Sécurité des implémentations cryptographiques face aux attaques par canaux auxiliaires basées sur des techniques d'apprentissage profond

Damien Robissout

N° d'ordre NNT : 2022LYSES005

THÈSE de DOCTORAT DE L'UNIVERSITÉ DE LYON

opérée au sein du
Laboratoire Hubert Curien

École Doctorale N° 488
Sciences Ingénierie Santé

Spécialité de doctorat :
Informatique

Soutenue publiquement le 10/02/2022, par :

Damien Robissout

Sécurité des implémentations cryptographiques face aux attaques par canaux auxiliaires basées sur des techniques d'apprentissage profond

---

Devant le jury composé de :

Fouque, Pierre-Alain	PR	Université Rennes 1	Rapporteur
Guilley, Sylvain	PR	Télécom Paris	Rapporteur
Fromont, Elisa	PR	IRISA Rennes 1	Examinatrice
Cagli, Eleonora	PhD	CEA LETI	Examinatrice
Grand, Michael	PhD	Serma	Examinateur
Bossuet, Lilian	PR	Université Jean Monnet	Directeur de thèse
Habrard, Amaury	PR	Université Jean Monnet	Directeur de thèse

Remerciements

Je souhaite tout d'abord prendre le temps d'adresser un remerciement général à toutes les personnes qui, tout au long de mon parcours, du lycée aux études supérieures, m'ont aidé et soutenu et sans qui je ne me serais probablement pas lancé dans cette thèse.

Je vais maintenant faire des remerciements personnels par ordre chronologique pour voir comment l'accumulation de petits coups de pouce à mener à l'écriture de cette thèse ! Je commence rapidement avec mes professeurs de maths au lycée qui m'ont donnée le goût des mathématiques ce qui m'a poussé dans la voie universitaire et également Fabien Pazuki qui m'a fait confiance et fait rentrer dans le parcours international de l'université de Bordeaux. C'est là que j'ai pu rencontrer mes camarades Simon et Louise, entre autres, avec qui j'ai partagé de nombreux cours, des mathématiques à l'épistémologie. Je remercie par la même occasion Joseph et Claudia qui m'ont énormément aidé à mon arrivée en Californie et durant l'année que j'y ai passé.

Je souhaite maintenant dire merci aux camarades de mon master. Ce sont ces années de master qui m'ont définitivement poussé vers la thèse même si l'idée d'en faire une était déjà présente dans ma tête depuis le lycée. Je remercie donc de nouveau Simon, Quentin, Ida, Corentin et Antoine pour leur aide que ce soit en programmation, pour comprendre d'obscures preuves de crypto ou lors de nos projets qui nous ont donnés un avant goût des challenges que l'on rencontrerait par la suite. Je remercie également les professeurs du master qui nous ont incités à suivre le cours de cartes à puce et plus particulièrement Alberto qui nous a insufflés un peu de sa passion pour les attaques par canaux auxiliaires et la sécurité matérielle.

Je remercie par la même occasion Vincent qui m'a fait confiance en m'accordant un stage de six mois et m'a donné un aperçu de ce que pouvait être la recherche dans ce domaine et également Marc d'avoir supporter ma présence dans son bureau pendant tout ce temps là.

Un grand merci à toute l'équipe SAS du CMP de Gardanne, en particulier Simon, Pierre-Alain et bien sûr Romain qui m'ont fait découvrir tous les aspects pratiques des attaques hardware. Sans oublier Brice à qui je dois beaucoup et qui m'a aiguillé vers cette présente thèse. Je le remercie également pour toute l'aide qu'il m'a fourni tout au long de la thèse et pour m'avoir aidé à me maintenir en forme avec nos courses du midi !

Je remercie maintenant tous mes camarades doctorants avec qui j'ai échangé durant ma thèse : Mehdi et Elie pour nos discussions, Oto pour son accueil, le coach Ugo, Tanguy pour ses conseils de Machine Learning, Boly d'avoir été Père Noël avant moi le temps d'une course.

Un grand merci à tous les membres de l'équipe SESAM : Gabriel, avec qui j'ai partagé mes débuts et dont les discussions que l'on a pu avoir m'ont fait comprendre beaucoup de

choses, Florent, qui m'a aidé à gérer mes cours, Pierre-Louis, qui avait toujours les bons mots de *soutien* que ce soit pour la thèse ou pour la course, Nathalie, qui m'a aidé à de nombreuses reprises à résoudre des problèmes, Alain, qui m'a fait découvrir un peu de la rivalité Sainté-Lyon, Vincent, qui m'a toujours *aidé* à tenir bon et *soutenu* à chaque instant, Viktor, le chef qui m'a fait confiance même sur un sujet aussi récent et qui nous surprenait toujours avec ses histoires à la pause café, et Jean-Jacques (même s'il ne fait pas partie de l'équipe) pour avoir agrémenté nos pauses café d'une compagnie *délicieuse*. Je n'oublie pas non plus la "nouvelle" génération de doctorants, Pierre-Antoine, Paul, Arturo et Mateus qui m'ont accompagné lors de la rédaction de ce manuscrit. Je sais que vos thèses seront un succès même si ce ne sera pas facile tous les jours !

