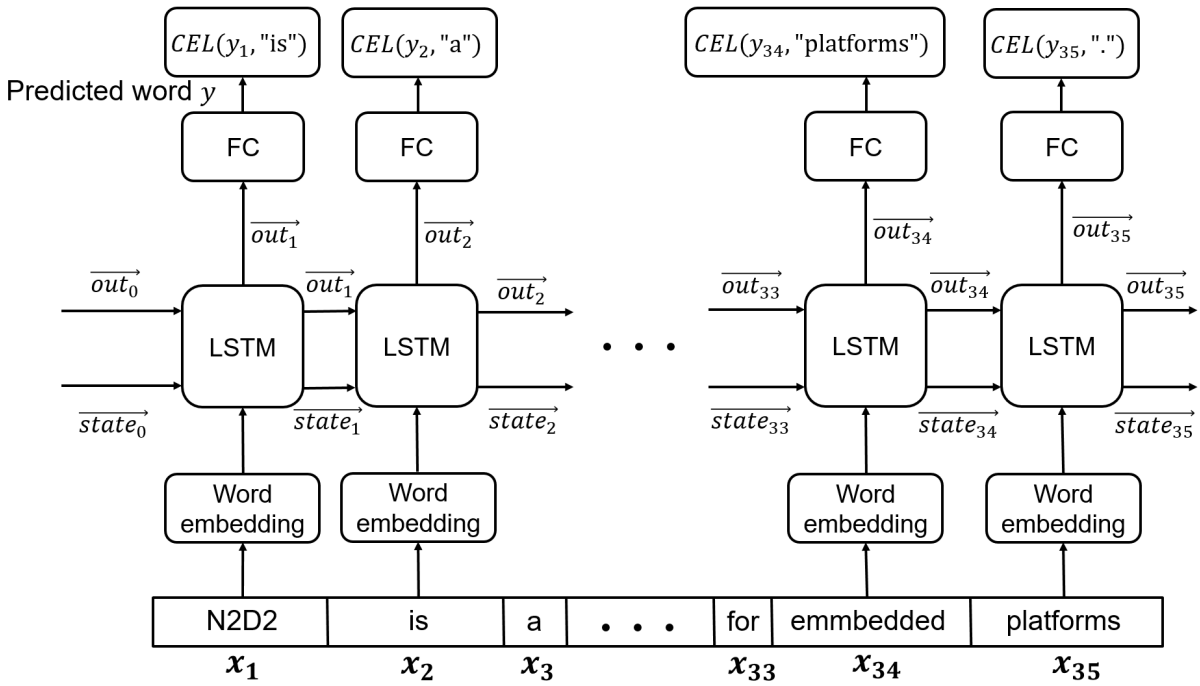



Figure 4.7: LSTM inference pipeline on Wikitext-2.

approach for the Language Modelling task. The network topology is the same as the Medium LSTM from the work of Inan *et al.* [45] and is composed of an embedding layer, two LSTM layers with a hidden dimension of 650 and a fully connected layer. We also use their weight sharing technique between the embedding layer and the weights of the fully connected layer as it reduces the number of parameters and allows for faster convergence. Accounting with respect to the weight sharing method, the total number of parameters is 26 Millions. The input sequences are composed with 35 words to follow the standard and compare our LSTM result with their Medium LSTM [45]. This fixed sequence length however prevents the LSTM from learning time dependencies exceeding 35 words. Each training uses the Cross Entropy Loss objective function, the Adam optimizer [53] with a learning rate of $10^{-3}$ subject to a cosine annealing strategy, $183k$ iterations with a batch size of 64 (*i.e.* 200 epochs). A random dropout mask with probability of 0.4 is used on each layer output (except the output of fully connected layer) to prevent the model from overfitting.

At the beginning of training and test time, the first sequence ($seq = 0$) starts with a zero state $h_0^{seq=0} = 0$. In the case of the LSTM, $h$ stands either for the cell state *state* and the output *out* and both of them are treated the same way. Each input recurrent state for the sequences following $h_0^{seq}$ follows the policy introduced by Melis *et al.* [71]:

$$h_0^{seq} = \begin{cases} h_{35}^{seq-1} & \text{with probability } p = 0.99 \\ 0 & \text{with probability } (1-p) = 0.01 \end{cases} \tag{4.1}$$

When $h_0^{seq} = h_{35}^{seq-1}$ with a probability of $p = 0.99$, the continuity of the text is preserved and context information is provided to the new sequence to make relevant first words predictions. Otherwise, the $h_0^{seq}$ is set to a constant zero state to bias the model towards being able to easily start from such a state at test time.

The performance of each network is evaluated with the perplexity metric and corresponds to the exponential of the CEL error. The lower the perplexity metric, the better the performance of a model on the language modelling task. The compression of the network is assessed by the sparsity percentage of the parameters, *i.e.*, the percentage of zero parameters in the network.

### 4.6.2 Convergence on Wikitext-2

In this section, we investigate the convergence of one dense LSTM model on the Wikitext-2 language modelling task. Our model produces a slightly better validation perplexity (on the validation Wikitext-2 set) and test perplexity (on the test Wikitext-2 set) perplexity

of the LSTM from the work of [45] and both scores are reported in Table 4.7. We then use this model as the baseline performance to compare to sparse models.

| Model | Perplexity | |
|---|---|---|
| | Val | Test |
| LSTM [45] | 100 | 95.3 |
| LSTM (Ours) | 97.9 | 94.4 |

Table 4.7: LSTM performance on Wikitext-2.

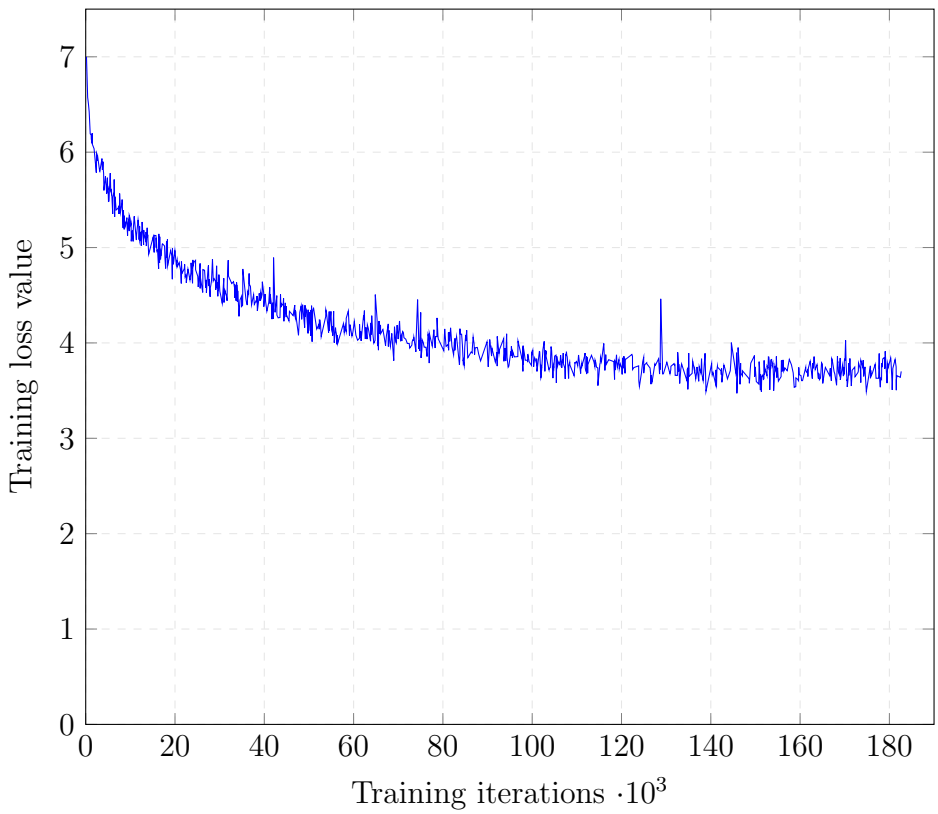



Figure 4.8: LSTM training loss on Wikitext-2.

The training loss convergence associated with our LSTM score is plotted in Fig. 4.8. Globally, the training loss clearly converges and the overall profile is smoother than the Sequential MNIST task. We assume the gradients are more stable for two main reasons. First, the LSTM has input errors from all time steps instead of only the last one, which thus provides much more information to compute the gradients from. Second, the sequence length is fixed at 35 instead of 784 for the Sequential MNIST, directly reducing the sensitivity towards Vanishing and Exploding gradients.

Locally, the loss is still unstable and we assume the main reason is the dropout regularization. Although necessary to prevent the model from overfitting, masking activations still filter information during the training.

### 4.6.3 Lottery ticket experiments

In this section, sparse LSTM models are learned with the following configurations: Rand_init, WT, WT with save at iteration 1830 (*i.e.*, the second epoch) and the IMP. The save iteration was chosen during the convergence of the dense model as seen in Fig. 4.8. All pruning experiments follow the same iterative pruning objectives $P_{obj} = (0.3,\ 0.5,\ 0.7,\ 0.8,\ 0.9,\ 0.95)$. We also using a threshold step $\epsilon = 3 \cdot 10^{-4}$ for the magnitude pruning of Algorithm 3.

Applying the pruning algorithm naively on the model only results in poor performance. We found that having one pruning objective for the whole model masked more connections in the LSTM than in the Embedding (and the tied fully connected) to the point that the recurrent structure could not learn anything with 50% or even 30% sparsity. To answer this limitation, we set two distinct pruning objectives for the tied parameters of the embedding and fully connected layers, and for the LSTM parameters. Also, we excluded the LSTM bias parameters from the pruning algorithm as it showed better results.

The results for each configuration are reported in Table 4.8 and plotted in Fig. 4.9. The dense LSTM test perplexity of 94.4 is considered as the baseline and all sparse networks outperforming it are highlighted in bold.

| Sparsity | Params | Rand_init | | WT | | WT_1830 | | IMP | |
|---|---|---|---|---|---|---|---|---|---|
| (%) | (Mi) | Val | Test | Val | Test | Val | Test | Val | Test |
| 0 | 26 | 97.9 | **94.4** | 97.9 | 94.4 | 97.9 | 94.4 | 97.9 | **94.4** |
| 30 | 18 | 118 | 113 | 95.3 | **91.4** | 94.8 | **91.1** | 106 | 103 |
| 50 | 13 | 120 | 114 | 97.1 | **92.8** | 96.2 | **92.8** | 122 | 117 |
| 70 | 7.6 | 124 | 117 | 106 | 101 | 107 | 103 | 125 | 120 |
| 80 | 5.1 | 125 | 117 | 111 | 105 | 112 | 107 | 127 | 120 |
| 90 | 2.6 | 139 | 131 | 139 | 130 | 139 | 131 | 164 | 152 |
| 95 | 1.3 | 164 | 154 | 156 | 145 | 160 | 150 | 210 | 193 |

Table 4.8: LSTM perplexity on Wikitext-2 with different pruning pipelines

As we read Table 4.8 from top to bottom, the networks are more and more sparse. The two configurations WT and WT_1830 (respectively in blue and red in Fig. 4.9) finds winning tickets outperforming the dense model perplexity with 30% and 50% sparsity. They even find similar winning tickets over the pruning objectives. Contrary to the loss profile on Sequential MNIST of Fig. 4.3, the loss profile on wikitext-2 of Fig. 4.8 shows convergence from the beginning of the of the training. Each retraining is intuitively expected to approximately retrace the loss profile of the dense model. The WT configuration retrains from the original initialization and is able to retrace this smooth early convergence

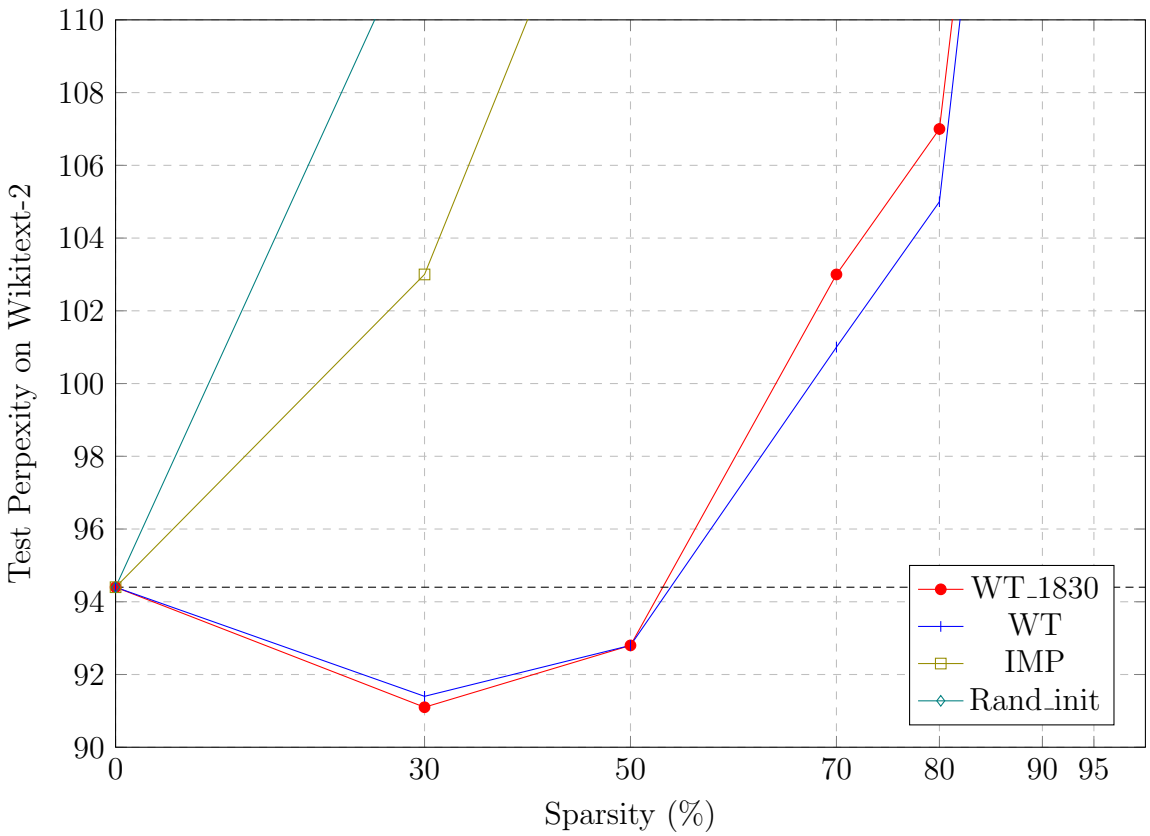



Figure 4.9: Profile of LSTM performance under increasing sparsity, trained on Wikitext-2 and following several lottery ticket configurations. Each point corresponds to a test accuracy reported in Table 4.3.

profile on wikitext-2 to find winning tickets. A similar observation can be made for the WT_1830 configuration. Under this observation, when the two configurations find similar winning tickets performance for each pruning objective on Fig. 4.8, they most likely retrace similar training loss convergence profile at each retraining.

Considering the Rand_init configuration (in teal on Fig. 4.9), the perplexity only increases compared to the dense model. Even if the loss profile has a smooth and early convergence, trying to find trainable sparse networks with similar performance fails when the initialization does not correspond to a state of the dense model. This observation on RNNs is aligned with the observations made by Frankle *et al.* [25] on Feed Forward Neural Network.

Contrary to the experiments on Sequential MNIST, the IMP configuration is not relevant on wikitext-2. The IMP method is too naive to prune the network while preserving the performance. A more advanced IMP method would find a much more relevant operating point. For example, Zhu and Gupta [125] introduced an advanced pruning method able to

find a large LSTM with 80% pruned parameters while preserving performance on another reference language modelling task.

## 4.7 Discussion and Perspectives

This chapter is dedicated to the investigation of practical RNN learning behaviour. The Lottery Ticket Hypothesis formulated by Frankle *et al.* [25] is introduced and we use their method as our tool to investigate the convergence of RNNs. We extend their experimental scope by assessing their algorithm with RNN on image classification and language modelling. When compared to the original work on MLP and CNN, it appears that RNN convergence profile is unstable. Coupling convergence analysis with relevant method configurations, we observed that, to find winning tickets on recurrent architectures, each retraining needs to be initialized with a dense network state from which the loss profile converges smoothly.

We proposed a sub-sampling method that enables faster convergence while maintaining the performance for recurrent architectures on the Sequential MNIST task. However, our method failed to shape the learning landscape to be friendly enough for the basic configuration to find winning tickets.

Magnitude pruning methods achieve significant model compression ratio while preserving application performance. Such model compression enables the storage of many models on memory-limited hardware. However, the resulting sparse architecture is not structured to be easily accelerated on target. Indeed, magnitude pruning finds its limitation when critical applications have latency and low power requirements. Regarding those criteria, DNN quantization is an elegant solution to both compress the memory storage of the network and enable an optimized inference with lower precision operators at inference time. This first work inspired us to pursue studying advanced quantization methods in the Chapter 5.

