



Institut Nationale des Sciences Appliqués et des Technologies
UNIVERSITE DE CARTHAGE

STAGE

Génie Logiciel

RN quantifié pour un contexte deep learning embarqué

Author:

Rami ZOUARI

Prof. Meriem JAIDANE
Supervisor, ENIT, Tunisia

Prof. Yosra BEN JEMAA
Supervisor, ENIS, Tunisia

2021/2022

Abstract

Les réseaux de neurones sont devenus très utilisés dans l'intelligence artificielle grâce à leurs performances de prédictions.

Ces réseaux de neurones sont parfois très complexes, et il n'est pas pratique de les utiliser dans des systèmes à ressources limitées.

Pour cela, dans ce rapport nous allons introduire les réseaux de neurones binaires (BNNs), les formaliser, étudier quelques modèles binarisés et dériver leurs formules, implémenter les résultats trouvés dans une bibliothèque qu'on va nommer **binaryflow**.

Et finalement, nous allons implémenter et entraîner des modèles binarisés pour la classification et régression, et nous allons comparer les performances de prédition, ainsi que les performances temps et mémoire de ces BNNs et leurs contrepart classique.

Acknowledgements

Je veux remercier mes encadreurs Mme Meriem JAÏDANE et Mme Yousra BEN JEMÂA, ainsi que toutes l'équipe de **dB Sense**, qui m'ont donné la chance de travailler sur ce projet intéressant, et leur expertise a été extrêmement précieuse dans la formulation des questions de recherche et de la méthodologie

Vos commentaires perspicaces m'ont poussés à affiner ma réflexion et ont fait passer le travail au niveau supérieur.

J'ai eu la chance de quitter ma zone de confort en travaillant sur ce sujet original, et je m'en suis très reconnaissants.

Contents

List of Figures	4
List of Tables	6
Introduction	7
1 Cadre du Stage	8
1.1 Présentation et activités de l'entreprise	8
1.2 Méthodologie de travail	9
2 Problématique	11
2.1 Introduction	11
2.2 Contexte	12
2.2.1 Histoire de l'apprentissage approfondie	12
2.2.2 Le coût de l'IA	13
2.3 Le Binary Neural Network	15
2.3.1 Introduction	15
2.3.2 La signification de "Binary"	15
2.4 BinaryFlow	16
2.4.1 Introduction	16
2.4.2 Raison d'exister	16
2.4.3 Signification du nom	16
2.4.4 Solutions offertes par BinaryFlow	16
3 Formalisation des BNNs	18
3.1 Introduction	18
3.2 Formalisation	18
3.2.1 Notations	18
3.2.2 Définition d'un BNN	20
3.2.3 Objectif	21
3.3 Entraînement	22
3.3.1 Difficulté du problème discret	22
3.3.2 Gradient Nul	22
3.4 Optimisations	24
3.4.1 Définitions	24
3.4.2 Codage sur 1bit	24
3.4.3 Utilisation de XNOR	24
3.4.4 Utilisation de POPCOUNT	26

3.4.5	Gain Temps & Mémoire	28
3.4.6	Algorithme Optimisé	29
4	Modèles des BNNs	30
4.1	Introduction	30
4.1.1	Importance de ce chapitre	30
4.1.2	Convention Utilisée	30
4.2	Histoire	31
4.3	Comparaison	32
4.3.1	Par quantification	32
4.3.2	Par architecture	32
4.4	BinaryNet	33
4.4.1	Conception	33
4.4.2	Topologie	33
4.4.3	Utilisation d'Estimateur Direct de gradient	33
4.4.4	Optimisations possible	35
4.5	XNOR-NET	36
4.5.1	Conception	36
4.5.2	Topologie	36
4.5.3	Réduction de l'erreur de quantification	36
4.5.4	Quantification optimale d'un vecteur	36
4.5.5	Quantification d'un produit scalaire de deux vecteurs	38
4.5.6	Quantification d'une couche dense par produits scalaires	40
4.5.7	Quantification d'une couche convolutionnelle par produits scalaires	40
4.5.8	Quantification d'une opération bilinéaire quelconque	43
4.6	ABCNet	46
4.6.1	Conception	46
4.6.2	Topologie	46
4.6.3	Objectif	46
4.6.4	Choix de binarisations de $\mathbf{W}^{(l)}$	47
4.6.5	Choix de binarisations de $a^{(l)}$	48
4.6.6	Quantification par tenseur binaire	48
4.6.7	Quantification par produit scalaire	48
4.7	BiRealNet	50
4.7.1	Conception	50
4.7.2	Topologie	50
4.7.3	Block BiRealNet	50
4.7.4	Liaison résiduelle	51
4.7.5	Choix de la quantification	51
5	Implémentation	52
5.1	Introduction	52
5.2	Paradigmes	53
5.2.1	Dans l'implémentation de BinaryFlow	53
5.2.2	Dans l'utilisation de BinaryFlow	53
5.2.3	Dans l'extension de BinaryFlow	54
5.3	Conception	54
5.4	Couches	55

5.4.1	BinaryNet	56
5.4.2	XnorNet	58
5.4.3	ABCNet	59
5.5	Block	60
5.6	Binarisations	61
5.6.1	Propagation en Avant	61
5.6.2	Propagation en arrière	63
5.6.3	Implémentation	63
5.7	Régularisations	64
5.7.1	Erreur de quantification des noeuds	64
5.7.2	Erreur de quantification des poids	64
5.7.3	Erreur de quantification de l'opération bilinéaire	64
6	Analyse	65
6.1	Introduction	65
6.2	Méthodologie adoptée	66
6.2.1	CRISP-DM	66
6.3	Régression de $f : x \rightarrow 2x^2 + 3x + 2$ sur $[-4, 4]$	67
6.3.1	Importance	67
6.3.2	Jeux de données	67
6.3.3	Modèle	67
6.3.4	Hypothèses	67
6.3.5	Problème Formel	67
6.3.6	Entraînement	67
6.3.7	Performances de prédiction	68
6.4	Classification MNIST	70
6.4.1	Introduction	70
6.4.2	Topologie	70
6.4.3	Modèle	71
6.4.4	Entraînement	71
6.4.5	Performances de prédictions	72
6.4.6	Performance mémoire	72
6.4.7	Performance temps	73
6.5	Classification Free Spoken Digits	74
6.5.1	Importance	74
6.5.2	Introduction	74
6.5.3	Structure	75
6.5.4	Analyse	75
6.5.5	Prétraitement	78
6.5.6	Topologie	81
6.5.7	Modèle	81
6.5.8	Entraînement	81
6.5.9	Performances de prédictions	82
6.5.10	Coût d'entraînement	84
Conclusion		86
Bibliographie		88

List of Figures

1.1	Équipe fondatrices de dB.Sense	8
1.2	ambiance dans dB.Sense	10
2.1	Lee Sedol analysant le tablier après sa première victoire du match	13
2.2	La compression et l'accélération des modèles	15
3.1	Un exemple d'une binarisation et sa dérivée	23
3.2	Table de multiplication en utilisant XNOR	24
3.3	Calcul du produit scalaires	26
4.1	Exemple d'un BinaryConnect à une seule couche cachée	33
4.2	Estimateur direct du gradient	34
4.3	Utilisation de facteurs α et β dans une couche dense	40
4.4	Utilisation de facteurs α et β dans une couche convolutionnelle	42
4.5	Exemple d'un ABCNet à une seule couche	46
4.6	Exemple d'une liaison résiduelle d'un BiRealNet	50
5.1	Structure de la bibliothèque	55
5.2	Le contenu de <code>binaryflow.layers</code>	55
5.3	Diagramme de classe du module BinaryNet	57
5.4	Diagramme de classe du module XnorNet	58
5.5	Diagramme de classe du module ABCNet	59
5.6	Le contenu de <code>binaryflow.blocs</code> , et de sous-module BiRealNet	60
5.7	Traçage des fonction Signe et Heaviside	61
5.8	Traçage de la fonction signe décalée à gauche par μ	62
5.9	Traçage de la fonction signe stochastique	62
5.10	Diagramme de classe global du module <code>binaryflow.quantizers</code>	63
6.1	Méthodologie CRISP-DM	66
6.2	Courbe de chaque modèle	68
6.3	Des images de MNIST	70
6.4	Architecture du modèle MNIST	70
6.5	Représentation des signals de FSD	74
6.6	Histogramme représentant la durée des signaux	75
6.7	Histogramme de la durée des signaux de chaque locuteur	76
6.8	Histogrammes représentant l'intensité maximale des signaux pour chaque locuteur	77
6.9	Exemple des spectrogrammes Mels et des MFCC	80
6.10	Architecture du modèle FSD	81

6.11	La fonction de coût en fonction de l'époque par modèle	82
6.12	Taux maximal de précision en fonction de l'époque par modèle	83
6.13	Énergie consommée par modèle durant l'entraînement	85
6.14	Masse de CO_2 dissipée durant l'entraînement	85

List of Tables

2.1	Coût des modèles de IA.	14
3.1	Terminologie des jeux de données	19
3.2	Terminologie de la division en Lots	19
3.3	Different machines characteristics	20
4.1	Le tableau d'avancement des BNNs	31
4.2	Approches de quantification	32
4.3	Approches architecturale	32
5.1	Paramètres de BinaryNet et BinaryConnect originaux	56
5.2	Les Les estimations de gradient présentes	63
6.1	Comparaison entre les modèles en utilisant Float32	72
6.2	Comparaison entre les modèles en utilisant Float8	73
6.3	Comparaison entre les modèles en utilisant Float8	73
6.4	Nombre de MACs des modèles	73
6.5	Statistique des durées par personne	76
6.6	Statistique des durées par personne	77
6.7	Taux maximal de précision	83

Introduction

With the rise of big data architectures, the need for creating large graphs has increased dramatically these last few years. And to process these large graphs, hardware limits (Memory, Computational power, ...) started to present a very large obstacle. Our personal computers have limited resources so we can only process very limited graphs.

With this paper, we came across this golden rule: how to create large graphs without exhausting the memory? This question could be answered by designing a couple of strategies for graph creation.

This report defines our approach to implement the different graph creation strategies and eventually our machine learning model. It is organized into 5 chapters:

In the first chapter, we will start by giving a brief history on graph theory, state its importance and then we will modelize our problem mathematically and state our hypothesis and different strategies.

In the second chapter, we will first detail the different paradigms and approaches we followed. Then we will present our system design and explain in details its different components.

In the third chapter, we will start by mentioning the different tools and languages used. And then we will describe our progress and the different challenges we have met.

In the fourth chapter, we will analyze our data in order to create an efficient model later on.

In the fifth chapter, we describe machine learning methodology and its different steps. Present our machine learning model in addition to the results.

Chapter 1

Cadre du Stage

Introduction

Ce chapitre est une introduction à **dB.Sense** et ses activités, nous abordons aussi la méthodologie de travail tout au long de ce stage.

1.1 Présentation et activités de l'entreprise

dB.Sense est une spin-off de l'Université de Tunis El Manar. Elle est incubée au Centre d'Innovation de l'UTM, fondée par des ingénieurs et docteurs dans le domaine des TIC, du génie électrique et des statistiques.

“Nous innovons, en symbiose avec nos partenaires dans les domaines de l'industrie et des services, en ayant constamment à l'esprit que l'humain doit être au centre de la technologie.”



Mériem Jaïdane

CEO, CO-FONDATRICE



Amira Ben Jemâa

CO-FOUNDRICE



Nader Mechergui

CO-FONDATEUR



Neska El Haouij

CO-FONDATRICE

Figure 1.1: Équipe fondatrices de dB.Sense

dB.Sense propose des solutions innovantes disponibles sous forme de prototypes. Ces solutions sont réalisées en collaboration avec des acteurs de l'industrie. dB.Sense met à la disposition des industriels et des entreprises son savoir-faire et ses compétences en matière de traitement du signal, d'intelligence artificielle (IA), d'analyse, et de modélisation de données pour le développement de solutions technologiques dans des domaines variés, tels que l'acoustique, la vibration, le traitement de l'information, et la complexité du vivant.

Activités

- Soutien technologique : Accompagnement des entreprises dans le développement d'outils innovants à forte valeur ajoutée dans leurs processus de production ou leurs services.
- Recommandations pour l'amélioration de l'environnement de travail : Solutions actives et passives pour améliorer le confort acoustique, analyse de l'environnement de travail et mesure du bien-être des travailleurs.
- Conception de protocoles : Protocoles objectifs centrés sur l'utilisateur pour la validation et le test de produits technologiques.
- Développement de PoCs : dB.Sense intervient dans la réalisation de prototypes à différents niveaux de maturité :
 - Collecte de données et enquêtes sur les besoins des utilisateurs.
 - Evaluations centrées sur l'utilisateur utilisant des biocapteurs pour l'objectivation de la perception.
 - Algorithmes hybrides signal et IA.
 - Développement d'algorithmes temps réel de faible complexité.

1.2 Méthodologie de travail

Le stage s'est déroulé d'une manière hybride : deux jours dans les locaux de dB.Sense, le reste de la semaine en ligne, une réunion de deux heures en ligne est tenue chaque mardi pour discuter de l'avancement et des problèmes rencontrés. Le stage a duré de deux mois et a été divisé en deux grandes phases :

- Une phase Recherche : plusieurs papiers de recherche scientifiques ont été lus et décortiqués, des présentations ont été préparées et partagées avec l'intégralité de l'équipe. C'est lors de ces présentations qu'il y ait eu l'échange d'idée et le partage des connaissances.
- Une phase Développement : des codes implémentants les différents algorithmes et modèles des papiers ont été testés et modifiés au besoin dans le but de concrétiser l'apprentissage acquis lors de la phase de recherche.

Autres activités

La vie au sein de dB.Sense dépasse le cadre du projet. Elle présente une ouverture sur d'autres disciplines et projets par exemple durant le stage nous avons eu l'occasion de rencontrer **M. Frédéric BERNAUDIEN** de Hitachi Rails qui nous a parlé de son entreprise et leur secteur d'activités. Nous avons eu le plaisir aussi d'assister à la présentation de **Mme Ayda BADRI** qui nous a introduit à l'application de l'IA dans la chimie. De plus, nous avons collaboré avec le bureau d'architecte Atmosphère dans le cadre de conceptualisation d'un arbre solaire pour le sommet de la francophonie.

Des activités de déroulement et de team-building après les heures de travail ont été aussi présente tel que des workshops de secourismes. Et pour cela



Figure 1.2: ambiance dans dB.Sense

Conclusion

Après avoir décrit dB.Sense brièvement dans ce chapitre, nous avons expliqué les étapes suivies dans ce stage les prochains chapitres seront plus orientés vers l'aspect techniques du projet.

Chapter 2

Problématique

2.1 Introduction

Dans ce chapitre, nous allons présenter brièvement l'histoire de l'intelligence artificielle afin de mettre la problématique en contexte.

Puis nous allons présenter les problèmes des modèles puissant de l'IA et leurs coût économique, énergétique et même environmental.

Après nous allons donner une solution qui est le Binary Neural Network (BNN), et en particulier notre implémentation des BNNs qui est **BinaryFlow**.

2.2 Contexte

2.2.1 Histoire de l'apprentissage approfondie

Cette section est basée sur l'article de Tim Dettmers [24]

Réseaux de Neurones

L'origine des réseaux de neurones, peut être considéré de Ivakhnenko et Lapa [11] qui ont réussi à implémenter un réseau neuronale à fonctions d'activations polynomiales. Mais ils n'ont pas utilisé la propagation en arrière qui n'était pas répandue dans ces années.

Propagation en Arrière

La propagation en arrière était dérivée dans les années 1960 mais sous une forme incomplète et inefficace. En outre, sa forme moderne était dérivée par Linnainmaa dans sa thèse de mastère [18] "Taylor expansion of the accumulated rounding error" en 1970. La propagation ne serait répandue que dans 1985 dans lequel les recherches ont montré qu'elle donne des représentations intéressantes des paramètres.

L'hiver de l'IA

Malgré les succès de la propagation en arrière, et son incorporation dans les couches convolutionnelles (LeNet [15]) et récurrentes (spécifiquement LSTM [9]), l'intérêt à l'intelligence artificielle était minimale dans les années 1980-1990, et les avancées dans ce domaine étaient minimales. De plus, les machines à vecteurs de supports (SVM [2]) étaient préférées au lieu des réseaux de neurones vu la complexité de ces derniers.

Réapparition grâce aux GPUs

Grâce au progrès des ordinateurs, et l'arrivée des cartes graphiques (GPUs), l'utilisation des réseaux de neurones était de plus en plus pratique, et progressivement, la complexité de ces réseaux croît.

Explosion de l'apprentissage approfondie

Le moment décisif à l'apprentissage approfondie était le succès de Krizhevsky, Sutskever et Hinton [14] à créer un réseau de neurones profonds avec lequel ils ont importé la compétition de ILSVRC-2012 ImageNet.

Ce moment a marqué l'abandon de l'ingénierie des fonctionnalités et le début de l'apprentissage des fonctionnalités. Et puis, Facebook, Google et Microsoft ont rapidement investi dans la recherche de l'apprentissage approfondie.

Un niveau surhumain

Dès les années 2010s, beaucoup de modèles de IA ont montré un niveau surhumain dans des tâches que nous avons cru impénétrable par les ordinateurs.

Comme un exemple pertinent, nous allons parler de la suite Alpha développée par l'équipe DeepMing de Google:

1. Le modèle AlphaGo était le premier agent ayant un élo surhumain, ceci est prouvé par son match historique en 2016 contre Lee Sedol¹ dans lequel AlphaGo a gagné 4-1 contre lui².
2. Le modèle AlphaGo Zero est une amélioration de AlphaGo. La majeure différence est dans son entraînement, qui n'est pas basé sur des jeux entre des humains comme le cas de AlphaGo, mais il s'est entraîné en jouant contre lui-même.
3. Le modèle AlphaZero est encore une amélioration de AlphaGo Zero, qui avait un niveau surhumain dans Go, Shogi et l'échec³



Figure 2.1: Lee Sedol analysant le tablier après sa première victoire du match
 Source de photo: time.com [6]

2.2.2 Le coût de l'IA

Cette explosion de l'IA a causé une explosion dans les coûts de leurs entraînement, déploiement et fonctionnement.

Le coût économique

Les modèles d'intelligences artificielles les plus robustes nécessitent des ressources énormes pour leur entraînement et même fonctionnement.

En effet:

- L'entraînement de AlphaGo Zero a coûté approximativement 25 millions de dollars [5].
- L'entraînement de NAS a coûté approximativement 3 millions de dollars [20].

¹Lee Sedol est un des joueurs de Go les plus accomplis dans l'histoire du jeu, avec 18 titres internationaux.

²Ce match était une cause pour laquelle Lee Sedol s'est retiré de Go en 2019. Il a cité que l'IA est "une entité qui ne pourrait être vaincue."

³Dans l'échec, il a même écrasé Stockfish en 2018 avec un résultat de (+155 -6 =839). Avant l'apparition de AlphaZero, Stockfish était le plus puissant moteur d'échec.

Le coût énergétique

L'utilisation gigantesque des ressources causent une consommation énorme d'énergie.
Par exemple:

- Dans le match contre Lee Sedol, AlphaGo a utilisé 1202 GPUs et 176 CPUs. Une estimation de la puissance consommé montre un chiffre colossal de 1 mégawatts [20].
- Pour entraîner NAS, l'énergie totale consommée est estimée à 656347 kWh [23].

Le coût environnemental

Le coût environnemental des modèles IA avancés est souvent très importants.
En effet, les estimations de CO₂ générés par ces algorithmes peut dépasser l'ordre de milliers de kilogrammes:

Modèle / Objet	Hardware	Puissance (Watt)	Temps de fonctionnement (Heures)	CO ₂ équivalents (kg)	Budget cloud (dollars américains)
Transformer _{base}	P100x8	1415.78	12	11.79	\$41 - \$410
Transformer _{big}	P100x8	1515.43	84	87.08	\$289 - \$981
ELMo	P100x3	1415.78	12	11.79	\$433 - \$1472
BERT _{base}	V100x64	12041.51	79	652.27	\$3751 - \$12571
NAS	P100x8	1515.43	274120	284019	\$942973 - \$3201722
AlphaGo		≈ 1000000	960	96000	≈ \$250000000
Climatiseur		840			
Refrégirateur		1233			
Avion SF → NY				900	
Personne durant 1 an				5000	
Voiture durant sa vie				57152	
Maison durant son construction [7]				50000 - 80000	

Table 2.1: Coût des modèles de IA.

2.3 Le Binary Neural Network

2.3.1 Introduction

Dans les derniers années, L'échelle de L'intelligence artificielle a été augmentée rapidement, et aussi la complexité des modèles utilisés.

Cette augmentation est l'une des causes des performances superbes des modèles état d'arts. Mais aussi, elle pose beaucoup de problèmes dans l'entraînement et déploiement de ces modèles dans un systèmes à complexité limitée.

Pour résoudre partiellement ce problèmes, beaucoup de solutions ont été proposées pour compresser et/ou accélérer les modèles.

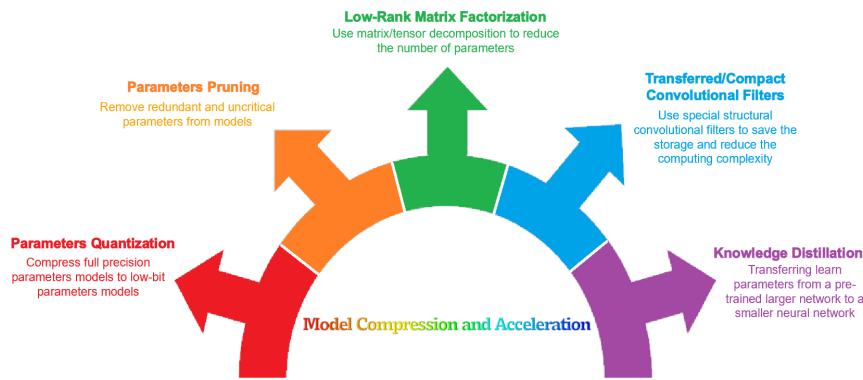


Figure 2.2: La compression et l'accélération des modèles

l'une de ces solutions est la quantification sur un nombre de bits limités indiqué par la flèche rouge dans la figure 2.2, dont les BNNs sont un cas extrême: La quantification est sur un seul bit.

2.3.2 La signification de "Binary"

Le mot **Binary** ne signifie pas nécessairement que les paramètres et/ou poids scalaires sont 0 ou 1. Il signifie en effet que chaque paramètre peut admettre uniquement 2 valeurs a, b qui sont distinctes.

2.4 BinaryFlow

2.4.1 Introduction

BinaryFlow est une bibliothèque de Deep Learning basée sur Larq et TensorFlow

2.4.2 Raison d'exister

Bien qu'il y a quelques solutions de BNNs, Elles sont:

- dispersées.
- anciennes et abandonnées.
- faites avec des bibliothèques et langages de programmations différentes.

Nous voulons implémenter une interface qui unifie les approches de BNNs sous une forme modulaire, extensibles et facile à utiliser.

En effet, la bibliothèque Larq est la plus proche à nos désirs, elle admet des implémentations importantes de quelques approches de BNNs, et pour cela binaryflow va être une extension de Larq.

2.4.3 Signification du nom

Le choix de nom "BinaryFlow" est influencé de nom "TensorFlow" de la fameuse bibliothèque d'apprentissage automatique et de programmation différentielle qui est créée et maintenue par Google.

TensorFlow se traduit en "flux de tenseurs" qui est l'utilisation excessive des tenseurs - qui sont grossièrement des tableaux numériques multidimensionnels - dans les calculs différentiels.

Par analogie, BinaryFlow se traduit en "flux binaire" qui signifie l'utilisation excessive des opérations booléennes sur des tenseurs binaires.