Un remerciement spécial à mes deux directeurs de thèses, Amaury et Lilian. À Amaury pour m'avoir beaucoup aidé et guidé pour comprendre un sujet nouveau pour moi au début de cette thèse, le machine learning, et pour la précision et la pertinence de ses corrections que ce soit pour la rédaction des articles ou du manuscrit de thèse. À Lilian pour tout ou presque, je pourrais faire une liste mais exhaustive mais je vais me contenter du plus important. Merci d'avoir vu en moi le potentiel de mener à bien cette thèse alors même qu'à l'époque je n'en étais moi-même pas certains. Merci d'avoir été là pour me guider dans l'exploration d'un sujet nouveau et de m'avoir appris à force de répétition à présenter mes résultats de manière digeste.

Merci également à mes camarades de l'ASEC, Aubin, Jules et Paul, ainsi qu'à ceux qui nous ont précédé dans la gestion de l'asso et ceux qui nous ont suivis.

Merci à Élodie, Fabien et Julian qui ont bien voulu de moi pour assurer leurs cours et qui, au passage, m'ont aidé et beaucoup appris.

Pour finir je remercie bien évidemment ma famille et mes amis, plus particulièrement mon père, ma mère, Xav, Laure, Anne et Max avec qui j'ai pu entraîner mes talents de vulgarisateur, avec plus ou moins de succès, et qui se sont toujours montrés intéressés lorsque je leur racontais mes péripéties de thésard.

Enfin, un remerciement spécial à Inés qui m'accompagne dans cette aventure depuis la première année. Merci pour ton soutient à chaque étape même lors de la plus difficile qu'a été ce doctorat. Si j'y suis arrivé et que je suis resté motivé, c'est en grande partie grâce à toi :

492646548712332194116258708795978360375382171971147132857326554124749759240032795669865644584391848110567327995083293248839922194779550360159760420169226090409492821711052641671790059937645277768763660822321961062923121127192756724173037106143572427801546734664454231422466792796620695381947136885458314808500076695284677165299132223831974198703081893049646766682285160116657671962964353978414964724261645321998708827094320551173775842689474411507831085819353297231779603781041454228916449658896520849749359293586996787767398417439496601 6

Table des matières

Liste des abréviations ix

Notation xi

Résumé xiii

I Introduction 1

1 Contexte 3
1.1 Sécurité des algorithmes de chiffrement . . . 3
1.1.1 Développement des standards cryptographiques . . . 3
1.1.2 Évaluation de la sécurité des implémentations d'algorithmes cryptographiques . . . 4
1.2 Introduction aux attaques par canaux auxiliaires . . . 6
1.2.1 Notations . . . 6
1.2.2 Principes des attaques par canaux auxiliaires . . . 6
1.2.3 Intégration du machine learning aux attaques profilées . . . 10
1.3 Contributions . . . 11

2 Attaques par canaux auxiliaires profilées et Deep Learning 15
2.1 Description de l'attaque Template . . . 15
2.1.1 Choisir le modèle de fuite . . . 15
2.1.2 Choisir la cible : l'avantage de la sortie de l'opération SubBytes . . . 16
2.1.3 Choix des points d'intérêt . . . 17
2.1.4 Construction des templates . . . 18
2.1.5 Réalisation de l'attaque et récupération de la clé . . . 19
2.1.6 Adaptation de cette attaque à l'utilisation de Deep Learning . . . 20
2.2 Machine Learning . . . 20
2.2.1 Concepts de base du Machine Learning . . . 20
2.2.2 Choix de la famille de fonctions . . . 23
2.2.3 Création de l'ensemble d'entraînement . . . 24
2.2.4 Fonction de perte . . . 25
2.2.5 Processus d'optimisation . . . 26
2.2.6 Hyperparamètres à optimiser . . . 28
2.3 Passage au Deep Learning . . . 29
2.3.1 Réseaux de neurones . . . 30

2.3.2 Évaluer et régulariser . . . 36
2.4 Le Deep Learning appliqué aux attaques par canaux auxiliaires . . . 38
2.4.1 Utilisations du Machine Learning et des réseaux de neurones pour des attaques par canaux auxiliaires . . . 38
2.4.2 Conclusion . . . 41

II Contributions **45**

3 Comprendre et évaluer les performances **47**
3.1 L'importance de l'évaluation des réseaux de neurones . . . 47
3.1.1 Utilisation des métriques pour l'évaluation des réseaux . . . 48
3.1.2 Métriques classiques utilisées pour le Machine Learning . . . 49
3.1.3 Lien entre l'évolution des métriques et les phénomènes d'underfitting et d'overfitting . . . 50
3.1.4 Technique d'early stopping et optimisation des performances . . . 52
3.2 Métrique existante peu efficace et exploration des possibilités . . . 53
3.2.1 Le problème de l'accuracy . . . 53
3.2.2 Utilisation de la théorie de l'information . . . 55
3.2.3 Cross-Entropy Ratio . . . 56
3.3 $\Delta^{d}_{train,val}$: métrique d'évaluation des algorithmes de deep learning appliqués aux attaques par canaux auxiliaires . . . 57
3.3.1 $\Delta^{d}_{train,val}$: détection de l'état de l'apprentissage . . . 57
3.3.2 Détection de l'overfitting et l'underfitting . . . 58
3.3.3 Technique d'early stopping basée sur la métrique $\Delta^{d}_{train,val}$ . . . 59
3.3.4 Parallélisation des calculs sur plusieurs epochs . . . 60
3.4 Résultats expérimentaux sur CNN_{best} . . . 61
3.4.1 Description du réseau et de la base de données utilisée . . . 61
3.4.2 Application de la métrique $\Delta^{1}_{train,val}$ et de la technique d'early stopping sur CNN_{best} . . . 63
3.4.3 Application de la métrique $\Delta^{1}_{train,val}$ sur d'autres réseaux . . . 64
3.5 Discussions sur la métrique $\Delta^{d}_{train,val}$ . . . 69
3.6 Conclusion . . . 70