# Chapter 5

# Disentangled Loss for Low-Bit Quantization Aware Training

## 5.1 Introduction

Many deep learning contributions rely on increasing the number of parameters and computation power to achieve better performance. With application performance as the one-trick-pony objective, deep learning models have become continually larger, more memory demanding and computationally heavier. Supporting this trend, Resnet [38], one of the most popular convolutional architectures, was designed to mitigate the vanishing gradient phenomenon in order to enable the convergence of very deep neural networks. This greedy approach is not limiting as long as the running environment offers sufficient resources. For that matter, cloud-based platforms offer a relevant strategy by centralizing all the computations resources.

However, cloud-based solutions cannot satisfy the requirements for all applications. Specifically, critical applications with real-time constraints such as memory, latency, power consumption, with a resource-scarce hardware target or with privacy issues, cannot be inferred on cloud. Instead, the requirements of those critical applications can be met with a local execution of the input data, *i.e.* on the edge. To be deployed on the edge, neural networks compromise between performance and the limited resources. Inventing new neural networks to meet specific needs requires a lot of effort. This is why compressing existing architectures is the preferred solution as they offer a good trade-off and they are reliable to transfer knowledge to down-stream tasks.

Therefore, various methods were designed to reduce memory and computational needs of these models, in order to use Deep Convolutional Neural Networks for low power and/or low memory applications. There are two major ways of reducing the size of a deep neural

network:

- Reducing the total number of parameters, with methods such as pruning, weight sharing, low-rank factorization, structured matrices, knowledge distillation. We refer to Wang *et al.* [113] for an in-depth explanation of each of these methods.
- Reducing the memory footprint of its parameters with quantization.

DNN quantization aims at reducing the number of bits to represent its parameters, while keeping the performance and quality of results as close as possible to the floating-point reference. For instance, the weights of a DNN can be stored as signed integers using 8 bits instead of 32 bits with the single-precision floating-point format, without any loss in application performance. Moreover, the added value of quantization lies with a fully quantized pipeline, *i.e.*, weights and activations, where, once engineered with hardware accelerators to reduce operator complexity, it achieves significant memory footprint reduction proportional to the reduction of the number of bits, along with energy savings and higher throughput.

In practice, the availability of training data and computation resources motivates the use of different quantization methods. With both those resources, the quantization process can rely on retraining the model from scratch with stochastic gradient descent. This class of methods is known as Quantization Aware Training (QAT), and the latest proposals lead to the best performance for settings from 8 bits down to 2 bits [23, 48]. Those methods rely on the CEL function, *i.e.* a combination of softmax and negative log likelihood, as it is the reference loss function for classification. A variation of the softmax was proposed by Liu *et al.* to encourage more discriminating features for image classification [69]. This research led to significant performance gains, especially in the face recognition domain [68, 114], where the number of classes is an order of magnitude higher than academic image classification tasks. Also, Wan *et al.* used Gaussian Mixtures to formalize the classification space and encourage more discriminating features [111].

To date, the effect of those loss functions on QAT remains unexplored. This chapter studies the quantization aware training with disentangled loss functions for settings down to binary weights. We empirically show that training a model to output discriminative features improves its resilience to quantization. Results on CIFAR-10, CIFAR-100 and ImageNet datasets show the clear advantage of our approach, with significant performance gains, especially for very low-bit settings.

This chapter is organized as follows. Section 5.2 presents some previous work on QAT as well as the foundation of disentangled loss functions. Section 5.3 introduces our method that takes advantage of both Additive Margin Softmax (AMS) loss and Gaussian Mixture

Loss (GML) to improve the QAT procedure. Section 5.4 presents our experimental setup and the results obtained on relevant datasets.

## 5.2 Previous Work

To better understand the intuition behind our approach, we first give a brief review of the state-of-the-art techniques on quantization-aware training and disentangled losses.

### 5.2.1 Quantization Aware Training

Given a network $f : \mathbb{R}^n \Rightarrow \mathbb{R}$ with its parameters $W$, an input $x \in \mathbb{R}^n$ and its corresponding label $y$, we refer to QAT for classification as finding the non-differentiable quantization function $q$ with the loss function $L$ as

$$\min_W L[f(x, q(W)), y]. \tag{5.1}$$

Bengio *et al.* proposed the Straight-Through-Estimator (STE) to enable training with backpropagation [10]. The STE method estimates the gradients of the quantized parameters assuming that the derivative of the quantization function $q$ is the identity function. QAT methods use this approximation tool to backpropagate errors and update a single-precision floating-point copy of the weights. Those updated weights are then being injected in the quantization function for the next inference. However, the approximation error grows bigger as the bitwidth goes smaller hence decreasing the performance for low-bit settings. Esser *et al.* tackled this issue by scaling dynamically the gradients with a learnable step [23]. Following their method, the gradient landscape is shaped to encourage the full precision parameters towards the quantized points. Doing so, the proposed Learned Step Size Quantization (LSQ) method implicitly reduces the approximation error introduced by the STE and shows substantially better results over the previous quantization techniques. Alternatively, the Scaled Adjust Training (SAT) method introduced by Jin *et al.* directly scales the weights instead of the gradients to control the training dynamics, which yields state-of-the-art results [48]. Their approach is driven by the distribution of parameters and gradients, which is summarized by two rules:

1. Keep the output of the linear layer from entering the saturation region of the CEL.
2. Keep the gradients of the weights at the same scale throughout the network.

To quantize a weight matrix $W_{ij}$, the SAT method relies on the DoReFa scheme [124]. First a weight matrix $W_{ij}$ is clamped in the range $[0, 1]$ as

$$\widetilde{W}_{ij} = \frac{1}{2}\left(\frac{\tanh(W_{ij})}{\max_{r,s} |\tanh(W_{rs})|} + 1\right). \tag{5.2}$$

The element-wise quantization function $q$ is applied on each weight $x$ towards $2^b$ quantized points using the factor $a = 2^b - 1$ as

$$q(x) = \frac{1}{a}\lfloor ax \rceil. \tag{5.3}$$

The final quantized weight matrix $Q_{ij}$ is obtained following

$$Q_{ij} = 2q(\widetilde{W}_{ij}) - 1. \tag{5.4}$$

To enable efficient quantization following their rules, they proposed the SAT method introducing constant rescaling [48]. For a layer with a quantized weight matrix $Q_{ij}$ and $n_{out}$ (number of output neurons), the scaled quantized weights $Q^*_{ij}$ are computed as

$$Q^*_{ij} = \frac{1}{\sqrt{n_{out}\mathbb{VAR}[Q_{rs}]}} Q_{ij}. \tag{5.5}$$

To quantize the activations, the authors replace the identity function for a better approximation of the quantization function derivative to upgrade the PACT [19] method. This new formulation especially reduces the approximation error for low-bit settings.

Concretely, they proposed the two-step training method for QAT illustrated in Fig. 5.1. During Training I, the network $f$ is learned with 32-bit floating-point parameters, while constraining the range of its parameters $W$ with clamping based on Eq. (5.2). The layers that are not followed by batch normalization [46] see their parameters scaled following Eq. (5.5). This first training step returns a converged network $f$ whose learned floating-point parameters $W_{clamp}$ are constraint and scaled in a quantization friendly range. Those learned parameters $W_{clamp}$ are then used as initialisation for Training II. During the second step, the network weights are quantized following the DoReFa scheme, and/or activations are quantized with an upgraded version of PACT [19, 48].

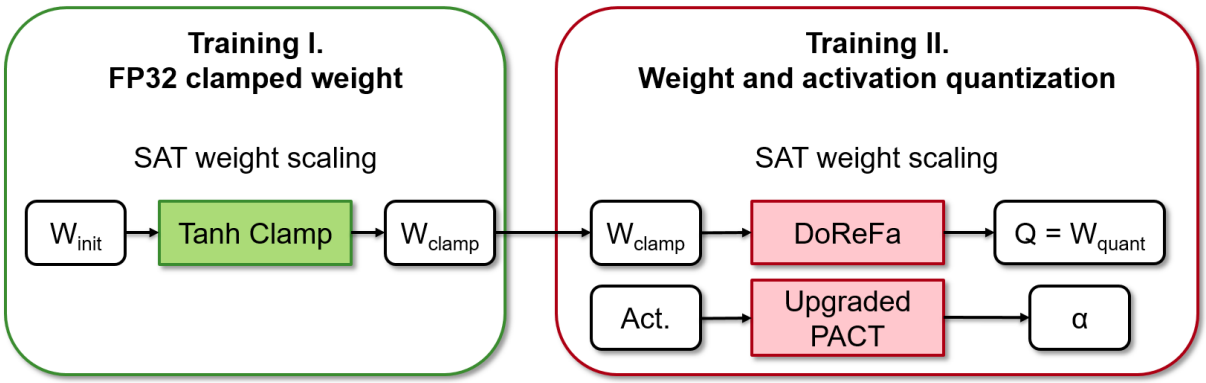



Figure 5.1: Scaled Adjust Training method.

### 5.2.2 Disentangled Losses

We call a loss "disentangled" as it encourages the network output space to be easily discriminated with linear functions.

#### Additive Margin Softmax

Inspired by Large-Margin Softmax [69] and Sphereface [68], Wang *et al.* proposed an intuitive formulation of the margin softmax loss function called Additive Margin Softmax [114]. The authors considered the propagation of features $f$ in the linear layer without bias as scalar products for each column $j$ of the weight matrix $W$. They used the geometric definition of the scalar product of Eq. (5.6), coupled with feature and weight normalization to rewrite the loss function applying a margin $m$ on the target logit $W_y^T f$ and a scaling factor $s$, following Eq. (5.7).

$$f \cdot W_j = \|W_j\| \|f\| cos(\theta_j) \tag{5.6}$$

$$L_{AMS} = -\log \frac{e^{s\cdot(\cos\theta_y - m)}}{e^{s\cdot(\cos\theta_y - m)} + \sum_{j=1,\ j\neq y}^{C} e^{s\cdot(\cos\theta_j)}} \tag{5.7}$$

The softmax output probabilities can be interpreted as a vector of dimension $C$, $C$ being the number of classes. The one-hot vectors encoding the different classes are the orthogonal vectors that construct the canonical basis of $\mathbb{R}^C$. Here, the subtracted margin $m$ acts as a classification boundary offset, forcing the network to output features that are closer to the orthogonal vector corresponding to their label, thus reducing the intra-class variance of each class cluster in the network.

#### Gaussian Mixture Loss

Wan *et al.* proposed to model the classification layer with Gaussian mixtures [111]. The Gaussian Mixture Loss (GML) draws the distances $d_j$ between features $f$ and the learned means $\mu_j$ to minimize the distance to the mean associated to the true label $d_y$. A positive margin factor $\alpha$ artificially inflates the distance $d_y$ to help regulate the convergence of the network. Under the assumption that the covariance matrix is isotropic, the GML can be rewritten as

$$L_{GM} = -\log \frac{e^{-d_y(1+\alpha)}}{e^{-d_y(1+\alpha)} + \sum_{j=1,\ j\neq y}^{C} e^{-d_j}} \tag{5.8}$$

$$\text{with } d_j = \frac{1}{2}\|f - \mu_j\|_2^2 \tag{5.9}$$

## 5.3 Disentangled Loss Quantization Aware Training

**Intuition.** Considering that features can be more discriminative with disentangled losses than with CEL, we assume that low-bit quantization-aware training can benefit from a disentangled loss. Indeed, a smaller intra-class variance and a bigger inter-class difference should be more robust to the quantization noise. With CEL, the inter-class features are optimized to be orthogonal without constraint on their actual distance in the output space. While it is also true for AMS, it still allows for an additional margin on the orthogonality. On the contrary, GML directly minimizes the distance between the features and their corresponding centroids, thus, minimizing the intra-class variance. The use of learned centroids instead of orthogonal features ensures that the distance between inter-class features is constrained by the distance of their respective centroids, as the features are attracted to their corresponding centroids. To reformulate, while AMS loss encourages a smaller intra-class variance than CEL, GML ensures both a smaller intra-class variance and a bigger inter-class difference than CEL. This motivates the formulation of our hypothesis along with our investigation combining several state-of-the-art methods: the presented disentangled loss functions with the SAT procedure [48].

**Hypothesis.** Disentangled losses encourage a small intra-class variance and a big inter-class difference. They are robust to quantization noise and benefits Quantization Aware Training.

**Method.** In order to assess our hypothesis, we introduce DL-QAT, a method applying the intuitive formulation of AMS or GML loss function with the quantization-aware training method SAT [48]. Algorithm 5 details the implementation of DL-QAT.

## 5.4 Experiments

### 5.4.1 Training setups

All experiments use a Resnet-18 [38] with the **CIFAR-10**, **CIFAR-100** [54] and **ILSVRC 2012 ImageNet-1k** dataset [22]. All the 8-bit images are divided by 255 and no standardization is applied. The train data are augmented with random rescaling, cropping and flipping. The learning strategy proposed by [48] is used for all experiments with different learning rates that we specify later. All networks are trained over 150 epochs.

For the experiments on **CIFAR-10** and **CIFAR-100** datasets [54], as our input images are 32×32, the Resnet-18 [38] is adapted with a first {3,3} kernel, stride= 1, which output is directly fed to the first residual block. The batch size is 768. When training is performed with SAT using the CEL or DL-QAT using the AMS loss, the learning rate is 0.01. When

**Algorithm 5** Disentangled Loss Quantization Aware Training

1: **Inputs:** a neural network $f$ and its FP32 parameters $W_{FP32}$, training data $x$ and its corresponding target $y$, the disentangled Loss $L$. Clamp() is Eq. (5.2), DoReFa is Eq. (5.3) and Eq. (5.4), Scaled-Adjust is Eq. (5.5).
2: **Outputs:** the quantized parameters $Q$ and the activation quantization learned parameters $\alpha$.
3: **DL-QAT**($f$,$W$,$x$,$y$,$L$):
4: Training I. FP32 clamped and scaled weights
5: Learn the network minimizing $L[f(x, Clamp(W_{FP32})), y]$
6: $W_{FP32} \leftarrow$ converged FP32 parameters of the first training
7: Training II. Quantized weight and activation
8: For each Quantization Aware Training iteration using input data $(x, y)$:
9: $W_{clamp} \leftarrow Clamp(W_{FP32})$
10: $Q \leftarrow DoReFa(W_{clamp})$
11: If No Batch Normalization:
12: $Q \leftarrow$ Scaled-Adjust($Q$)
13: $Out \leftarrow f(x, Q)$. *(Propagate and quant. the activations on the fly with PACT($\alpha$))*
14: $Error \leftarrow L(Out, y)$
15: Backpropagate the error. *(The quant. functions are approximated as detailed in [48])*
16: Update $W_{FP32}$ and $\alpha$ with SGD and their respective gradients: $\frac{\partial Error}{\partial W}$, $\frac{\partial Error}{\partial \alpha}$
17: **return** $Q$, $\alpha$

the training is performed with DL-QAT using the GML, the learning rate is 0.2. We use the same AMS scale $s = 30$ from Eq. (5.7) hyperparameter from the best results of the original paper [114]. Preliminary experiments were conducted to find the best AMS loss additive margin $m$ from Eq. (5.7) as well as the best GML multiplicative margin $\alpha$ from Eq. (5.8) and are respectively reported in Table 5.2 and Table 5.1. We chose the margin parameters $\alpha = 0.7$ and $m = 0.35$ for all experiments on **CIFAR-10** & **CIFAR-100** based on the best floating-point performance.

For the experimentations on **ILSVRC 2012 ImageNet-1k** dataset, the Resnet-18 is the original ResNet proposed by He *et al.* [38]. The batch size is 1024. The learning rate is 0.02 for both SAT using CEL and DL-QAT using GML. The GML multiplicative margin is $\alpha = 0$ from Eq. (5.8) as it gives best results.

As it is common practice in the previous quantization approaches [23, 48], the precision of filters from the first convolution, the weights of the last layer and the activation preceding the last layer are fixed to 8 bits. Also, all batch normalization layers and the bias in the linear layer are not quantized. For the experiments using the AMS loss function, we do not use the bias of the linear layer, in order to respect the geometric definition of the scalar product as previously introduced in Section 5.2.2.