2.4.4 Solutions offertes par BinaryFlow

Couches

BinaryFlow supporte toutes les couches offertes par TensorFlow et Larq. Mais il aussi supporte:

- BinaryNet: Dense, ConvND, TransposedConvND, SeparableConvND.
- XnorNet: Dense, ConvND.
- XnorNet++: Dense, ConvND.
- ABCNet: Dense, Conv1D, Conv2D, Conv3D
- BiRealNet: Dense, Conv1D, Conv2D, Conv3D
- MeliusNet: Dense, Conv1D, Conv2D, Conv3D

Binarisations

BinaryFlow supporte toutes les binarisations standard offertes par Larq, et les étend en ajoutant:

- Binarisation décalée: qui est une métá-binarisation, qui prend une binarisation Ψ et donne $\Psi_\mu = x \rightarrow \Psi(x + \mu)$ avec $\mu \in \mathbb{R}$ entraînable
- Binarisation stochastique: qui est une métá-binaristation qui prend une distribution de probabilité réelle \mathcal{D} en paramètre, et une binarisation Ψ , et donne la fonction: $x \rightarrow \Psi(x - z)$ avec $z \sim U$
- Binarisation stochastique décalée

Optimiseurs

BinaryFlow offre les optimiseurs suivants

- SGD
- Adam[13]
- Bob[8]

Déploiement

BinaryFlow conserve tous les optimisations offerte par Larq

Chapter 3

Formalisation des BNNs

3.1 Introduction

In this chapter we will start by presenting the used paradigms and approaches, the global domain Diagram used in the analysis phase. And then we will explain the different components and their behavior.

3.2 Formalisation

Le BNN, bien que son objectif est clair, elle n'a pas d'approche triviale pour arriver à un réseau "binaire". Pour cela, on va tout d'abord formaliser notre objectif, est après ça, on va essayer de définir un BNN

3.2.1 Notations

Pour simplifier, nous allons commencer par la terminologie des réseaux à perceptrons multicouches.

Observations

r	nombre d'exemplaires du jeux de données
$\mathbf{X}^{\{k\}}$	l'entrée du $k^{\text{ème}}$ exemplaire
$\mathbf{X} = \left(\mathbf{X}^{\{1\}}, \dots, \mathbf{X}^{\{r\}} \right)$	le tenseur des entrées
$\mathbf{Y} = \left(\mathbf{Y}^{\{1\}}, \dots, \mathbf{Y}^{\{r\}} \right)$	le tenseur des résultats
$\mathbf{U} = \left(\mathbf{U}^{\{1\}}, \dots, \mathbf{U}^{\{r\}} \right)$	le tenseur des observations
$\hat{\mathbf{Y}}$	Une estimation de \mathbf{Y}
\mathcal{L}	Une fonction objective
$\mathcal{L}(\hat{\mathbf{Y}}, \mathbf{Y})$	L'erreur d'estimation de Y

Table 3.1: Terminologie des jeux de données

Division en Lots

m	nombre de lots
s_k	la taille du $k^{\text{ème}}$ lot
$\mathbf{X}^{[k]}$	le $k^{\text{ème}}$ lot d'entrées
$\mathbf{Y}^{[k]}$	le $k^{\text{ème}}$ lot de sorties
$\mathbf{U}^{[k]}$	le $k^{\text{ème}}$ lot des observation
$\mathcal{L}(\hat{\mathbf{Y}}^{[k]}, \mathbf{Y}^{[k]})$	l'erreur d'estimation de $k^{\text{ème}}$ lot

Table 3.2: Terminologie de la division en Lots

Remarque 1 *Par défaut, on va diviser les lots en une taille fixe s .*

Comme une exception, le dernier lot va avoir le reste de cette division

MLP et CNN

\mathcal{N}	Un réseau de neurones
$\mathcal{N}_{\mathbf{W}}$	Un réseau de neurones défini par le tenseur \mathbf{W}
$\mathbf{W}^{(l)}$	La matrice de liaison de la $l^{\text{ème}}$ couche
$\sigma^{(l)}$	la fonction d'activation de la $l^{\text{ème}}$ couche
$z^{(l)} = \mathbf{W}^{(l)} \star a^{(l-1)}$	le contenu de la couche l avant l'activation
$a^{(l)} = \sigma^{(l)}(z^{(l)}) = \sigma^{(l)}(\mathbf{W}^{(l)} \star a^{(l-1)})$	le contenu de la couche l après l'activation
$W_{i,j}^{(l)}$	le poids de neurone $a_j^{(l-1)}$ dans la $i^{\text{ème}}$ neurones avant l'activation
$x = a^{(0)}$	L'entrée du réseau de neurones
$\hat{y} = a^{(l)} = \mathcal{N}_{\mathbf{W}}(x)$	La sortie du réseau de neurones
n_l	le nombre de neurones de la $l^{\text{ème}}$ couche

Table 3.3: Different machines characteristics

Remarque 2 Le terme $\mathbf{W}^{(l)} \star a^{(l-1)}$ est interprété comme:

1. une multiplication matricielle $\mathbf{W}^{(l)} \cdot a^{(l-1)}$ dans le cas d'une couche dense
2. une opération de convolution $\mathbf{W}^{(l)} * a^{(l-1)}$ dans le cas d'une couche convolutionnelle

3.2.2 Définition d'un BNN

Avant de définir un BNN, nous allons introduire le concept des binarisations et des quantifications:

Définition 1 Une binarisation est une fonction $\Psi : \mathbb{R} \rightarrow \mathcal{B}$ avec \mathcal{B} un ensemble à deux éléments. Dans le cas d'un tenseur, la binarisation est appliquée élément par élément. On va distinguer essentiellement deux valeurs particulières de \mathcal{B} :

- $\mathcal{B} = \{\pm 1\}$
- $\mathcal{B} = \{0, 1\}$

Définition 2 Un scalaire u est dit binarisé s'il varie dans un ensemble \mathcal{B} à deux éléments. Un tenseur de rang r et de dimension $n_1 \times \dots \times n_r$ est dit binarisé s'il varie dans un ensemble $\mathcal{B}^{n_1 \times \dots \times n_r}$ avec \mathcal{B} à deux éléments.

Définition 3 Une quantification est un opérateur Ψ appliqué à un tenseur juste avant l'opération bilinéaire en conservant ses dimensions.

Remarque 3 Dans les réseaux de neurones classiques, il n'y a pas de quantifications. En fait, l'absense d'une quantification est équivalente à l'utilisation de la quantification `id`: $x \rightarrow x$

Dans la littérature

Le BNN n'admet pas d'une définition unique. En contre partie, dans la littérature[26], la plupart des réseaux sont nommées binaires puisque :

1. Pour chaque paire de nœuds connexe (u, v) . Le poids de la liaison est $\omega(u, v) = \pm 1$
2. Pour chaque couche cachée, la fonction d'activation est la fonction signe sign

Notre définition

Nous avons proposé cette définition pour essaier d'unifier les approches des BNNs:

Définition 4 *Un BNN est tout réseau de neurones admettant au moins une couche dont les quantifications sont des binarisations*

Un BNN idéal est un réseau de neurones dont toutes les quantifications sont des binarisations

Remarque 4 *Dans ce stage, nous allons concentrer sur les binarisations dont $\mathcal{B} = \{\pm 1\}$, et surtout la binarisation sign*

Remarque 5 *Bien que la définition est un peu permissive, nous n'allons considérer que les BNNs qui ont un nombre "important"¹ de binarisations.*

3.2.3 Objectif

Notations

- Soit $\mathcal{D} = (\mathbf{X}, \mathbf{Y})$ le jeu de données
- Soit $\mathcal{N}_{\mathbf{W}}$ un réseau de neurones admettant le tenseur de paramètres \mathbf{W} .

Problème d'optimisation

L'objectif "idéal" est la résolution du problème d'optimisation suivant:

$$\mathbf{W}^* = \underset{\mathbf{W}}{\operatorname{argmin}} \mathcal{L}(\mathcal{N}_{\mathbf{W}}(\mathbf{X}), \mathbf{Y}) \quad (3.1)$$

L'objectif pour un BNN est aussi la minimisation de (3.1), mais la seule différence est que le réseau admet des binarisations comme quantifications.

¹Grosso modo, le nombre de binarisations doit être aussi important pour justifier les apports des binarisations.

3.3 Entraînement

Notre définition de BNN pose beaucoup de problème dans l'entraînement, dont la résolution exige des approches sophistiquées.

3.3.1 Difficulté du problème discret

Soit \mathcal{N} un réseau de neurones binaire idéal ayant une topologie fixe, et soit η le nombre total de ses paramètres binarisés.

Dans le plus mauvais cas, la complexité de trouver les valeurs optimales des paramètres pour ce réseau est $\mathcal{O}(2^\eta)$ qui n'est pas du tout pratique.

Deux approches se présentent:

Optimisation discrète Cette approche est directe, elle prend compte de la nature discrète du problème d'optimisation.

Deux techniques importantes sont applicable dans cette approche:

- Méthode Hill-Climbing
- Méthode Simulated-Annealing

Bien que cette approche soit intéressante, on ne va pas la considérer dans ce stage vu le problème d'intégration de ces méthodes dans TensorFlow.

Optimisation différentiable Étant l'approche utilisée dans la plus part des problématique d'apprentissage automatique², et puisque les optimiseurs par défaut de TensorFlow sont de premier ordre, nous allons considérer cette approche.

Dans cette approche, nous allons utiliser essentiellement la famille des méthodes basées sur la descente du gradient.

En effet, elle va être utilisée durant ce stage.

3.3.2 Gradient Nul

Nous allons opter pour la deuxième solution dans ce stage. Mais, malheureusement, on ne peut pas appliquer directement les méthodes d'optimisations de premier ordre, ou même d'ordre supérieur. Le problème majeur provient de la lemme suivante:

Lemme 1 *Toute binarisation continue presque partout³ admet une dérivée nulle presque partout*

Preuve 1 *Soit $\Phi : \mathbb{R} \rightarrow \{a_0, a_1\}$ une binarisation continue partout sauf un ensemble \mathcal{D} de mesure nulle, avec $a_0, a_1 \in \mathbb{R}$ et $a_0 \neq a_1$.*

Soit $x_0 \in \mathbb{R}$ tels que Φ est continue en x_0 , et soit $0 < \epsilon < |a_1 - a_0|$ et $\delta \in \mathbb{R}_+^$ tel que*

$$x \in \mathcal{B}(x_0, \delta) \implies |\Phi(x) - \Phi(x_0)| < \epsilon$$

²En effet, dans l'apprentissage automatique, la plus part des problèmes différentiables sont résolu à l'aide de la descente du gradient ou l'une de ces variantes

³Le mot "presque partout" est interprété au sens de la théorie des mesures, et précisément la mesure naturelle dans \mathbb{R} (mesure de Leibniz). Par exemple, dans un sens, la "dérivée" de la fonction sign est la fonction de Dirac δ

On a nécessairement, $\Phi(\mathcal{B}(x_0, \epsilon)) = \{a_i\}$ avec $i \in \{0, 1\}$, et donc Φ est constante sur $\mathcal{B}(x_0, \delta)$, et ainsi dérivable sur cet interval et en particulier en x_0 , et on a:

$$\Phi'(x_0) = 0$$

Ainsi:

$$\forall x \in \mathbb{R}, \quad \Phi \text{ continue en } x \implies \Phi \text{ dérivable en } x \text{ et } \Phi'(x) = 0$$

Finallement, on a Φ est continue sur $\mathcal{S} = \mathbb{R} \setminus \mathcal{D}$, avec \mathcal{D} est de mesure nulle, et donc Φ est dérivable sur \mathcal{S} et par conséquent $\Phi' = 0$ sur \mathcal{S} ■

Cette lemme pose un problème puisque la mise à jour des paramètres sera impossible avec des gradients nuls presque partout.

Pour résoudre ce problème, nous pouvons:

1. Faire une approximation de la binarisation par une fonction plus régulière, surtout dans la propagation en arrière.
2. Faire une relaxation concernant la notion de gradient à l'aide d'un opérateur \mathcal{G} adéquat " avec lequel on peut appliquer les méthodes de premier ordre.

Dans le chapitre suivant, nous allons voir chacune des approches 1 et 2.

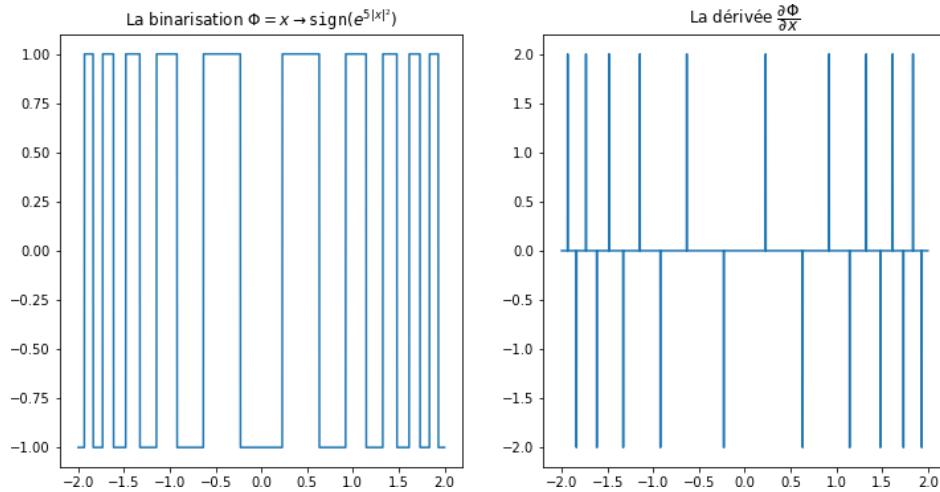


Figure 3.1: Un exemple d'une binarisation et sa dérivée

3.4 Optimisations

3.4.1 Définitions

On note par:

- $(\mathbb{B}, +, \cdot, \bar{\cdot}, 0, 1)$ une algèbre de boole.
- l'opérateur **XOR** par \oplus défini par:

$$a \oplus b = \text{XOR}(a, b) = a \cdot \bar{b} + \bar{a} \cdot b = \bar{a} \oplus \bar{b}$$
- l'opérateur **XNOR** par \odot défini par:

$$a \odot b = \text{XNOR}(a, b) = \overline{\text{XOR}(a, b)} = (a + \bar{b})(\bar{a} + b) = a \cdot b + \bar{a} \cdot \bar{b} = \bar{a} \odot \bar{b}$$

3.4.2 Codage sur 1bit

Puisque les poids et noeuds des couches cachées sont tous binarisés, on peut coder chaque valeur sur un seul bit. Ainsi on peut introduire le codage:

$$\Psi : \begin{cases} -1 & \rightarrow 0 \\ 1 & \rightarrow 1 \end{cases}$$

En fait, on regroupe chaque 8 variables dans la même case mémoire.

3.4.3 Utilisation de XNOR

Transformation de \times en XNOR

Dans le cas d'un BNN, les noeuds et les poids sont binarisés. Ainsi toute multiplication est entre deux éléments de $\{-1, 1\}$.

Or, par la transformation bijective:

$$\Psi : \begin{cases} 1 & \rightarrow 1 \\ -1 & \rightarrow 0 \end{cases}$$

On a l'égalité suivante:

$$\forall a, b \in \{-1, 1\}, \quad \Psi(a \times b) = \Psi(a) \odot \Psi(b)$$

Encoding (Value)	XNOR (Multiply)
0 (-1)	0 (-1)
0 (-1)	1 (+1)
1 (+1)	0 (-1)
1 (+1)	1 (+1)

Figure 3.2: Table de multiplication en utilisant XNOR

Preuve 2 On a le groupe (\mathbb{C}_2, \times) avec $\mathbb{C}_2 = \{z \in \mathbb{C} / z^2 = 1\} = \{-1, 1\}$ est cyclique et d'ordre 2. Ainsi, il est isomorphe au groupe (\mathbb{F}_2, \oplus) où $\mathbb{F}_2 = \mathbb{Z}/2\mathbb{Z} = \{0, 1\}$, et \oplus est l'addition modulo 2. L'isomorphisme entre eux est:

$$\Psi_1 = \begin{cases} -1 & \rightarrow 1 \\ 1 & \rightarrow 0 \end{cases}$$

De plus, le corps $(\mathbb{F}_2, \oplus, \times)$ induit l'algèbre de boole $(\mathbb{F}_2, +, \times, \bar{}, 0, 1)$ avec:

$$\begin{aligned} \bar{a} &= a \oplus 1 \\ a + b &= \overline{\bar{a} \times \bar{b}} \end{aligned}$$

Or, par dualité de l'algèbre booléenne, l'opérateur de négation $\Psi_2 : a \rightarrow \bar{a}$ est un isomorphisme entre $(\mathbb{F}_2, +, \times, \bar{}, 0, 1)$ et $(\mathbb{F}_2, \times, +, \bar{}, 1, 0)$

Ainsi on a:

$$\begin{aligned} \forall a, b \in \{-1, 1\}, \quad \Psi_2 \circ \Psi_1(a \times b) &= \Psi_2(\Psi_1(a \times b)) \\ &= \Psi_2(\Psi_1(a) \oplus \Psi_1(b)) \text{ isomorphisme 1} \\ &= \overline{\Psi_1(a) \oplus \Psi_1(b)} \\ &= \Psi_1(a) \odot \Psi_1(b) \text{ isomorphisme 2} \\ &= \overline{\Psi_1(a)} \odot \overline{\Psi_1(b)} \\ &= \Psi_2 \circ \Psi_1(a) \odot \Psi_2 \circ \Psi_1(b) \end{aligned}$$

Maintenant, il ne reste qu'à vérifier que $\Psi = \Psi_2 \circ \Psi_1$:

$$\begin{aligned} \Psi_2 \circ \Psi_1(-1) &= \Psi_2(\Psi_1(-1)) \\ &= \Psi_2(1) \\ &= 0 \\ \Psi_2 \circ \Psi_1(1) &= \Psi_2(\Psi_1(1)) \\ &= \Psi_2(0) \\ &= 1 \quad \blacksquare \end{aligned}$$

Avantages

En fait, dans les circuits intégrés, l'opération XNOR est plus rapide que \times . En fait, elle:

- consomme moins d'énergie
- peut être faite 32 fois dans un seul cycle. Par comparaison, une seule opération \times peut dépasser 1 cycle

3.4.4 Utilisation de POPCOUNT

Définition de POPCOUNT

POPCOUNT est une instruction qui permet de compter les bits qui sont mis à 1 dans un vecteur de bits v .

Remarque 6 v peut être la représentation binaire d'un nombre n

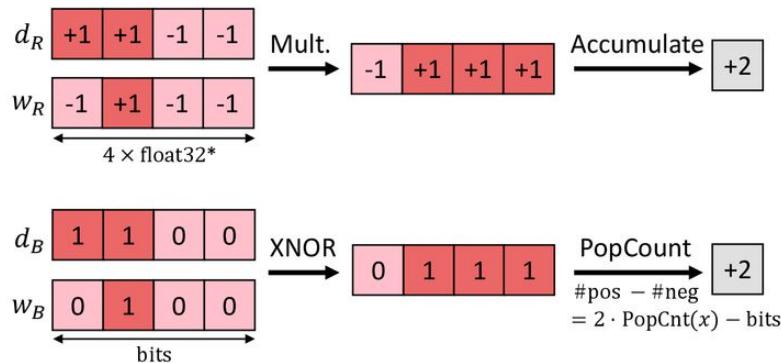
Produit scalaire de deux vecteurs binarisés

Soit $u, v \in \{-1, 1\}^n$. On a:

$$\begin{aligned}
 \langle u, v \rangle &= \sum_{i=1}^n u_i v_i \\
 &= \sum_{i=1}^n \Psi^{-1}(\Psi(u_i) \odot \Psi(v_i)) \\
 &= \text{POPCOUNT}(\Psi(u) \odot \Psi(v)) \cdot \Psi^{-1}(1) + (n - \text{POPCOUNT}(\Psi(u) \odot \Psi(v))) \cdot \Psi^{-1}(0) \\
 &= \text{POPCOUNT}(\Psi(u) \odot \Psi(v)) - (n - \text{POPCOUNT}(\Psi(u) \odot \Psi(v))) \\
 &= 2 \cdot \text{POPCOUNT}(\Psi(u) \odot \Psi(v)) - n \\
 &= (\text{POPCOUNT}(\Psi(u) \odot \Psi(v)) \ll 1) - n
 \end{aligned}$$

Convolution with bitwise operations

Multiplication and addition are replaced by bitwise XNOR and PopCount.



11

Figure 3.3: Calcul du produit scalaires

Estimation de coût

On suppose pour $m \in \mathbb{N}^*$, que notre processeur supporte dans une seule instruction, un **XNOR** bit-wise des deux vecteurs bits $u, v \in \{0, 1\}^m$

On suppose aussi pour cette même valeur de m que notre processeur supporte dans une seule instruction, un **POPCOUNT** bit-wise d'un vecteur bits $u \in \{0, 1\}^m$

Soit $n = mr$, $r \in \mathbb{N}^*$, et soit $u, v \in \{-1, 1\}^n$

Pour $p \in \{1, r\}$, soit:

$$\begin{cases} u^{[p]} = (u_{(p-1)m+1}, \dots, u_{pm}) \\ v^{[p]} = (v_{(p-1)m+1}, \dots, v_{pm}) \end{cases}$$

On a:

$$\begin{aligned} \langle u, v \rangle &= \sum_{i=1}^n u_i v_i \\ &= \sum_{p=1}^r \langle u^{[p]}, v^{[p]} \rangle \\ &= \sum_{p=1}^r 2 \cdot \text{POPCOUNT} \left(\Psi(u^{[p]}) \odot \Psi(v^{[p]}) \right) - m \\ &= 2 \sum_{p=1}^r \text{POPCOUNT} \left(\Psi(u^{[p]}) \odot \Psi(v^{[p]}) \right) - \sum_{p=1}^r m \\ &= \left(\sum_{p=1}^r \text{POPCOUNT} \left(\Psi(u^{[p]}) \odot \Psi(v^{[p]}) \right) \ll 1 \right) - n \end{aligned}$$

Ainsi, on a optimisé n multiplications et $n - 1$ additions en:

- $r - 1 = \frac{n}{m} - 1$ additions.
- r instructions **XNOR**
- 1 décalage à gauche.
- 1 soustraction.

3.4.5 Gain Temps & Mémoire

Gain Mémoire

Les implémentations classiques des RN utilisent des paramètres en flottantes à précision simple. c'est à dire chaque paramètre prends 32 bits.

Avec le codage considéré, on peut avoir un gain de mémoire $\alpha_{\text{mémoire}} = 32$

Gain Temps

Le gain de l'opération de produit scalaire est:

$$\begin{aligned}\alpha_{\langle \cdot, \cdot \rangle} &= \frac{n + n - 1}{\frac{n}{m} - 1 + \frac{n}{m} + 1 + 1} \\ &= \frac{2n - 1}{2\frac{n}{m} + 1} \\ &= \frac{2 - \frac{1}{n}}{2\frac{1}{m} + \frac{1}{n}} \\ &= \frac{2m - \frac{m}{n}}{2 + \frac{m}{n}} \\ &= \frac{m - \frac{m}{2n}}{1 + \frac{m}{2n}} \\ &= m - \frac{m(m + 1)}{2n} + o\left(\frac{m^3}{n^2}\right)\end{aligned}$$

On a:

- Pour un PC personnel, ou même téléphone: $m = 64$.
- Pour les réseaux de neurones standards, $32 \leq n \leq 1024$

Ainsi une estimation de α_{temps} est :

$$31.5 \leq \alpha_{\text{temps}} \leq \frac{2047}{32} \approx 62.03$$

3.4.6 Algorithme Optimisé

(À modifier)...

Algorithm 1 Interférence Optimisée

Require: K the size of the window

Require: $p \in \mathbb{N}^*$ the number of random crops per input

Require: X, Y dataset with size n

Ensure: X', Y' dataset with size pn

```
 $X' \leftarrow []$ 
 $y' \leftarrow []$ 
for  $x, y \in X, Y$  do
    for  $i \in \{1, \dots, p\}$  do
         $L \leftarrow \text{length}(X_i)$ 
        if  $L \geq K$  then
             $r \sim \mathcal{U}(0, L - K)$ 
             $x' \leftarrow x[r, \dots, r + K]$ 
        else
             $r \sim \mathcal{U}(0, K - R)$ 
             $a \leftarrow \text{zeros}(r)$ 
             $b \leftarrow \text{zeros}(K - R - r)$ 
             $x' \leftarrow \text{concat}(a, x, b)$ 
        end if
         $X'.\text{append}(x')$ 
         $y'.\text{append}(y)$ 
    end for
end for
```

Chapter 4

Modèles des BNNs

4.1 Introduction

Dans ce chapitre, on va étudier les approches faites dans la conception et implémentation des réseaux de neurones binaires, dériver les formules nécessaires dans les cas des couches dense et convolutionnelles, et dans le cas générique.

4.1.1 Importance de ce chapitre

Dans les articles de BNNs considérés, les formules données ne sont généralement applicable qu'aux couches convolutionnelles à 2 dimensions.

Nous chercherons ici à trouver une approche générique avec laquelle on peut dériver pour chaque modèle considéré les équations des couches dense, convolutionnelles, récurrentes, etc...