4 Études de différentes techniques d'optimisation des réseaux de neurones **73**
4.1 Techniques d'amélioration de performance utilisées pour les applications des réseaux en analyses de canaux auxiliaires . . . 73
4.2 Introduction de la batch normalization . . . 75
4.2.1 Processus de normalisation . . . 75
4.2.2 Couche de normalisation . . . 76
4.2.3 Lissage de la fonction de perte . . . 76
4.3 Introduction aux différentes formes de régularisation . . . 78
4.3.1 Discussion sur les formes de régularisation déjà explorées . . . 78
4.3.2 Régularisation par norme L1 et par norme L2 . . . 80

4.3.3 Dropout . . . 81
4.4 Résultats expérimentaux . . . 83
4.4.1 Description de l'ensemble de données ASCAD clé variable . . . 83
4.4.2 Exécution d'attaques sur des traces à clé variable . . . 84
4.4.3 Étude de l'évolution de $\Delta^1_{train,val}$ pour CNN_{best} en fonction des ensembles considérés et application de la batch normalization . . . 86
4.4.4 Application des techniques de régularisation au réseau CNN_{bn} . . . 88
4.4.5 Comparaison avec les autres techniques utilisées dans l'état de l'art . 93
4.5 Conclusion . . . 95

5 Classement des traces à partir des prédictions 97
5.1 Problématique soulevée . . . 97
5.1.1 Modification du schéma d'attaque classique . . . 98
5.1.2 Formalisation du problème . . . 98
5.2 Distinction de la qualité des prédictions . . . 101
5.2.1 Quelques méthodes pour distinguer la qualité des prédictions . . . 102
5.2.2 Résultats expérimentaux sur des réseaux de neurones . . . 105
5.2.3 Résultats expérimentaux sur des attaques template . . . 115
5.2.4 Conclusion des tests . . . 119
5.3 Utilisation de réseaux de neurones pour résoudre le problème . . . 120
5.3.1 L'approche Learning to rank . . . 120
5.3.2 Utilisation d'un réseau siamois . . . 122
5.3.3 Choix des labels des paires de prédictions . . . 122
5.3.4 Adaptation de la ranking loss . . . 127
5.3.5 Adaptation au problème de classement des prédictions . . . 128
5.3.6 Équivalence au cas 1 . . . 131
5.4 Expériences sur l'application de la Scoring Loss . . . 131
5.4.1 ASCAD clé fixe . . . 133
5.4.2 ASCAD clé variable . . . 135
5.5 Conclusion du chapitre . . . 137

Conclusion 141

Communications et publications 145

A Networks 147

Table des figures 163

Liste des tableaux 167

Liste des abréviations

AES	**A**dvanced **E**ncryption **S**tandard (en français : standard de chiffrement avancé)
ANSSI	**A**gence **N**ationale de la **S**écurité des **S**ystèmes d'**I**nformation
CCE	**C**ategorical **C**ross **E**ntropy (en français : entropie croisée catégorique)
CER	**C**ross **E**ntropy **R**atio (en français : ratio d'entropie croisée)
CESTI	**C**entre d'**É**valuation de la **S**écurité des **T**echnologies de l'**I**nformation
DES	**D**ata **E**ncryption **S**tandard (en français : standard de chiffrement des données)
FIA	**F**ault **I**njection **A**ttack (en français : attaques par injection de fautes)
HD	**H**amming **D**istance (en français : distance de Hamming)
HW	**H**amming **W**eight (en français : poids de Hamming)
MLP	**M**ulti-**L**ayer **P**erceptron (en français : perceptron multi-couches)
MSE	**M**ean **S**quare **E**rror (en français : erreur quadratique moyenne)
NIST	**N**ational **I**nstitute for **S**tandards and **T**echnology (en français : institut national des normes et technologie)
NSA	**N**ational **S**ecurity **A**gency (en français : agence nationale de la sécurité)
PoI	**P**oint **o**f **I**nterest (en français : point d'intérêt)
ReLU	**Re**ctified **L**inear **U**nit (en français : unité linéaire rectifiée)
RF	**R**andom **F**orest (en français : forêt aléatoire)
SCA	**S**ide-**C**hannel **A**nalysis (en français : analyse par canaux auxiliaires)
SPA	**S**imple **P**ower **A**nalysis (en français : analyse simple de consommation de courant)
SR	**S**uccess **R**ate (en français : taux de succès)
SVM	**S**upport **V**ector **M**achine (en français : machine à vecteurs de support)