All experimentations presented in this chapter were done in the Neural Network Design

& Deployment (N2D2) framework (Section 3.1.3).

| Margin $\alpha$ | $Acc_{GML}^{float32}$ | $Acc_{GML}^{2bits}$ |
|---|---|---|
| 0 | 70.5 | 68.6 |
| 0.1 | 71.3 | 69.5 |
| 0.3 | 72.0 | 70.7 |
| 0.5 | 72.8 | 70.8 |
| 0.7 | **73.6** | **71.9** |
| 1.0 | 72.9 | 71.3 |

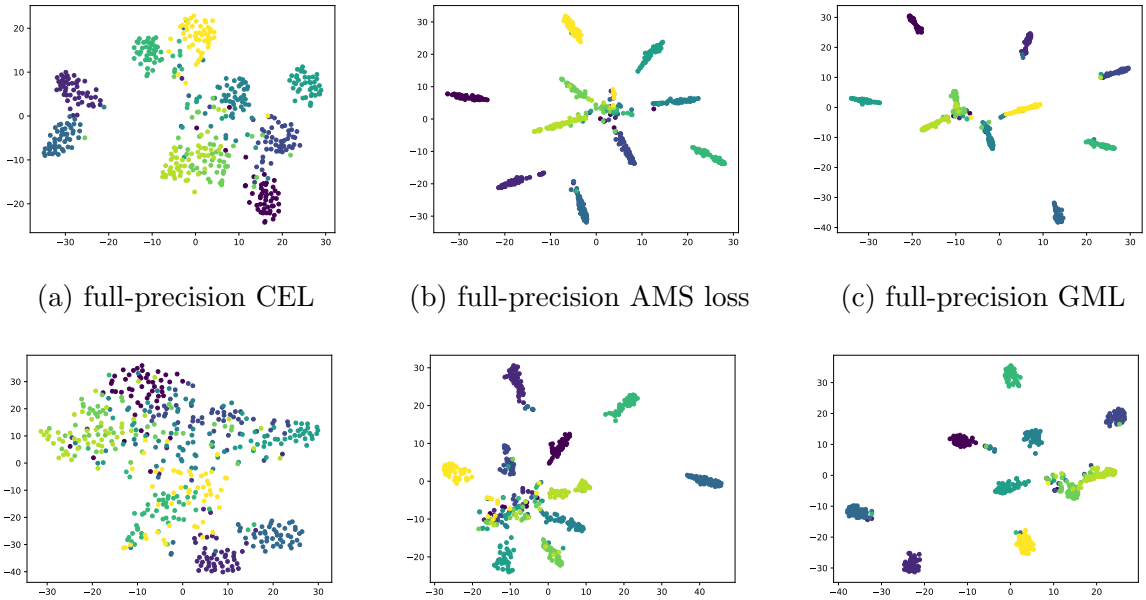
Table 5.1: 2-bit weight and activation (W&A) quantization of Resnet-18 on CIFAR-100 with various GML margins.

| Margin $m$ | $Acc_{AMS}^{float32}$ | $Acc_{AMS}^{2bits}$ |
|---|---|---|
| 0 | 65.2 | 62.2 |
| 0.1 | 67.2 | 65.1 |
| 0.2 | 68.1 | 65.8 |
| 0.35 | **68.7** | 66.1 |
| 0.5 | 68.6 | **66.4** |

Table 5.2: 2-bit weight and activation (W&A) quantization of Resnet-18 on CIFAR-100 with various AMS margins.

### 5.4.2 Results and analysis

To better visualize the contribution of the AMS loss and the GML during quantization, we performed dimension reduction with the t-sne algorithm [107] over the input features of the linear classifier. The features fed to the t-sne algorithm are extracted from the converged Resnet-18 inferring with the same sets of 50 test images for each class. The 2D visualisations from full precision and 2-bit Resnet-18 for CEL, AMS loss and GML are plotted in Fig. 5.2. As expected, the full precision Resnet-18 clusters with AMS loss (b) and GML (c) are more compact than with CEL (a). It manifests that separating the clusters thanks to straight lines modeled by the linear classifier will be easier. Comparing full precision in Fig. 5.2.(a-b-c) to 2-bit quantization in Fig. 5.2.(d-e-f), the clusters with the quantized version are less compact, and we can interpret this as the effect of the quantization. In Fig. 5.2, comparing (d) to (e) and (f), the plots show that the ambiguities caused by the quantization are reduced thanks to the disentangled losses.



(a) full-precision CEL (b) full-precision AMS loss (c) full-precision GML

(d) 2-bit W&A CEL (e) 2-bit W&A AMS loss (f) 2-bit W&A GML

Figure 5.2: Dimension reduction with the t-sne algorithm representing the input features of the linear classifier from **CIFAR-10** test data. The corresponding top-1 test accuracies are reported in Table 5.3. t-sne performed over 1000 iterations and a perplexity of 30.

The top-1 test accuracies on **CIFAR-10** and **CIFAR-100** of the proposed DL-QAT method ($Acc_{AMS}$) and ($Acc_{GML}$) compared to the SAT method ($Acc_{CEL}$) are reported in Table 5.3. The lines with 32 correspond to single-precision floating-point, which is considered as the full precision baseline. Table 5.3 also reports the $\Delta P_{loss}$ quality metric to compare the QAT methods defined as

$$\Delta P_{loss} = Acc_{loss}^{float32} - Acc_{loss}^{quant}. \tag{5.10}$$

$\Delta P_{loss}$ measures the drop in top-1 accuracy between the full precision version and a quantized version of a network trained with the same loss function. Given $\Delta P_{GML}$ and $\Delta P_{CEL}$, we can better compare the quantization resilience between disentangled losses and CEL.

One of the main results is that Resnet-18 with binary weights and 2-bit activations trained with GML (71.3%) outperforms the full precision Resnet-18 trained with CEL (66.2%). We also want to emphasize that $\Delta P_{CEL} > \Delta P_{AMS}$ and $\Delta P_{CEL} > \Delta P_{GML}$ for all settings. As the precision is reduced, the drop in top-1 accuracy grows larger. Our approach especially well limits the drop in top-1 accuracy for low-bit settings. Hence, the discriminative features, enforced by the AMS loss or the GML, enable more resilient quantization-aware training than the CEL, especially for low-bit settings. Overall, a clear

| Dataset | W [bits] | A [bits] | SAT [48] $Acc_{CEL}$ | DL-QAT (ours) $Acc_{AMS}$ | DL-QAT (ours) $Acc_{GML}$ | SAT [48] $\Delta P_{CEL}$ | DL-QAT (ours) $\Delta P_{AMS}$ | DL-QAT (ours) $\Delta P_{GML}$ |
|---|---|---|---|---|---|---|---|---|
| CIFAR-10 | 32 | 32 | 89.4 | 91.7 | **93.0** | – | – | – |
| | 2 | 2 | 76.5 | 89.1 | **91.3** | 12.9 | 2.6 | **1.7** |
| | binary | 2 | 72.4 | 88.3 | **91.2** | 17.0 | 3.4 | **1.8** |
| CIFAR-100 | 32 | 32 | 66.2 | 68.7 | **73.6** | – | – | – |
| | 8 | 8 | 65.8 | 68.5 | **73.1** | 0.4 | **0.2** | 0.5 |
| | 4 | 4 | 65.4 | 68.4 | **72.6** | 0.8 | **0.3** | 1.0 |
| | 3 | 3 | 65.1 | 68.2 | **73.3** | 1.1 | 0.5 | **0.3** |
| | 2 | 2 | 61.1 | 66.1 | **71.9** | 5.1 | 2.6 | **1.7** |
| | binary | 8 | 63.9 | 67.9 | **72.5** | 2.3 | **0.8** | 1.1 |
| | binary | 4 | 63.2 | 67.1 | **72.5** | 3.0 | 1.6 | **1.1** |
| | binary | 3 | 62.4 | 67.0 | **72.0** | 3.8 | 1.7 | **1.6** |
| | binary | 2 | 59.0 | 65.5 | **71.3** | 7.2 | 3.2 | **2.3** |

Table 5.3: CIFAR-10 and CIFAR-100 test top-1 Accuracy for extreme quantization settings of ResNet-18.

tendency appears where $Acc_{CEL} < Acc_{AMS} < Acc_{GML}$. Indeed, GML minimizes the intra-class variance and constrains the distances of inter-class features while the AMS loss only minimizes the intra-class variance. Those results confirm our hypothesis on the loss function that both intra-class variance and inter-class difference need to be constrained.

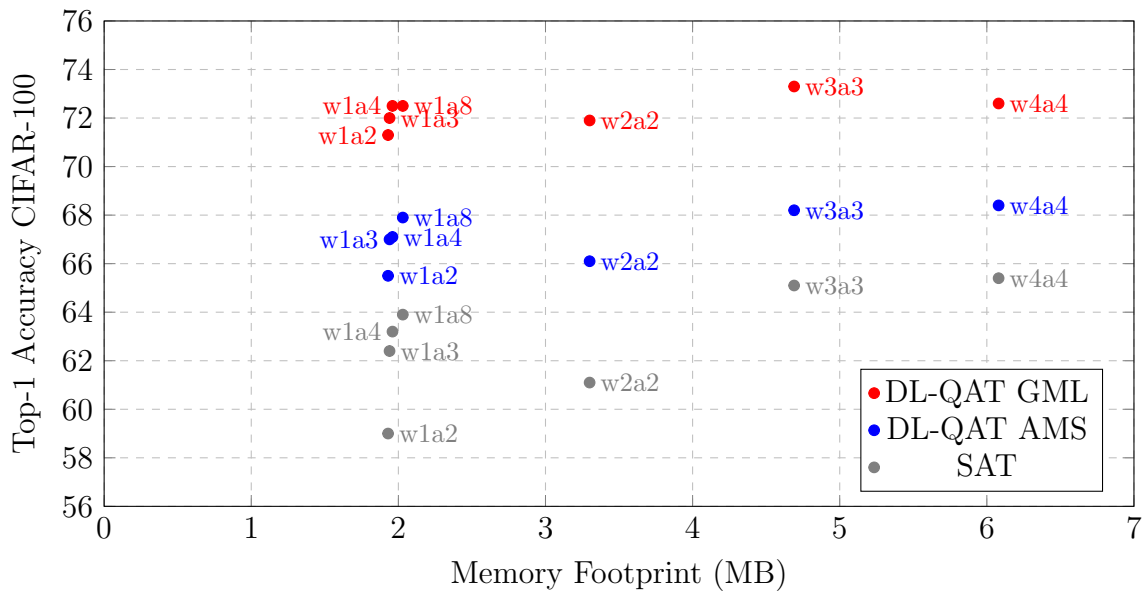



Figure 5.3: Memory footprint (Mega-Bytes) comparison of different precision settings from Table 5.3. Each point is one quantized network associated to the bit precision of its weight and activation.

Fig. 5.3 maps the performance of each ResNet-18 with respect to their memory footprint.

The memory footprint of a quantized neural network is approximated by summing the total memory storage for all the parameters at their respective bit-representation with the memory needed to contain the biggest activation. With this perspective, the reliable score and competitive memory footprint showed by the binary versions of ResNet-18 better stands out. Moreover, we can better grasp the opportunities that DL-QAT creates. The binary versions of ResNet-18 are suitable to infer on microcontrollers such as STM32 chips as their memory constraint is met with less than 2Mb. The image classification task on **CIFAR-100** is a representative task for the Internet of Things. Our approach brings us one step closer to the inference on microcontrollers of IoT tasks.

### ImageNet-1k

In this section, we evaluate the performance of our method using the **ImageNet-1k** dataset. Considering the results on **CIFAR** and our hypothesis on the losses, we chose to focus on the GML for **ImageNet-1k** experiments. We report the top-1 test accuracy on **ImageNet-1k** of our method DL-QAT using the GML and the SAT method [48] using CEL and other state-of-the-art approaches in Table 5.4.

As we read Table 5.4 from left to right, the quantization is more and more aggressive. Considering our experimental results only (DL-QAT using GML and SAT using CEL), the gap between the disentangled loss GML and the CEL is getting bigger as the settings reach more extreme quantization. Ultimately, in the binary weights and 2-bit activation setting, our approach reaches an accuracy of $64,0\%$, improving by $0.9\%$ the CEL score of $63.1\%$.

When comparing our method to the other approaches, the version of Resnet-18 and the quantization method matter. Notably, the Resnet-18 results reported in Esser *et al.* [23] use pre-activation quantization scaling and thus keep the residual connections in the same precision as the accumulation (*i.e.*, 32 bits). While this significantly improves the final accuracy in low precision, the actual precision of the dataflow is not strictly the activation's precision. For this reason, we have chosen to keep Resnet-18 with post-activation for our experiments, which makes it however not fully comparable with the LSQ reported results. For 2-bit weights, our method achieves substantial improvement over ABC-Net[66] and INQ[123], while the setting is more constraining on the activations. One noticeable result over the binary weights experiments is that our method with 4-bit activations reaches $67.2\%$ and surpasses all other approaches with full precision or 8-bit activations. Looking at the stricter quantization setting with binary weights and 2-bit activations, our approach achieves the highest performance with $64\%$ top-1 accuracy. Over all approaches, our method demonstrates the best performance on **ImageNet-1k** for extreme quantization.

| Method | Weights [bits] | Activations [bits] | Top-1Accuracy |
|---|---|---|---|
| Baseline | 32 | 32 | 70.2 |
| LSQ [23] | 4 | 4/32* | **71.1** |
| HAWQ-V3[118] | 4 | 4 | 68.5 |
| SAT[48] | 4 | 4 | 70.0 |
| DL-QAT GML (ours) | 4 | 4 | **70.1** |
| ABC-Net [66] | 2 | 32 | 63.7 |
| INQ [123] | 2 | 32 | 66.0 |
| SAT[48] | 2 | 8 | 67.4 |
| DL-QAT GML (ours) | 2 | 8 | **67.9** |
| LSQ [23] | 2 | 2/32* | **67.6** |
| SAT[48] | 2 | 2 | 63.1 |
| DL-QAT GML (ours) | 2 | 2 | **64.0** |
| BWN [86] | binary | 32 | 60.8 |
| ABC-Net [66] | binary | 32 | 62.8 |
| BWNH [43] | binary | 32 | 64.3 |
| DSQ [29] | binary | 32 | 63.7 |
| Q-Networks [117] | binary | 32 | 66.5 |
| IR-Net [85] | binary | 32 | 66.5 |
| SYQ [24] | binary | 8 | 62.9 |
| SAT[48] | binary | 8 | **67.5** |
| DL-QAT GML (ours) | binary | 8 | **67.5** |
| SAT[48] | binary | 4 | 66.9 |
| DL-QAT GML (ours) | binary | 4 | **67.2** |
| PACT [19] | binary | 2/32* | 62.9 |
| LQ-Net [120] | binary | 2 | 62.6 |
| SAT[48] | binary | 2 | 63.1 |
| DL-QAT GML (ours) | binary | 2 | **64.0** |

* For LSQ and PACT, the residual connections remain in the accumulation dynamic.

Table 5.4: **ImageNet-1k** Top-1 Accuracy for extreme quantization settings of Resnet-18. DL-QAT and SAT results are obtained from our experiments, all the other results are reported from the original papers. DL-QAT and SAT use original Resnet-18. LSQ and PACT use full pre-activation Resnet-18. LQ-Net use Resnet-18 type-A shortcut. BWN use Resnet-18 type-B shortcut.

Figure 5.4 shows the results from Table 5.4 in a plot with the memory footprint of the network on the $x$ axis and the ImageNet top-1 test accuracy on $y$ axis. The memory footprint of a quantized neural network is approximated by summing the total memory storage for all the parameters at their respective bit-representation with the memory needed to contain the biggest activation. We considered that every method quantized the first and last layers parameters to 8bits. To achieve lower memory footprint quantized neural networks, the quantization settings get more and more extreme and the application performance drops significantly. Comparing DL-QAT to all other methods, it is visually

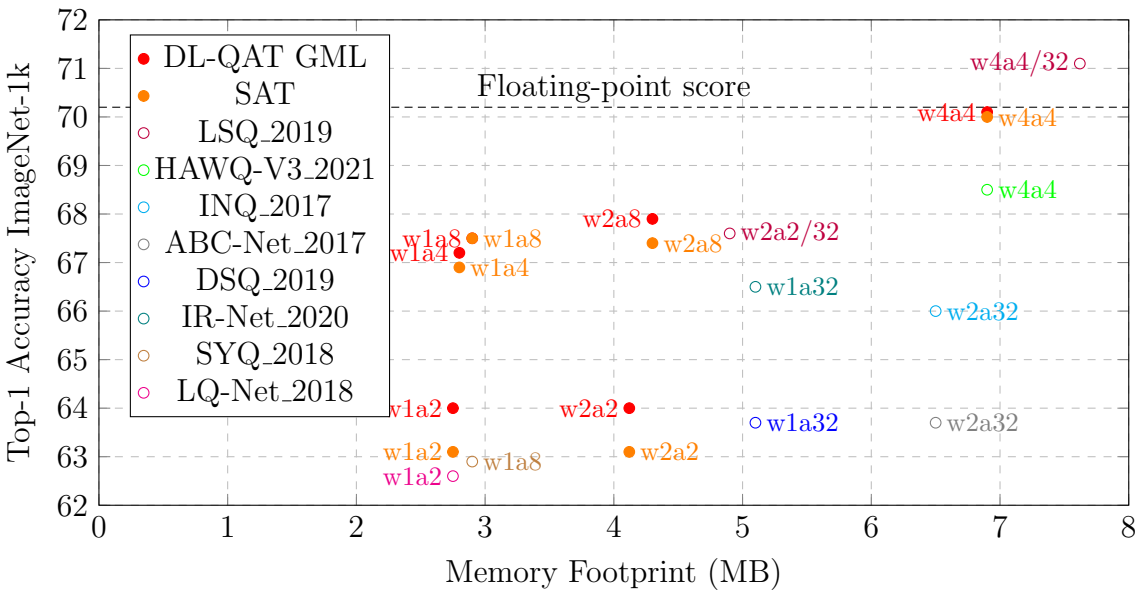



Figure 5.4: Memory footprint (Mega-Bytes) comparison of different precision settings from Table 5.4. Each point is one quantized network associated to the bit precision of its weight and activation. The filled round marks are results from our experiments while the empty round marks are results reported from the State of The Art.

clear that our method provides the best application performance over memory footprint trade-off.