Notre approche admet principalement 2 caractéristiques:

Rigueur

Pour établir les équations nécessaires, nous avons suivi les étapes suivantes:

1. Définir l'utilité potentielle du modèle
2. Traduire l'objectif en un problème d'optimisation mathématique
3. Définir les hypothèses nécessaires
4. Etablir les formules

Flexibilité

Grâce au niveau de rigueur utilisé, on peut facilement modifier et même améliorer les équations. Dans notre implémentation de chaque modèle, nous allons se baser sur les formules de ce chapitre en offrant la possibilité de faire quelques modifications.

4.1.2 Convention Utilisée

Dans les démonstrations, nous allons suivre les conventions mathématiques de l'algèbre linéaire et de l'optimisation mathématique. Mais dans les implémentations, nous allons suivre les conventions de TensorFlow.

4.2 Histoire

Modèle	Année	Idée(s)
BinaryConnect[4]	2015	<ul style="list-style-type: none"> • Binarisation des poids en ± 1 • Optimisation des instructions MAC¹ en des sommes. • Estimateur direct du gradient pour résoudre le problème d'annulation de gradient.
BinaryNet[3]	2015	<ul style="list-style-type: none"> • Binarisation des paramètres et noeuds en ± 1 • Optimisation des instructions MAC² en des XNOR et POPCOUNT.
XNOR-NET[21]	2016	Introduction de deux facteurs α et β à chaque produit scalaire pour minimiser l'erreur de l'opérateur bilinéaire \star
ABCNet[17]	2017	<ul style="list-style-type: none"> • Utilisation de $n \in \mathbb{N}^*$ binarisations pour les paramètres: $\widetilde{\mathbf{W}}^{(l),1}, \dots, \widetilde{\mathbf{W}}^{(l),n}$ • Utilisation de $m \in \mathbb{N}^*$ binarisations pour les noeuds: $\tilde{a}^{(l),1}, \dots, \tilde{a}^{(l),m}$ • Décomposition de l'opération bilinéaire $\mathbf{W}^{(l)} \star a^{(l-1)}$ en une combinaison linéaire des $\widetilde{\mathbf{W}}^{(l),i} \star \tilde{a}^{(l-1),j}$ pour $i \in \{1, \dots, n\}$ et $j \in \{1, \dots, m\}$
BiRealNet[19]	2019	<ul style="list-style-type: none"> • Ajout d'une architecture résiduelle • Approximation régulière plus fine de la fonction sign
MeliusNet	2020	<ul style="list-style-type: none"> • Utilisation des blocs denses pour augmenter la capacité des caractéristiques • Utilisation des blocs d'améliorations pour augmenter la qualité des caractéristiques

Table 4.1: Le tableau d'avancement des BNNs

4.3 Comparaison

4.3.1 Par quantification

Modèle	Avantages	Inconvénients(s)
BinaryNet[3]	<ul style="list-style-type: none"> • Le plus léger • Interférence rapide 	<ul style="list-style-type: none"> • Induit des modèles très simple. • Admet des mauvaises performances de prédictions pour les tâches difficiles³. • Instable.
XNOR-NET[21]	<ul style="list-style-type: none"> • Le premier BNN performant. • Interférence rapide. 	<ul style="list-style-type: none"> • Réintroduit des multiplication et divisions. • Performances modestes.
ABCNet[17]	<ul style="list-style-type: none"> • Très performant. • Il est stable par rapport aux autres. • Admet une justification théorique grâce à l'algèbre linéaire. 	<ul style="list-style-type: none"> • Le temps d'interférence est en $\mathcal{O}(nm)$. • La mémoire est en $\mathcal{O}(n + m)$

Table 4.2: Approches de quantification

4.3.2 Par architecture

Modèle	Avantages	Inconvénients(s)
BiRealNet[19]	<ul style="list-style-type: none"> • BNN performant • Conserve la rapidité d'interférence. • Modèle plus stable 	<ul style="list-style-type: none"> • Conçu spécifiquement pour les architectures convolutionnelles • Instable.
MeliusNet	<ul style="list-style-type: none"> • Plus performant que BiRealNet • Conserve la rapidité d'interférence. • Modèle plus stable 	<ul style="list-style-type: none"> • Plus lourds que BiRealNet • Conçu spécifiquement pour les architectures convolutionnelles

Table 4.3: Approches architecturale

4.4 BinaryNet

4.4.1 Conception

BinaryNet[3] est le premier réseau de neurones totalement binaires dans ces couches cachées, c'est à dire tout passage d'une couche cachée à une autre se fait par des accumulations de nombres binarisés (± 1).

Autrement dit, c'est le premier réseau de neurones binaires conforme à notre définition.

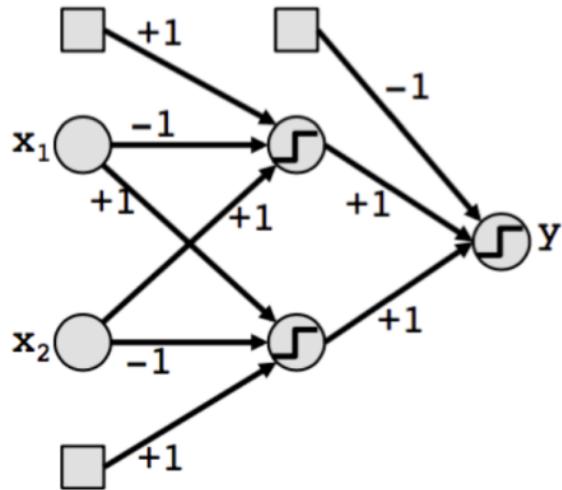


Figure 4.1: Exemple d'un BinaryConnect à une seule couche cachée

4.4.2 Topologie

BinaryNet est indépendant de l'architecture choisie: Elle peut être dense, convolutionnelle, récurrente, etc...

Mais en pratique, il n'est testé que dans le cas dense et convolutionnel.

4.4.3 Utilisation d'Estimateur Direct de gradient

Problématique

Puisque la fonction signe est constante par morceau, son dérivé est donc nulle presque partout (dans le sens de la théorie des mesures).

Ainsi, tout algorithme basé sur le gradient doit mettre en considération

Principe

L'entraînement d'un BNN est similaire à un RN classique:

1. Une propagation en avant pour calculer la valeur de la fonction objective
2. Une propagation en arrière pour calculer le gradient des paramètres
3. La mise à jour des paramètres

4. Refaire l'étape 1, jusqu'à satisfaire un critère bien déterminé (convergence, dépassement de la limite des époches, taux d'entraînement négligeable, etc...)

L'estimateur direct de gradient [1] sert à corriger le problème du gradient nul en faisant une estimation plus régulière de la fonction signe dans la propagation arrière (la fonction signe reste toujours utilisée dans la propagation avant)

L'estimation utilisée de la fonction signe est la fonction hardtanh définie par:

hardtanh : $\mathbb{R} \rightarrow \mathbb{R}$

$$x \rightarrow \begin{cases} -1 & \text{si } x < -1 \\ x & \text{si } |x| \leq 1 \\ 1 & \text{si } x > 1 \end{cases}$$

En fait son dérivé est:

$$\begin{aligned} \forall x \in \mathbb{R}, \quad \frac{\partial \text{hardtanh}}{\partial x}(x) &= \mathbb{1}_{|x| \leq 1}(x) \\ &= \begin{cases} 1 & \text{si } |x| \leq 1 \\ 0 & \text{sinon} \end{cases} \end{aligned}$$

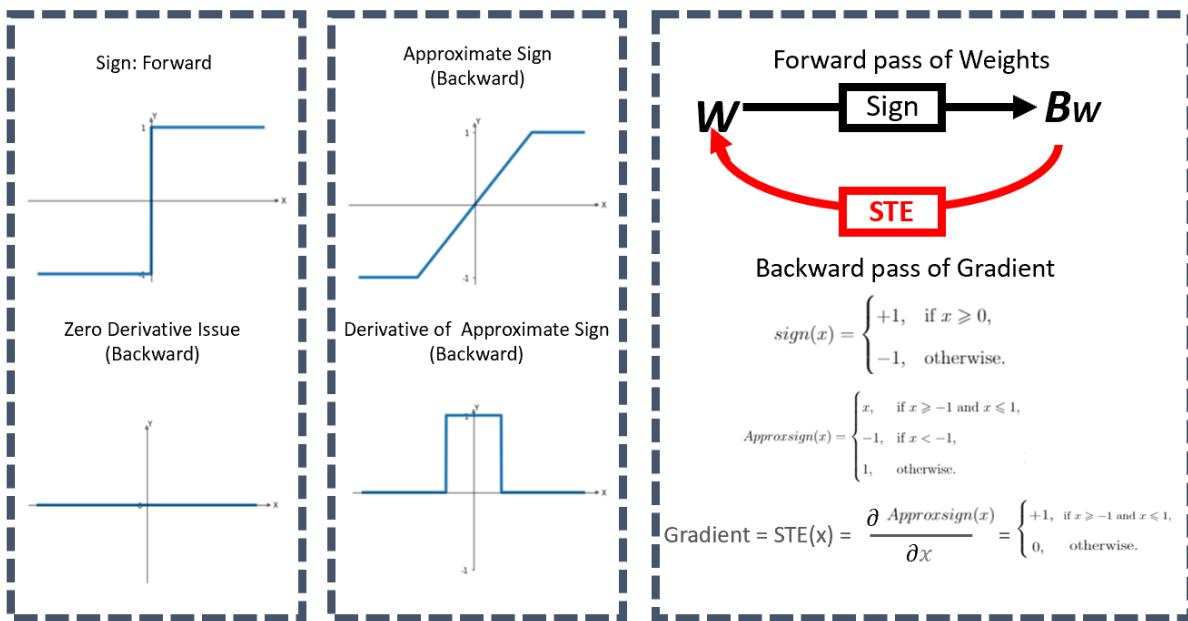


Figure 4.2: Estimateur direct du gradient

Estimation du gradient

Soit f une fonction. Soit \mathbf{x}, \mathbf{y} deux tenseurs tels que $f(\mathbf{x}) = y$. On note par $\mathcal{G}_{\mathbf{x}}$ l'estimation de gradient $\frac{\partial \mathcal{L}}{\partial \mathbf{x}}$ (par la méthode STE) et on la définit par:

$$\mathcal{G}_{\mathbf{x}} = \begin{cases} \frac{\partial \mathcal{L}}{\partial \mathbf{x}} \odot \mathbb{1}_{|\mathbf{x}| \leq 1} & \text{si } f = \text{sign} \\ \mathcal{G}_y \cdot \frac{\partial \mathbf{y}}{\partial \mathbf{x}} & \text{sinon} \end{cases}$$

Remarque 7 *Le calcul de $\mathbb{1}_{|\mathbf{x}| \leq 1}$ se fait élément par élément.*

Remarque 8 *L'opérateur \odot dénote la multiplication élément par élément*

La mise à jour des paramètres se fait en remplaçant chaque gradient par son estimation.

Justification Théorique

4.4.4 Optimisations possible

Etant un réseau totalement binarisé dans les couches cachées, BinaryConnect peut avoir plusieurs optimisations, y compris:

- Utilisation du Codage binaire
- Utilisation des instructions XNOR et POPCOUNT pour accélérer la propagation avant
- Utilisation de ADA-Max pour accélérer la mise à jour des paramètres
- Utilisation de "Shifted Batch Normalisation" pour accélérer le calcul de la normalisation par lots.

4.5 XNOR-NET

4.5.1 Conception

XNOR-NET[21] est le premier réseau de neurones binaires admettant des performances acceptables. Il est basé sur BinaryNet, et il met l'accent sur l'utilisation de XNOR et POPCOUNT [21].

De plus, Il introduit une nouvelle approche pour réduire l'erreur de quantification

4.5.2 Topologie

Comme BinaryConnect, XNOR-NET[21] est indépendant de l'architecture et la topologie utilisée.

En autre partie, il n'est pratiquement implémenté que dans les architectures denses et convolutionnelles.

4.5.3 Réduction de l'erreur de quantification

Pour $\mathbf{W}^{(l)}$ et $a^{(l-1)}$, on veut trouver deux facteurs $\alpha, \beta \in \mathbb{R}_+$, et deux binarisations $\tilde{\mathbf{W}}^{(l)}$ et $\tilde{a}^{(l-1)}$ respectivement de $\mathbf{W}^{(l)}$ et $a^{(l-1)}$ tels que:

$$\mathbf{W}^{(l)} \approx \alpha \tilde{\mathbf{W}}^{(l)} \quad (4.1)$$

$$\tilde{a}^{(l-1)} \approx \beta \tilde{a}^{(l-1)} \quad (4.2)$$

$$\mathbf{W}^{(l)} \star a^{(l-1)} \approx \alpha \beta \tilde{\mathbf{W}}^{(l)} \star \tilde{a}^{(l-1)} \quad (4.3)$$

Dans l'objectif décrit, les binarisations ne sont pas nécessairement la fonction sign, mais on montrera que c'est toujours un bon choix.

4.5.4 Quantification optimale d'un vecteur

Dans cette section, nous allons trouvé les paramètres α et β optimaux des équation respectifs (4.1) et (4.2) sans tenir compte de l'équation (4.3).

C'est à dire, nous allons dériver les quantification optimales de $\mathbf{W}^{(l)}$ et $a^{(l-1)}$ sans tenir compte de la qualité de la quantification (4.3) de $z^{(l)} = \mathbf{W}^{(l)} \star a^{(l-1)}$.

En effet, nous allons raisonner d'une manière générale. Et puis les deux résultats vont être immédiatement déduits.

Notations

On va dénoter par:

- $n \in \mathbb{N}^*$ la dimension de notre espace vectoriel.
- w un vecteur de \mathbb{R}^n .
- $\tilde{w} \in \{-1, 1\}^n$ une binarisation de w .
- $\gamma \in \mathbb{R}_+$ un facteur d'échelle.

Objectif

L'objectif est de trouver \tilde{w} et γ qui minimisent $\|w - \gamma\tilde{w}\|_2^2$ pour un u donné:

$$(\gamma, \tilde{w}) = \underset{\gamma, \tilde{w}}{\operatorname{argmin}} \|w - \gamma\tilde{w}\|_2^2 \quad (4.4)$$

Résolution du problème

On a:

$$\begin{aligned} \forall \gamma \in \mathbb{R}, \forall \tilde{w} \in \{-1, 1\}^n, \quad \|w - \gamma\tilde{w}\|_2^2 &= \langle w, w \rangle - 2\gamma\langle w, \tilde{w} \rangle + \gamma^2\langle \tilde{w}, \tilde{w} \rangle \\ &= \langle w, w \rangle - 2\gamma\langle w, \tilde{w} \rangle + n\gamma^2 \\ \text{car } \tilde{w} \in \{-1, 1\}^n, \quad \|\tilde{w}\|_2^2 &= \sum_{i=1}^n \tilde{w}_i^2 = n \end{aligned}$$

Maintenant, on a $\langle w, \tilde{w} \rangle$ est maximisé pour $\tilde{w} = \operatorname{sign} w$, et par conséquent puisque $\gamma \geq 0$ $\langle w, w \rangle - 2\gamma\langle w, \tilde{w} \rangle + n\gamma^2$ est minimisé pour $\tilde{w} = \operatorname{sign} w \quad \forall \gamma \in \mathbb{R}_+$.

Ainsi le problème est réduit à la minimisation de la fonction quadratique suivante:

$$H(\gamma) = \langle w, w \rangle - 2\gamma\langle w, \operatorname{sign} w \rangle + n\gamma^2 \quad (4.5)$$

En effet, H étant une fonction convexe sur \mathbb{R} , admet un unique minimum local en $x^* = \frac{\langle w, \operatorname{sign} w \rangle}{n}$.
Or, on a:

$$\begin{aligned} x^* &= \frac{\langle w, \tilde{w} \rangle}{n} = \frac{1}{n} \sum_{i=1}^n w_i \operatorname{sign} w_i \\ &= \frac{1}{n} \sum_{i=1}^n |w_i| = \frac{\|w\|_1}{n} \geq 0 \end{aligned}$$

Ainsi, $\gamma = x^*$, et donc l'expression de γ est:

$$\gamma = \frac{\|w\|_1}{n} \quad (4.6)$$

Quantification de $\mathbf{W}^{(l)}$

On a $\mathbf{W}^{(l)} \in E$ où $E \cong \mathbb{R}^{n_1}$ avec $n_1 = \dim \mathbf{W}^{(l)}$.

Ainsi, on a $\|\mathbf{W}^{(l)} - \alpha\tilde{\mathbf{W}}^{(l)}\|_2^2$ est minimisé pour:

$$\begin{cases} \alpha &= \frac{\|\mathbf{W}^{(l)}\|}{n_1} \\ \tilde{\mathbf{W}}^{(l)} &= \operatorname{sign} \mathbf{W}^{(l)} \end{cases} \quad (4.7)$$

Quantification de $a^{(l)}$

On a $a^{(l)} \in F$ où $F \cong \mathbb{R}^{n_2}$ avec $n_2 = \dim a^{(l)}$.

Ainsi, on a $\|a^{(l)} - \beta \tilde{a}^{(l)}\|_2^2$ est minimisé pour:

$$\begin{cases} \beta &= \frac{\|a^{(l)}\|}{n_2} \\ \tilde{a}^{(l)} &= \text{sign } a^{(l)} \end{cases} \quad (4.8)$$

4.5.5 Quantification d'un produit scalaire de deux vecteurs

Dans cette partie, nous allons tenir compte de la quantification de $z^{(l)}$, c'est à dire nous allons minimiser l'erreur de l'équation (4.3).

La résolution de cette équation n'est pas aussi triviale que (4.1) et (4.2), et donc nous allons majorer l'erreur de quantification $\|\langle u, v \rangle - \langle \alpha \tilde{u}, \beta \tilde{v} \rangle\|_2^2$ par une fonction plus facile à optimiser, et nous allons retrouver les expressions (4.7) et (4.8) de la section précédente.

Objectif

- Soit $n \in \mathbb{N}$ la dimension de l'espace euclidien.
- Soit $u, v \in \mathbb{R}^n$

Notre objectif est de trouver deux facteurs $\alpha, \beta \in \mathbb{R}_+$ et deux vecteurs $u, v \in \{-1, 1\}^n$ qui réduisent $\|\langle u, v \rangle - \langle \alpha \tilde{u}, \beta \tilde{v} \rangle\|_2^2$

Une Borne supérieure de l'erreur

Une formule exacte pour le problème originale est un peu difficile. Pour cela, on pose $w = u \odot v$, et on cherche une borne supérieure de $\|\langle u, v \rangle - \langle \alpha \tilde{u}, \beta \tilde{v} \rangle\|_2^2$ en fonction de $\|w - \alpha \beta \tilde{u} \odot \tilde{v}\|_2^2$ [21]:

$$\begin{aligned} \|\langle u, v \rangle - \langle \alpha \tilde{u}, \beta \tilde{v} \rangle\|_2^2 &= (\langle u, v \rangle - \langle \alpha \tilde{u}, \beta \tilde{v} \rangle)^2 \\ &= \left(\sum_{i=1}^n u_i v_i - \alpha \beta \tilde{u}_i \tilde{v}_i \right)^2 \\ &= \left| \sum_{i=1}^n \sum_{j=1}^n (u_i v_i - \alpha \beta \tilde{u}_i \tilde{v}_i)(u_j v_j - \alpha \beta \tilde{u}_j \tilde{v}_j) \right| \\ &\leq \sum_{i=1}^n \sum_{j=1}^n |u_i v_i - \alpha \beta \tilde{u}_i \tilde{v}_i| |u_j v_j - \alpha \beta \tilde{u}_j \tilde{v}_j| \\ &\leq \sum_{i=1}^n \sum_{j=1}^n \|u \odot v - (\alpha \tilde{u}) \odot (\beta \tilde{v})\|_\infty^2 \\ &\leq n^2 \|u \odot v - (\alpha \tilde{u}) \odot (\beta \tilde{v})\|_\infty^2 \\ &\leq n^2 \|u \odot v - (\alpha \tilde{u}) \odot (\beta \tilde{v})\|_2^2 \\ &\leq n^2 \|w - \alpha \beta \tilde{u} \odot \tilde{v}\|_2^2 \end{aligned}$$

Avec ce résultat, on est sûr qu'en minimisant $\|w - \alpha \beta \tilde{u} \odot \tilde{v}\|_2^2$, on minimise la borne supérieure de l'erreur de quantification.

Minimisation de $\|w - \alpha\beta\tilde{u} \odot \tilde{v}\|_2^2$

En effet, Puisque $\tilde{u} \odot \tilde{v}$ génère tout les vecteurs binaires, et $\alpha\beta$ génère tous les réels positifs, on peut trouver $\underset{\alpha, \beta, \tilde{u}, \tilde{v}}{\operatorname{argmin}} \|w - \alpha\beta\tilde{u} \odot \tilde{v}\|_2^2$ à partir de $\underset{\gamma, \tilde{w}}{\operatorname{argmin}} \|w - \gamma\tilde{w}\|_2^2$.

Or, on a déjà trouvé ce dernier. Ainsi on a:

$$\begin{cases} \tilde{u} \odot \tilde{v} = \tilde{w} = \operatorname{sign} w = \operatorname{sign} u \odot \operatorname{sign} v \\ \alpha\beta = \gamma = \frac{\|w\|_1}{n} \end{cases} \quad (4.9)$$

Valeur de α, β, \tilde{u} et \tilde{v}

Pour trouver chacune de α, β on fait les hypothèses suivantes:

- u_1, \dots, u_n et \mathbf{x} suivent une distribution \mathcal{U} à moment absolu de premier ordre fini.
- v_1, \dots, v_n et \mathbf{y} suivent une distribution \mathcal{V} à moment absolu de premier ordre fini.
- $\forall i, j \in \{1, \dots, n\}$, u_i et v_j sont indépendents
- α et \tilde{u} ne dépendent que de u
- β et \tilde{v} ne dépendent que de v

En exploitant les hypothèses proposées, on peut facilement trouver \tilde{u} et \tilde{v} :

$$\begin{cases} \tilde{u} = \operatorname{sign} u \\ \tilde{v} = \operatorname{sign} v \end{cases} \quad (4.10)$$

Pour α et β , on donne la démonstration suivante:

$$\begin{aligned} \mathbb{E}[\gamma] &= \mathbb{E}\left[\frac{\|w\|_1}{n}\right] = \frac{1}{n}\mathbb{E}[\|w\|_1] \\ &= \frac{1}{n}\sum_{i=1}^n \mathbb{E}[|u_i v_j|] = \frac{1}{n}\sum_{i=1}^n \mathbb{E}[|u_i|]\mathbb{E}[|v_j|] \\ &= \frac{1}{n}\sum_{i=1}^n \mathbb{E}[|\mathbf{x}|]\mathbb{E}[|\mathbf{y}|] = \mathbb{E}[|\mathbf{x}|]\mathbb{E}[|\mathbf{y}|] \\ &= \mathbb{E}\left[\frac{\|u\|_1}{n}\right] \times \mathbb{E}\left[\frac{\|v\|_1}{n}\right] \\ &= \mathbb{E}[\alpha]\mathbb{E}[\beta] \end{aligned}$$

Ainsi avec les hypothèses proposées on trouve $\mathbb{E}[\alpha] = \mathbb{E}[|\mathbf{u}|]$ et $\mathbb{E}[\beta] = \mathbb{E}[|\mathbf{v}|]$, et donc:

$$\begin{cases} \alpha \approx \frac{\|u\|_1}{n} \\ \beta \approx \frac{\|v\|_1}{n} \end{cases} \quad (4.11)$$

En conclusion, on trouve[21]:

$$\begin{cases} u \approx \frac{\|u\|_1}{n} \operatorname{sign} u \\ v \approx \frac{\|v\|_1}{n} \operatorname{sign} v \end{cases} \quad (4.12)$$

4.5.6 Quantification d'une couche dense par produits scalaires

Dans ce cas, on a:

$$\mathbf{W}^{(l)} a^{(l-1)} = \begin{pmatrix} \langle \mathbf{W}_1^{(l)}, a^{(l-1)} \rangle \\ \vdots \\ \langle \mathbf{W}_{n_l}^{(l)}, a^{(l-1)} \rangle \end{pmatrix}$$

On applique la réduction d'erreur de quantification élément par élément:

$$\begin{cases} a^{(l-1)} \approx \beta \operatorname{sign} a^{(l-1)} & \beta = \frac{\|a^{(l-1)}\|_1}{n_{l-1}} \\ \forall i \in \{1, \dots, n_l\}, \quad \mathbf{W}_i^{(l)} \approx \alpha_i \operatorname{sign} \mathbf{W}_i^{(l)} & \alpha_i = \frac{\|\mathbf{W}_i^{(l)}\|_1}{n_{l-1}} \end{cases} \quad (4.13)$$

Ainsi

$$\begin{aligned} \mathbf{W}^{(l)} a^{(l-1)} &\approx \begin{pmatrix} \beta \alpha_1 \langle \operatorname{sign} \mathbf{W}_1^{(l)}, \operatorname{sign} a^{(l-1)} \rangle \\ \vdots \\ \beta \alpha_{n_l} \langle \operatorname{sign} \mathbf{W}_{n_l}^{(l)}, \operatorname{sign} a^{(l-1)} \rangle \end{pmatrix} \\ &\approx \beta \left(\operatorname{sign} \mathbf{W}^{(l)} \operatorname{sign} a^{(l-1)} \right) \odot \boldsymbol{\alpha} \end{aligned} \quad (4.14)$$

Full-precision		Binary		Binary with scaling	
Input	Weight	Input	Weight	Input Scaling	Weight Scaling
0.1	0.5 -0.5 -0.1	1	1 -1 -1	0.4	0.33 0.45 0.4
-0.7	-0.1 0.5 0.5	-1	-1 1 1		
0.5	-0.4 -0.7 0.3	1	-1 -1 1		
0.3	0.3 -0.1 -0.7	1	1 -1 -1		
Result	0.01 -0.78 -0.42	Result	2 -4 -2	Result	0.26 -0.72 -0.32
		Error	1.99 3.22 1.58	Error	0.25 0.06 0.1
Normalized	1.01 -0.95 -0.06	Normalized	1.25 -1.0 -0.25	Normalized	1.2 -1.06 -0.14
		Error	0.24 0.05 0.19	Error	0.19 0.11 0.08

Figure 4.3: Utilisation de facteurs α et β dans une couche dense

4.5.7 Quantification d'une couche convolutionnelle par produits scalaires

Remarque 9 Nous allons raisonner sur les convolutions à 2 dimensions. Mais en effet, les résultats peuvent être trivialement généralisés.