Notation

b	Octet de clé attaqué
$\mathbb{E}$	Estimateur de l'espérance mathématique
f	Primitive cryptographique/Modèle entraîné
f^*	Fonction cible
$\mathbf{g}$	Vecteur contenant le classement des hypothèses de clé résultant d'une attaque
$\mathbf{k}^*$	Clé utilisée pour le chiffrement
$\mathcal{K}$	Ensemble des clés
$\mathcal{L}$	Fonction de perte/valeur de fuite
$\mathcal{N}(\mu, \sigma)$	Loi normale de moyenne μ et d'écart-type σ
$\mathcal{M}$	Modèle de fuite choisi
$\mathbf{p}$	Message d'entré du chiffrement
$\mathcal{P}$	Ensemble des messages
θ	Ensemble des paramètres d'un modèle
$\mathcal{T}$	Ensemble des traces
$\mathcal{Y}$	Ensemble des labels d'entraînement
Z	Variable sensible
$\hat{\rho}_{\mathbf{x},\mathbf{y}}$	Estimateur de la corrélation de Pearson entre les variables $\mathbf{x}$ et $\mathbf{y}$
∇_θ	Gradient en fonction des paramètres θ

Résumé

Les attaques par canaux auxiliaires peuvent être considérées comme une des principales menaces contre les implémentations de systèmes de chiffrement et de protocoles sécurisés. Elles se basent sur l'exploitation de fuites d'information, liées à des valeurs sensibles des algorithmes de chiffrement, qui sont contenues dans des grandeurs physiques telles que la consommation de courant ou l'émanation électromagnétique. Les principaux problèmes rencontrés dans la mise en place de ces attaques sont la présence de bruit dans les grandeurs physiques lors de leurs acquisitions et la gestion de la dimension de ces acquisitions. Récemment, des travaux ont été menés sur l'application d'algorithmes d'apprentissage profond pouvant aider à résoudre ces problèmes. Dans cette thèse, nous poursuivons l'étude de cette application des algorithmes d'apprentissage profond en essayant de comprendre plus précisément leur fonctionnement afin d'évaluer la menace qu'ils représentent. Nous commençons par mettre au point une métrique de performances permettant d'évaluer le comportement de ces algorithmes au cours de leur apprentissage et ainsi d'en déduire l'état de l'entraînement. Cette métrique est ensuite utilisée pour déterminer l'influence de techniques d'amélioration de performances issues du machine learning sur un réseau de neurones et caractériser les gains de performances. Pour finir, nous explorons une nouvelle façon de réaliser les attaques par canaux auxiliaires se basant sur l'ordonnancement des traces de consommation de courant utilisées en fonction des prédictions du réseau de neurones utilisé. Cela nous permet d'obtenir un rang minimal de la clé inférieur au rang final dans la plupart des scénarios explorés et a le potentiel de remettre en cause un certain nombre d'évaluations de sécurité.

Abstract

Side-channel analysis can be considered as one of the most significant threat against the implementation of encryption schemes and secure protocols. It is based on the exploitation of information leakages of sensitive variables present in different physical values such as the power consumption or the electromagnetic emanation of the secure system. The main difficulties faced during the realisation of those attacks are the presence of noise during the acquisition of the physical values and the high dimension of the acquisitions. Recently, researchers started to explore the application of machine learning algorithms in order to solve those problems. In this thesis, we continue this work and try to understand more precisely how those algorithms work to better judge the threat they represent. We start by developing a performance metric that allows us to evaluate the behavior of the algorithms during their training phase and thus deduce the state of the learning process. We then use this metric to understand the influence of performance improvement techniques

usually considered in machine learning on a neural network used to perform side-channel attacks and characterize the improvement of the results. Finally, we explore a new way of performing side-channel attacks in which we determine an ordering on the traces to be processed based on the predictions from the neural network. It allows us to obtain a minimal value for the rank of the key lower than the final rank in most scenarios and this method has the potential to question a certain number of security evaluations.

Partie I

Introduction

Chapitre 1

Contexte

1.1 Sécurité des algorithmes de chiffrement

Depuis la fin de la 2nde guerre mondiale et l'avènement de l'informatique, la cryptographie a joué un rôle majeur dans la sécurisation des données sensibles et des communications. Pour cela, des algorithmes de chiffrement se sont succédés apportant toujours plus de garanties théoriques de sécurité. Ces garanties sont basées sur un principe proposé par Kerckhoffs puis Shannon établissant que la sécurité des algorithmes doit dépendre uniquement de la clé secrète et non de la non-connaissance de l'algorithme utilisé. La sécurité du chiffrement est donc uniquement lié au secret de la clé. Il est alors nécessaire à un attaquant de retrouver cette clé pour casser le chiffrement. Cela implique notamment que le chiffré issu du chiffrement ne doit pas apporter d'information sur la clé secrète. Les algorithmes suivant ce principe se basent donc sur deux méthodes différentes : soit sur un problème mathématique difficile pour lequel aucune solution facilement calculable n'existe, comme le problème du logarithme discret [1], soit sur un ensemble d'opérations successives créant assez de confusion pour que la dépendance sur la clé secrète ne soit pas détectable. De nos jours, la majorité de ces algorithmes ont été étudiés en détails, sont théoriquement sûrs et laissent peu de place pour des attaques par cryptanalyse, c'est-à-dire les attaques capables de retrouver le message clair à partir du message chiffré sans connaître la clé au préalable. Parmi ces algorithmes, ceux jugés les plus sûrs sont standardisés afin d'unifier à la fois les moyens de communications et la recherche autour de ces standards et de garantir leur sécurité.