### 5.4.3 Discussion and Perspectives

In this chapter, we target very-low-bit settings and study multiple losses to further reduce the gap in accuracy of quantization. Quantization-aware training techniques and margin-based losses are reviewed in order to show that QAT may benefit from disentangled losses. We introduce DL-QAT, a method combining quantization-aware training and disentangled losses, as our tool to investigate the contribution of those different loss functions for extreme quantization. Preliminary experiments on **CIFAR-10** and **CIFAR-100** are conducted to visualise and lighten the advantage of our method. We reduce the drop in top-1 accuracy on Resnet-18 with binary weights and 2-bit activations from 7.2% to 2.3%. Further results on **ImageNet-1k** show that our approach improves by nearly 1% the top-1 accuracy of Resnet-18 with binary weights and 2-bit activations. Overall, the experiments confirm our hypothesis and encourage future use and research of disentangled losses for Quantization Aware Training.

A first natural future work would first experiment our method on the variety of classical models for the image classification task. We would also like to experiment on other classification tasks as we think they would benefit from DL-QAT.

The approach of this contribution aims at further improving the Quantization Aware Training process to reach substantial improvement for very low-bit settings. One may consider the low-bit quantization from a representation capacity perspective. Indeed, when the model has binary parameters, its representation capabilities are far lesser than one with 8-bits or even 4-bits parameters. This might explain the significant drop in application performance for the most extreme quantization settings like binary or 2-bits models despite the progress of quantization methods. Trying to answer this limitation, some methods proposed to offer more flexibility by quantizing different parameter sets of the model to different bit-width. Further work could investigate the benefits of disentangled losses with the mixed-precision approach.

DL-QAT contributes to make the CNN inference further accessible for edge applications. As more and more neural network-based applications are deployed, DNN are more and more exposed to security threats like adversarial attacks. Understanding the vulnerability of those neural networks towards adversarial attacks and how to address it is more than ever a crucial problem. We address this question by proposing a defense system in the next chapter. Moreover, we provide first experiments to measure the impact of DL-QAT on the robustness of neural networks in Appendix A.

# Chapter 6

# Adversarial Robustness

## 6.1 Introduction

The very active research around neural networks and their compression make DNNs competitive candidates in many areas. As the logical next step, many applications are starting to be deployed on both edge and cloud. This industrial transfer is only going to grow even more in the next few years. In this context, DNNs are facing more and more threats from any attacker who wishes the system to misbehave. Understanding the security properties of deep learning is a crucial question in this area. Based on the definition of Confidentiality, Integrity, Availability (CIA) model, the security of neural systems can be defined by three aspects:

- Confidentiality: The neural model would leak sensitive data.
- Integrity: An attacker would alter the prediction of a classification model.
- Availability: An attacker would compromise the system into a failsafe mode.

In this chapter, we focus on the vulnerability of DNNs towards small perturbations. Such vulnerability is exploited by attackers to compromise the model integrity and to force models to make wrong predictions only by perturbing the inputs. To that end, an attacker computes adversary examples, as introduced by Szegedy *et al.* [103]. The purpose of the attack is to find a small variation of the input that is imperceptible to the human perception and yet that fools the neural model prediction. Taking an example from Fig. 6.1, the model correctly classifies the original image as a panda. But, adding the small perturbation to the original image, an adversary image is generated, which is misclassified as a gibbon. Despite the original image and the adversary image being indistinguishable to the human eye, the small perturbation is enough to fool the neural model prediction.

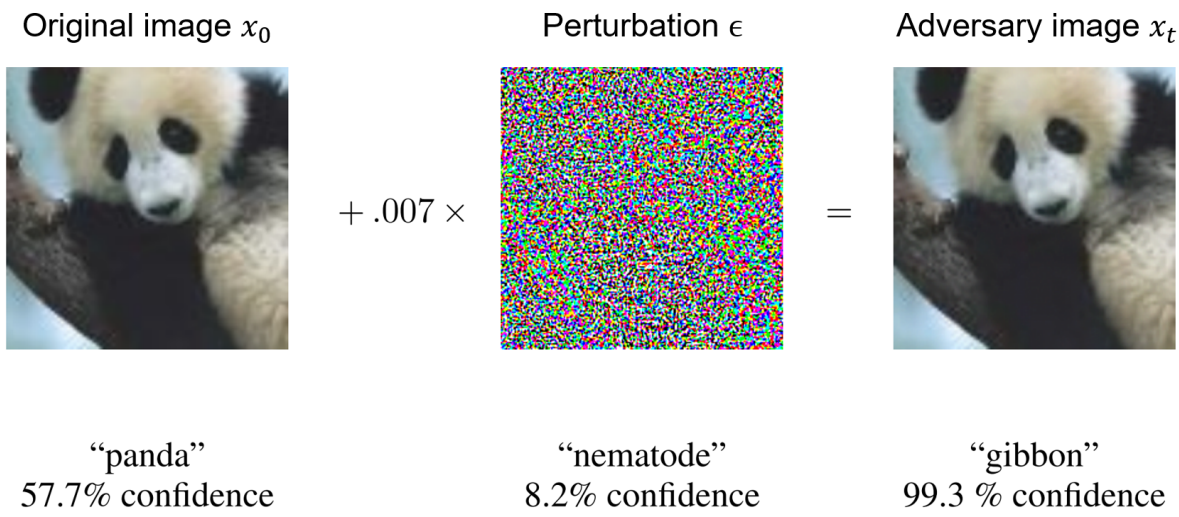



Figure 6.1: Example of an adversary image. Illustration from [32]

This chapter focuses on image classification tasks and is organized as follows. Section 6.3 presents the state of the art on adversarial attacks and defense mechanisms. Section 6.4 exposes the EHD system as the main contribution of this chapter with relevant experimentation.

## 6.2 Common concepts for adversarial attacks

An adversarial attack is an adversarial generation method targeting a specific model.

### 6.2.1 Distance metrics

In order to ensure that the perturbation to the original image remains indistinguishable to the human eye, all adversary generation methods constrains the perturbation. In the literature, distance metrics are used to bound the perturbation and all are $L_p = \|x - x^{adv}\|_p$ norms with the $p$-norm defined as:

$$\|v\|_p = \left( \sum_{i=1}^{n} |v_i|^p \right)^{\frac{1}{p}} \tag{6.1}$$

Three distance metrics are commonly used:

- The $L_0$ distance metric measures the number of pixels that have been changed by the adversary example. However, it does not constrain how much each pixel was changed.
- The $L_2$ distance metric measures the euclidean distance between $x$ and $x^{adv}$. This metric can remain small when many pixels have a small change, but it can also hide

some singular pixel big change due to the global pixel average in the root-mean-square process.

- The $L_\infty = \|x - x^{adv}\|_\infty = max(|x_1 - x_1^{adv}|..., |x_n - x_n^{adv}|)$ distance metric measures the maximum change above all pixels. Applying a bound on this distance directly limits the maximum change on all pixels.

### 6.2.2 Attacker goals

The attackers may have different reasons to target a specific algorithm. When an adversarial attack targets a classification model, it can choose between two goals:

- **Targeted attack.** The adversarial generation method tries to misguide the model to a particular class other than the true class.
- **Untargeted attack.** The adversarial generation method tries to misguide the model to predict any of the incorrect classes.

### 6.2.3 Attacker knowledge

Adversarial attacks can access different resources and there are two standard resource settings in the literature: white-box attacks and black-box attacks:

- **White-box attack.** The adversarial generation method has access to the model internals like its architecture and parameters as well as the training dataset.
- **Black-box attack.** The adversarial generation method only has access to the model inputs and outputs by submitting inference queries.

## 6.3 State of The Art

This section presents adversarial attacks, defense mechanism and some works studying the effect of quantization on adversarial robustness as well as the current limitations. In the past decade, the research in this field has been stacking many attack methods to counter the latest defense mechanisms. We present the ones that had the most impact in the domain.

### 6.3.1 White-box attacks

DNNs are end-to-end differentiable to enable learning with the backpropagation algorithm. With a white-box setting, the attacks dispose of all resources needed to use the backpropagation algorithm and find small perturbations very efficiently. To some extent, one could

say that DNN are vulnerable to adversarial attacks by construction.

The most basic white-box attack was introduced by Goodfellow *et al.* [32] in 2014, the Fast Gradient Sign Method (FGSM) attack. As described in Algorithm 6, it performs a gradient ascent based on the signs of the input image gradients. For instance on one image, the gradients of each pixel are computed with backpropagation and the signs of each pixel gradient decide to either add or subtract a small $\delta$ to each pixel. This method was designed to produce an adversary image fast rather than optimal. As the adversary image is generated with a one shot strategy, the perturbation on each pixel is bounded by the $\epsilon$ argument, *i.e.* the $L_\infty$ bound.

**Algorithm 6** The FGSM attack

1: **Inputs:** the target model $f$, the original image $x$ and $y$ its corresponding label, the loss function $L$, the perturbation on each pixel $\epsilon$.
2: **Outputs:** the adversary image $x^{adv}$
3: **FGSM**($f$,$x$,$y$,$L$,$\epsilon$):
4: $cost \leftarrow L(f(x), y)$
5: Backpropagation to obtain $x_{grad}$ based on *cost*
6: $x^{adv} \leftarrow x + \epsilon * Sign(x_{grad})$
7: $x^{adv} \leftarrow Clamp(x^{adv}, 0, 1)$
8: **return** $x^{adv}$

The Projected Gradient Descent (PGD) attack, introduced by Kurakin *et al.* [57], extends the FGSM attack with an iterative process. As described in Algorithm 7, the adversary image is generated iteratively with a small $\alpha$ and an $L_2$ bound or, more commonly, an $L_\infty$ bound. Obviously, the PGD attack proved to find better adversary examples than FGSM.

**Algorithm 7** The PGD attack

1: **Inputs:** the target model $f$, the original image $x$ and $y$ its corresponding label, the loss function $L$, the number of iterations *steps*, the iteration step $\alpha$, the maximum perturbation per pixel $\epsilon_{max}$.
2: **Outputs:** adversary image $x^{adv}$.
3: **PGD**($f$,$x$,$y$,$L$,$\alpha$, $\epsilon_{max}$):
4: $x^{adv} \leftarrow x$
5: **For** _ **in range**(steps):
6: $cost \leftarrow L(f(x^{adv}), y)$
7: Backpropagation to obtain $x^{adv}_{grad}$ based on *cost*
8: $x^{adv} \leftarrow x^{adv} + \alpha * Sign(x^{adv}_{grad})$
9: Clamp the perturbation following $\|x - x^{adv}\|_\infty < \epsilon_{max}$
10: $x^{adv} \leftarrow Clamp(x^{adv}, 0, 1)$
11: **return** $x^{adv}$

There are many other white-box adversarial attacks relying on backpropagation that

propose different objective functions to find adversary examples [17, 73, 81, 101]. Noticeably, Carlini and Wagner [17] introduced the C&W attack that is now a standard in the community. The proposed attack relies on a two-component objective function to balance between minimizing the perturbation and the adversary objective. For an in-depth presentation, we refer to the original paper [17].

### 6.3.2 Black-box attacks

Even without the knowledge of the internal structure and parameters of a targeted DNN model, *i.e.*, the backpropagation algorithm to compute gradients from the target model, black-box attacks can still exploit DNN vulnerability using different approaches. Those approaches are reviewed from the most basic one to the most elegant one in the rest of this section.

#### Random search

A random search attack is the brute force approach towards finding adversary examples. Random small perturbations are generated and added to the original image until one successfully fools the target model prediction. This attack is time and computation intensive. Still *et al.* [2] proposed an upgraded algorithm based on random search that converges faster than the naive random search.

#### Gradient estimation

While the backpropagation algorithm cannot be used to compute the gradient of the image, the gradient can still be approximated. Based on the definition of the derivative, the numerical gradient approximation is often used to perform gradient checking in the DNN domain. The idea is to consider the target model as the function and to approximate its gradient. Note that a DNN is a complex and highly non-linear function. Intuitively, estimating a DNN gradient is much harder than estimating the gradient of a classical function. To overcome this issue, Spall *et al.* [97] proposed an attack combining a relevant choice of perturbators and multiple approximations in an iterative procedure. The Simultaneous Perturbation Stochastic Approximation (SPSA) algorithm manages to converge effectively on target models. We further detail this attack in Appendix B. Despite being computation-extensive, this algorithm is still very efficient even with many defense mechanisms [77].

#### Attack by transfer

In 2016, Papernot, McDaniel, and Goodfellow [79] uncovered that the adversary examples can be transferred from one machine learning system to another. The transfer property

means that an adversary example generated to misclassify a target model can also misclassify another model. Most surprisingly, they also find that not only adversarial samples are transferable across models trained using the same machine learning technique, but also across models trained by different techniques.

Consequently, Papernot *et al.* [80] proposed a black-box attack relying on this transfer property. The strategy of the attack is to learn a substitute model from the queries delivered by the target model. Using the queries as training data, *i.e.*, inputs and corresponding outputs, the substitute model learns the classification problem and approximates the target model. The substitute model is transparent to the attacker and it is used to craft the adversary images with any adversarial attack, in particular, the efficiency of the backpropagation algorithm can be exploited. The target model is then expected to misclassify the adversary examples generated from the substitute model thanks to the transferability between architectures.

### 6.3.3 Defense mechanisms

Answering those attacks threats, many defenses were proposed in the past decade and this section overviews the main contributions.

#### Gradient masking

A gradient masking method deteriorates or invalidates the gradient computed with the backpropagation algorithm. This deteriorated gradient only serves poor purpose in the convergence of a classical white-box attack and those methods show good robustness. Several methods rely on randomization at inference time. Guo *et al.* [35] apply random transformations to the input, Xie *et al.* [116] add a randomized zero-padding before the classifier, and Pinot *et al.* [84] conduct a theoretical analysis of those randomization techniques. Some other methods rely on vanishing and exploding gradients or non-differentiable operations[96, 91].

However, Athalye *et al.* [4] show that these methods can be countered by attacks that bypass the gradient masking. In the case of randomization at inference time, they bypass the defense mechanism by estimating the gradient of the stochastic function with the Expectation Over Transformation (EOT) algorithm [5]. In the case of methods relying on vanishing and exploding gradients or non-differentiable operations, they propose to approximate the function that is non-differentiable or that introduces instability with tools such as the STE [10]. Finally, the authors argue that the evaluation of defense system robustness should also consider counter attacks that could exploit the knowledge of the defense system.

#### Robust optimization

Robust optimization makes the model itself more robust using optimization methods. Such a defense strategy is composed of methods that improve the optimization function either by adding regularization terms, certification bounds or adversarial examples in the training process. For a thorough review of those methods, we refer to the work of Silva and Najafirad [95]. Instead, we want to highlight that these methods search for a trade-off between robustness either with the application performance, as discussed in the contribution of Zhang *et al.* [121], or with computation and memory overhead at inference time. Taking the proposed MMC of Pang *et al.* [77] as an example, models trained with their loss are significantly more robust to white-box and black-box attacks than models trained with CEL. However, the CIFAR-100 score of their Resnet trained with MMC drops by 1% compared to the one trained with CEL. Some works based on an ensemble approach show both good robustness and application performance while increasing both the storage needed for each model instead of one and the inference computational cost as one input image will infer on each model [105, 78, 93, 108, 70].

#### Attack detection

Instead of a proactive approach to make the DNN robust to adversarial attacks, several works proposed a reactive approach where a defense system would seek to detect adversarial examples. Detecting adversarial examples usually involves statistical tools or an ensemble strategy that can distinguish between perturbed and normal images. Noticeably, Li *et al.* [63] proposed *Blacklight*, a black-box attack detection system using a fingerprint buffer. *Blacklight* distinguishes attack queries from benign queries by searching for this invariant in a model's stream of input queries. As it would be a very computational extensive task to perform on the inputs directly, *Blacklight* proposes a probabilistic algorithm that detects highly similar images using probabilistic fingerprints. The probabilistic fingerprint is a set of hash key computed from defined segments of the input image with a cryptographic hash function. They identify similar images, *i.e.*, the adversary images, when their respective sets of hash keys have a higher collision frequency than the average collision frequency.

However, *Blacklight* requires to store query fingerprints in order to compare the different fingerprints and detect an adversarial attack. Such a system becomes memory extensive as the number of queries increase. Indeed, for scalability purposes, they propose to reset the fingerprint storage every once in a while. In the end, it is still possible for an adversary to perform an undetected black-box attack.