Remarque 10 Pour trouver des expressions de α et β , nous allons raisonner sur une convolution

sans strides et sans padding. On va proposer ensuite un truc qui permet de recouvrir les formules de α et β dans ces cas.

Soit $\mathbf{W}^{(l)}$ une couche convolutionnelle qui prends C_{in} canaux et donne C_{out} canaux et dont les noyaux sont de dimensions (w, h) .

La formule de $z^{(l)}$ est:

$$\forall c \in \{1, \dots, C_{\text{out}}\}, \quad z_c^{(l)} = \sum_{c'=1}^{C_{\text{in}}} \mathbf{W}_{c,c'}^{(l)} * a_{c'}^{(l-1)} \quad (4.15)$$

Remarque 11 On peut aussi écrire (4.15) sous la forme compacte:

$$z^{(l)} = \mathbf{W}^{(l)} * a^{(l-1)} \quad (4.16)$$

où $\mathbf{W}^{(l)}$ est de dimension $C_{\text{out}} \times C_{\text{in}} \times w \times h$, et $a^{(l-1)}$ est de dimension $C_{\text{in}} \times w \times h$

Pour simplifier, nous allons raisonner sur le cas $C_{\text{out}} = 1$, puis nous allons généraliser:

Cas d'un seul canal de sortie: $C_{\text{out}} = 1$

On commence par le cas $C_{\text{out}} = 1$: On a:

$$\alpha = \frac{\|\mathbf{W}^{(l)}\|_1}{w \cdot h \cdot C_{\text{in}}} \quad (4.17)$$

$$\forall i, j \quad \beta_{i,j} = \frac{\|a_{[i,w],[j,h]}^{(l-1)}\|_1}{w \cdot h \cdot C_{\text{in}}} \quad (4.18)$$

avec $a_{[i,w],[j,h]}^{(l-1)}$ est le sous-tenseur de dimensions $C_{\text{in}} \times w \times h$ qui commence à la position i dans son deuxième axe, et j dans son troisième axe.

En faite, $(\beta_{i,j})_{i,j}$ constitue un tenseur β ⁴ de dimension $w \times h$, et la formule de $z^{(l)}$ sera:

$$z^{(l)} = (\text{sign } \mathbf{W}^{(l)} * \text{sign } a^{(l-1)}) \odot \beta \alpha \quad (4.19)$$

Cas de plusieurs canaux de sortie

Dans le cas où $C_{\text{out}} > 1$, on décompose cette couche convolutionnelle en plusieurs convolutions. et on dérive la formule de chacune:

$$\forall c \in \{1, \dots, C_{\text{out}}\}, \quad z_c^{(l)} = (\text{sign } \mathbf{W}_c^{(l)} * \text{sign } a^{(l-1)}) \odot \beta \alpha_c \quad (4.20)$$

Finalement, on peut l'écrire dans la forme compacte:

$$z^{(l)} = (\text{sign } \mathbf{W}^{(l)} * \text{sign } a^{(l-1)}) \odot (\boldsymbol{\alpha} \otimes \boldsymbol{\beta}) \quad (4.21)$$

où \otimes est le produit extérieur entre deux tenseurs.

⁴Le temps de calcul de β peut être optimisé grâce aux techniques de la programmation dynamique

Cass général

En général, la convolution n'est pas nécessairement 2 dimensionnelle, et la convolution peut contenir des strides.

Dans ce cas, l'expression (4.21) restera toujours valable, mais avec une modification pour le facteur β (La formule (4.19) pour α restera le même).

En effet, en posant:

- \mathbf{J} un tenseur des 1 admettant la même dimension de $\mathbf{W}^{(l)}$
- \mathbf{Q} le tenseur de même dimensions que $\mathbf{W}^{(l)} * a^{(l-1)}$ avec $Q_{i,j,\dots}$ représente la taille du plus petit sous-tenseur de $a^{(l-1)}$ avec lequel on peut calculer le terme $(\mathbf{W}^{(l)} * a^{(l-1)})_{i,j,\dots}$
- $|a^{(l-1)}|$ la valeur absolue terme à terme.

On a:

$$\beta = (\mathbf{J} * |a^{(l-1)}|) \oplus \mathbf{Q} \quad (4.22)$$

Avec \oplus la division terme à terme.

En effet, dans le cas où la convolution n'admet pas de padding, \mathbf{Q} sera un tenseur dont tous les termes sont égaux à $n = \frac{1}{\dim \mathbf{W}^{(l)}}$.

Dans ce cas la formule (4.22) peut être simplifié à

$$\beta = \frac{1}{n} \mathbf{J} * |a^{(l-1)}| \quad (4.23)$$

En outre, même si la convolution admet un padding, on peut utiliser la formule (4.23) comme une approximation de (4.22)

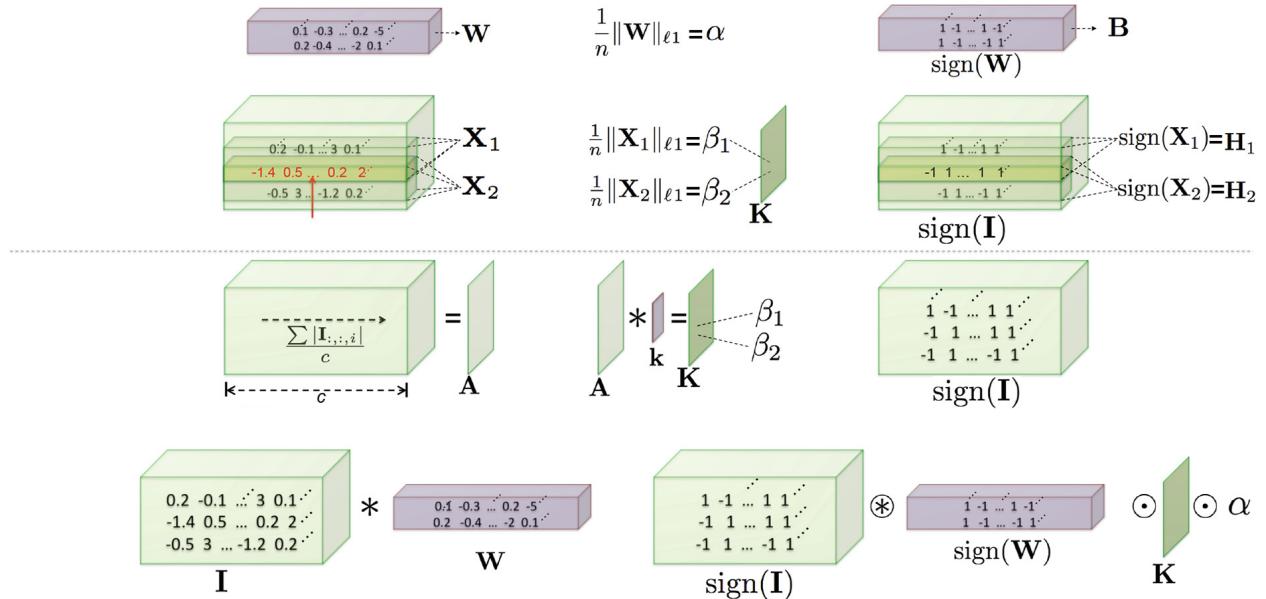


Figure 4.4: Utilisation de facteurs α et β dans une couche convolutionnelle

Dans cette figure, I représente $a^{(l-1)}$, W représente $\mathbf{W}^{(l)}$, et k représente $\frac{1}{n} \mathbf{J}$. La convolution contre A constitue une optimisation potentielle de notre formule en terme de complexités de calculs.

4.5.8 Quantification d'une opération bilinéaire quelconque

Hypothèses

Les hypothèses suivantes ne posent aucune contrainte supplémentaires, en effet on les pose pour formaliser notre démonstration:

- On suppose que $\mathbf{W}^{(l)}$ et $a^{(l-1)}$ sont deux membres des espaces vectoriels réels respectifs E_l et F_{l-1} .
- On suppose que $\star : E_l \times F_{l-1} \rightarrow H_l$ est bilinéaire avec H_l est un espace vectoriel réel de dimension $\dim H_l = s$.
- On suppose que \mathcal{B}_1 et \mathcal{B}_2 sont les bases respectifs de E_l et F_{l-1} .

Décomposition de \star en formes bilinéaires

On a \star peut être décomposé élément par élément en s forme bilinéaire $\star_1, \dots, \star_s : E_l \times F_{l-1} \rightarrow \mathbb{R}$. Soit $M_i = \text{mat}(\star_i, \mathcal{B}_1, \mathcal{B}_2)$. On a donc:

$$\mathbf{W}^{(l)} \star a^{(l-1)} = \begin{pmatrix} \mathbf{W}^{(l)} \star_1 a^{(l-1)} \\ \vdots \\ \mathbf{W}^{(l)} \star_s a^{(l-1)} \end{pmatrix} = \begin{pmatrix} \mathbf{W}^{(l)T} M_1 a^{(l-1)} \\ \vdots \\ \mathbf{W}^{(l)T} M_s a^{(l-1)} \end{pmatrix}$$

Décomposition en valeurs singulières

Dans cette section, nous allons quantifier chaque forme bilinéaire indépendamment de l'autre. Soit $i \in \{1, \dots, s\}$. On a d'après la décomposition en valeurs singulières:

$$\exists V_i \in \mathcal{O}(\mathbb{R}, m), \exists U_i \in \mathcal{O}(\mathbb{R}, n), \exists \Sigma_i \in \mathcal{M}(\mathbb{R}, n, m) \text{ diagonal} / \quad \begin{cases} M_i = U_i^T \Sigma_i V_i \\ \Sigma_{i,p,p} = \sigma_{i,p} \geq 0 \quad \forall p \in \{1, \dots, \min(n, m)\} \\ \Sigma_{i,p,p} = \sigma_{i,p} = 0 \quad \forall p > \min(n, m) \end{cases}$$

On peut encore factoriser Σ_i en deux matrices diagonales $A_i \in \mathcal{M}(\mathbb{R}, \min(n, m), n)$ et $B_i \in \mathcal{M}(\mathbb{R}, \min(n, m), m)$ tels que:

$$\begin{cases} A_{i,p,p} = B_{i,p,p} = \sqrt{\sigma_{i,p}} \quad \forall p \in \{1, \dots, \min(n, m)\} \\ \Sigma_i = A_i^T B_i \end{cases} \quad (4.24)$$

On a donc:

$$\begin{aligned} \mathbf{W}^{(l)} \star_i a^{(l-1)} &= \mathbf{W}^{(l)T} M_i a^{(l-1)} \\ &= \mathbf{W}^{(l)T} U_i^T A_i^T B_i V_i a^{(l-1)} \\ &= (A_i U_i \mathbf{W}^{(l)})^T (B_i V_i a^{(l-1)}) \\ &= \langle A_i U_i \mathbf{W}^{(l)}, B_i V_i a^{(l-1)} \rangle \end{aligned} \quad (4.25)$$

Pour quantifier ce produit scalaire, on doit d'abord généraliser l'équation (4.4) en:

$$(\gamma, \tilde{w}) = \underset{\gamma, \tilde{w}}{\text{argmin}} \|A U w - \gamma A U \tilde{w}\|_2^2 \quad (4.26)$$

Avec

- $A \in \mathcal{M}(\mathbb{R}, m, n)$ une matrice diagonale dont les coefficient diagonaux $A_{p,p} = \sqrt{\sigma_p}$ sont positifs
- $U \in \mathcal{O}(\mathbb{R}, n)$ une matrice orthogonale.

Quantification optimale de AUw

Avant de résoudre ce problème, nous allons utiliser la propriété suivante:

$$U^T A^T A U = A^T A = D^2 \quad (4.27)$$

Avec $D \in \mathcal{M}(\mathbb{R}, m)$ la matrice diagonale tels que $D_{p,p} = \sqrt{\sigma_p}$

En effet, en exploitant (4.27) on a:

$$\begin{aligned} \|AUw - \gamma AU\tilde{w}\|_2^2 &= \langle AUw, AUw \rangle - 2\gamma \langle AUw, AU\tilde{w} \rangle + \gamma^2 \langle AU\tilde{w}, AU\tilde{w} \rangle \\ &= \langle AUw, AUw \rangle - 2\gamma \langle AUw, AU\tilde{w} \rangle + \gamma^2 \langle AU\tilde{w}, AU\tilde{w} \rangle \\ &= \langle w, U^T A^T AUw \rangle - 2\gamma \langle w, U^T A^T AU\tilde{w} \rangle + \gamma^2 \langle \tilde{w}, U^T A^T AU\tilde{w} \rangle \\ &= \langle w, D^2 w \rangle - 2\gamma \langle w, D^2 \tilde{w} \rangle + \gamma^2 \langle \tilde{w}, D^2 \tilde{w} \rangle \\ &= \langle Dw, Dw \rangle - 2\gamma \langle Dw, D\tilde{w} \rangle + \gamma^2 \langle D\tilde{w}, D\tilde{w} \rangle \end{aligned}$$

Or on a:

$$\langle D\tilde{w}, D\tilde{w} \rangle = \sum_{i=1}^n \sigma_i \tilde{w}_i^2 = \sum_{i=1}^n \sigma_i$$

Et donc on a:

$$\|\Sigma Vw - \gamma \Sigma V\tilde{w}\|_2^2 = \langle Dw, Dw \rangle - 2\gamma \langle Dw, D\tilde{w} \rangle + \gamma^2 \sum_{i=1}^n \sigma_i$$

Maintenant, d'une façon analogue à la démonstration de 4.5.4, on peut montrer que $\langle Dw, D\tilde{w} \rangle$ est maximisé pour $\tilde{w} = \text{sign } w$, et donc puisque $\gamma \geq 0$, on a $\|AUw - \gamma AU\tilde{w}\|_2^2$ est minimisé pour $\tilde{w} = \text{sign } w$ indépendamment de γ .

En s'inspirant de 4.5.4, on trouve:

$$\begin{cases} \gamma &= \frac{\sum_{i=1}^n \sigma_i |w_i|}{\sum_{i=1}^n \sigma_i} \\ \tilde{w} &= \text{sign } w \end{cases} \quad (4.28)$$

Quantification optimale de $A_i U_i \mathbf{W}^{(l)}$

D'après la résolution du problème (4.26), et en exploitant le résultat (4.28):

$$\forall i \in \{1, \dots, r\} \begin{cases} \alpha_i &= \frac{\sum_{j=1}^n \sigma_{i,j} |\mathbf{W}_j^{(l)}|}{\sum_{j=1}^n \sigma_{i,j}} \\ \tilde{\mathbf{W}}^{(l)} &= \text{sign } \mathbf{W}^{(l)} \end{cases} \quad (4.29)$$

Quantification optimale de $B_i V_i a^{(l-1)}$

De même, on trouve que:

$$\forall i \in \{1, \dots, s\} \begin{cases} \beta_i &= \frac{\sum_{j=1}^n \sigma_{i,j} |a_j^{(l-1)}|}{\sum_{j=1}^n \sigma_{i,j}} \\ \tilde{a}^{(l-1)} &= \text{sign } a^{(l-1)} \end{cases} \quad (4.30)$$

Quantification de l'opérateur bilinéaire

Le résultat final est peut être écrit sous cette forme compacte:

$$\mathbf{W}^{(l)} \star a^{(l-1)} \approx \left(\text{sign } \mathbf{W}^{(l)} \star \text{sign } a^{(l-1)} \right) \odot \boldsymbol{\alpha} \odot \boldsymbol{\beta} \quad (4.31)$$

4.6 ABCNet

4.6.1 Conception

ABCNet[17] est un réseau de neurones binaires basé sur XNOR-Net, son nom est une abréviation de "Accurate Binary Convolutional Network", et donc comme son nom suggeste, il est très performant (comparable à un CNN classique).

Il peut être considéré comme une amélioration directe de XnorNet, puisqu'il réduit l'erreur de quantification d'une variable en faisant une somme pondérée de binarisations prédefinie.

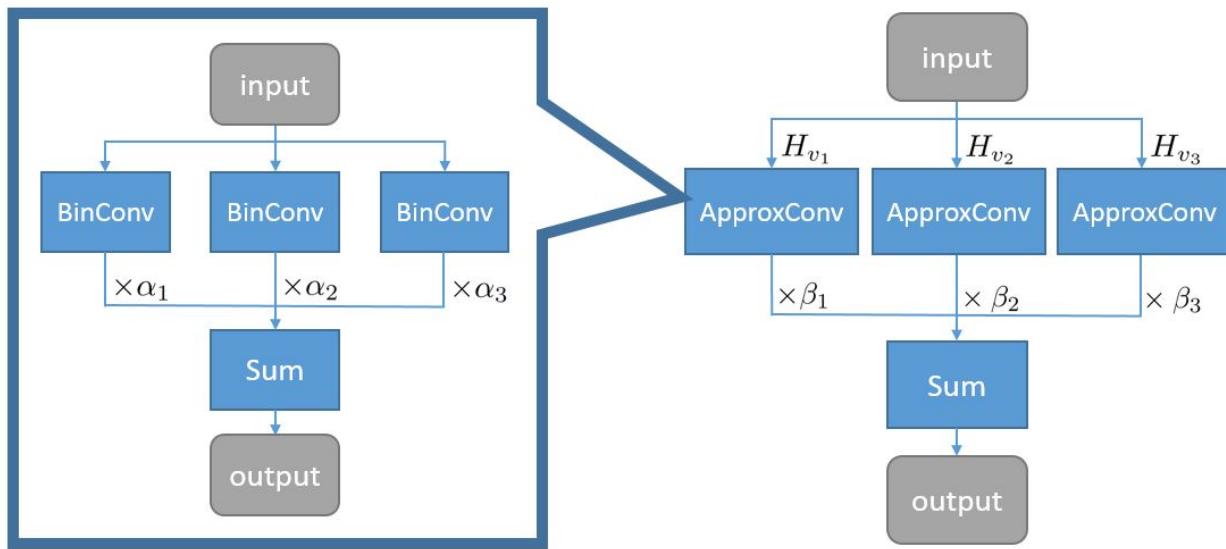


Figure 4.5: Exemple d'un ABCNet à une seule couche

4.6.2 Topologie

Le ABCNet[17] est conçu explicitement pour les réseaux de neurones binaires convolutionnelles.

Mais comme le XNOR-NET, il peut être généralisé à n'importe quel architecture (dense, récurrente, etc...)

4.6.3 Objectif

Notations

On dénote par

1. Ψ_1, \dots, Ψ_n des binarisations de $\mathbf{W}^{(l)}$
2. Φ_1, \dots, Φ_m des binarisations de $a^{(l-1)}$

Approximation par combinaisons linéaires

L'objectif d'un ABCNet est d'approximer $\mathbf{W}^{(l)}$ en une combinaison linéaires de n binarisations, et $a^{(l-1)}$ en une combinaison linéaires de m binarisations.

On veut trouver $(\Psi_1, \dots, \Psi_n), (\Phi_1, \dots, \Phi_m), (\alpha_1, \dots, \alpha_n) \in \mathbb{R}_+^n, (\beta_1, \dots, \beta_m) \in \mathbb{R}_+^m$ tel que:

$$\mathbf{W}^{(l)} \approx \sum_{i=1}^n \alpha_i \Psi_i(\mathbf{W}^{(l)}) \quad (4.32)$$

$$a^{(l-1)} \approx \sum_{i=1}^m \beta_i \Phi_i(a^{(l-1)}) \quad (4.33)$$

$$\mathbf{W}^{(l)} \star a^{(l-1)} \approx \sum_{i=1}^n \sum_{j=1}^m \alpha_i \beta_j \Psi_i(\mathbf{W}^{(l)}) \star \Phi_j(a^{(l-1)}) \quad (4.34)$$

4.6.4 Choix de binarisations de $\mathbf{W}^{(l)}$

Il y'a plusieurs méthodes pour choisir les binarisations de $\mathbf{W}^{(l)}$

Base orthogonale de Hadamard

Cette technique crée des binarisations qui ne dépendent pas de $\mathbf{W}^{(l)}$. En interprétant $\mathbf{W}^{(l)}$ comme un vecteur d'un espace euclidien E de dimension $r = 2^s \geq n$. On cherche la matrice de Hadamard H_s , puis on choisit n colonnes (ou lignes) Ψ_1, \dots, Ψ_n .

Dans ce cas, la famille (Ψ_1, \dots, Ψ_n) induit un sous-espace F de E de dimension n .

Ainsi, le problème est réduit à une projection orthogonale de E vers F

En notant $K = \text{mat}(\Psi_1, \dots, \Psi_n)$, la valeur optimale de α qui minimise l'erreur quadratique de quantification est:

$$\alpha = K^T \mathbf{W}^{(l)} \quad (4.35)$$

Famille de fonctions sign décalées

Cette technique crée des binarisations de la forme:

$$\Psi_i(\mathbf{W}^{(l)}) = \text{sign}(\mathbf{W}^{(l)} + \mu_i) \quad (4.36)$$

Avec $\mu_1, \dots, \mu_n \in \mathbb{R}$ un paramètre de décalage. Il peut être entraînable ou non.

Approximation par distribution

Cette technique se base sur l'observation que la distribution des poids (non-binarisés) est dense[17], et elle est proche d'une distribution normale. L'expression de Ψ_i est:

$$\Psi_i(\mathbf{W}^{(l)}) = \text{sign}\left(\mathbf{W}^{(l)} - \mathbb{E}[\mathbf{W}^{(l)}] + \mu_i \sqrt{\mathbb{V}[\mathbf{W}^{(l)}]}\right) \quad (4.37)$$

Où

- $\mathbb{E}[\mathbf{W}^{(l)}]$ est un estimateur de l'espérance de la distribution des paramètres: c'est la valeur moyenne de tous ces paramètres.
- $\mathbb{V}[\mathbf{W}^{(l)}]$ est une estimation de leur variance.
- μ_i est fixe ou entraînable.

Quelque soit le cas, on peut initialiser μ_i [17] par:

$$\mu_i = -1 + 2 \cdot \frac{i-1}{n-1} \quad (4.38)$$

4.6.5 Choix de binarisations de $a^{(l)}$

Puisque $a^{(l)}$ dépend toujours de l'entrée x . Pour cela $a^{(l)}$ va généralement varier d'une inférence à une autre.