1.1.1 Développement des standards cryptographiques

L'algorithme *Lucifer* (1971), utilisé dans un premier temps par IBM, peut être considérer comme un des premiers standards de chiffrement. Mis au point par Horst Feistel et basé sur les *réseaux de Feistel*, il est adapté par la suite par la *NSA* (*National Security Agency*) pour devenir le premier algorithme officiellement standardisé sous le nom de *DES*, *Data Encryption Standard*, en 1977. Le DES est un algorithme de chiffrement par bloc qui consiste à effectuer les mêmes calculs sur différents blocs, aussi appelés états, de données, *e.g.* un octet, et il utilise un système de clé symétrique, ce qui veut dire que la même clé est utilisée pour chiffrer et pour déchiffrer. De plus, un chiffrement complet consiste en la répétition d'opérations successives où chaque entrée est la sortie de l'opération précédente.

1. Problème basé sur la difficulté de retrouver l'exposant d'une exponentiation.

Une exécution de toutes les opérations est appelé un tour ou *round*. Malgré les critiques sur sa taille de clé, 56 bits, jugée trop petite, le DES est resté un standard utilisé jusqu'au début des années 2000. Il sera en partie remplacé par le *triple DES*, ou 3DES, une variante combinant trois DES et donc avec une complexité et une taille de clé plus grande.

En 2002, un nouveau standard, l'AES ou *Advanced Encryption Standard*, est établie par le *NIST*, *National Institute for Standards and Technology*. Ce nouvel algorithme, issu d'un concours ouvert à tous, est basé sur une proposition de Joan Daemen et Vincent Rijmen, il est nommé Rijndael [DR99]. Il reprend le principe de chiffrement par bloc et de clé symétrique, avec une clé de taille variable, 128, 192 et 256 bits, selon les besoins en sécurité. L'AES est aujourd'hui un des algorithmes les plus utilisés en sécurité et donc aussi un des plus étudiés que ce soit par la recherche académique ou industrielle. En voici une description générale :

Description de l'AES : Advanced Encryption Standard Similaire au DES, l'AES repose sur un principe de tour et sur quatre opérations différentes, chacune ayant un rôle précis :

- l'opération *AddRoundKey*, sert à combiner l'entrée avec la clé ;
- l'opération *SubBytes*, ajoute de la non linéarité aux calculs ;
- l'opération *ShiftRows*, procède à un décalage des états du chiffrement ;
- l'opération *MixColumns*, mélange les valeurs de quatre différents états.

L'algorithme commence avec une opération AddRoundKey puis enchaîne des tours composés en premier d'une opération SubBytes puis d'une opération ShiftRows et d'une opération MixColumns et finalement d'une opération AddRoundKey à l'exception du dernier tour qui ne contient pas d'opération MixColumns mais conserve les autres opérations. Le nombre de tours varie entre 10, 12 et 14 en même temps que la taille de la clé qui se compose de respectivement 128, 192 et 256 bits. La forme la plus répandue de l'AES est celle à 128 bits puisqu'elle procure déjà un très bon niveau de sécurité [Age03].

De plus, comme ce type d'algorithme ne repose pas sur un problème calculatoire, comme le problème du *logarithme discret* par exemple, il n'est donc pas inquiété par les attaques par ordinateur quantique basées sur l'algorithme de Shor [CCJ+16]. Les seules attaques connues pouvant effectivement réduire la sécurité de l'AES sont les attaques qui ciblent l'implémentation matérielle ou logicielle de l'algorithme.

Ces nouveaux vecteurs d'attaques ont rapidement été inclus dans les évaluations de sécurités des standards dû à leur efficacité bien supérieure aux attaques classiques.

1.1.2 Évaluation de la sécurité des implémentations d'algorithmes cryptographiques

De nos jours, un ensemble de laboratoires, supervisé par l'*ANSSI* (Agence Nationale de la Sécurité des Systèmes d'Information), est appelé à effectuer des évaluations de sécurité afin de certifier les composants et autres outils liés à la sécurité. Ces laboratoires sont appelés *Centre d'Évaluation de la Sécurité des Technologies de l'Information* ou *CESTI*. Ils effectuent une batterie de tests pour évaluer la résistance de différents composants sécurisés et c'est sur ces évaluations que repose le principe de certification. Les fabricants sont obligés de faire passer ces tests à leurs composants avant leur mise sur le marché. Un composant

suffisamment résistant à certains types d'attaques sera certifié pour des utilisations dans des contextes sensibles.

Un des principaux exemples est la carte à puce, notamment celles dédiés aux applications bancaires. Le besoin de sécurité y est très important, elles doivent donc être résistantes aux attaques complexes qui ciblent l'implémentation des algorithmes de chiffrement. En effet, lorsqu'un algorithme est utilisé sur un composant dont l'accès physique est possible, cela ouvre de nouvelles possibilités d'attaques. L'attaquant, plutôt que d'attaquer l'algorithme d'un point de vue calculatoire, peut chercher à récupérer des informations sur l'exécution des calculs au travers des grandeurs physiques émises par le matériel pour réduire le niveau de sécurité proposé par l'algorithme voir même dans certains cas récupérer la clé de chiffrement. Ces attaques se nomment *analyses par canaux auxiliaires*, ou *Side-Channel Analysis* (SCA). Ce nouveau genre d'attaques, découvert dans les années 90 par Paul Kocher [Koc96], a radicalement changé la façon de considérer la sécurité des composants embarqués. Une autre possibilité en lien avec l'accès physique du composant est l'*attaque par injection de fautes*, ou *Fault Injection Attack* (FIA), qui consiste à créer des fautes dans l'exécution des calculs nécessaires à l'algorithme. Ces fautes peuvent venir, par exemple, de tirs lasers créant des courants transitoires ou bien de changements dans l'alimentation du composant dans le but de générer des erreurs dans les calculs. Ces erreurs modifient le résultat du chiffrement et il est alors possible, en comparant ce résultat fauté au même résultat non-fauté par exemple, de retrouver la clé utilisée.