### 6.3.4 Current limitations

Overall, defense mechanisms either seek to increase models robustness towards adversarial attacks with specific learning mechanisms or to detect adversarial attacks. While applying

those defenses mechanisms, two main limitations arise:

- The robustness gains trade-off with either the application performance [121] or computation and memory overhead at inference time [105, 78, 93, 108, 70].
- The defense mechanisms remain more or less vulnerable to gradient estimation attacks like SPSA [77].

In the next section, we then propose a defense system addressing those two limitations.

## 6.4 Ensemble Hash Defense (EHD)

Answering the limitations presented in Section 6.3.4, we propose EHD, a defense system to improve DNN robustness towards black-box attacks based on gradient estimation while maintaining the application performance.

### 6.4.1 Concept

EHD undermines gradient approximation with a two-step defense mechanism based on model ensemble and cryptographic hash functions.

**1. Learning phase**

Learn $k$ models. Each model needs to achieve similar performance.

**2. Inference phase**

At inference time, EHD combines a model ensemble strategy with an inference model selection process. Based on one input image, the selection process chooses one model from the $k$ models to infer the input image. Figure 6.2 illustrates the EHD inference pipeline for $k = 3$ models.

### 6.4.2 Diversity hypothesis

We formulate the hypothesis that the EHD system benefits from the diversity of the models involved. That is to say, combining models with more output diversity would introduce more misleading information during the attack. With more misleading information, the attack is less likely to converge, and so the EHD system shows better robustness. We identify the learning phase as the main way to induce diversity between the models. We list some training configuration to induce model diversity that we find interesting to investigate:

- Train the models with different objective functions.

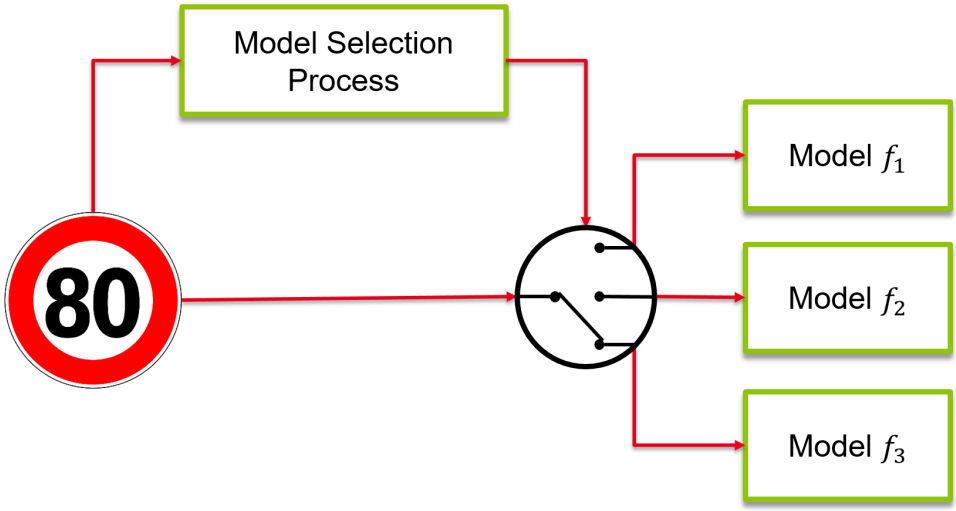



Figure 6.2: Ensemble Hash Defense principle

- Train the models with an ensemble objective instead of training the models independently.
- Train the models with robust optimization mechanisms from the state of the art.
- Train the models with different weight and activation quantization bit-width.
- Use different topologies for models.

### 6.4.3 Model selection process

This section details the model selection process and explains the relevance of each components. Figure 6.3 details the process of the model selection. Each input image is flattened into a sequence of bytes and encoded by a cryptographic hash function (*e.g.*, SHA-256 that is propagated with a random value chosen by the defender). Operating through a modulo $k$ operator, the resulting hash key determines the inference model $k_{infer}$. The input image is then only inferred on the $k_{infer}$ model.

The use of a cryptographic hash function is justified by its properties that are presented by order of importance:

- It is a one-way function, *i.e.*, a function for which it is practically infeasible to reverse the computation.
- Deterministic, *i.e.*, one input image will always have the same hash key. The hash function will always generate the same hash key from one input image, that is to say, one input image will always be inferred on the same model. In the end, this property prevents an adversary from using multiple inferences to target one of the models.

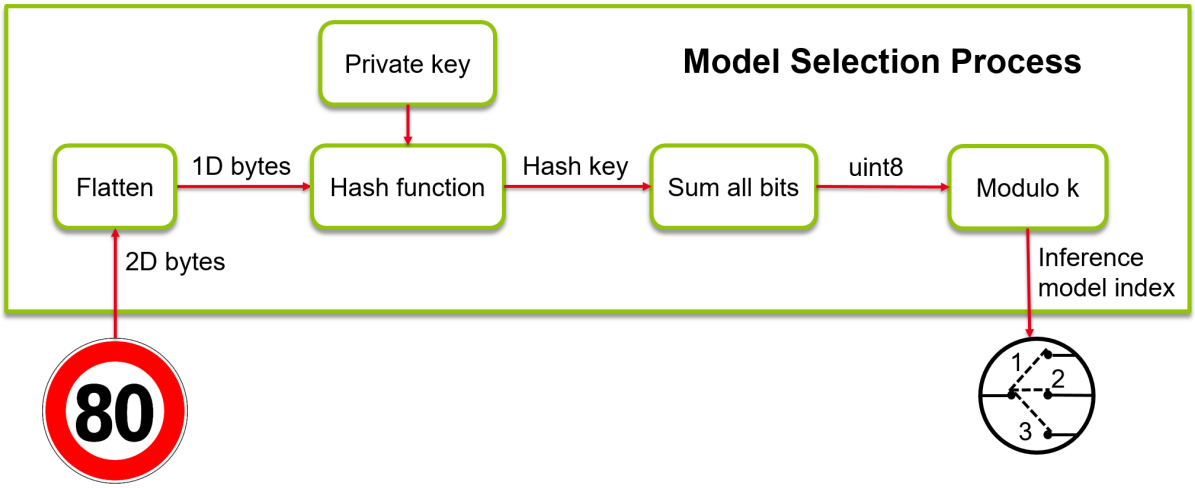



Figure 6.3: Model selection process.

- Easy to compute, so we consider the computation overhead negligible compared to the model inference.
- Any slight modification of the input image would change dramatically the hash key and so the inference model.
- It is impossible to create an image having the same hash key as another image. With our method perspective, one adversary would only need to create an image with the same modulo $k$. However, regarding the facts that an adversary would not have access to the number of models $k$, the random private key, and that, in order to complete the attack, he would need to keep a coherence for its attack to converge, it is practically impossible to force EHD.

The choice of the cryptographic hash function matters in terms of security and representation potential. Taking SHA-256 and SHA-512 Secure Hash Algorithm (SHA) as examples:

- Security: The number of operations needed to brute force revert the bit operator drastically increases from SHA-256 to SHA-512.
- Representation potential: The hash key size is 32 bits with SHA-256 while it is 64 bits with SHA-512. A higher number of bits allows a bigger representation potential and so less collision in the hash key domain.

Again, the main purpose of the hash function is to choose the inference model in an unpredictable and deterministic manner. The inference model is selected based on the hash key. The modulo operator truncates the hash key to a distribution of $k$ elements and directly returns the index of the selected inference model. We sum the bits with value 1 from the hash key to obtain the number of bits to 1 and apply the modulo $k$ operator.

### 6.4.4 Defense example

In order to understand the intuition of our approach, we propose to detail how our defense system functions while targeted by one iteration of SPSA attack. We use the following notations:

- *image* the original image,
- $f_1, f_2$ and $f_3$ the $k = 3$ models composing the defense system,
- $\delta$ the perturbation to perform the gradient approximation, and
- $n$ the size of the gradient approximation batch.

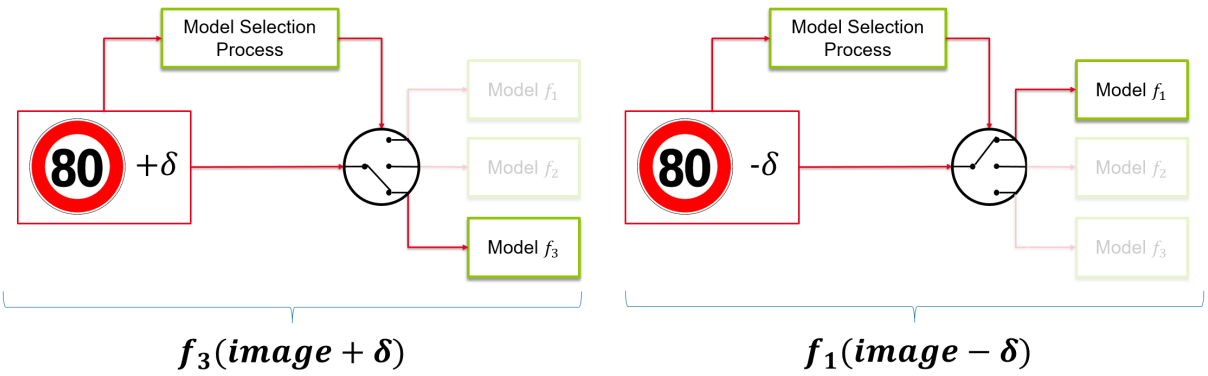



Figure 6.4: Ensemble Hash Defense targeted by SPSA attack.

To approximate the gradient, the attack performs two inferences with two inputs $image + \delta$ and $image - \delta$. Figure 6.4 shows a example where the model selection process chooses the model $f_3$ for the first input $image + \delta$ and the model $f_1$ for the second input $image - \delta$. The numerical approximation of the input gradient is:

$$Grad_{approx} = \frac{f_3(image + \delta) - f_1(image - \delta)}{2\delta} \tag{6.2}$$

For the SPSA attack to converge, the gradient approximation must carry relevant information. However, in the configuration described by Eq. (6.2), the formulation of the gradient approximation is not correct. In practice, one can distinguish two cases:

- Wrong gradient approximation: The chosen inference models are different with a frequency of $\frac{k-1}{k}$:

$$f_i \leftrightarrow f_j \text{ for } i, j \in [1..k] \text{ and } i \neq j$$

- Correct gradient approximation: The chosen inference models are the same with a frequency of $\frac{1}{k}$:

$$f_i \leftrightarrow f_i \text{ for } i \in [1..k]$$

The attack averages a batch of approximations of size $n$ to obtain a more precise gradient.

$$Grad^{avg}_{approx} = \frac{1}{n}\sum_{1}^{n}\frac{f_i(image+\delta) - f_j(image-\delta)}{2\delta} \; with \; i,j \in [1..k] \tag{6.3}$$

This average gradient contains wrong approximations as well as correct approximations. It is used to update the input image with a step of ADAM optimizer. Wrong gradient approximations are meant to mislead the attack convergence. Also, while correct gradient approximation carries relevant information for the attack on one specific model, different models can show opposite directions with their respective correct gradient approximation. In the end, even if the attack manages to converge, one adversary image might infer a model that would not be fooled. To summarize, EHD limits the convergence of SPSA attack by introducing misleading gradient approximation.

### 6.4.5 Advantages and limitations

The main advantage of our method Ensemble Hash Defense is that it can be combined with any robust optimization method in the recent SoTA to further enhance the final defense system robustness towards adversarial attacks.

A first limitation is the requirements to train $k$ models. Firstly, it requires more training resources and the usage of different training methods (examples in Section 6.4.2) to induce diversity in the response of the different models. Secondly, each model needs to reach similar application performance in order for the defense system to maintain the performance. As soon as one model has lower application performance, its usage with EHD at inference time will hinder the overall performance of the system.

Other defense systems that rely on model ensemble [105, 78, 93, 108, 70] infer one input on all models to produce one output. Such a pipeline leads to memory overhead for each model storage and computation overhead for each model inference. Whereas our method infer one input on only one model. Therefore, there is no computation overhead compared to a traditional model. However, the EHD system still has a memory overhead to store $k$ models at inference time. Finally, the Model Selection Process infers each input image with a cryptographic hash function like SHA-256. The computation overhead involved with the cryptographic hash function is considered negligible compared to the model inference.

## 6.5 Experiments

### 6.5.1 Evaluation setup

All experiments are based on residual blocks[38] with the **CIFAR-100**[54] dataset. All experiments are using the same ResNet-32 as the one used by Pang *et al.* [77]. All the 8-bit images are divided by 255 and no standardization is applied. The train data are augmented with random rescaling, cropping and flipping. The learning strategy is different according to the objective function used for training. All models trained with the CEL and the GML use a learning rate of 0.1 with a cosine annealing strategy over 150 epochs. All models trained with the MMC uses a learning rate of 0.01 with a step decay strategy factor of 0.1 on epoch 100 and 150 over a total epoch of 200. Also, all experiments with GML use a multiplicative margin $\alpha = 0.3$.

To evaluate the robustness of our EHD system, we compare how successful SPSA attacks are between EHD and single models. An SPSA attack generates an adversary test set specific to the targeted model from each image of the test set of CIFAR-100. To measure how successful is one attack targeting a model, we compute the accuracy obtained by the model with its corresponding adversary test set. The lower the accuracy, the more the model has been fooled by the adversary images and so, the more successful is the attack.

All SPSA attacks presented in this work are conducted in the untargeted mode and the $L_\infty$ distance between the original and adversary image bound by $\epsilon_{max} = 8/255$. In the untargeted mode, each adversary image only needs to fool the prediction of the model to any arbitrary class. In the experimentation, the intensity of the SPSA attacks are controlled with two parameters:

- The number of iterations, *i.e.*, the number of times the adversary images are updated by the ADAM optimizer. The more iteration steps, the better the convergence of the SPSA attack, and the stronger the attack.

- The gradient approximation batch size that determines how many gradient approximation are averaged for one update by the ADAM optimizer. The bigger the gradient approximation batch, the more precise the gradient approximation, the more meaningful each update, and the stronger the attack.

In this chapter, most of the results presented in the tables are SPSA attacks. All tables have the same format, from left to right, the columns show results with increasing number of iterations, and, from top to bottom, the lines show results with increasing batch

approximation size. In the end, the top left result is the less intense SPSA attack and the bottom right result is the most intense SPSA attack.

### 6.5.2 First results

In this section, our EHD system is configured with 3 ResNets that are trained independently with the CEL and a slightly different parameters initialization. Different parameters initialization are generated with a different random seed. Each ResNet converges to a similar top-1 accuracy on CIFAR-100, as reported in Table 6.1.

| Resnets | Top-1 Accuracy |
|---|---|
| ResNet-32 0 | 72.7 |
| ResNet-32 1 | 73.4 |
| ResNet-32 2 | 72.8 |

Table 6.1: ResNet performance on CIFAR-100 for three independent trainings

The SPSA attack results on the ResNet-32 0 are presented in Table 6.2. We only show the results on model 0 because models 1 and 2 have similar results. The less intense SPSA attack brings the accuracy down from 72.7% to 26.4%. As the intensity of SPSA attacks increases, the score of the attacks reaches an asymptote around $9,6\%$. It shows how vulnerable are vanilla image classification models to the SPSA attack.

| Approximation batch size | Number of iterations | | | |
|---|---|---|---|---|
| | 10 | 20 | 50 | 100 |
| 10 | 26.4 | 16.1 | 10.5 | 9.7 |
| 128 | 10.6 | 9.8 | 9.7 | 9.7 |
| 1024 | 9.8 | 9.7 | 9.7 | 9.6 |

Table 6.2: SPSA attack targeting the ResNet-32 0 with 72.7% top-1 test accuracy trained on CIFAR-100 with the cross entropy loss.

The SPSA attack results on the EHD system that uses the three Resnets (model 0, model 1 and model 2) are presented in Table 6.3. The defense system preserves the application performance of the best model with a top-1 accuracy of 73.4%. Comparing Table 6.2 with Table 6.3, the EHD system offers a better robustness for all settings of the SPSA attack. The EHD system also offers a better resilience to SPSA attacks than the single model. However, the SPSA attack with 100 iterations and a batch approximation size of 1024 targeting the EHD achieves 10.8% that is only 1% over the single model. The SPSA attack targeting the EHD still manages to converge well with enough iterations and a significant batch approximation size. We explain this with the configuration of the models

being too naive. Indeed, when trained independently with a slightly different initialization as the only difference in the learning strategy, the adversary images fooling one model are very likely to fool the other ones. One could speculate that in such configuration, multiple models would converge to a very similar solution, and that the optimization problems that those models are solving are not intricate enough to show drastically different solutions.

| Approximation batch size | Number of iterations | | | |
|---|---|---|---|---|
| | 10 | 20 | 50 | 100 |
| 10 | 52.2 | 45.8 | 37.5 | 33.7 |
| 128 | 32.8 | 23.4 | 17.0 | 15.8 |
| 1024 | 17.8 | 13.9 | 11.5 | 10.8 |

Table 6.3: SPSA attack targeting the EHD mechanism that achieves 73.4% top-1 test accuracy combining three ResNets-32 trained on CIFAR-100 with different initialisations and the cross entropy loss.

### 6.5.3 EHD with different objective functions

This section investigates the use of different objective functions to independently train the EHD models while inducing diversity. In order to assess our hypothesis, we selected the following objective functions:

- The classic CEL.