Pour cela, on doit créer des binarisations qui dépendent de $a^{(l)}$. Par exemple, on peut choisir:

Famille de fonctions sign décalées

Cette technique crée des binarisations de la forme:

$$\Psi_j(a^{(l)}) = \text{sign}(a^{(l)} + \kappa_j) \quad (4.39)$$

Avec $\kappa_1, \dots, \kappa_m \in \mathbb{R}$ un paramètre de décalage. Il peut être entraînable ou non.

Approximation par distribution

$$\Phi_j(a^{(l)}) = \text{sign}\left(a^{(l)} - \mathbb{E}[a^{(l)}] + \kappa_j \sqrt{\mathbb{V}[a^{(l)}]}\right) \quad (4.40)$$

Où

- $\mathbb{E}[a^{(l)}]$ est une estimation de l'espérance de $a^{(l)}$ dans le lot courant.
- $\mathbb{V}[a^{(l)}]$ est une estimation de la variance de $a^{(l)}$ dans le lot courant
- κ_j est fixe ou entraînable.

Dans la phase d'inférence, pour éviter le coût de calcul des paramètres statistiques, on peut remplacer:

- $\mathbb{E}[a^{(l)}]$ du lot courant par $K_1 = \mathbb{E}_B[\mathbb{E}[a^{(l)}]]$, qui est la valeur moyenne de tous les lots du jeu de données.
- $\mathbb{V}[a^{(l)}]$ du lot courant par $K_2 = \frac{m}{m-1} \mathbb{E}_B[\mathbb{V}[a^{(l)}]]$, qui est une estimation sans biais de la variance de $a^{(l)}$ dans tous les lots du jeu de données, avec m la taille de lot.

4.6.6 Quantification par tenseur binaire

C'est la méthode où on fait les quantifications de $\mathbf{W}^{(l)}$ et $a^{(l-1)}$ en suivant les équations respectifs (4.32) et (4.33)

La quantification de $z^{(l)}$ va respecter l'équation (4.34).

4.6.7 Quantification par produit scalaire

La quantification par produit scalaire est une variante dans laquelle les variables $\alpha_1, \dots, \alpha_n$ de (4.32) et β_1, \dots, β_m de (4.33) sont calculées pour chaque produit scalaire.

C'est à dire, L'équation (4.34) est décomposée terme par terme en des produits scalaires, dans chacune on cherche les $(\alpha_i)_{i \in \{1, \dots, j\}}$ et $(\beta_j)_{j \in \{1, \dots, m\}}$.

Couche Dense

C'est une généralisation de l'équation (4.14) de XnorNet.

On a:

$$\begin{aligned}
 \mathbf{W}^{(l)} \cdot a^{(l-1)} &\approx \begin{pmatrix} \left\langle \sum_{i=1}^n \alpha_{i,1} \Psi_i \left(\mathbf{W}_1^{(l)} \right), \sum_{j=1}^m \beta_j \Phi_j \left(a^{(l-1)} \right) \right\rangle \\ \vdots \\ \left\langle \sum_{i=1}^n \alpha_{i,n_l} \Psi_i \left(\mathbf{W}_{n_l}^{(l)} \right), \sum_{j=1}^m \beta_j \Phi_j \left(a^{(l-1)} \right) \right\rangle \end{pmatrix} \\
 &\approx \begin{pmatrix} \sum_{i=1}^n \sum_{j=1}^m \left\langle \alpha_{i,1} \Psi_i \left(\mathbf{W}_1^{(l)} \right), \beta_j \Phi_j \left(a^{(l-1)} \right) \right\rangle \\ \vdots \\ \sum_{i=1}^n \sum_{j=1}^m \left\langle \alpha_{i,n_l} \Psi_i \left(\mathbf{W}_{n_l}^{(l)} \right), \beta_j \Phi_j \left(a^{(l-1)} \right) \right\rangle \end{pmatrix} \\
 &\approx \sum_{i=1}^n \sum_{j=1}^m \begin{pmatrix} \left\langle \alpha_{i,1} \Psi_i \left(\mathbf{W}_1^{(l)} \right), \beta_j \Phi_j \left(a^{(l-1)} \right) \right\rangle \\ \vdots \\ \left\langle \alpha_{i,n_l} \Psi_i \left(\mathbf{W}_{n_l}^{(l)} \right), \beta_j \Phi_j \left(a^{(l-1)} \right) \right\rangle \end{pmatrix} \\
 &\approx \sum_{i=1}^n \sum_{j=1}^m \beta_j \left(\Psi_i \left(\mathbf{W}^{(l)} \right) \cdot \Phi_j \left(\text{sign } a^{(l-1)} \right) \right) \odot \boldsymbol{\alpha}_i
 \end{aligned} \tag{4.41}$$

Couche convolutionnelle

C'est une généralisation de l'équation (4.19) de XnorNet[21].

On a:

$$\mathbf{W}^{(l)} a^{(l-1)} \approx \sum_{i=1}^n \sum_{j=1}^m \left(\Psi_i \left(\mathbf{W}^{(l)} \right) * \Phi_j \left(a^{(l-1)} \right) \right) \odot (\boldsymbol{\alpha}_i \otimes \boldsymbol{\beta}_j) \tag{4.42}$$

Opérateur bilinéaire quelconque

C'est une généralisation de 4.5.8, l'équation (4.31) va être généralisé en:

$$\mathbf{W}^{(l)} \star a^{(l-1)} \approx \sum_{i=1}^n \sum_{j=1}^m \left(\Psi_i \left(\mathbf{W}^{(l)} \right) \star \Phi_j \left(a^{(l-1)} \right) \right) \odot \boldsymbol{\alpha}_i \odot \boldsymbol{\beta}_j \tag{4.43}$$

4.7 BiRealNet

4.7.1 Conception

BiRealNet[19] est un réseau de neurones binarisés basés sur BinaryNet.

Son nom signifie le fait que s'il est binaire, il a aussi le comportement continu présent dans les réseaux de neurones classiques à valeurs réelles.

Il n'est pas basé sur la quantification comme les autres BNNs, mais sur l'architecture du réseau. En effet il réduit la perte d'information causée par la quantification par l'ajout des connexions résiduelles. Dans BiRealNet les connexions résiduelles sont sous la forme d'une somme simple.

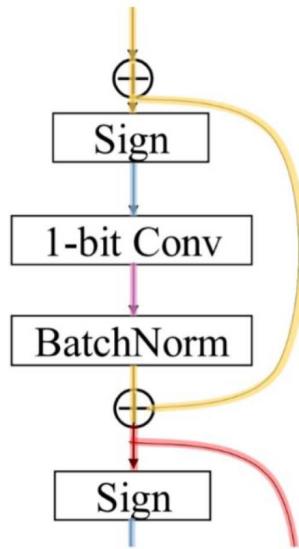


Figure 4.6: Exemple d'une liaison résiduelle d'un BiRealNet

4.7.2 Topologie

BiRealNet[19] est pratiquement indépendant de la topologie elle-même. Mais la présence des liaisons résiduelles y induit quelques contraintes.

La principale contrainte est que la couche résiduelle et la couche actuelle doivent avoir les mêmes dimensions.

4.7.3 Block BiRealNet

Pour bien augmenter l'apport d'informations avec BiRealNet, la connexion résiduelle est faite entre deux blocs connexes et qui vérifient la contrainte sur les dimensions.

De plus, le résidu doit avoir la possibilité de varier sur un ensemble réel. C'est à dire varier sur un grands ensemble de valeurs.

Un exemple concis est la figure 4.6 qui montre une connexion résiduelle pour chaque bloc de:

- Quantification par signe
- Convolution binarisée
- Normalisation par Lots

En effet, selon cette figure, la connexion va être faite entre le résultat du bloc actuel et le bloc précédent.

4.7.4 Liaison résiduale

Notation

Soit $\mathcal{R}(l)$ l'ensemble de couches qui ont une liaison résiduale avec la $l^{\text{ème}}$ couche.

Contraintes sur $\mathcal{R}(l)$

Dans BiRealNet, la seule contrainte présente est que le réseau de neurone doit rester acyclique:

$$\forall l, \forall l' \in \mathcal{R}(l), \quad l' < l \quad (4.44)$$

Equation

La liaison résiduale induit un changement de la formule de $z^{(l)}$ trouvée dans le tableau 3.3 situé dans la section 3.2.1. La nouvelle expression de $z^{(l)}$ est:

$$z^{(l)} = \mathbf{W}^{(l)} \tilde{\star} a^{(l-1)} + \sum_{l' \in \mathcal{R}(l)} z^{(l')} \quad (4.45)$$

Dans cette équation, $\tilde{\star}$ signifie que l'opérateur \star est quantifié avec une quantification quelconque. Dans la plus part des cas, $\mathcal{R}(l)$ est un singleton qui contient seulement la couche du dernier bloc, ce qui est le cas par exemple pour la figure 4.6

4.7.5 Choix de la quantification

Dans l'équation (4.45), nous n'avons explicité aucune quantification, et c'est l'une des avantages de BiRealNet: il est orthogonal au type de binarisation⁵.

Ainsi, l'opérateur quantifié $\tilde{\star}$ peut être basé sur BinaryNet, XnorNet, ABCNet, etc...

⁵En effet, BiRealNet est un modèle basé sur l'architecture, et ces modèles sont généralement orthogonaux aux modèles basées sur la quantification, dans le sens où on peut créer un modèle en exploitant en même temps une quantification d'un et une architecture de l'autre.

Chapter 5

Implémentation

5.1 Introduction

Dans ce chapitre, on va implémenter les modèles qu'on a décri dans le chapitre précédent tout en suivant les paradigme que Keras et Larq respectent.

Définition des types Bien que Python est dynamiquement typé, nous allons inférer dans ce chapitre et dans notre bibliothèque un typage statique pour faciliter la compréhension.

Nous allons dénoter pour un ensemble de types $T, T_1, \dots, T_n, Q, Q_1, \dots, Q_m$:

- $\text{List}[T]$ pour n’importe quel itérable sur T
- $\text{Tuple}[T_1, \dots, T_n]$ pour un tuple contenant des éléments de T_1, \dots, T_n dans l’ordre indiqué.
- $\mathcal{T}[T] = \text{Tensor}[T]$ pour un type compatible avec `tensorflow.Tensor` avec `dtype=T`
- $\mathcal{T}[T, n] = \text{Tensor}[T, n]$ pour un type compatible avec `tensorflow.Tensor` avec un rang égal à n et avec `dtype=T`
- $\mathcal{V}[T] = \text{Vector}[T]$ pour un type compatible avec `tensorflow.Tensor` avec un rang égal à 1 et avec `dtype=T`
- $\mathcal{M}[T] = \text{Matrix}[T]$ pour un type compatible avec `tensorflow.Tensor` avec un rang égal à 2 et avec `dtype=T`
- $\mathcal{R}[T] = \text{Random}[T]$ pour une fonction aléatoire sans arguments.
- $\mathcal{F}[T_1, \dots, T_n \rightarrow Q_1, \dots, Q_m] = \text{Function}[T_1, \dots, T_n \rightarrow Q_1, \dots, Q_m]$ pour une fonction dont les arguments sont de types respectifs T_1, \dots, T_n et dont les valeurs sonts de types respectifs Q_1, \dots, Q_m
- $\text{End}[T] = \text{Function}[T_1, \dots, T_n \rightarrow T_1, \dots, T_n]$ pour une fonction dont l’argument et les valeurs sont de types respectifs T_1, \dots, T_n .
- $\mathcal{P}[T_1, \dots, T_n] = \text{Predicate}[T_1, \dots, T_n]$ pour un prédictat (fonction booléenne) acceptant n arguments de types respectifs T_1, \dots, T_n

5.2 Paradigmes

5.2.1 Dans l'implémentation de BinaryFlow

Dans l'implémentations de la bibliothèque BinaryFlow, nous avons adopté principalement:

- La programmation orientée objet
- La programmation générique

Programmation orientée objet

La programmation orientée objet (POO) est un paradigme informatique consistant à définir et à faire interagir des objets grâce à différentes technologies, notamment les langages de programmation (Python dans notre cas).

On appelle objet, un ensemble de variables complexes et de fonctions, comme par exemple la couche dense `binaryflow.BinaryNet.Dense`.

Presque tout peut être considéré comme un objet. L'objectif de la programmation orientée objet est de se concentrer sur l'objet lui-même et les données, plutôt que sur la logique nécessaire et les actions à mener pour faire cette manipulation.

Programmation générique

Un avantage de la programmation générique consiste à abstraire un ensemble de concepts cohérents pour construire des algorithmes au dessus indépendamment de leur implémentation.

L'implémentation sera essentiellement agnostique au type, à condition qu'il vérifie le contrat imposé.

5.2.2 Dans l'utilisation de BinaryFlow

Etant une extension de Keras et Larq, nous recommanderions l'utilisateur à suivre les bonnes pratiques de ces bibliothèques.

En effet nous encouragerions à suivre une de ces interfaces:

- L'interface Séquentielle
- L'interface fonctionnelle
- L'interface orientée objet

Interface séquentielle

C'est l'interface la plus facile à exploiter, mais naturellement elle ne supporte pas les couches résiduelles.

L'utilisation de cette interface sert à créer une objet de type `tensorflow.keras.Sequential` contenant la séquence des couches dont le type est `List[tensorflow.keras.layers.Layer]`

Interface fonctionnelle

Cette interface met en valeur le fait qu'un modèle se traduit en des fonctions évaluer dans un certain ordre à un tenseur initial X .

Pour cela, cette interface introduit un tenseur symbolique \mathbf{X} qui décrit l'entrée du réseau de neurones. Puis les fonctions sont appliquées au tenseur \mathbf{X} pour établir le modèle et fixer les dimensions de:

- Ces paramètres
- L'entrée de chaque fonction
- La sortie de chaque fonction

Une fois le modèle est construit, on peut l'entraîner, et puis le déployer.

Interface orientée objet

Cette interface met en valeur l'aspect orienté objet de TensorFlow (et aussi BinaryFlow), il se base sur le fait que qu'un utilisateur peut définir son modèle en héritant de la classe `tensorflow.keras.Model`. En utilisant cette interface, l'utilisateur peut surcharger les méthodes de `tensorflow.keras.Model`, ce qui lui permet de personnaliser:

- L'entraînement du modèle
- L'inférence du modèle
- L'évaluation du modèle
- La fonction coût du modèle
- ...

Interface bas niveaux

Cette interface utilise directement les fonctionnalités de TensorFlow sans exploiter Keras. Elle est aussi supportée par BinaryFlow, mais on ne la recommande que pour les utilisateurs avancés.

5.2.3 Dans l'extension de BinaryFlow

Pour étendre notre bibliothèques, nous recommanderions de suivre les paradigmes utilisées dans l'implémentation qui sont décrite dans la section 5.2.1.

5.3 Conception

BinaryFlow est implémentés sous formes des modules suivants:

- Module des couches
- Module des blocs
- Module des quantificateurs
- Module des coûts

Dans les sections suivantes nous allons parler de chaque module.

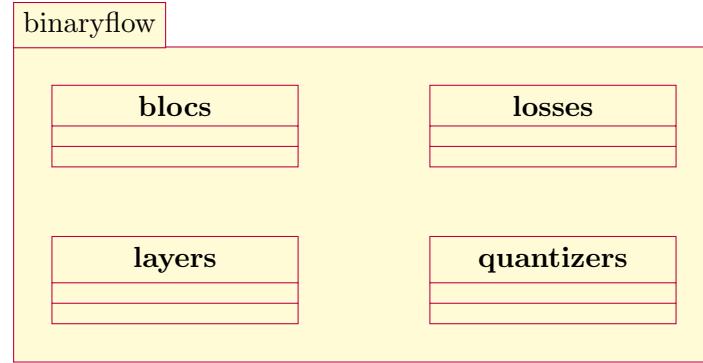


Figure 5.1: Structure de la bibliothèque

5.4 Couches

Ce module est l'implémentation des couches des modèles suivant:

- BinaryNet
- XnorNet
- ABCNet

Dans ce module, chaque couche admet un nom de la forme `NomModèle.TypeCouche` avec:

- `NomModèle` $\in \{\text{BinaryNet}, \text{XnorNet}, \text{ABCNet}\}$
- `TypeCouche` $\in \{\text{Dense}, \text{Conv1D}, \text{Conv2D}, \text{Conv3D}\}$

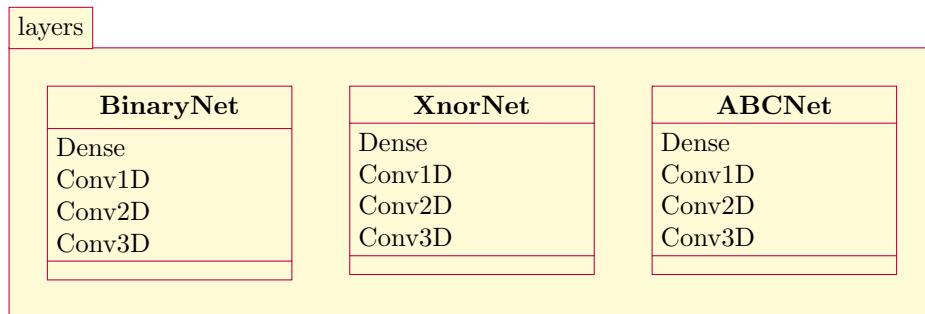


Figure 5.2: Le contenu de `binaryflow.layers`

5.4.1 BinaryNet

Le module `BinaryNet` contient 4 classes qui sont: `Dense`, `Conv1D`, `Conv2D` et `Conv3D`. Nous avons définir ces classes comme des synonymes respectivement de `Larq.layers.QuantDense`, `Larq.layers.QuantConv1D`, et `Larq.layers.QuantConv3D`

API commun

Ces 4 couches admettent des attributs similaires, qui sont:

- `kernel_quantizer` qui est la quantification de poids.
- `input_quantizer` qui est la quantification des noeuds.
- `kernel_constraint` qui est la contrainte sur les poids.

Pour conformer à la formulation originale de `BinaryNet` [3] et `BinaryConnect` [4] ces valeurs doivent être initialisés à

Attribut	Valeur
<code>kernel_quantizer</code>	<ul style="list-style-type: none">• <code>Larq.quantizers.SteSign</code>• <code>BinaryFlow.quantizers.StochasticSteSign</code>
<code>input_quantizer</code>	<ul style="list-style-type: none">• <code>Larq.quantizers.SteSign</code>• <code>BinaryFlow.quantizers.StochasticSteSign</code>
<code>kernel_constraint</code>	<ul style="list-style-type: none">• <code>weight_clip</code>

Table 5.1: Paramètres de `BinaryNet` et `BinaryConnect` originaux

Dense

Les attributs définissant cette couches sont:

- `units` qui est la dimension de sortie
- `kernel` qui est la matrice de paramètres

ConvND

Les couches convolutionnelles admettent des attributs similaires.

Pour une couche convolutionnelle à $N \in \{1, 2, 3\}$ dimensions, on a:

- `filters` est le nombre de canaux de sorties.
- `kernel_size` est la taille du noyau de convolution

- **padding** est le padding utilisée dans la convolution.
- **strides** est la taille de pas entre deux convolutions successives

BinaryNet

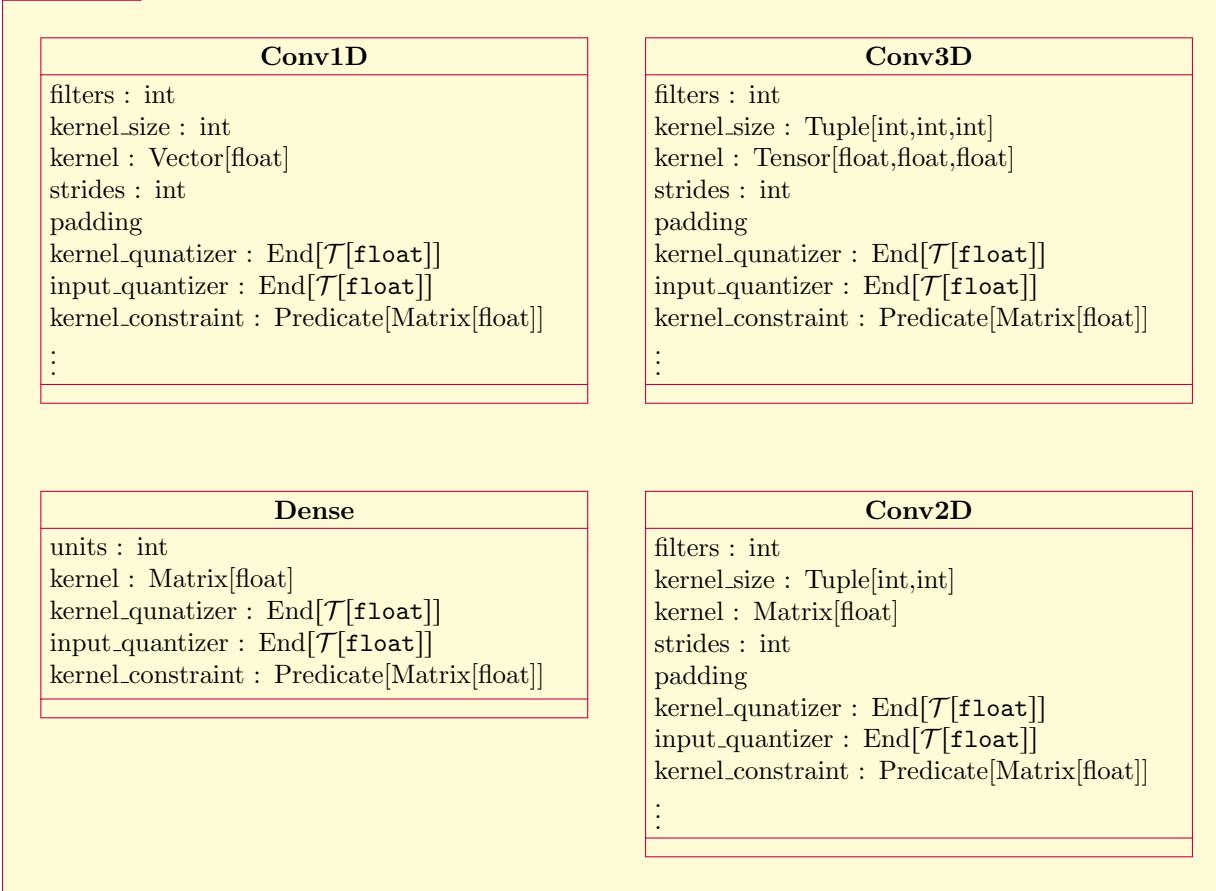


Figure 5.3: Diagramme de classe du module BinaryNet

5.4.2 XnorNet

Le module `XnorNet` contient 4 classes qui sont: `Dense`, `Conv1D`, `Conv2D` et `Conv3D`.

Chacune de ces classes est une classe fille de son contrepart dans le module `BinaryNet` présenté dans 5.4.1.

Attributs Ajouté

Le seul attribut ajouté est `alpha_trainable` qui demande si α doit être entraîné ou non. Par défaut il est égal à `False`.

Initialisaiton de α

Indépendemment de `alpha_trainable`, α est toujours initialisé à 4.14 dans le cas d'une couche dense, et à 4.14 dans le cas d'une couche convolutionnelle.