L'AES, décrit en Section 1.1.1, est généralement la cible de deux types d'attaques par canaux auxiliaires : les attaques par message clair choisi, *chosen plaintext attack* en anglais, et les attaques par chiffré choisi, ou *chosen ciphertext attack*. Dans le premier, on suppose avoir la connaissance et la maîtrise du message d'entrée de l'algorithme. Cela veut dire que, si l'attaquant arrive à retrouver une valeur intermédiaire utilisée dans les opérations du premier tour de l'AES précédant l'opération MixColumns, *e.g.* la sortie de la première opération SubBytes, il est capable de récupérer la valeur de l'octet de clé correspondant. Dans le deuxième type d'attaque, ce sont la connaissance et la maîtrise du chiffré en sortie d'algorithme qui sont considérées. Dans ce cas de figure, l'attaquant remonte un peu plus loin dans l'algorithme et va chercher une valeur intermédiaire utilisée dans les calculs après l'opération AddRoundKey du tour précédant, *e.g.* la valeur d'entrée de la dernière opération SubBytes. Ce faisant, il lui suffit de dérouler l'algorithme avec cette valeur étant donné que le dernier tour ne contient pas d'opération MixColumns et ainsi de retrouver la valeur de clé correspondante. Ces deux méthodes sont assez similaires et seule l'hypothèse de départ va décider laquelle est applicable. De plus, la récupération de la valeur ciblée se fait dans les deux cas par une analyse des canaux auxiliaires. Dans le reste du manuscrit, nous nous concentrerons sur le premier type d'attaque.

Un des aspects considérés lors des évaluations est la difficulté à mettre en place ces attaques. Une attaque plus complexe, nécessitant un niveau d'expertise plus élevé et un coût monétaire plus important, sera jugée comme moins impactante vis-à-vis de la sécurité d'un composant. C'est pour cette raison que les améliorations d'attaques peuvent avoir une influence importante sur les niveaux de certification. Cela a notamment été le cas récemment avec l'introduction des techniques d'apprentissage profond, ou *deep learning*,

pour faciliter les attaques par canaux auxiliaires. Ces nouvelles techniques doivent donc être étudiées en profondeur afin de juger de l'impact quelles auront, ce qui est un des objectifs de cette thèse. Pour faire cela, nous commencerons par discuter des attaques par canaux auxiliaires avant d'aborder les techniques d'apprentissage profond utilisables afin d'établir le contexte dans lequel se situe ces travaux de thèse.

1.2 Introduction aux attaques par canaux auxiliaires

Depuis leur introduction à la fin des années 90, les analyses par canaux auxiliaires, aussi connues sous le nom d'attaques par canaux auxiliaires, font l'objet d'importantes recherches qui ont mises en avant les failles que ces attaques introduisent dans des systèmes, matériels ou logiciels, jusqu'alors jugés sécurisés. Il ne suffit donc plus d'étudier la sécurité théorique des algorithmes de chiffrement d'un point de vue mathématiques mais il faut aussi s'assurer que la façon dont les calculs s'effectuent en pratique n'apporte pas d'information sur la clé secrète utilisée.

Les attaques par canaux auxiliaires se basent sur le fait qu'avec une étude précise de certaines grandeurs physiques, il est possible de déterminer à quel moment un calcul utilisant des valeurs sensibles est effectué et donc localiser les fuites d'informations contenues dans certaines de ces grandeurs physiques. Par exemple, une de ces attaques, appelé *timing attack*, utilise le temps que met un algorithme à s'exécuter afin d'en déduire des informations sur le secret utilisé. Les attaques qui vont nous intéresser dans ce manuscrit sont les attaques basées sur l'exploitation des grandeurs comme la consommation de courant ou le rayonnement électromagnétique d'un appareil [KJJ99]. La première nécessite un contact direct avec une partie du composant qui va permettre d'acquérir cette mesure ce qui n'est pas toujours facile à mettre en place. La seconde peut se faire à distance en plaçant des sondes au-dessus du composant mais les mesures sont plus sensibles aux bruits environnants et il faut aussi un travail préalable de recherche des zones de fuites [GMO01]. À des fins de simplification, dans le reste du manuscrit, nous parlerons de consommation de courant à la place de grandeur physique.