- The GML proposed by Wan *et al.* [111]. We use the same formulation as in Section 5.2.2:

$$L_{GM} = -\log \frac{e^{-d_y(1+\alpha)}}{e^{-d_y(1+\alpha)} + \sum_{j=1,\ j\neq y}^{C} e^{-d_j}} \tag{6.4}$$

$$\text{with } d_j = \frac{1}{2}(f - \mu_j)^2 \tag{6.5}$$

  The GML draws the distances $d_k$ between output features $f$ and the learned means $\mu_k$ to minimize the distance to the mean associated to the true label $d_{z_i}$. A positive margin factor $\alpha$ artificially inflates the distance $d_{z_i}$ to help regulate the convergence of the network.

- The MMC loss proposed by Pang *et al.* [77]:

$$L_{MMC} = \frac{1}{2}\|f - \mu_y^*\|_2^2 \tag{6.6}$$

  The MMC loss directly minimizes the square of the $L_2$ distance between output features $f$ and the preset center $\mu_y^*$ associated to the true label $y$.

Each objective function is used to train one model. Tables 6.2, 6.4 and 6.5 report the results of SPSA attacks targeting models trained with CEL, GML and MMC, respectively. While SPSA attacks on CEL and GML show very similar results, the MMC loss shows a better robustness for every setting of the SPSA attack. However, the SPSA attack with 100 iterations and a batch approximation size of 1024 targeting the MMC achieves 10.0%, which is similar to the other objective functions. Our results on the MMC loss extends the analysis of Pang *et al.* [77] with SPSA attacks on CIFAR-100. Under intense attack settings, the MMC is not more robust than CEL or GML.

| Approximation batch size | Number of iterations | | | |
|---|---|---|---|---|
| | 10 | 20 | 50 | 100 |
| 10 | 25.8 | 16.2 | 10.2 | 9.1 |
| 128 | 10.8 | 9.5 | 9.3 | 9.2 |
| 1024 | 9.2 | 9.3 | 9.3 | 9.3 |

Table 6.4: SPSA attack targeting a ResNet-32 with 73.5% top-1 test accuracy trained on CIFAR-100 with the gaussian mixture loss (GML).

| Approximation batch size | Number of iterations | | | |
|---|---|---|---|---|
| | 10 | 20 | 50 | 100 |
| 10 | 33.6 | 26.1 | 19.2 | 15.6 |
| 128 | 22.1 | 17.6 | 12.1 | 10.4 |
| 1024 | 16.7 | 12.9 | 10.5 | 10.0 |

Table 6.5: SPSA attack targeting a ResNet-32 with 71.9% top-1 test accuracy trained on CIFAR-100 with the max mahalanolis center (MMC) loss.

The EHD system then uses the three models trained with CEL, GML and MMC, achieving top-1 accuracies of 72.7%, 73.5% and 71.9%, respectively. The SPSA attacks targeting the EHD with the different objective function configurations are reported in Table 6.6. The defense system shows a top-1 accuracy of 73.2% that is lower than the best performance of 73.5% from the GML model. However, it still outperforms the top-1 accuracy of the CEL and MMC models. This configuration achieves significant robustness improvement over all the other experiments so far. Comparing the first configuration based on different parameter initialization in Table 6.3 to this objective function configuration in Table 6.6, there is a substantial gain of 30% in the most intense SPSA attack. These results strongly encourage our hypothesis on model diversity for EHD robustness.

### 6.5.4 Influence of the number of models

To further study the proposed defense system, we compare the robustness of the EHD systems with an increasing number $k$ of models. Tables 6.7 and 6.8 report the results of

| Approximation batch size | Number of iterations | | | |
|---|---|---|---|---|
| | 10 | 20 | 50 | 100 |
| 10 | 58.5 | 55.8 | 52.8 | 50.5 |
| 128 | 54.4 | 49.9 | 47.0 | 45.3 |
| 1024 | 48.5 | 45.0 | 41.9 | 41.0 |

Table 6.6: SPSA attack targeting the EHD mechanism that achieves 73.2% top-1 test accuracy combining three ResNets-32 trained on CIFAR-100 with three different loss functions: CEL, GML and MMC.

EHD systems with respectively $k = 4$ and $k = 5$ models, using the same configuration as in Section 6.5.2, *i.e.*, models trained independently with the CEL and a slightly different parameter initialization.

| Approximation batch size | Number of iterations | | | |
|---|---|---|---|---|
| | 10 | 20 | 50 | 100 |
| 10 | 54.1 | 49.4 | 42.0 | 38.0 |
| 128 | 37.3 | 28.0 | 20.0 | 17.8 |
| 1024 | 20.7 | 15.7 | 12.7 | 11.6 |

Table 6.7: SPSA attack targeting the EHD mechanism that achieves 73.4% top-1 test accuracy combining $k = 4$ ResNets-32 trained on CIFAR-100 with different initialisations and the cross entropy loss.

| Approximation batch size | Number of iterations | | | |
|---|---|---|---|---|
| | 10 | 20 | 50 | 100 |
| 10 | 55.6 | 51.3 | 44.3 | 41.0 |
| 128 | 39.0 | 30.5 | 22.3 | 19.1 |
| 1024 | 22.3 | 16.8 | 12.8 | 12.0 |

Table 6.8: SPSA attack targeting the EHD mechanism that achieves 73.1% top-1 test accuracy combining $k = 5$ ResNets-32 trained on CIFAR-100 with different initialisations and the cross entropy loss.

Comparing the EHD system with 3, 4 and 5 models, with the respective Tables 6.3, 6.7 and 6.8, and with the naive configuration, only limits slightly the success of the SPSA attacks. Such configuration offers a poor trade-off between the memory storage and the robustness. However, increasing the number of models could still offer an interesting trade-off with other configurations such as different objective functions or an ensemble objective. Based on this work, a first investigation would be to add a model learned with another loss function, e.g. the AMS loss used in Chapter 5, to the EHD system from Section 6.5.3.

## 6.6 Discussion and perspectives

In this chapter, we introduced some of the key contributions from the literature on adversarial attacks. We then uncover from the State of The Art that defense methods tarde-off robustness either over application performance or computation and memory overhead at inference time. Also, many defense methods are still vulnerable to adversarial attacks based on gradient estimation such as SPSA. To overcome those shortcomings, we propose the EHD system that enables better resilience to adversarial attacks based on gradient approximation while preserving application performance and only requiring a memory overhead at inference time. In the best EHD configuration, our system achieves a resilience of 41% compared to a resilience of 10.0% for the MMC loss defense [77] under the most intense SPSA attack. We believe that this result is only the first step towards more robust systems based on the simple concept introduced with EHD.

We identify the memory overhead induced by the model redundancy as the main bottleneck for the deployment on edge of the EHD system. The concept of EHD makes it compatible with neural network compression methods like weight sharing and quantization. A first idea would be to use weight sharing between the models. A naive solution would simulate different models by considering a set of different layers inside a common topology. During inference, the Model Selection Process would select one processing path from the set of different layers instead of selecting a model. Only redundant layers would need to be stored instead of redundant models thus drastically reducing the memory overhead at the inference phase. Using quantization on each model in the EHD system would drastically reduce the memory storage overhead as well as the memory bandwidth and computational workload required for inference.

Future work would also investigate more configurations listed in Section 6.4.2: an ensemble objective to train the models and induce diversity, use robust optimization mechanism from the state of the art or use different topologies for the models.

We investigated the work of Pang *et al.* [78], who proposed to pair an ensemble loss with a term to encourage the logits of each model to be orthogonal. Preliminary experimentation based on this loss showed that their method cannot be directly applied with EHD. Indeed, the global objective allows good average performance with model diversity but undermines the performance of each model. Thus, using those models with the EHD system only results in poor application performance. Despite this first experimentation, we emphasize on the potential of combining a diversity ensemble loss with the EHD system in order to further extend resilience of neural networks to adversarial attacks. For further investigation, we find the contribution of Huang *et al.* [44] to be an interesting lead as they proposed a diversity loss based on the gradient of the input.

Another perspective would be to evaluate the resilience of the EHD system when targeted by other black-box and white-box attacks. The transfer black-box attack from Papernot *et al*. [80] is a good first candidate for this investigation as it would give feedback on the transferability between the models involved in the EHD system. We think that diversity of the different inference models is a key point to induce poor transferability of the adversary images between models and so good resilience towards many black-box and white-box adversarial attacks. Bernhard *et al*. [13] also discussed the quantization shift phenomenon as a way to induce poor transferability. Their results further motivate the use of quantized models in the EHD system to reduce the memory storage overhead and induce poor transferability at the same time. As a first step, we provide experiments to measure the impact of our proposed quantization method DL-QAT on the robustness of neural networks in Appendix A. Further experiments could use $k$ models quantized with DL-QAT at different precision settings into the EHD defense system to evaluate if the quantization shift phenomenon adapts to the EHD method.

# Chapter 7

# Conclusion

## 7.1 Summary of our results

This research project contribution is twofold, on the one hand, the compression and acceleration of neural networks for deployment on edge devices and on the other hand, the robustness of neural networks towards adversarial attacks. In particular, we addressed the problematic of extreme quantization of convolutional neural networks in order to enable inference on resource-scarce targets such as microcontrollers. Then we addressed the problematic of neural network robustness towards adversarial attacks so that it requires less resources for inference on edge.

In Chapter 2, we introduced typical neural network types like deep feed forward neural networks, convolutional neural networks and recurrent neural networks as well as different methods to compress neural networks. Among the neural network compression and acceleration methods, we highlight the advantages of quantization. The parameters of the neural network are compressed to make it less demanding on memory and once the quantization approximation is applied to the inference dataflow and operators, it increases throughput while being more energy efficient on real-time platforms with limited computing resources.

In Chapter 3, we introduced the rich development of deep learning libraries for neural network training and deployment on edge devices alongside the tasks and datasets used in this work. The maturity of deep learning frameworks makes neural networks design easily accessible and benefits both research and the usage of neural network for industrial applications. However, the deployment of deep neural networks on edge is still a challenging task. So far, the current neural network deployment libraries propose solutions for a restricted set of targets. The contribution of N2D2 is to unify and normalize processes for deep neural network deployment on edge on a large set of hardware targets. This work contributes to this effort as the proposed quantization method DL-QAT is implemented in N2D2.

In Chapter 4, we proposed to transfer an existing weight compression method based on pruning from CNN to RNN, the winning ticket. We investigated the convergence of RNN and its influence on pruning method performance. In our experiment settings, RNN showed an unstable convergence profile and we observed that it directly impacted the performance of the pruning method. In the end, we found winning tickets on recurrent architectures bypassing those instabilities. While magnitude pruning methods achieve significant model compression, the resulting sparse architecture cannot be accelerated easily. To overcome this limitation, we chose to focus on a different class of methods : quantization.

In Chapter 5, we detailed our main contribution, Disentangled Loss Quantization Aware Training, which we used to further improve the performance of convolutional neural networks under extreme quantization settings by relying on quantization friendly loss functions. Our experiments, conducted on CIFAR-10, CIFAR-100 and Imagenet-1k datasets, showed superior results compared to other state of the art approaches. Our interpretation is that extreme quantization benefits from the better clustering of the classes provided by those disentangled loss functions in comparison to the classic cross entropy loss. This work contributes to make the CNN inference further accessible for edge applications. As more and more neural network-based applications are deployed, DNN are more and more exposed to security threats like adversarial attacks. Understanding the vulnerability of those neural networks towards adversarial attacks and how to address it is more than ever a crucial question.

Finally, Chapter 6 we proposed Ensemble Hash Defense, a defense system that enables better resilience to adversarial attacks based on gradient approximation while preserving application performance and only requiring a memory overhead at inference time. In the best EHD configuration, our system achieves significant robustness gains compared to baseline models and a loss function-driven approach. Moreover, any existing robust optimization mechanism can be used to further enhance the robustness of the final system. We discuss its main limitation on memory overhead at inference time. In fact, we put in perspective the EHD system inference on edge thanks to its compatibility with compression methods like quantization or weight sharing. Taking a step back from this work, as one considers to use neural networks for an application, he would need to take in account the risks of adversarial attacks and if an existing robust method allow to reduce the resulting criticality to an acceptable level.

## 7.2 Potential improvements

While the proposed methods for quantization, DL-QAT and for adversarial robustness, EHD presents interesting performance and advantages, we also highlight some current limitations that may serve as a starting point for future improvement.

### 7.2.1 Mixed precision

With DL-QAT, our approach focuses on quantizing all weights and activations to the same precision so that the inference dataflow has an homogeneous precision throughout the network and simplifies the hardware implementation. However, when convolutional neural networks reach extreme quantization settings, we found that some activation needs higher precision to maintain the application performance, like the residual connection in ResNets. To overcome this limitation, mixed precision quantization seems to be a relevant future direction to investigate.

### 7.2.2 Transferability of adversarial examples

In 2016, Papernot N., McDaniel P. and Goodfellow I. [79] uncovered that the adversary examples can be transferred from one machine learning system to another. The transfer property means that an adversary example generated to misclassify a target model can also misclassify another model. Meanwhile, Bernhard *et al.* [13] discussed that models quantized to different precisions show a poor transferability of adversarial examples. We think there is a potential improvement for the EHD in this direction.

## 7.3 Future research directions

Adding to the limitations and perspectives of the different contributions discussed in the chapters and in the previous section, I provide in this section my thoughts on research directions that I find relevant.

### 7.3.1 Neural architecture search for embedded applications

So far, most of the approaches to make neural networks viable for embedded constraints focused on creating efficient structures like ResNets, MobileNets, SqueezeNet or EfficientNet and on the design of methods to compress and/or accelerate the network neural networks like pruning, quantization or knowledge distillation.

In contrast, neural architecture search approach automatizes the architecture search. Neural Architecture search methods explore a lot of potential architecture solutions in order to find efficient architectures that could not be designed by hand. However those methods require a tremendous amount of computing to explore the search space. Yet, several recent approaches of neural architecture search overcome this complexity and show interesting results on making neural networks viable for edge deployment :

- Cai *et al.* [15] proposed to train a once-for-all network that supports diverse architectural settings. Their shrinking algorithm allows to efficiently find specialized architectures that are able to fit different hardware platforms and latency constraints

while maintaining the same level of accuracy as training from scratch on a classic topology.

- Zeghidour *et al.* [87] proposed DiffStride, a method that downsample the feature maps by learning the convolution stride with the gradient-based optimization strategy. One of the main advantages is the fast convergence thanks to the gradient based learning. Using this method, the authors are able to generate semi-specialized CNN architecture from existing CNN topology like ResNet that have a lower computational cost for the same application performance.

Those NAS approaches show very competitive results compared to traditional approaches and have a high potential for future deployment of neural network applications on edge.

### 7.3.2 Self-supervised learning

Self supervised learning has the main advantage that data does not need to be labeled for a neural network to be learned. As such, it carries the hope to better exploit the potential of the colossal amount of unlabeled data. Self supervised learning unlocks training on much bigger datasets than supervised learning. In the recent years, many works investigated self supervised learning and proposed improvements in generative learning, contrastive learning and task learning. The state of the art shows that further scaling up the models to learn data representations on such big datasets enable better performance (the language model GPT-3, speech representations with Wav2vec 2.0 [92] and stable diffusion [88]) and unlocks new exciting usecases (text-to-image, text-to-audio ...). Thanks to the quality of the representations learned by Wav2vec 2.0, the model easily transfers to downstream tasks with a limited amount of data. This would benefit many niche applications including the ones inferred on the edge. How to transfer and downscale the knowledge of those new blocks for edge inference is a relevant problema to be explored in further research.

# Appendix A

# Quantization and Adversarial Robustness

## A.1 Previous work

In Chapter 5, our work on quantization aware training contributes to learning more efficient DNN for deployment. Understanding the security properties of those deployed models is a crucial question. Here, our motivation is to study a possible bridge between quantization and robustness to enable efficient and secure DNN deployment. We present some works that studied this topic. The early work of Galloway *et al.* [27] empirically showed on MNIST that stochastic quantization and especially binarization offer interesting robustness to white-box attacks. On the contrary, preliminary experiments with FGSM targeting Wide ResNets that were trained on CIFAR10 of Lin *et al.* [65] showed that the more extreme the quantization, the less robust the model. Moreover, Bernhard *et al.* [13] highlighted that adversarial examples have poor transferability between full precision models and quantized models. Based on this phenomenon, Bernhard *et al.* [13] and Sen *et al.* [93] both proposed a defense with an ensemble of quantized models with different precision settings. These defense mechanisms achieve substantially better robustness towards white-box attacks, while requiring a computation and memory overhead at inference time due to the ensemble strategy.

## A.2 DL-QAT and Adversarial Robustness

In this section, we extend the previous studies covering quantization and adversarial robustness by evaluating the robustness of the models learned with our DL-QAT method on the CIFAR100 dataset. Adding to the GML [111], the loss proposed by Pang *et al.* [77] is used in our quantization method. The MMC is designed to learn more robust models

and is formulated as followed:

$$L_{MMC} = \frac{1}{2}\|f - \mu_y^*\|_2^2 \tag{A.1}$$

The MMC loss directly minimizes the square of the $L_2$ distance between output features $f$ and the preset center $\mu_y^*$ associated to the true label $y$. This loss does not rely on softmax.