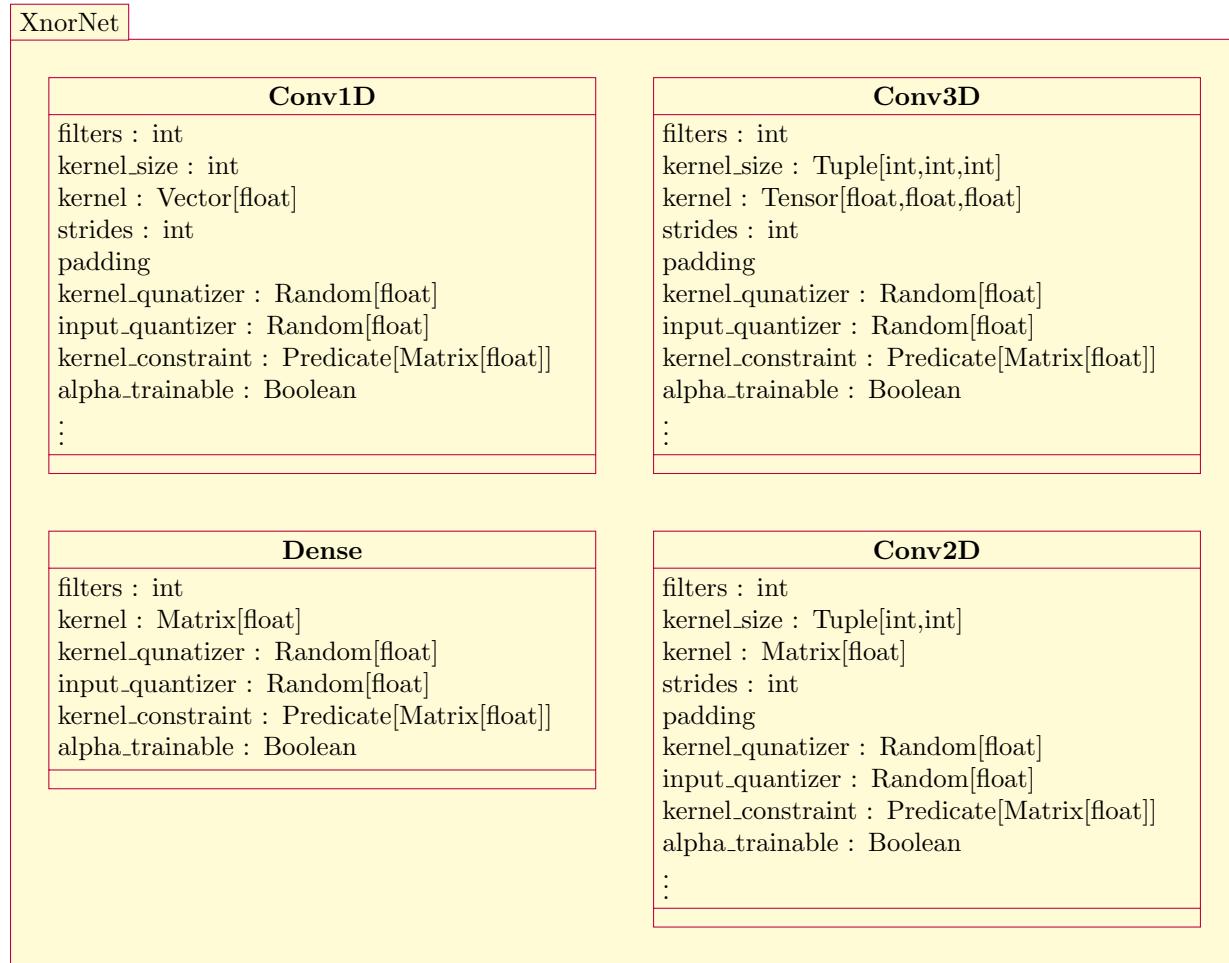


Figure 5.4: Diagramme de classe du module XnorNet

5.4.3 ABCNet

Le module `ABCNet` contient 4 classes qui sont: `Dense`, `Conv1D`, `Conv2D` et `Conv3D`. Chacune de ces classes est une classe fille de `tensorflow.keras.Model`.

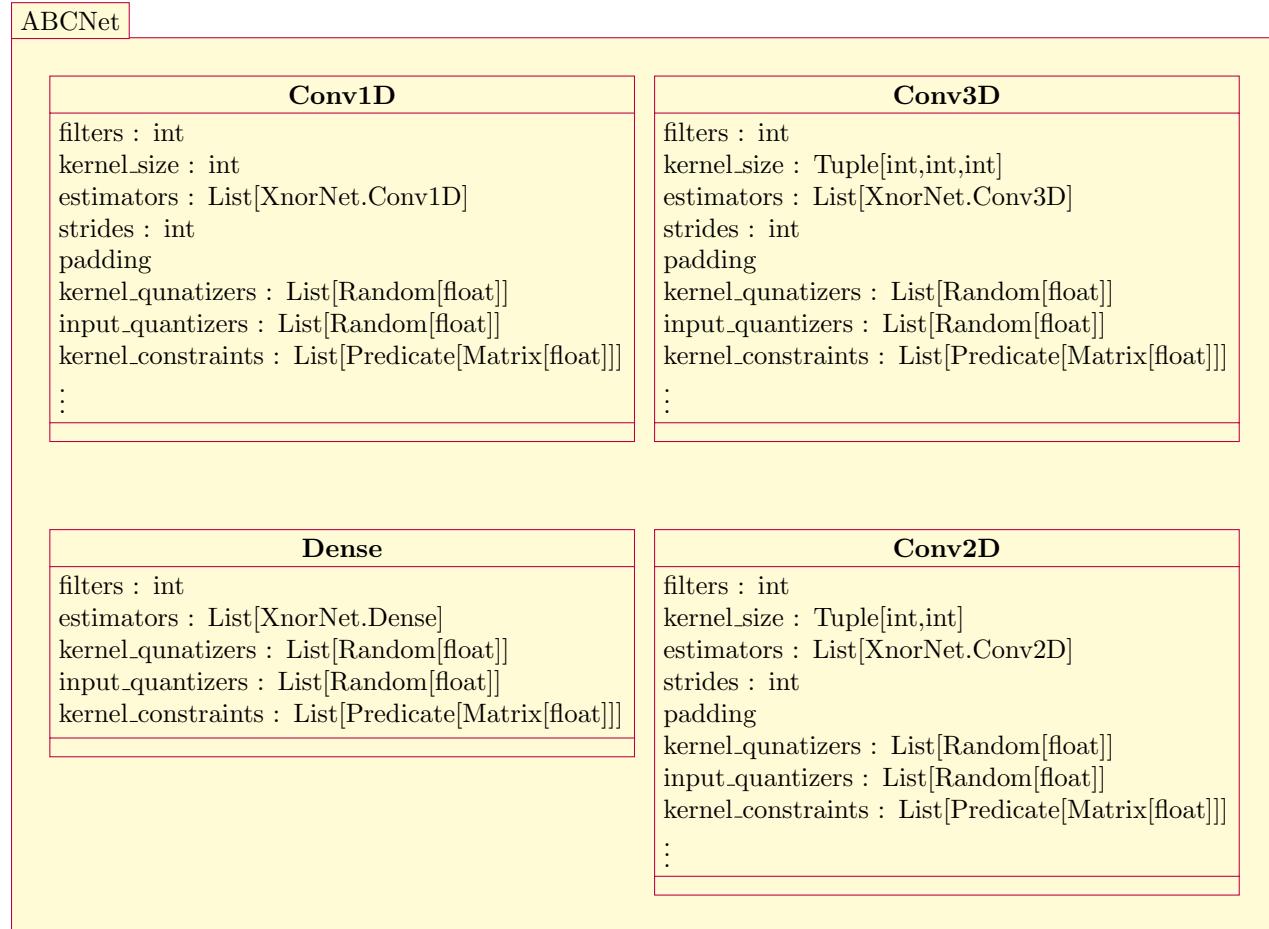


Figure 5.5: Diagramme de classe du module ABCNet

5.5 Block

Ce module est l'implémentation des "couches" des modèles suivant:

- BiRealNet
- MeliusNet

Vue leur nature résiduelle, une "couche" de l'un de ces modèles constitue effectivement un bloc.

Remarque 12 *Ce module n'est nécessaire que dans l'utilisation de l'interface séquentielle.*

Dans ce module, chaque bloc admet un nom de la forme `NomBloc.TypeCouche` avec:

- `NomBloc` $\in \{\text{Sequential}, \text{BiRealNet}, \text{MeliusNet}\}$
- `TypeCouche` $\in \{\text{Dense}, \text{Conv1D}, \text{Conv2D}, \text{Conv3D}\}$

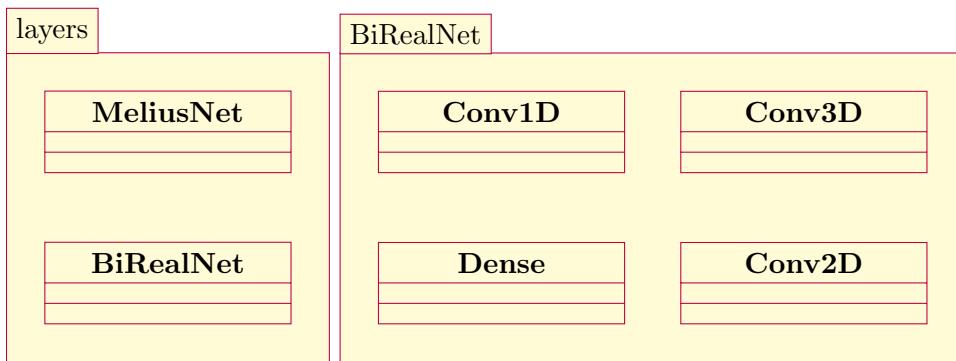


Figure 5.6: Le contenu de `binaryflow.blocs`, et de sous-module `BiRealNet`

5.6 Binarisations

Dans ce qui précède, nous n'avons parlé que de la fonction signe avec STE dans la propagation en arrière. Mais en effet BinaryFlow, supporte plusieurs binarisations, et ces binarisations peuvent être utilisées dans n'importe quel couche grâce à l'aspect modulaire.

Pour chaque binarisation, on peut aussi personnaliser l'estimation de son gradient, ce qui donne une flexibilité dans le choix de la binarisation.

Pour cela, nous allons étudier les binarisations que nous avons implémentées, et les approximations de leurs gradients

5.6.1 Propagation en Avant

Définition 5 *Une méta-binarisation est un opérateur \mathcal{K} qui prend une binarisation Ψ et donne une autre binarisation $\mathcal{K}(\Psi)$*

Nous avons les binarisations suivantes

Fonction signe

C'est la binarisation la plus utilisée dans la littérature, et en particulier les papiers de BinaryNet [3], XnorNet [21] et ABCNet [17].

L'expression de cette fonction est:

$$x \rightarrow \text{sign}(x) \quad (5.1)$$

Fonction heavyside

C'est une autre binarisation très connue qui peut être utilisée.

L'expression de cette fonction est:

$$x \rightarrow H(x) = \mathbf{1}_{x \geq 0}(x) \quad (5.2)$$

Cette binarisation doit être utilisée avec attention, car son utilisation rend plusieurs formules analytiques de XnorNet et ABCNet invalides.

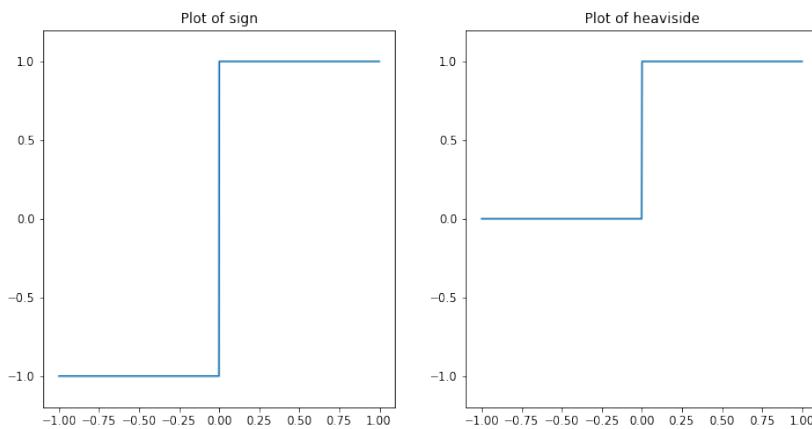


Figure 5.7: Traçage des fonction Signe et Heaviside

Binarisation décalée

C'est une meta-binarisation qui prend une binarisation Ψ et donne Ψ décalé par un paramètre μ :

$$x \rightarrow \Psi(x + \mu) \quad (5.3)$$

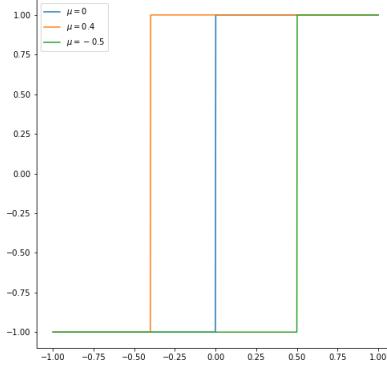


Figure 5.8: Traçage de la fonction signe décalée à gauche par μ

Le paramètre μ peut être fixe ou entraîné.

Binarisation stochastique

C'est aussi une meta-binarisation. Elle prend une binarisation Ψ et donne Ψ décalé par une variable aléatoire z suivant une distribution \mathcal{D} :

$$x \rightarrow \Psi(x + z) \quad (5.4)$$

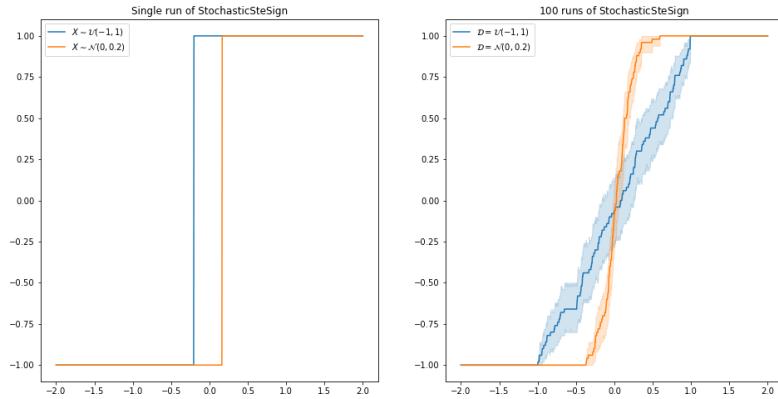


Figure 5.9: Traçage de la fonction signe stochastique

Dans cette figure, $\mathcal{N}(0, 0.2)$ dénote la distribution normale centrée avec une deviation standard $\sigma = 0.2$.
 $\mathcal{U}(-1, 1)$ dénote la distribution uniforme dans l'intervalle $[-1, 1]$

- La variable aléatoire z suit une distribution \mathcal{D} qui peut être fixe, ou même entraînée.
- Cette binarisation peut être utilisée comme une forme de régularisation pour le modèle.

5.6.2 Propagation en arrière

On peut résumer les approximations du gradient par:

Estimation	Expression	Binarisation correspondante
STE	$1_{ x } \leq 1$	Signe, Heavyside.
ApproxSign	$1_{ x } \leq 1 \cdot (2 - x)$	Sign
SwishSigne	$\frac{\beta(2 - \beta x \tanh(\frac{\beta x}{2}))}{1 + \cosh(\beta x)}$	Signe.

Table 5.2: Les estimations de gradient présentes

5.6.3 Implémentation

Dans `binaryflow`, les binarisations sont implémentées dans le module `quantizers`

Toute binarisation implémentée hérite de la classe `larq.quantizers.Quantizer`, et elle admet aussi un attribut de classe `precision` qui est égal à 1.

Cet attribut permet à Larq de faire les optimisations adéquates dans la phase de déploiement.

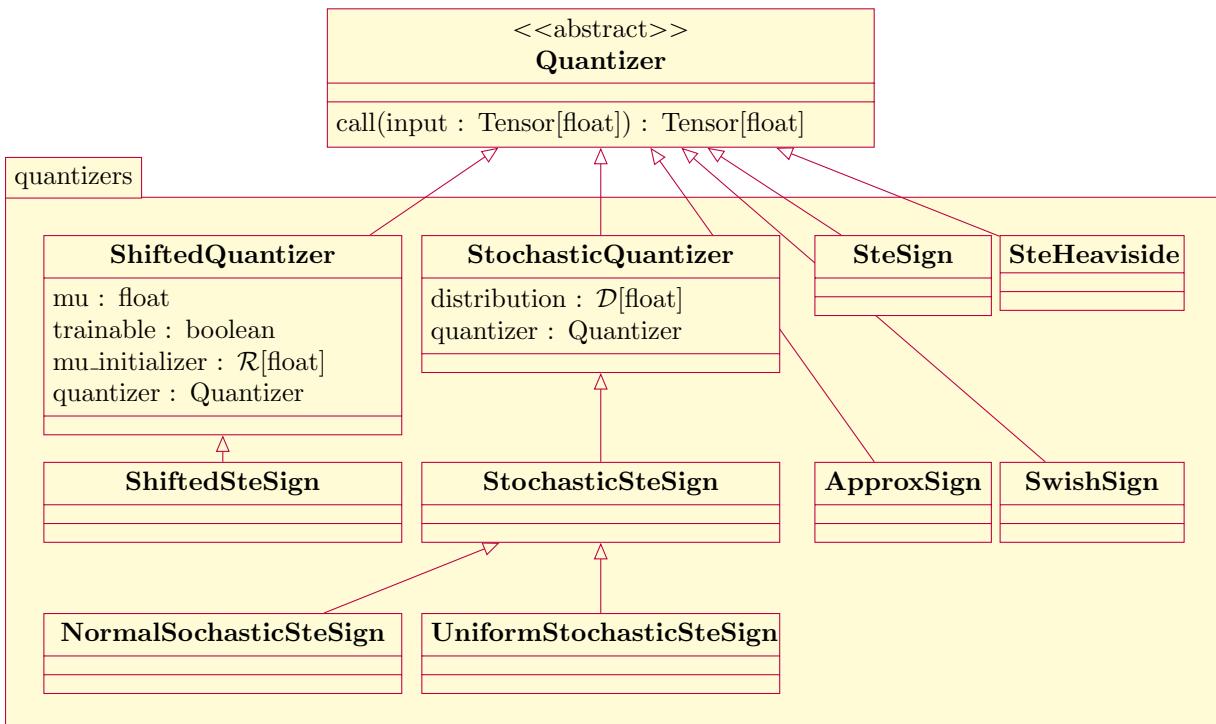


Figure 5.10: Diagramme de classe global du module `binaryflow.quantizers`

5.7 Régularisations

Les BNNs supportent les régularisations offertes par les réseaux de neurones classiques. De plus, ils supportent une autre forme de régularisation appelée régularisation de quantification, qui est un terme $\mathcal{L}_{\text{quantification}}$ ajouté à la fonction objective pour réduire l'erreur de quantification:

$$\mathcal{L} = \mathcal{L}_{\text{model}} + \mathcal{L}_{\text{quantification}} \quad (5.5)$$

Le terme $\mathcal{L}_{\text{quantification}}$ comporte une somme pondérée de:

5.7.1 Erreur de quantification des noeuds

C'est égal à

$$\mathcal{L}_{\text{weight}} = \sum_{a^{(l)} \text{quantified}} \|\tilde{a}^{(l)} - a\|_p^p \quad (5.6)$$

5.7.2 Erreur de quantification des poids

C'est égal à

$$\mathcal{L}_{\text{input}} = \sum_{\mathbf{W}^{(l)} \text{quantified}} \|\tilde{\mathbf{W}}^{(l)} - \mathbf{W}^{(l)}\|_p^p \quad (5.7)$$

5.7.3 Erreur de quantification de l'opération bilinéaire

C'est aussi l'erreur de quantification de $z^{(l)}$.

Cette erreur est l'erreur la plus importante car elle agit directement sur les performances du modèle. Elle est égale à:

$$\mathcal{L}_{\text{output}} = \sum_{\mathbf{z}^{(l)} \text{quantified}} \|\tilde{\mathbf{z}}^{(l)} - \mathbf{z}^{(l)}\|_p^p = \sum \|\tilde{\mathbf{W}}^{(l)} \star \tilde{a}^{(l-1)} - \mathbf{W}^{(l)} \star a^{(l-1)}\|_p^p \quad (5.8)$$

Chapter 6

Analyse

6.1 Introduction

Dans ce chapitre, on va résoudre 3 problèmes en utilisant les approches classiques d'apprentissages profonds, et aussi avec les réseaux de neurones binarisés. On va essentiellement résoudre:

1. Régression de $f : x \rightarrow 2x^2 + 3x + 2$ sur $[-4, 4]$
2. Classification des chiffres en exploitant le jeu de données MNIST
3. Classification des chiffres en exploitant le jeu de données Free Spoken Digits

6.2 Méthodologie adoptée

Dans les problèmes, qui se suivent, nous avons suivi la méthodologie CRSIP-DM[10].

6.2.1 CRISP-DM

CRISP-DM signifie Cross Industry Standard Process for Data Mining (CRISP-DM) est une méthode mise à l'épreuve sur le terrain permettant d'orienter les travaux d'exploration de données.

Cycle CRISP-DM

1. Connaissance du métier.
2. Connaissance des données.
3. Préparation des données.
4. Modélisation des données.
5. Évaluation.
6. Déploiement.

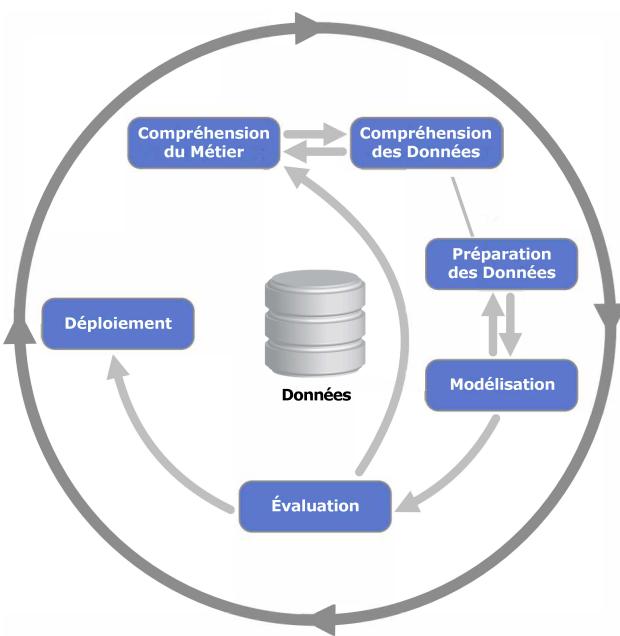


Figure 6.1: Méthodologie CRISP-DM

6.3 Régression de $f : x \rightarrow 2x^2 + 3x + 2$ sur $[-4, 4]$

6.3.1 Importance

Ce problème est fait artificiellement étudier la robustesse les BNNs implémentées.

6.3.2 Jeux de données

Dans cette problématique, le jeu de données est généré avec $n = 20000$ exemplaires:

- $x_1, \dots, x_n \sim \mathcal{U}(-4, 4)$
- $\epsilon_1, \dots, \epsilon_n \sim \mathcal{N}(0, 0.1)$
- y_1, \dots, y_n avec $y_i = f(x_i) + \epsilon_i \quad \forall i \in \{1, \dots, n\}$.

6.3.3 Modèle

Notre modèle \mathcal{M}_θ est un estimateur de f paramétrisé par $\theta \in S$. L'ensemble S va dépendre de type du modèle.

Dans notre cas, l'architecture du modèle est décrite par la figure ci-dessous.

Pour la binarisation, nous avons utilisé la fonction sign dans la propagation avant, et nous avons utilisé la méthode STE bornée sur $[-1, 1]$ pour la propagation en arrière.

6.3.4 Hypothèses

On a fait les hypothèses suivantes:

1. H1: x_1, \dots, x_n sont mutuellement indépendents
2. H2: Les bruits sont mutuellement indépendentes.
3. H3: Le model \mathcal{M} ne connaît sur la fonction f que $f(x_i) \approx y_i$, et il ne tient pas compte de l'erreur ϵ_i .

6.3.5 Problème Formel

On va chercher θ^* tel que:

$$\theta^* = \underset{\theta \in S}{\operatorname{argmin}} \|\mathbf{y} - \mathcal{M}_\theta(\mathbf{x})\|_2^2 = \underset{\theta \in S}{\operatorname{argmin}} \sum_{i=1}^n (y_i - \mathcal{M}_\theta(x_i))^2 \quad (6.1)$$

6.3.6 Entraînement

On a entraîné 3 types de modèles binaires:

- BinaryNet
- XnorNet
- ABCNet

Pour chacun, on a utilisé:

- L'optimiseur Adam, avec `learning_rate` = 10^{-2} et `decay` = 10^{-4}
- La fonction objective MSE comme décri dans l'équation (6.1)
- Taille de lot égale à 128
- Une limite de 30 époches. .

Nous avons aussi entraîné un MLP avec précision complète ayant la même architecture comme référence.