1.2.1 Notations

Avant de détailler les grands principes des attaques par canaux auxiliaires, nous allons établir les notations qui seront utilisées dans la suite du manuscrit. Les ensembles sont désignés par des lettres calligraphiques comme $\mathcal{X}$, les variables aléatoires, elles, s'écrivent à l'aide d'une majuscule X et les vecteurs aléatoires en majuscule et en gras $\mathbf{X}$ tandis que leurs réalisations sont en minuscules x (et $\mathbf{x}$). La $i^{\text{ème}}$ valeur d'un vecteur est représenté $\mathbf{X}[i]$ et la $i^{\text{ème}}$ observation d'une variable aléatoire x_i. La probabilité qu'une variable aléatoire X prenne la valeur x s'écrit donc $\Pr[X = x]$.

1.2.2 Principes des attaques par canaux auxiliaires

Avant d'aborder les attaques en elles-mêmes, il faut introduire les principes théoriques qui les régissent. Premièrement, les grandeurs mesurées sont stockées sous forme de vec-

teurs temporels appelés *traces*, dénoté **t** pour représenter une trace, et sont regroupés dans des ensembles $\mathcal{T} \in \mathbb{R}^{N \times D}$ où N est le nombre de traces de l'ensemble et D la dimension des traces, *i.e.* le nombre de points temporels qu'elle contient. La valeur intermédiaire ciblée générant la fuite est notée $Z = f(P, K)$, où f est une primitive cryptographique, typiquement l'AES ou une partie de l'AES, P ($\in \mathcal{P}$) est une variable publique, *e.g.* le message clair ou le chiffré, et K ($\in \mathcal{K}$) est une partie de la clé, *e.g.* un octet. Le but de l'attaque est de retrouver la valeur de k^*, la clé secrète utilisée par l'algorithme de chiffrement. Pour se faire, la méthode généralement utilisée est celle de *diviser pour régner*, en anglais *divide and conquer*, qui consiste à trouver des fractions de la clé, par exemple un bit ou un octet, séparément puis de les combiner pour obtenir la clé complète. Deux méthodes peuvent être utilisées pour faire cette récupération en fonction du contexte de l'attaque : l'analyse par canaux auxiliaires *profilée*, pour laquelle l'attaquant a accès à une copie de la cible qu'il attaque en version *boîte blanche*, c'est-à-dire qu'il connaît les valeurs de tous les calculs effectués, et l'analyse par canaux auxiliaires *non profilée*, pour laquelle l'attaquant a uniquement accès à la cible qu'il attaque. Nous commencerons par décrire cette dernière attaque avant de nous attarder sur les analyses par canaux auxiliaires profilées.

Analyse par canaux auxiliaires non profilée Les analyses par canaux auxiliaires non profilées sont un type d'attaques qui reposent sur certaines hypothèses nécessaires aux attaquants :

- les fuites d'information présentes dans les mesures sont directement corrélées avec les valeurs intermédiaires sensibles manipulées par l'algorithme ;
- les valeurs qui fuitent sont reliées au secret ou une partie du secret ;
- l'adversaire a accès aux grandeurs physiques provenant du composant pendant le chiffrement.

Ces analyses sont faites dans un contexte de *boîte noire*, c'est-à-dire qu'aucune information sur les valeurs manipulées par l'algorithme de chiffrement n'est supposée connue. Seul l'algorithme et l'entrée de cet algorithme, *i.e.* le message en clair, sont connus par l'attaquant. Ces attaques ne seront pas abordées en détail dans ce manuscrit mais permettent d'offrir plus de contexte sur les différentes façons qui existent d'exploiter les informations mesurables sur les canaux auxiliaires. Les attaques non profilées, comme leur nom le laisse deviner, ne reposent pas, contrairement aux attaques profilées, sur un profilage préalable de la consommation de courant d'un composant. L'attaquant doit alors faire des hypothèses sur la valeur de l'octet de clé et calculer les valeurs intermédiaires liées aux différentes hypothèses. Une fois ces valeurs calculées, il peut les confronter aux traces qu'il a récupérées depuis le composant attaqué et chercher à établir une corrélation entre les valeurs intermédiaires issues des différentes hypothèses et les valeurs de consommation des traces. L'hypothèse montrant la corrélation la plus élevée est considérée comme étant l'octet de clé utilisé lors du chiffrement.

Analyse par canaux auxiliaires profilée Le but des attaques profilées est aussi de récupérer une valeur secrète, ou une partie de cette valeur, utilisée dans le calcul d'un algorithme de chiffrement. Cette récupération mène à la possibilité de déchiffrer le message d'entrée,

que ce soit un déchiffrement complet, partiel ou une réduction du problème sur lequel se base l'algorithme. Dans le cadre d'un AES, une telle réduction peut prendre la forme de la récupération d'un octet de la clé. Ce type d'attaque a besoin d'une hypothèse supplémentaire par rapport aux attaques non profilées qui est : l'adversaire possède un composant identique à celui qu'il cherche à attaquer et il contrôle les valeurs intermédiaires manipulées. Ce composant, contrairement à celui attaqué, peut être vu comme une boîte blanche. Il est aussi parfois appelé composant *ouvert*. Cela signifie que toutes les valeurs manipulées lors du chiffrement sont connues.