The GML and the MMC share the same minimization idea as they both minimize distances between features and class centers with the difference that MMC pre-defines fixed class centers while GML learns those centers. We refer to Pang *et al.* [77] for the definition of the centers.

All the presented models are the ResNet architecture used by Pang *et al.* [77] and are based on bottleneck-like blocks. All the 8-bit images are divided by 255 and no standardization is applied. The train data are augmented with random rescaling, cropping and flipping. The learning strategy is different according to the objective function used for training. All models trained with the CEL and the GML uses a learning rate of 0.1 with a cosine annealing strategy over 150 epochs. All models trained with the MMC uses a learning rate of 0.01 with a step decay strategy factor of 0.1 on epoch 100 and 150 over a total epoch of 200. Also, all experiments with GML use a multiplicative margin $\alpha = 0.7$ as it gives best quantization results.

To evaluate the robustness of the models quantized with our DL-QAT method, we compare how successful PGD (Algorithm 7) attacks are between models learned with DL-QAT and models learned with SAT [48]. The PGD attack generates an adversary test set specific to the targeted model from each image of the test set of CIFAR-100. To measure how successful is one attack targeting a model, we compute the accuracy obtained by the model with its corresponding adversary test set. The lower the accuracy, the more the model has been fooled by the adversary images and so, and the more successful the attack.

All PGD attacks presented in this work are conducted in the untargeted mode and the $L_\infty$ distance between the original and adversary image bound by $\epsilon_{max} = 8/255$. The attacks are all performed over a total of 10 iterations with an iteration step $\alpha = 2/255$. In the untargeted mode, each adversary image only needs to fool the prediction of the model to any arbitrary class.

The results of PGD attacks targeting models learned with SAT method and our DL-QAT method, in full precision and several low-bit settings, are reported in Table A.1.

The top-1 test accuracies on **CIFAR-100** of the proposed DL-QAT method with the

| | | SAT[48] | | DL-QAT GML | | DL-QAT MMC | |
|---|---|---|---|---|---|---|---|
| W [bits] | A [bits] | Acc | PGD | Acc | PGD | Acc | PGD |
| 32 | 32 | 72.9 | 5.9 | **73.8** | 10.1 | 70.8 | **17.3** |
| 4 | 4 | 72.1 | 7.9 | **73.0** | 10.9 | 69.4 | **19.1** |
| 2 | 2 | 69.8 | 8.9 | **70.5** | 10.7 | 67.9 | **18.5** |
| binary | 4 | 70.7 | 9.2 | **72.2** | 10.7 | 69.3 | **18.1** |
| binary | 2 | 68.2 | 7.1 | **69.8** | 10.7 | 66.1 | **17.8** |

Table A.1: Top-1 accuracy (%) on the white-box adversarial examples crafted on the test set of CIFAR-100 targeting ResNet-32 quantized with DL-QAT.

Gaussian Mixture Loss (DL-QAT GML) and the Max Mahalanolis Loss (DL-QAT MMC) compared to the Scaled Adjust Training (SAT) method are reported in the corresponding *Acc* columns of Table A.1. The line with 32 corresponds to single-precision floating-point, which is considered as the full precision baseline. Table 5.3 also reports the white-box attack Projected Gradient Descent (PGD) attack targeting each model. The resulting accuracy displayed in the PGD columns corresponds to the score of the targeted model on the adversary examples generated from the test set. The higher the score, the more robust the model towards the PGD attack.

As we read Table 5.3 from top to bottom, the quantization is more and more aggressive. In accordance with our previous results in Chapter 5, the performance (*Acc*) of DL-QAT GML method outperforms both SAT and DL-QAT MMC in all precision settings. Looking at the PGD scores, all the models are vulnerable with a drop in performance ranging from around 50% for DL-QAT MMC to $60 - 65\%$ for SAT and DL-QAT GML. Nevertheless, the MMC loss, initially designed to promote robustness, shows a better robustness towards PGD attack than SAT and DL-QAT, and our experiment further extends the scope of the original paper with quantized models. However, the main drawback of the MMC Loss is the compromise between application performance and the robustness gain. Indeed, each model trained with DL-QAT MMC shows almost always a 2% drop in score than the SAT with the cross-entropy loss.

Contrary to other state-of-the-art works [65, 27, 13], we find that using DL-QAT to quantize resnets with more and more aggressive precision settings has little to no influence on the robustness of the models towards PGD attack. One relevant direction to investigate would be constructing the Ensemble Hash Defense system proposed in Chapter 6 with quantized models of different precision to exploit the potential poor transferability hypothesis formulated in Bernhard *et al.* [13].

# Appendix B

# Simultaneous Perturbation Stochastic Approximation

This appendix presents the concept of the adversarial attack SPSA introduced by Spall *et al.* [97] along with one iteration example. In an image classification context, an adversarial attack generates adversary images, *i.e.*, slight variations from original images, that fool the prediction of a target model, while the differences between adversary and original images are imperceptible to the human perception. Following Fig. B.1, the adversary attack tries to find a small perturbation $\epsilon$ which, once added to the original image $x_0$, gives the adversary image $x_t$ and fools the model $f$ prediction during inference. A human easily identifies both original and adversary images as pandas, whereas the model is fooled by the perturbation $\epsilon$ and predicted a gibbon.

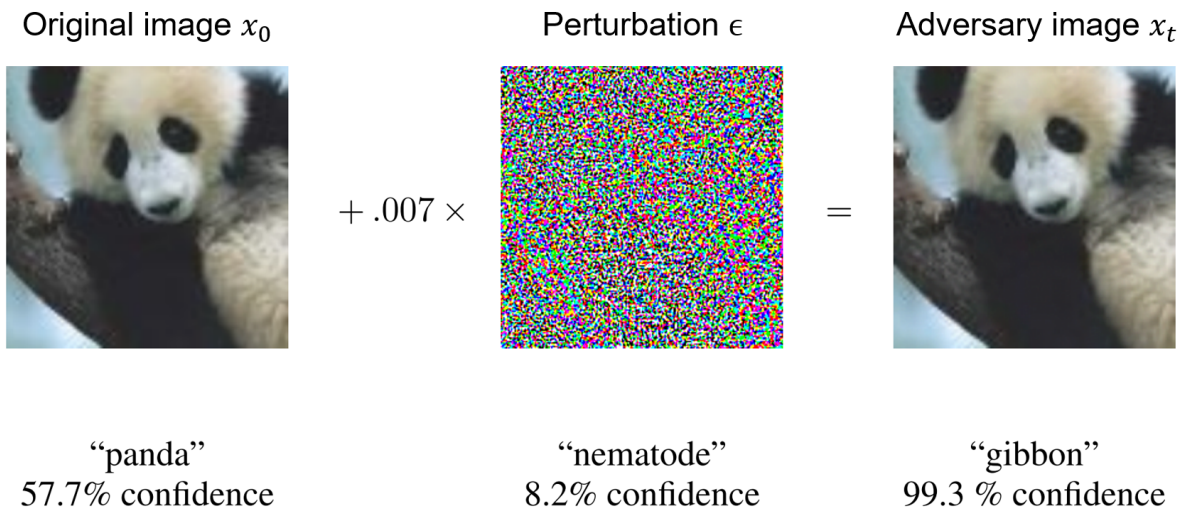



Figure B.1: Example of an adversary image. Illustration from [32]

The SPSA black-box attack that relies on the derivative definition to approximate the gradient of the perturbation $\epsilon$. The perturbation is iteratively updated with the stochastic

gradient descent. In order to keep the perturbation $\epsilon$ imperceptible to the human eye, the $L_\infty$ distance between the original and adversary image is bound by $\epsilon_{max}$. For a better understanding, we take the example of the iteration $t$ with:

- $x_t$ the adversary image at iteration $t$ ($x_0$ being the original image).
- $f$ the model.
- $out = f(x)$ the output vector of the model and $y_{pred}$, $y_{true}$ the predicted label and the true label of the input $x$. $top1$ and $top2$ refer to the maximum value and second maximum value of a vector.
- $\delta$ the perturbation to perform the gradient approximation. Each pixel will be affected by a value of $\pm 0.01$ from a Bernoulli law.
- $n$ the size of the gradient approximation batch.
- $\alpha$ the learning step.
- $\epsilon_{max}$ the perturbation bound.

The error function $L$ aims at fooling the network prediction with the closest second prediction of the network:

$$L(out, y_{true}) = \begin{cases} 0 & if\ y_{pred} \neq y_{true} \\ out_{top1} - out_{top2} & otherwise \end{cases} \tag{B.1}$$

For the gradient approximation to be meaningful, the algorithm performs $n$ approximations to average. The bigger the gradient approximation batch, the more precise the average gradient approximation, and the more meaningful one update.

$$Grad^{avg}_{approx} = \frac{1}{n}\sum_{i=1}^{n} \frac{L(f(x_t + \delta_i), y_{true}) - L(f(x_t - \delta_i), y_{true})}{2\delta_i} \tag{B.2}$$

This average gradient approximation is then used to update the adversary image as

$$x_t^{'} = x_t - \alpha Grad^{avg}_{approx}. \tag{B.3}$$

The adversary image is then clamped according to the $L_\infty$ bound and the image bound as

$$\|x_0 - x_t^{'}\|_\infty < \epsilon_{max}\ \&\ x_t^{'} \in [0, 1]. \tag{B.4}$$

This $L_\infty$ bound on images can be interpreted as a limit to the variation of $\epsilon_{max}$ in absolute value on each pixel, without any limit on the number of pixels that are modified. Once

the update is bounded, the adversary image is ready for another iteration:

$$x_{t+1} = x_t^{'} \tag{B.5}$$

One noticeable aspect of the SPSA attack is its computational needs. For $k$ iterations, the number of inferences needed to complete the attack follows $\mathcal{O}(kn)$. Increasing both the number of iterations and the gradient approximation batch greatly increases the computational needs. In practice, the Adam optimizer is used instead of the stochastic gradient descent as it makes the attack converge with less iterations. For an in-depth explanation of the attack, we refer to Spall *et al*. [97] and Uesato *et al*. [106].

# Acronyms

**AMS** Additive Margin Softmax 90, 93–98, 119

**BPTT** Backpropagation Through Time 48, 49, 77

**CEL** Cross Entropy Loss 76, 84, 90, 91, 94–97, 99, 109, 115–119, 128

**CNN** Convolutional Neural Networks 8, 61, 62, 88

**CNNs** Convolutional Neural Networks 28

**DL-QAT** Disentangled Loss Quantization Aware Training 22, 94, 95, 99–102, 121, 123–125, 127, 128

**DNN** Deep Neural Networks 27, 53, 61, 62, 72, 88, 90, 102, 103, 105–107, 109, 110, 124, 127

**EHD** Ensemble Hash Defense 13, 104, 110, 114–121, 124, 125

**EOT** Expectation Over Transformation 108

**FGSM** Fast Gradient Sign Method 106, 127

**GML** Gaussian Mixture Loss 90, 93–99, 115, 117, 118, 127, 128

**LSTM** Long Short Term Memory 28, 30, 49–51, 69, 71–73, 76, 78, 80, 83

**MLP** MultiLayer Perceptrons 8, 88

**MMC** Max-Mahalanobolis Center loss 13, 109, 115, 117, 118, 120, 127, 128

**MNIST** Modified National Institute of Standards and Technology 68

**N2D2** Neural Network Design & Deployment 95

**PGD** Projected Gradient Descent 106, 128

**QAT** Quantization Aware Training 90–92, 94

**RNN** Recurrent Neural Networks 8, 46, 48, 61, 62, 71–73, 87, 88

**RNNs** Recurrent Neural Networks 28, 30

**SAT** Scaled Adjust Training 22, 60, 91, 92, 94, 95, 97, 99, 100, 128

**SHA** Secure Hash Algorithm 112

**SPSA** Simultaneous Perturbation Stochastic Approximation 107, 113–116, 120, 130

# Bibliography


[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous systems, 2015.

[2] Maksym Andriushchenko, Francesco Croce, Nicolas Flammarion, and Matthias Hein. Square attack: a query-efficient black-box adversarial attack via random search. In *European Conference on Computer Vision*, pages 484–501. Springer, 2020.

[3] Martin Arjovsky, Amar Shah, and Yoshua Bengio. Unitary evolution recurrent neural networks. In *International Conference on Machine Learning*, pages 1120–1128. PMLR, 2016.

[4] Anish Athalye, Nicholas Carlini, and David Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. In *International conference on machine learning*, pages 274–283. PMLR, 2018.

[5] Anish Athalye, Logan Engstrom, Andrew Ilyas, and Kevin Kwok. Synthesizing robust adversarial examples. In *International conference on machine learning*, pages 284–293. PMLR, 2018.

[6] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[7] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[8] Ron Banner, Yury Nahshan, and Daniel Soudry. Post training 4-bit quantization of convolutional networks for rapid-deployment. *Advances in Neural Information Processing Systems*, 32, 2019.

[9] Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, James Bergstra, Ian Goodfellow, Arnaud Bergeron, Nicolas Bouchard, David Warde-Farley, and Yoshua Bengio. Theano: new features and speed improvements. *arXiv preprint arXiv:1211.5590*, 2012.

[10] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

[11] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[12] James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. Theano: A cpu and gpu math compiler in python. In *Proc. 9th python in science conf*, volume 1, pages 3–10, 2010.

[13] Remi Bernhard, Pierre-Alain Moellic, Jean-Max Dutertre, and France Gardanne. Adversarial robustness of quantized embedded neural networks. *Computer & Electronics Security Applications Rendezvous*, pages 1–33, 2019.

[14] Olivier Bichler, David Briand, Vincent Lorrain, Vincent Templier, Inna Kucher, Cyril Moineau, Johannes Thiele, Thibaut Goetghebuer-Planchon, et al. N2d2 : Neural network design and deployment. https://github.com/CEA-LIST/N2D2.

[15] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once-for-all: Train one network and specialize it for efficient deployment. *arXiv preprint arXiv:1908.09791*, 2019.

[16] A. Carbon, J.-M. Philippe, O. Bichler, R. Schmit, B. Tain, D. Briand, N. Ventroux, M. Paindavoine, and O. Brousse. Pneuro: A scalable energy-efficient programmable hardware accelerator for neural networks. In *2018 DATE*, pages 1039–1044, 2018.

[17] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *2017 ieee symposium on security and privacy (sp)*, pages 39–57. IEEE, 2017.

[18] Jean-Jacques Ruch-Marie-Line Chabanol. Chaînes de markov. https://www.math.u-bordeaux.fr/~mchabano/Agreg/ProbaAgreg1213-COURS5-CM.pdf.

[19] Jungwook Choi, Zhuo Wang, Swagath Venkataramani, Pierce I-Jen Chuang, Vijayalakshmi Srinivasan, and Kailash Gopalakrishnan. Pact: Parameterized clipping activation for quantized neural networks. *arXiv preprint arXiv:1805.06085*, 2018.

[20] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[21] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. *Advances in neural information processing systems*, 28, 2015.

[22] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *IEEE CVPR*, pages 248–255, 2009.

[23] Steven K Esser, Jeffrey L McKinstry, Deepika Bablani, Rathinakumar Appuswamy, and Dharmendra S Modha. Learned step size quantization. In *ICLR*, 2020.

[24] Julian Faraone, Nicholas Fraser, Michaela Blott, and Philip HW Leong. Syq: Learning symmetric quantization for efficient deep neural networks. In *IEEE CVPR*, pages 4300–4309, 2018.

[25] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *ICLR 2019*, 2018.

[26] Jonathan Frankle, Gintare Karolina Dziugaite, Daniel M Roy, and Michael Carbin. The lottery ticket hypothesis at scale. *arXiv preprint arXiv:1903.01611*, 2019.

[27] Angus Galloway, Graham W Taylor, and Medhat Moussa. Attacking binarized neural networks. *arXiv preprint arXiv:1711.00449*, 2017.

[28] Aidan Gomez. Backpropogating an lstm : A numerical example. https://medium.com/@aidangomez/let-s-do-this-f9b699de31d9, 2016.

[29] Ruihao Gong, Xianglong Liu, Shenghu Jiang, Tianxiang Li, Peng Hu, Jiazhen Lin, Fengwei Yu, and Junjie Yan. Differentiable soft quantization: Bridging full-precision and low-bit neural networks. In *IEEE ICCV*, pages 4852–4861, 2019.

[30] Yunchao Gong, Liu Liu, Ming Yang, and Lubomir Bourdev. Compressing deep convolutional networks using vector quantization. *arXiv preprint arXiv:1412.6115*, 2014.

[31] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[32] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[33] Alex Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

[34] Alex Graves, Navdeep Jaitly, and Abdel-rahman Mohamed. Hybrid speech recognition with deep bidirectional lstm. In *2013 IEEE workshop on automatic speech recognition and understanding*, pages 273–278. IEEE, 2013.