6.3.7 Performances de prédiction

On a tracé la courbe de chaque modèle en le comparant avec f :

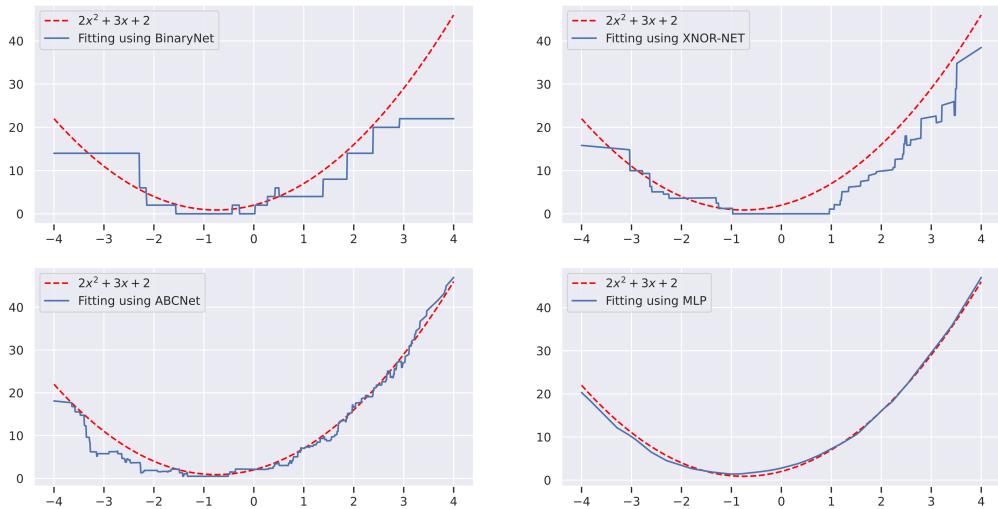


Figure 6.2: Courbe de chaque modèle

Pour mesurer la qualité de régression, nous avons utilisé 3 métriques:

Distance \mathcal{L}^∞

C'est la distance \mathcal{L}^∞ dans l'espace des fonctions continues $\mathcal{C}([-4, 4])$:

$$\mathcal{L}_{\text{test}}^\infty(y, \mathcal{M}_\theta) = \sup_{x \in [-4, 4]} |\mathcal{M}_\theta(x) - f(x)| \quad (6.2)$$

Distance \mathcal{L}^1

C'est la distance \mathcal{L}^1 dans l'espace $\mathcal{C}([-4, 4])$:

$$\mathcal{L}_{\text{test}}^1(y, \mathcal{M}_\theta) = \int_{-4}^4 |\mathcal{M}_\theta(x) - f(x)| dx \quad (6.3)$$

Distance \mathcal{L}^2

C'est la distance \mathcal{L}^2 dans l'espace $\mathcal{C}([-4, 4])$:

$$\mathcal{L}_{\text{test}}^2(y, \mathcal{M}_\theta) = \left(\int_{-4}^4 (\mathcal{M}_\theta(x) - f(x))^2 dx \right)^{\frac{1}{2}} \quad (6.4)$$

Dans la pratique, nous allons estimer les 2 intégrales en utilisant la méthode Romb avec $n = 2^{20} + 1$ points.

On a trouvé les résultats suivant:

Modèle	\mathcal{L}^∞	\mathcal{L}^1	\mathcal{L}^2 Carré	\mathcal{L}^2
BinaryNet	24.000	4.322	42.524	6.521
XNOR-NET	13.852	3.531	19.671	4.435
ABCNet	7.395	1.421	4.024	2.01
MLP	1.692	0.616	0.562	0.750

Dans le tableau ci-dessus, MLP est le modèle classique sans utilisation de binarisations, et il sert comme référence.

6.4 Classification MNIST

6.4.1 Introduction

MNIST[16] est un jeu de données des images de chiffres écrits à la main. Il admet:

- 60000 exemplaires pour l'entraînement
- 10000 exemplaires pour le test.

Les images sont de tailles 28×28 et en niveau de gris, dans lesquels les chiffres sont centrés.

Ce jeu de données sert comme un test de performance des modèles de Machine Learning.

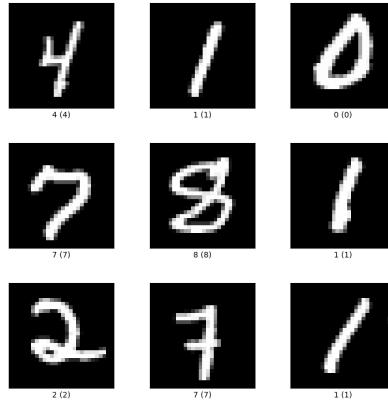


Figure 6.3: Des images de MNIST

6.4.2 Topologie

Nous avons utilisé l'architecture ci-dessous:

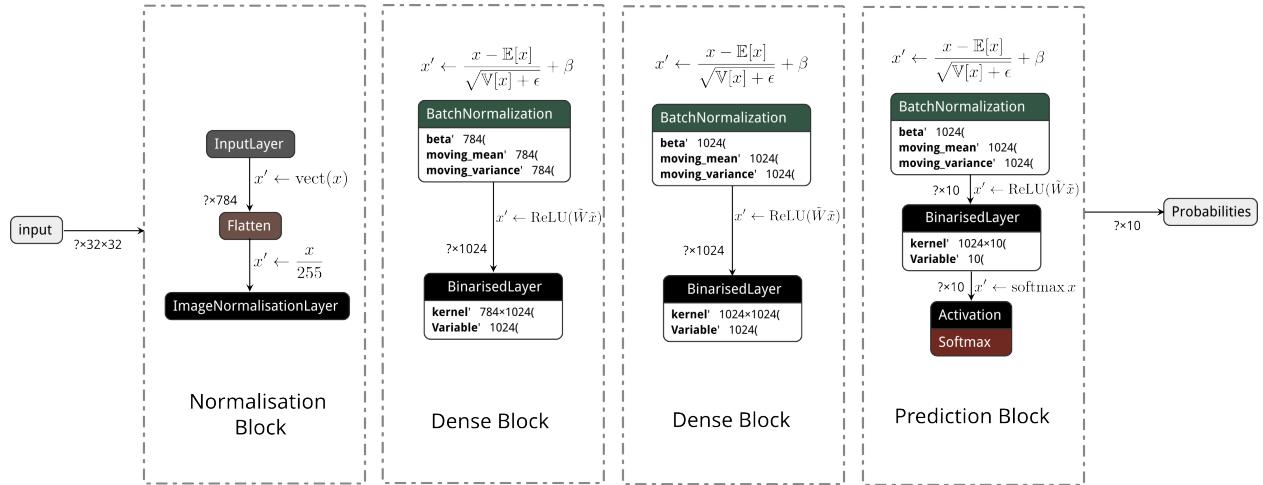


Figure 6.4: Architecture du modèle MNIST

BinarisedLayer désigne ici le type de couche dense binarisé qui est utilisé. Il s'agit de:

- QuantDense pour BinaryNet
- ScaledQuantDense pour XnorNet
- ABCDense pour ABCNet

Dans la figure, les dimensions de couche sont décrites explicitement avec leurs équations, et le point d'interrogation '?' indique que le modèle est indifférent à la taille de lot.

6.4.3 Modèle

On a utilisé 3 modèles pour l'architecture donnée:

- BinaryNet, avec:
 - Binarisation: sign avec STE
 - Activation: ReLU
- XnorNet, avec:
 - Binarisation: sign avec STE
 - Activation: ReLU
 - α et β sont calculés par produit scalaire.
 - Le facteur α est entraînable, initialisé à (4.12)
 - Le facteur β est calculé, initialisé à (4.12)
- ABCNet avec:
 - Binarisation: sign décalée avec STE.
 - Activation: ReLU.
 - $n = 3$ binarisations des poids, suivant
 - $m = 1$ binarisations des noeuds, suivant
 - μ_i et κ_j sont entraînables
 - α_i et β_j sont calculés par produit scalaire.

6.4.4 Entraînement

On a entraîné 3 types de modèles binaires:

- BinaryNet
- XnorNet
- ABCNet

Pour chacun, on a utilisé:

- L'optimiseur Adam, avec `learning_rate = 10-2` et `decay = 10-4`.
- La fonction objective MSE.

- Taille de lot égale à 96.
- Une limite de 30 époches.

Nous avons aussi entraîné un MLP avec précision complète ayant la même architecture comme référence.

6.4.5 Performances de prédictions

6.4.6 Performance mémoire

les performances mémoire sont mesurées avec la taille totale des paramètres, qui donnent une approximation de l'utilisation mémoire du modèle à son déploiement.

Nous avons fait la comparaison de 2 versions de chaque modèle:

- Les paramètres (non-binarisés) sont en float à 32 bits.
- Les paramètres (non-binarisés) sont en float à 8 bits.

Comparaison Float 32

Les paramètres de type float sont naturellement encodés en 64 bits dans TensorFlow. En apportant des binarisations, on va avoir le tableau suivant

Modèle	Nombre de paramètres binarisés	Nombre de paramètres non-binarisés	Taille de mémoire KB T	Taux de compression $\tau = \frac{T_{\text{MLP}}}{T_{\text{Model}}}$
MLP	0	1'869'354	7302.16	1
BinaryNet	1'861'632	5'664	249.38	29.28
XnorNet	1'861'632	7'722	257.41	28.36
ABCNet	5'584'896	11'838	727.99	10.03

Table 6.1: Comparaison entre les modèles en utilisant Float32

Ce tableau nous montre une grande compression de mémoire en utilisant les BNNs, surtout les modèles BinaryNet et XnorNet qui sont près de la limite théorique qui est égale à 32 qu'on a trouvé dans 3.4.5

Pour ABCNet, on remarque que le taux de compression est inversement proportionnel aux nombre de binarisations des poids, qui est dans notre cas égal à 3.

En général, pour un ABCNet à n binarisations de poids et m binarisations des noeuds, le gain est asymptotiquement en $\mathcal{O}(\frac{1}{n})$

Comparaison Float 8

On peut aussi compresser les paramètres float en utilisant Float 8 (Minifloat) dans TensorFlow. Ainsi, on va avoir le tableau suivant:

Modèle	Nombre de paramètres binarisés	Nombre de paramètres non-binarisés	Taille mémoire en KB T	Taux de compression $\tau = \frac{T_{\text{MLP}}}{T_{\text{Model}}}$
MLP	0	1'869'354	7302.16	1
BinaryNet	1'861'632	5'664	249.38	29.28
XnorNet	1'861'632	7'722	257.41	28.36
ABCNet	5'584'896	11'838	727.99	10.03

Table 6.2: Comparaison entre les modèles en utilisant Float8

Quantification de Float 32 à Float 8

Avec les deux tableaux précédents, on peut étudier l'apport de la réduction de précision de float à minifloat pour les 4 modèles considérés en terme de mémoire.

Modèle	Taille avec Float 32 T_{32}	Taille avec Float 8 T_8	Rapport de Taille $\tau = \frac{T_8}{T_{32}}$
MLP	7302.16	1825.54	0.25
BinaryNet	249.38	232.78	0.9334
XnorNet	257.41	234.79	0.9121
ABCNet	727.99	693.31	0.9523

Table 6.3: Comparaison entre les modèles en utilisant Float8

On remarque que contrairement aux modèles à précision complète, l'utilisation de Float 8 ne va pas donner une bonne compression du modèle pour les BNNs, et c'est expliqué par le fait que le nombres de paramètres de types Float 32 dans un BNN est généralement négligeable au nombres de paramètres binarisés, ainsi la compression des paramètres Float 32 ne sera pas très utile dans ce cas.

6.4.7 Performance temps

les performance temps sonts mesurée avec le nombre d'instructions multiply & accumulate (MAC) nécessaires pour chaque modèle à son déploiement.

Modèle	Nombre de MACs binarisés N_B	Nombre de MACs non-binarisés N_F	Instructions équivalentes: $I = \frac{N_B}{64} + N_F$	Taux de MACs binarisés $\alpha = \frac{N_B}{N_B + N_F}$	Taux de gain: $\tau = \frac{I_{\text{MLP}}}{I_{\text{Model}}}$
MLP	0	1'861'632	1'861'632	0	1
BinaryNet	1'861'632	2058	29'088	1	64
XnorNet	1'861'632	7'722	31'064	0.9989	59.92
ABCNet	5'584'896	11'838	93'438	0.9989	19.92

Table 6.4: Nombre de MACs des modèles

6.5 Classification Free Spoken Digits

6.5.1 Importance

Dans la littérature, les BNNs utilisés principalement avec les jeux de données images. Ainsi, notre objectif est l'utilisation d'un BNN pour résoudre les problèmes de traitement des signaux à faible complexité.

Nous avons choisis Free Spoken Digits[12] comme une preuve de concept de l'applicabilité des BNNs dans ce genre de problèmes. De plus, nous allons utiliser l'expertise de dB Sense pour extraire intelligemment les informations utiles d'une parole afin de construire un classificateur performant et à faible complexité.

6.5.2 Introduction

Free Spoken Digits[12] (FSD) est un jeu de données audio qui consiste d'un ensemble de fichiers wav à une fréquence d'échantillonage de 8 kHz. Les fichiers sont coupés pour minimiser le silence.

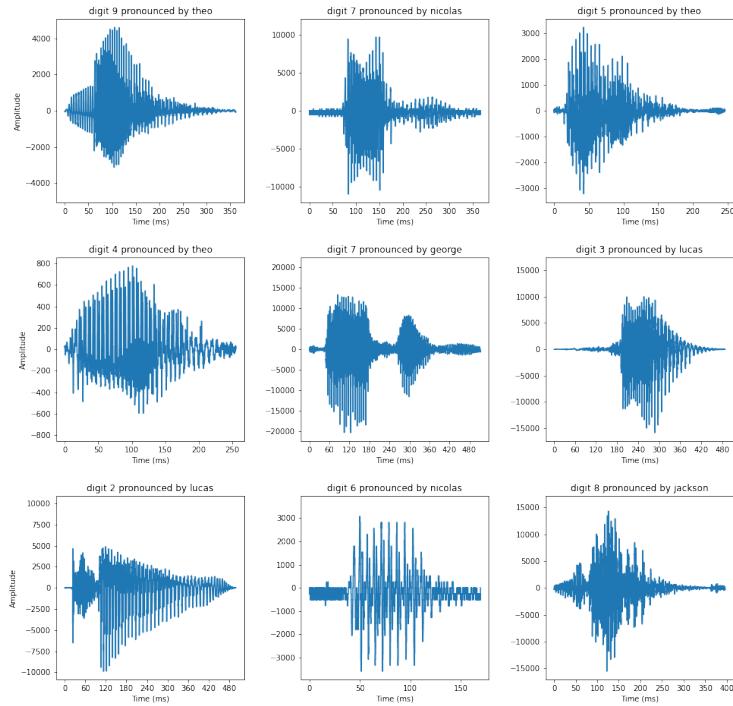


Figure 6.5: Représentation des signaux de FSD

6.5.3 Structure

FSD est enregistré à l'aide d'un microphone monophonique, et il admet 6 orateurs qui ont chacun 50 enregistrements par chiffre.

En total, chaque orateur admet 500 enregistrements, et donc ce jeu de données admet 3000 fichiers audio.

De plus, chaque enregistrement admet un nom significatif de la forme `digit_person_id.wav` avec:

- `digit` est le chiffre prononcé dans l'enregistrement.
- `person` est le nom de l'orateur.
- `id` est l'identificateur de l'enregistrement, sachant la personne et le chiffre, appartenant à l'ensemble $\{0, \dots, 49\}$

6.5.4 Analyse

Avant la création d'un modèle, nous devons analyser les données pour comprendre leur nature et caractéristiques.

Nous allons étudier:

- La durée de chaque signal
- L'amplitude maximale en valeur absolue de chaque signal

Durée du signal

Notre modèle \mathcal{M} va dépendre d'une fenêtre de taille fixe T . Pour bien choisir cette fenêtre, nous devons étudier la distribution statistique des durées dans le jeu de données.

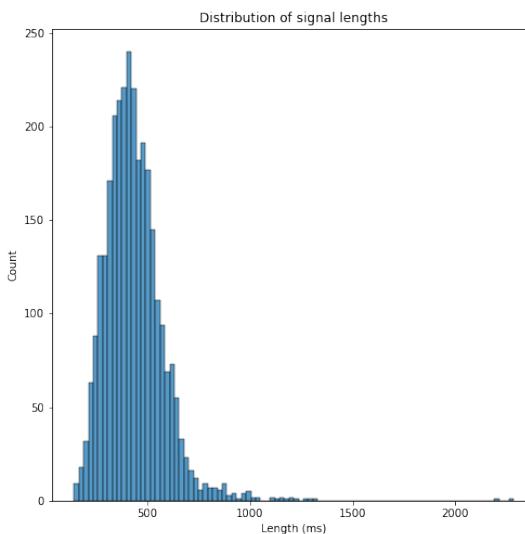


Figure 6.6: Histogramme représentant la durée des signaux

La figure 6.6 nous montre que la durée de la majorité des fichiers audio est inférieure à 1000 ms avec quelques exemplaires aberrants ayant une durée supérieure à 2200 ms
 En détaillant notre analyse par personne, nous allons remarquer que ces histogrammes varient par personnes.

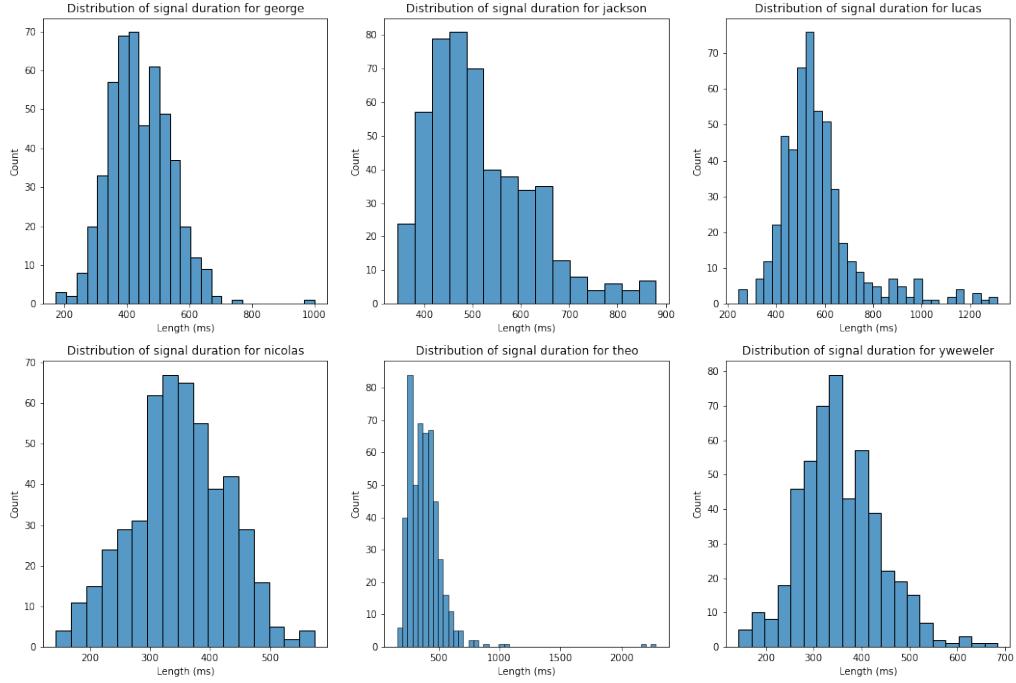


Figure 6.7: Histogramme de la durée des signaux de chaque locuteur

La figure 6.7 montre une large différence entre les histogrammes par personne, et c'est tout à fait normal puisque les expériences ne sont pas fait dans les mêmes conditions.
 Le tableau suivant va donner quelques statistiques par personne.

Personne	Nombre	Moyenne	Std	Minimum	25%	50%	75%	Maximum
George	500	441.718	97.588	174.875	374.219	430.1250	506.844	999.875
Jackson	500	516.460	107.457	344.125	436.344	491.688	581.31250	879.750
Lucas	500	574.211	161.731	245.625	479.906	541.8750	616.875	1313.000
Nicolas	500	349.188	79.630	143.625	299.719	348.688	402.813	574.250
Theo	500	388.862	168.451	161.000	282.906	369.625	449.875	2282.750
Yweweler	500	354.168	83.247	143.500	299.156	347.000	406.625	683.750

Table 6.5: Statistique des durées par personne

À partir des figures 6.6, 6.7 et le tableau 6.5, nous avons conclu qu'une fenêtre $T \in [250 \text{ ms}, 750 \text{ ms}]$ est suffisante.

Nous avons finalement choisis $T = 250 \text{ ms}$ pour minimiser la complexité du modèle.

Amplitude maximale du signal

Notre modèle \mathcal{M} doit mettre en considération les variations des intensités sonores entre les observations.

Pour cela, nous allons étudier la distribution de l'amplitude maximale du signal.

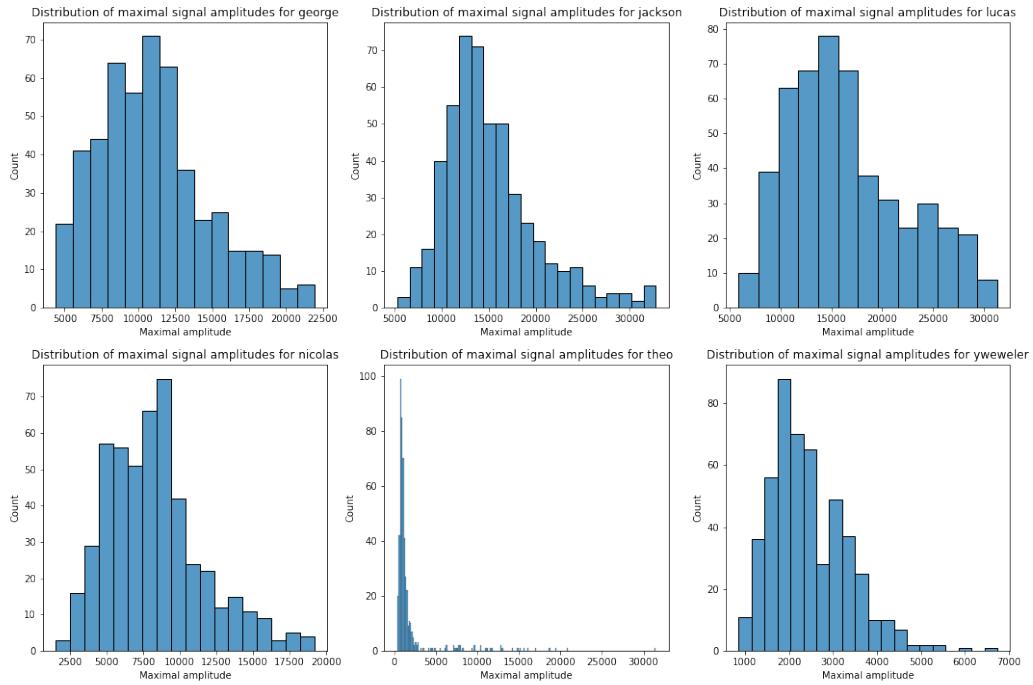


Figure 6.8: Histogrammes représentant l'intensité maximale des signaux pour chaque locuteur

Le tableau suivant va donner quelques statistiques par personne.

Personne	Nombre	Moyenne	Std	Minimum	25%	50%	75%	Maximum
George	500	11024.848	3741.403	4408.000	8228.250	10683.000	13059.500	21942.000
Jackson	500	15129.334	4941.939	5326.000	11872.500	14076.500	17263.250	32767.000
Lucas	500	16557.842	5797.445	5880.000	12060.500	15519.500	20200.000	31343.000
Nicolas	500	8288.256	3295.527	1536.000	5888.000	7936.000	9984.000	19200.000
Theo	500	1836.706	3117.099	343.000	744.750	967.500	1356.750	31504.000
Yweweler	500	2448.588	882.629	854.000	1828.500	2294.000	3007.250	6742.000

Table 6.6: Statistique des durées par personne

6.5.5 Prétraitement

Après l'étude du jeu de données, nous allons appliquer quelques traitements et transformations pour aboutir à une représentation utilisable

Recadrement Aléatoire

Dans la section 6.5.4, nous avons analysé la distribution des durées des fichiers audios, et puis nous avons choisi une fenêtre de longueur 250 ms. En effet, puisque la fréquence est 8 kHz, cette longueur correspond à un échantillon de taille $K = 2000$

En notant L la taille de l'échantillon audio, dans le recadrement, deux cas se présentent:

1. $L \geq K$: Dans ce cas, on choisit la fenêtre commençant à r avec r une variable aléatoire suivant la distribution uniforme $\mathcal{U}(0, L - K)$
2. $L < K$: Dans ce cas, on fait un padding à gauche de taille r et un padding à droite de taille $K - L - r$, avec r une variable aléatoire suivant la distribution uniforme $\mathcal{U}(0, K - L)$

Augmentation stochastique

Puisque le jeu de données est un peu petit, nous avons fait 3 itérations de recadrement aléatoire pour chaque signal audio.

Ainsi, nous allons trouver 9000 exemplaires dans le nouveau jeu de données.