[35] Chuan Guo, Mayank Rana, Moustapha Cisse, and Laurens Van Der Maaten. Countering adversarial images using input transformations. *arXiv preprint arXiv:1711.00117*, 2017.

[36] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[37] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[38] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE CVPR*, pages 770–778, 2016.

[39] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European conference on computer vision*, pages 630–645. Springer, 2016.

[40] Sepp Hochreiter. Untersuchungen zu dynamischen neuronalen netzen. *Diploma, Technische Universität München*, 91(1), 1991.

[41] Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, Jürgen Schmidhuber, et al. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies, 2001.

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

[43] Qinghao Hu, Peisong Wang, and Jian Cheng. From hashing to cnns: Training binary weight networks via hashing. In *AAAI*, 2018.

[44] Bo Huang, Zhiwei Ke, Yi Wang, Wei Wang, Linlin Shen, and Feng Liu. Adversarial defence by diversified simultaneous training of deep ensembles. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 7823–7831, 2021.

[45] Hakan Inan, Khashayar Khosravi, and Richard Socher. Tying word vectors and word classifiers: A loss framework for language modeling. *arXiv preprint arXiv:1611.01462*, 2016.

[46] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[47] Yangqing Jia and Eric Shelhamer. Caffe: An open source convolutional architecture for fast feature embedding (2013), 2013.

[48] Qing Jin, Linjie Yang, and Zhenyu Liao. Towards efficient training for neural network quantization. *arXiv preprint arXiv:1912.10207*, 2019.

[49] Longlong Jing and Yingli Tian. Self-supervised visual feature learning with deep neural networks: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 43(11):4037–4058, 2020.

[50] Rafal Jozefowicz, Wojciech Zaremba, and Ilya Sutskever. An empirical exploration of recurrent network architectures. In *International conference on machine learning*, pages 2342–2350. PMLR, 2015.

[51] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.

[52] Nikhil Ketkar and Jojo Moolayil. Introduction to pytorch. In *Deep learning with python*, pages 27–91. Springer, 2021.

[53] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[54] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images, 2009.

[55] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

[56] David Krueger, Tegan Maharaj, János Kramár, Mohammad Pezeshki, Nicolas Ballas, Nan Rosemary Ke, Anirudh Goyal, Yoshua Bengio, Aaron Courville, and Chris Pal. Zoneout: Regularizing rnns by randomly preserving hidden activations. *arXiv preprint arXiv:1606.01305*, 2016.

[57] Alexey Kurakin, Ian J Goodfellow, and Samy Bengio. Adversarial examples in the physical world. In *Artificial intelligence safety and security*, pages 99–112. Chapman and Hall/CRC, 2018.

[58] Aditya Kusupati, Manish Singh, Kush Bhatia, Ashish Kumar, Prateek Jain, and Manik Varma. Fastgrnn: A fast, accurate, stable and tiny kilobyte sized gated recurrent neural network. In *Advances in Neural Information Processing Systems*, pages 9017–9028, 2018.

[59] Quoc V Le, Navdeep Jaitly, and Geoffrey E Hinton. A simple way to initialize recurrent networks of rectified linear units. *arXiv preprint arXiv:1504.00941*, 2015.

[60] Yann LeCun, Yoshua Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

[61] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[62] Edward H Lee, Daisuke Miyashita, Elaina Chai, Boris Murmann, and S Simon Wong. Lognet: Energy-efficient neural networks using logarithmic computation. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5900–5904. IEEE, 2017.

[63] Huiying Li, Shawn Shan, Emily Wenger, Jiayun Zhang, Haitao Zheng, and Ben Y Zhao. Blacklight: Defending black-box adversarial attacks on deep neural networks. *arXiv preprint arXiv:2006.14042*, 2020.

[64] Zhengang Li, Mengshu Sun, Alec Lu, Haoyu Ma, Geng Yuan, Yanyue Xie, Hao Tang, Yanyu Li, Miriam Leeser, Zhangyang Wang, et al. Auto-vit-acc: An fpga-aware automatic acceleration framework for vision transformer with mixed-scheme quantization. *arXiv preprint arXiv:2208.05163*, 2022.

[65] Ji Lin, Chuang Gan, and Song Han. Defensive quantization: When efficiency meets robustness. *arXiv preprint arXiv:1904.08444*, 2019.

[66] Xiaofan Lin, Cong Zhao, and Wei Pan. Towards accurate binary convolutional neural network. *arXiv preprint arXiv:1711.11294*, 2017.

[67] Zachary C Lipton, John Berkowitz, and Charles Elkan. A critical review of recurrent neural networks for sequence learning. *arXiv preprint arXiv:1506.00019*, 2015.

[68] Weiyang Liu, Yandong Wen, Zhiding Yu, Ming Li, Bhiksha Raj, and Le Song. Sphereface: Deep hypersphere embedding for face recognition. In *IEEE CVPR*, pages 212–220, 2017.

[69] Weiyang Liu, Yandong Wen, Zhiding Yu, and Meng Yang. Large-margin softmax loss for convolutional neural networks. In *ICML*, volume 2, page 7, 2016.

[70] Kaleel Mahmood, Phuong Ha Nguyen, Lam M Nguyen, Thanh Nguyen, and Marten van Dijk. Buzz: Buffer zones for defending adversarial examples in image classification. *arXiv preprint arXiv:1910.02785*, 2019.

[71] Gábor Melis, Chris Dyer, and Phil Blunsom. On the state of the art of evaluation in neural language models. In *International Conference on Learning Representations*, 2018.

[72] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.

[73] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: a simple and accurate method to fool deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2574–2582, 2016.

[74] Markus Nagel, Rana Ali Amjad, Mart Van Baalen, Christos Louizos, and Tijmen Blankevoort. Up or down? adaptive rounding for post-training quantization. In *International Conference on Machine Learning*, pages 7197–7206. PMLR, 2020.

[75] Markus Nagel, Mart van Baalen, Tijmen Blankevoort, and Max Welling. Data-free quantization through weight equalization and bias correction. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1325–1334, 2019.

[76] Sharan Narang, Erich Elsen, Gregory Diamos, and Shubho Sengupta. Exploring sparsity in recurrent neural networks. In *Proceedings of the ICLR*, 2017.

[77] Tianyu Pang, Kun Xu, Yinpeng Dong, Chao Du, Ning Chen, and Jun Zhu. Rethinking softmax cross-entropy loss for adversarial robustness. *arXiv preprint arXiv:1905.10626*, 2019.

[78] Tianyu Pang, Kun Xu, Chao Du, Ning Chen, and Jun Zhu. Improving adversarial robustness via promoting ensemble diversity. In *International Conference on Machine Learning*, pages 4970–4979. PMLR, 2019.

[79] Nicolas Papernot, Patrick McDaniel, and Ian Goodfellow. Transferability in machine learning: from phenomena to black-box attacks using adversarial samples. *arXiv preprint arXiv:1605.07277*, 2016.

[80] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia conference on computer and communications security*, pages 506–519, 2017.

[81] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. In *2016 IEEE European symposium on security and privacy (EuroS&P)*, pages 372–387. IEEE, 2016.

[82] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017.

[83] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[84] Rafael Pinot, Laurent Meunier, Alexandre Araujo, Hisashi Kashima, Florian Yger, Cédric Gouy-Pailler, and Jamal Atif. Theoretical evidence for adversarial robustness through randomization. *Advances in Neural Information Processing Systems*, 32, 2019.

[85] Haotong Qin, Ruihao Gong, Xianglong Liu, Mingzhu Shen, Ziran Wei, Fengwei Yu, and Jingkuan Song. Forward and backward information retention for accurate binary neural networks. In *IEEE CVPR*, pages 2250–2259, 2020.

[86] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *ECCV*, pages 525–542. Springer, 2016.

[87] Rachid Riad, Olivier Teboul, David Grangier, and Neil Zeghidour. Learning strides in convolutional neural networks. *arXiv preprint arXiv:2202.01653*, 2022.

[88] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models, 2021.

[89] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[90] Hasim Sak, Andrew W Senior, and Françoise Beaufays. Long short-term memory recurrent neural network architectures for large scale acoustic modeling, 2014.

[91] Pouya Samangouei, Maya Kabkab, and Rama Chellappa. Defense-gan: Protecting classifiers against adversarial attacks using generative models. *arXiv preprint arXiv:1805.06605*, 2018.

[92] Steffen Schneider, Alexei Baevski, Ronan Collobert, and Michael Auli. wav2vec: Unsupervised pre-training for speech recognition. *arXiv preprint arXiv:1904.05862*, 2019.

[93] Sanchari Sen, Balaraman Ravindran, and Anand Raghunathan. Empir: Ensembles of mixed precision deep networks for increased robustness against adversarial attacks. *arXiv preprint arXiv:2004.10162*, 2020.

[94] Connor Shorten and Taghi M Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of big data*, 6(1):1–48, 2019.

[95] Samuel Henrique Silva and Peyman Najafirad. Opportunities and challenges in deep learning adversarial robustness: A survey. *arXiv preprint arXiv:2007.00753*, 2020.

[96] Yang Song, Taesup Kim, Sebastian Nowozin, Stefano Ermon, and Nate Kushman. Pixeldefend: Leveraging generative models to understand and defend against adversarial examples. *arXiv preprint arXiv:1710.10766*, 2017.

[97] James C Spall et al. Multivariate stochastic approximation using a simultaneous perturbation gradient approximation. *IEEE transactions on automatic control*, 37(3):332–341, 1992.

[98] Suraj Srinivas and R Venkatesh Babu. Data-free parameter pruning for deep neural networks. *arXiv preprint arXiv:1507.06149*, 2015.

[99] Pierre Stock, Armand Joulin, Rémi Gribonval, Benjamin Graham, and Hervé Jégou. And the bit goes down: Revisiting the quantization of neural networks. *arXiv preprint arXiv:1907.05686*, 2019.

[100] Ruslan Leont'evich Stratonovich. Conditional markov processes. In *Non-linear transformations of stochastic processes*, pages 427–453. Elsevier, 1965.

[101] Jiawei Su, Danilo Vasconcellos Vargas, and Kouichi Sakurai. One pixel attack for fooling deep neural networks. *IEEE Transactions on Evolutionary Computation*, 23(5):828–841, 2019.

[102] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, 27, 2014.

[103] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[104] Tijmen Tieleman and Geoffrey Hinton. Rmsprop: Divide the gradient by a running average of its recent magnitude. coursera: Neural networks for machine learning. *COURSERA Neural Networks Mach. Learn*, 2012.

[105] Florian Tramèr, Alexey Kurakin, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel. Ensemble adversarial training: Attacks and defenses. *arXiv preprint arXiv:1705.07204*, 2017.

[106] Jonathan Uesato, Brendan O'donoghue, Pushmeet Kohli, and Aaron Oord. Adversarial risk and the dangers of evaluating against weak attacks. In *International Conference on Machine Learning*, pages 5025–5034. PMLR, 2018.

[107] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

[108] Gunjan Verma and Ananthram Swami. Error correcting output codes improve probability estimation and adversarial robustness of deep neural networks. *Advances in Neural Information Processing Systems*, 32, 2019.

[109] Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and tell: A neural image caption generator. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3156–3164, 2015.

[110] Andrew Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE transactions on Information Theory*, 13(2):260–269, 1967.

[111] Weitao Wan, Yuanyi Zhong, Tianpeng Li, and Jiansheng Chen. Rethinking feature distribution for loss functions in image classification. In *IEEE CVPR*, pages 9117–9126, 2018.

[112] Di Wang and Eric Nyberg. A long short-term memory model for answer sentence selection in question answering. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 707–712, Beijing, China, July 2015. Association for Computational Linguistics.

[113] Erwei Wang, James J Davis, Ruizhe Zhao, Ho-Cheung Ng, Xinyu Niu, Wayne Luk, Peter YK Cheung, and George A Constantinides. Deep neural network approximation for custom hardware: Where we've been, where we're going. *ACM Computing Surveys (CSUR)*, 52(2):1–39, 2019.

[114] Feng Wang, Jian Cheng, Weiyang Liu, and Haijun Liu. Additive margin softmax for face verification. *IEEE Signal Processing Letters*, 25(7):926–930, 2018.

[115] Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

[116] Cihang Xie, Jianyu Wang, Zhishuai Zhang, Zhou Ren, and Alan Yuille. Mitigating adversarial effects through randomization. *arXiv preprint arXiv:1711.01991*, 2017.

[117] Jiwei Yang, Xu Shen, Jun Xing, Xinmei Tian, Houqiang Li, Bing Deng, Jianqiang Huang, and Xian-sheng Hua. Quantization networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7308–7316, 2019.

[118] Zhewei Yao, Zhen Dong, Zhangcheng Zheng, Amir Gholami, Jiali Yu, Eric Tan, Leyuan Wang, Qijing Huang, Yida Wang, Michael Mahoney, et al. Hawq-v3: Dyadic neural network quantization. In *International Conference on Machine Learning*, pages 11875–11886. PMLR, 2021.

[119] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*, 2014.

[120] Dongqing Zhang, Jiaolong Yang, Dongqiangzi Ye, and Gang Hua. Lq-nets: Learned quantization for highly accurate and compact deep neural networks. In *ECCV*, pages 365–382, 2018.

[121] Hongyang Zhang, Yaodong Yu, Jiantao Jiao, Eric Xing, Laurent El Ghaoui, and Michael Jordan. Theoretically principled trade-off between robustness and accuracy. In *International conference on machine learning*, pages 7472–7482. PMLR, 2019.

[122] Yichi Zhang, Zhiru Zhang, and Lukasz Lew. Pokebnn: A binary pursuit of lightweight accuracy. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12475–12485, 2022.

[123] Aojun Zhou, Anbang Yao, Yiwen Guo, Lin Xu, and Yurong Chen. Incremental network quantization: Towards lossless cnns with low-precision weights. *arXiv preprint arXiv:1702.03044*, 2017.

[124] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160*, 2016.

[125] Michael Zhu and Suyog Gupta. To prune, or not to prune: exploring the efficacy of pruning for model compression. *arXiv preprint arXiv:1710.01878*, 2017.

[126] Konrad Zolna, Devansh Arpit, Dendi Suhubdy, and Yoshua Bengio. Fraternal dropout. *arXiv preprint arXiv:1711.00066*, 2017.

COLLEGE DOCTORAL BRETAGNE | MATHS, TELECOMS INFORMATIQUE, SIGNAL SYSTEMES, ELECTRONIQUE

Université de Rennes

**Titre :** Quantification et robustesse aux attaques adverses d'algorithmes neuronaux profonds embarqués

**Mots clés :** Réseaux neuronaux; Quantificateurs; Commande robuste; Systèmes embarqués

**Résumé :** Les réseaux de neurones convolutifs (CNN) et les réseaux de neurones récurrents (RNN) sont largement utilisés dans de nombreux domaines. Ce projet de recherche vise à rendre les réseaux de neurones profonds (DNNs) plus robustes face aux attaques adverses et plus faciles à déployer sur les plateformes embarquées. Après une revue de la littérature, nous proposons trois contributions sur la compression et la robustesse des DNNs : le lottery ticket sur les RNNs, Disentangled Loss Quantization Aware Training (DL-QAT) et Ensemble Hash Defense (EHD). L'étude du lottery ticket analyse la convergence des RNN et son impact sur le pruning. Ce travail nous a conduit vers des méthodes de quantifications en raison de leurs avantages pour l'inférence des DNNs. Nous proposons DL-QAT, une méthode de quantification avancée avec fonctions de coût adaptées qui permet d'atteindre des paramètres binaires sur les CNNs. Ce travail propose ensuite EHD, un méchanisme de défense permettant une meilleure résistance aux attaques adverses tout en préservant les performances et ne nécessitant qu'une surcharge mémoire lors de l'inférence. Ces travaux réduisent l'écart entre l'état de l'art des DNNs et leur exécution sur des cibles embarquées.

**Title :** Quantization and Adversarial Robustness of Embedded Deep Neural Networks

**Keywords :** Neural Networks; Quantization; Adversarial Attacks; Embedded systems

**Abstract :** Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) have been broadly used in many fields. This PhD research project tackles how to make the Deep Neural Networks (DNNs) more robust towards adversarial attacks and easier to deployed on the resource limited platforms. After a literature review, we propose three contributions on compression and robustness of DNNs: lottery tickets on RNNs, Disentangled Loss Quantization Aware Training (DL-QAT), and Ensemble Hash Defense (EHD). The investigation of lottery tickets analyzes the convergence of RNNs and study its impact when subject to pruning on image classification and language modelling. We then study quantization because of its advantages for DNNs inference. DL-QAT further improve an advanced quantization method with quantization friendly loss functions to reach binary parameters on CNNs where the application performance is the most impacted. We finally study neural networks robustness toward adversarial attacks and we present the EHD defense mechanism. EHD enables better resilience to adversarial attacks based on gradient approximation while pre- serving application performance and only requiring a memory overhead at inference time. All these contributions further reduce the gap between DNNs state of the art and their execution on edge devices.