Algorithm 2 Data augmentation with random crop algorithm

Require: K the size of the window

Require: $p \in \mathbb{N}^*$ the number of random crops per input

Require: X, Y dataset with size n

Ensure: X', Y' dataset with size pn

```
 $X' \leftarrow []$ 
 $y' \leftarrow []$ 
for  $x, y \in X, Y$  do
    for  $i \in \{1, \dots, p\}$  do
         $L \leftarrow \text{length}(X_i)$ 
        if  $L \geq K$  then
             $r \sim \mathcal{U}(0, L - K)$ 
             $x' \leftarrow x[r, \dots, r + K]$ 
        else
             $r \sim \mathcal{U}(0, K - L)$ 
             $a \leftarrow \text{zeros}(r)$ 
             $b \leftarrow \text{zeros}(K - L - r)$ 
             $x' \leftarrow \text{concat}(a, x, b)$ 
        end if
         $X'.append(x')$ 
         $y'.append(y)$ 
    end for
end for
```

MFCC [22]

Le Mel Frequency Cepstrum Coefficients (MFCC) est une méthode très utilisée la reconnaissance du locuteur.

Elle est également populaire en raison de l'efficacité schémas de calcul disponibles pour celui-ci et sa robustesse.

Elle peut être résumée par les étapes suivantes:

1. Une fenétrage du signal
2. Un padding si nécessaire
3. Calcul de la DFT
4. Calcul de la puissance spectrale
5. Filtrage de la puissance spectrale avec p filtres triangulaires
6. Calcul du logarithme de la puissance filtrée
7. Calcul de la DCT du logarithme trouvé

Dans notre cas nous avons utilisé la fonction `librosa.feature.mfcc` pour générer le MFCC à partir du signal audio en utilisant les hyperparamètres suivant:

- La fréquence `sr=8000`
- La norme orthogonale
- Une taille de la FFT `n_fft=256`
- Le décalage en indice entre 2 MFCC successifs: `hop_length=64`
- Le nombre de coefficients mel cepstrum générés: `n_mfcc=32`

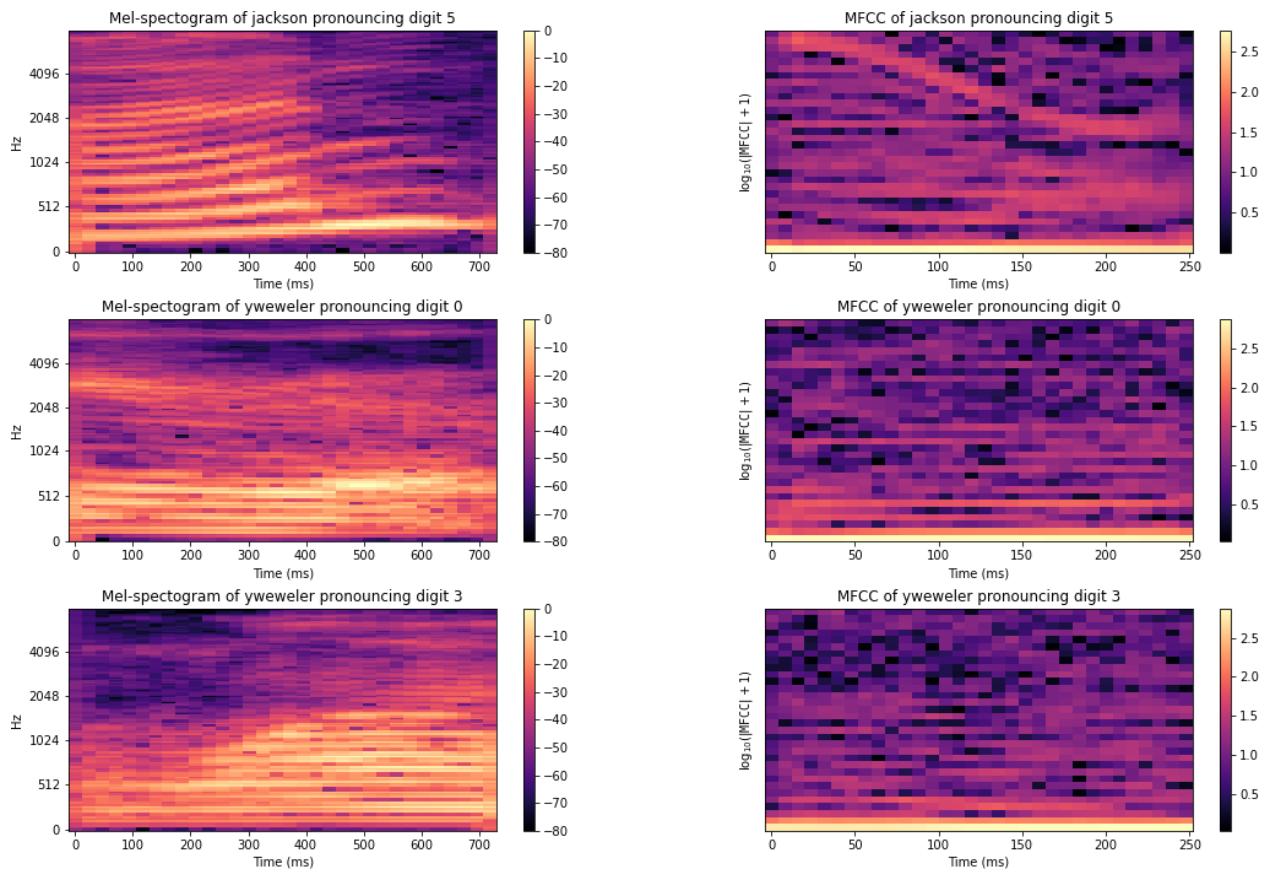


Figure 6.9: Exemple des spectrogrammes Mels et des MFCC

Normalisation des MFCC

Même si les MFCC donnent une bonne représentation d'une parole. Cette représentation doit être indépendante de l'intensité sonore du signal, car le chiffre prononcé va être toujours le même en changeant cette intensité.

Pour cela, nous avons apporté une normalisation dans l'axe des fréquences.

6.5.6 Topologie

La topologie utilisée dans ce modèle est une variante de

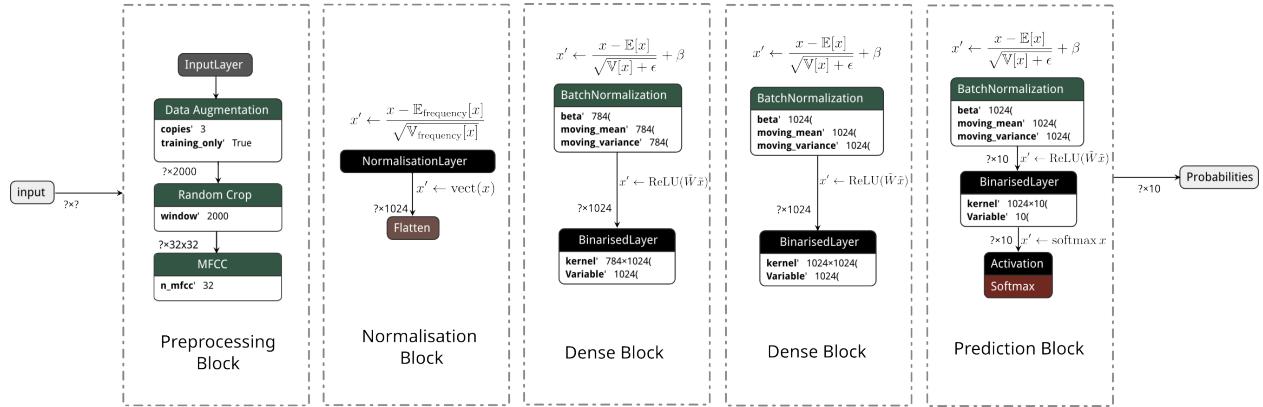


Figure 6.10: Architecture du modèle FSD

6.5.7 Modèle

Nous avons utilisé les mêmes modèles que la section 6.4.3.

6.5.8 Entraînement

Nous avons suivi les mêmes conditions que 6.4.4 dans l'entraînement de nos modèles.

6.5.9 Performances de prédictions

Nous avons suivi les performances des modèles après chaque époque. Les métriques mesurés sont:

- Valeur de la fonction de coût dans le jeu de données d'entraînement
- Précision de prédiction dans le jeu de données test.

Fonction de coût

Dans la phase d'apprentissage, la fonction de coût donne un aperçu sur la capacité du modèle à apprendre.

En effet, dans notre cas, le comportement de la fonction de coût contre chacune des modèles est similaires. Et de plus, elle est presque dans le même ordre de grandeur, comme le montre la figure 6.11 ci-dessous.

On peut conclure, que dans la phase d'apprentissage, que même si les BNNs sont théoriquement inférieurs en performances de prédictions que son contrepart classique. Il peuvent avoir des performances proches du modèle classique correspondant.

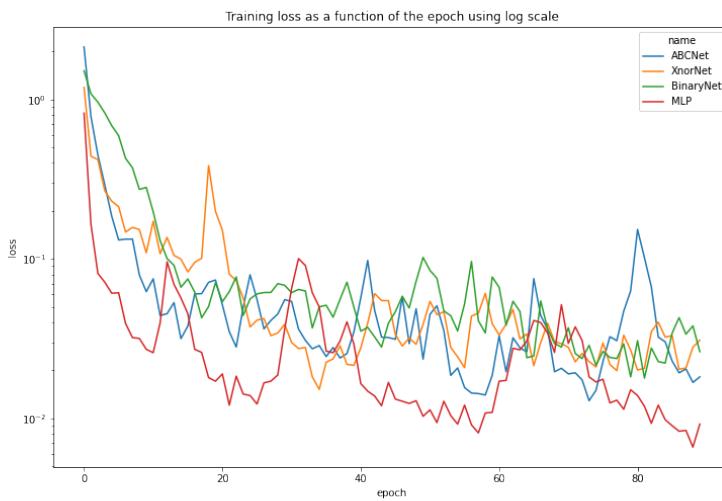


Figure 6.11: La fonction de coût en fonction de l'époque par modèle

Précision de prédiction

Le métrique précision de prédiction donne le taux d'exemplaires correctement classifiés.

Dans la phase de test, ce métrique donne un aperçu sur la capacité du modèle à généraliser, car il n'a pas vu ses données dans sa phase d'entraînement.

Dans notre cas, nous avons comparer le taux maximal de précision¹

En effet, comme le montre la figure 6.12 ci-dessous, les modèles BinaryNet et XnorNet ont réussi à dépasser le 92%, et ABCNet avait même surperformé MLP durant 50 époques.

¹Dans ce contexte, le taux maximal de précision pour une époque est le maximum des taux de précisions du modèle dès le début de son entraînement jusqu'à cette époque

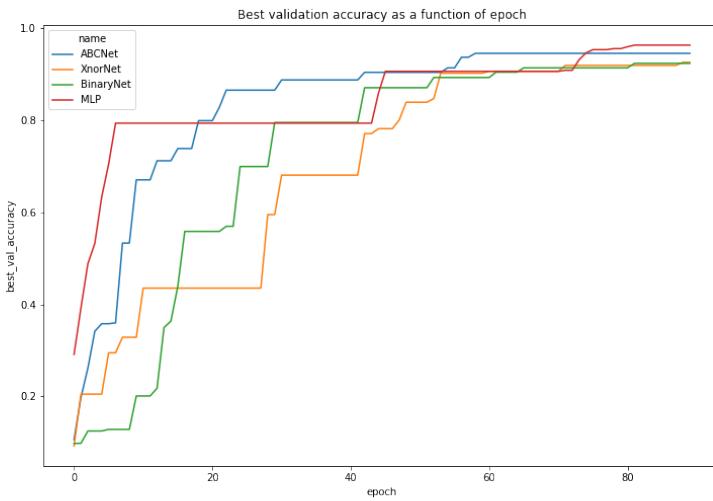


Figure 6.12: Taux maximal de précision en fonction de l'époque par modèle

Les performances trouvés sont:

Modèle	Taux de Précision
MLP	0.9627
BinaryNet	0.9231
XnorNet	0.9253
ABCNet	0.9449

Table 6.7: Taux maximal de précision

6.5.10 Coût d'entraînement

Introduction

L'entraînement des réseaux de neurones engendre une utilisation d'énergie qui peut être importante, et donc peut laisser une empreinte de CO₂ appelée code carbone. Le tableau 2.1 dans le premier chapitre montre les coûts de quelques modèles connues.

Dans cette section, nous allons estimer le coût énergétique et le code carbone des BNNs entraînés avec le jeux de données FSD.

Le code carbone

Le code carbone est un métrique représentant la quantité de carbone produite.

Par rapport à un algorithme bien déterminé, Il signifie la quantité du carbone produite durant son exécution.

Le calcul exact de ce métrique est difficile, pour cela on va l'estimer en utilisant **comet.ml**. C'est une bibliothèque de Python qui calcule automatiquement le code carbone quelques étapes:

- Il prend les puissances dissipées par le CPU $(c_i)_{i \in \{0, \dots, n-1\}}$, RAM $(r_i)_{i \in \{0, \dots, n-1\}}$ et GPU $(g_i)_{i \in \{0, \dots, n-1\}}$ dans des instants $(t_i)_{i \in \{0, \dots, n\}}$, avec n le nombre d'échantillons en excluant l'échantillon à l'instant $t_0 = 0$.
- Il calcule la masse de carbone générée à l'aide de la formule suivante

$$M(CO_2) = C \cdot P$$
$$P = \sum_{i=0}^{n-1} (\Delta t)_i (c_i + r_i + g_i)$$

Avec:

P l'énergie totale dissipée en kWh.

C l'intensité de carbone par rapport à l'énergie consommée, en kg/kWh

$M(CO_2)$ la masse de carbone générée, en kg

$(\Delta t)_i = t_{i+1} - t_i$

En effet, le paramètre C est difficile à estimer, et **comet.ml** lui fait automatiquement une estimation selon le pays, la région et l'infrastructure utilisée.

Code carbone des modèles

Dans l'entraînement de chaque modèle, le tracking de l'émission commence juste avant l'exécution `fit` et se termine juste après son exécution.

La figure ci-dessous montre une comparaison des énergies totales consommée pour chaque modèle durant l'entraînement.

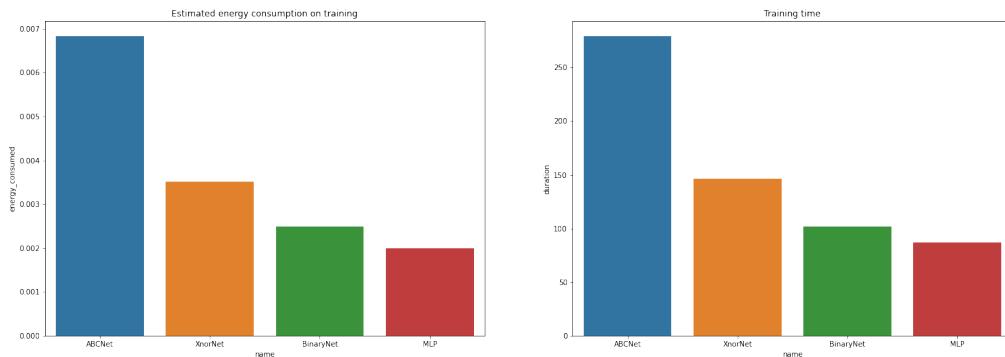


Figure 6.13: Énergie consommée par modèle durant l'entraînement

Finalement, à partir de l'énergie totale consommée, **comet.ml** calcule la masse de CO_2 potentiellement générée pour chaque modèle qu'on va montrer dans la figure ci-dessous:

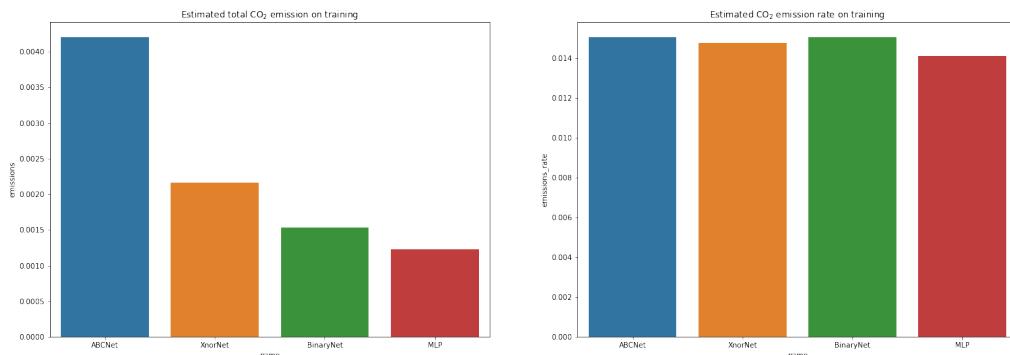


Figure 6.14: Masse de CO_2 dissipée durant l'entraînement

Conclusion

Durant ce stage, nous avons étudier les BNNs, implémenté une bibliothèque de BNNs basée sur tensorflow et larq qu'on a nommé **binaryflow**.

Dans le premier chapitre, nous avons présenté **dB Sense** et ses activités.

Dans le deuxième chapitre, nous avons présenté le problème de la grande complexité, introduit le concept des BNNs et proposé **binaryflow** comme notre solution pour les BNNs

Dans le troisième chapiter, nous avons formalisé les BNNs, et nous avons décris leurs optimisations possible, et les problèmes rencontrés dans leur implémentation.

Dans le quatrième chapitre, nous avons présenté des modèles BNNs, chacun en dérivant ses formules

Dans le cinquième chapitre, nous avons donné notre implémentation de **binaryflow** en jusitifiant les paradigmes utilisés

Dans le sixième chapitre, nous avons analysé 3 jeux de données en implémentant des modèles BNNs pour chacune de ces 3 jeux de données, et en comparant les performances de prédiction et la complexité temps et mémoire des modèles entraînés.

Notre travail n'est qu'une petite introduction des BNNs. En effet, la liste des BNNs proposés dans la littérature est très vaste, et il existe plusieurs autres approches pour faciliter l'entraînement et l'interférence des BNNs que nous n'avons pas considéré vu les contraintes de stage, y parmi:

1. Les méthodes d'optimisations discrètes
2. Les optimiseurs dédiés au BNNs
3. Les binarisations entraînables
4. Les méthodes ensemblistes pour régulaliser les BNNs

De plus, nous avons réussi à vérifier l'optimisation de la multiplication matricielle (L'exécution est parfois 30 fois plus rapide), mais nous n'avons pas pu intégrer cette optimisation aux modèles tensorflow. Et malgré que larq supporte lui même un déploiement optimisé, nous n'avons pas aussi pu l'exploiter puisque ce déploiement ne supporte que les processeurs ARMv8, et nous utilisons une machine avec un processeur d'architecture x86-64.

Finalement, nous avons fait une petite intégration du code carbone comme une mesure de coût d'entraînement. Une fois le problème de déploiement est résolu, nous recommanderions l'utilisation de ce même métrique pour estimer le coût d'interférence qui va justifier l'utilisation des BNNs.

Bibliography

- [1] Yoshua Bengio, Nicholas Léonard, and Aaron C. Courville. “Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation”. In: *CoRR* abs/1308.3432 (2013). arXiv: 1308.3432. URL: <http://arxiv.org/abs/1308.3432>.
- [2] Corinna Cortes and Vladimir Vapnik. “Support-vector networks”. In: *Machine learning* 20.3 (1995), pp. 273–297.
- [3] Matthieu Courbariaux and Yoshua Bengio. “BinaryNet: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1”. In: *CoRR* abs/1602.02830 (2016). arXiv: 1602.02830. URL: <http://arxiv.org/abs/1602.02830>.
- [4] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. “BinaryConnect: Training Deep Neural Networks with binary weights during propagations”. In: *CoRR* abs/1511.00363 (2015). arXiv: 1511.00363. URL: <http://arxiv.org/abs/1511.00363>.
- [5] Elizabeth Gibney. “Self-taught AI is best yet at strategy game Go”. In: *Nature* (Oct. 2017). DOI: 10.1038/nature.2017.22858.
- [6] *Go Player Finally Defeats Google’s AlphaGo After 3 Losses*. 14 Mar 2016.
- [7] Citu Group. *What is the carbon footprint of a house?* URL: <https://citu.co.uk/citu-live/what-is-the-carbon-footprint-of-a-house>.
- [8] Koen Helwegen et al. “Latent Weights Do Not Exist: Rethinking Binarized Neural Network Optimization”. In: *CoRR* abs/1906.02107 (2019). arXiv: 1906.02107. URL: <http://arxiv.org/abs/1906.02107>.
- [9] Sepp Hochreiter and Jürgen Schmidhuber. “Long short-term memory”. In: *Neural computation* 9.8 (1997), pp. 1735–1780.
- [10] IBM. *Présentation générale de CRISP-DM*. <https://www.ibm.com/docs/fr/spss-modeler/saas?topic=dm-crisp-help-overview>. 2017.
- [11] A.G. Ivakhnenko et al. *Cybernetics and Forecasting Techniques*. Modern analytic and computational methods in science and mathematics. American Elsevier Publishing Company, 1967. ISBN: 9780444000200. URL: <https://books.google.tn/books?id=rGFgAAAAMAAJ>.
- [12] Zohar Jackson. *Free Spoken Digits Dataset*. <https://github.com/Jakobovski/free-spoken-digit-dataset>. 2019.
- [13] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *CoRR* abs/1412.6980 (2015).
- [14] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “ImageNet Classification with Deep Convolutional Neural Networks”. In: *Advances in Neural Information Processing Systems*. Ed. by F. Pereira et al. Vol. 25. Curran Associates, Inc., 2012. URL: <https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf>.

- [15] Y. Lecun et al. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324. DOI: 10.1109/5.726791.
- [16] Yann LeCun. *THE MNIST DATABASE*. <http://yann.lecun.com/exdb/mnist>. 1998.
- [17] Xiaofan Lin, Cong Zhao, and Wei Pan. “Towards Accurate Binary Convolutional Neural Network”. In: *CoRR* abs/1711.11294 (2017). arXiv: 1711.11294. URL: <http://arxiv.org/abs/1711.11294>.
- [18] Seppo Linnainmaa. “Taylor Expansion of the Accumulated Rounding Error”. In: *BIT* 16.2 (1976), 146–160. ISSN: 0006-3835. DOI: 10.1007/BF01931367. URL: <https://doi.org/10.1007/BF01931367>.
- [19] Zechun Liu et al. “Bi-Real Net: Enhancing the Performance of 1-bit CNNs With Improved Representational Capability and Advanced Training Algorithm”. In: *CoRR* abs/1808.00278 (2018). arXiv: 1808.00278. URL: <http://arxiv.org/abs/1808.00278>.
- [20] Kim Martineau. *Shrinking deep learning’s carbon footprint*. 7Aug 2021. URL: <https://news.mit.edu/2020/shrinking-deep-learning-carbon-footprint-0807>.
- [21] Mohammad Rastegari et al. “XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks”. In: *CoRR* abs/1603.05279 (2016). arXiv: 1603.05279. URL: <http://arxiv.org/abs/1603.05279>.
- [22] Md. Sahidullah and Goutam Saha. “Design, analysis and experimental evaluation of block based transformation in MFCC computation for speaker recognition”. In: *Speech Communication* 54.4 (2012), pp. 543–565. ISSN: 0167-6393. DOI: <https://doi.org/10.1016/j.specom.2011.11.004>. URL: <https://www.sciencedirect.com/science/article/pii/S0167639311001622>.
- [23] Emma Strubell, Ananya Ganesh, and Andrew McCallum. “Energy and Policy Considerations for Deep Learning in NLP”. In: *CoRR* abs/1906.02243 (2019). arXiv: 1906.02243. URL: <http://arxiv.org/abs/1906.02243>.
- [24] Tim Dettmers. *Deep Learning in a Nutshell: History and Training*. <https://developer.nvidia.com/blog/deep-learning-nutshell-history-training/>. 16 Dec. 2015.
- [25] Aimee Wynsberghe. “Sustainable AI: AI for sustainability and the sustainability of AI”. In: *AI and Ethics* 1 (Feb. 2021). DOI: 10.1007/s43681-021-00043-6.
- [26] Chunyu Yuan and Sos S. Agaian. “A comprehensive review of Binary Neural Network”. In: *CoRR* abs/2110.06804 (2021), pp. 3–4. arXiv: 2110.06804. URL: <https://arxiv.org/abs/2110.06804>.
- [27] Chunyu Yuan and Sos S. Agaian. “A comprehensive review of Binary Neural Network”. In: *CoRR* abs/2110.06804 (2021). arXiv: 2110.06804. URL: <https://arxiv.org/abs/2110.06804>.