Les attaques profilées sont constituées de deux phases : d'abord une phase de profilage qui repose sur ce composant ouvert puis une phase de correspondance. Durant la phase de profilage, illustrée par la Figure 1.1, l'adversaire utilise le composant ouvert, qui est une copie du composant qu'il veut attaquer, pour déterminer quand la variable sensible Z ($\in \mathcal{Z}$) fuit. Pour cela, il effectue des chiffrements pour lesquels il connaît toutes les valeurs des variables intermédiaires et mesure la consommation de courant pour chacun des chiffrements. Ces mesures forment l'ensemble de profilage. Il va ensuite déterminer un ensemble de point, appelé *points d'intérêt* ou *Point of Interest* (PoI), qui représente au mieux la fuite d'information. Ces points sont généralement choisis en calculant, par exemple, la corrélation entre les valeurs de consommation de courant et les valeurs de la variable intermédiaire ciblée Z. Ils permettent à l'attaquant de construire des modèles $F_k : (\mathbf{t}, p) \rightarrow \Pr[\mathbf{T} = \mathbf{t} | (P, K) = (p, k)]$ pour chaque valeur possible de k afin d'estimer la probabilité $Pr[\mathbf{T}|Z = z]$, c'est-à-dire la probabilité, étant donné un ensemble de traces $\mathcal{T}$, que la valeur de la variable sensible soit z. Ces modèles F_k peuvent être étendus et servir à estimer des probabilités $\Pr[\mathbf{T} = \mathbf{t} | \phi(P, K) = z]$ sans perte de généralité. Dans notre cas, la fonction ϕ est remplacée par la primitive cryptographique f.

Une fois les modèles construits, l'attaquant peut s'en servir pour retrouver la clé secrète utilisée par cible de l'attaque, c'est la phase de correspondance, illustrée par la Figure 1.2. Pour cela, il fait faire à la cible un ensemble de chiffrements pour lesquels il connaît le message clair et il enregistre la consommation de courant. Cela constitue l'ensemble de traces d'attaque $\mathcal{T}_a$. L'attaquant peut ensuite comparer les valeurs aux points d'intérêt de ces traces à chacun de ses modèles F_k afin de déterminer lequel correspond au mieux aux traces d'attaque et ainsi retrouver la clé utilisée par la cible.

Ces attaques sont cependant particulièrement sensibles à un phénomène appelé *désynchronisation*. Il correspond à un décalage entre chaque trace des valeurs de consommation de courant. Cela veut dire que chaque point temporel d'une trace ne correspond pas nécessairement aux mêmes opérations dans les autres traces. Les traces de l'ensemble de profilage ne présentent généralement pas de désynchronisation étant donné que l'attaquant a la maîtrise de la copie de la cible. Toutefois ce n'est pas toujours le cas des traces d'attaques. En effet, la cible se comportant en boîte noire, il est possible que lors de l'acquisition des traces de consommation de courant, il y ait un décalage temporel entre les traces. L'application des modèles n'est alors plus possible dû au fait que les points d'intérêt des traces d'attaque ne sont pas situés au même endroit. Il faut donc veiller à ce que les traces d'attaques soient synchronisées, soit via une méthode d'acquisition très précise, soit en appliquant des techniques de resynchronisation des traces *a posteriori*. Ces techniques sont

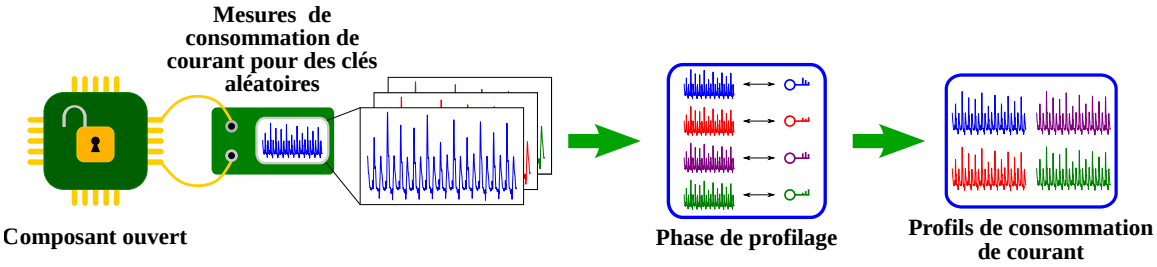


FIGURE 1.1: Phase de profilage des traces.

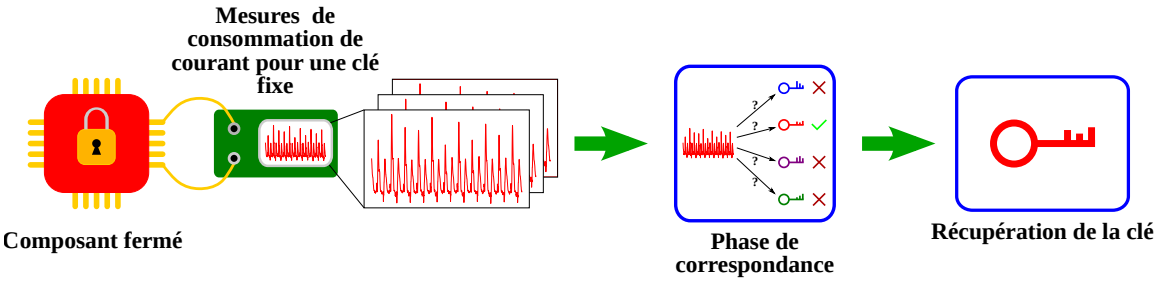


FIGURE 1.2: Phase de correspondance entre les traces et les profils.

nombreuses et variées, nous ne rentrerons donc pas dans leurs détails dans ce manuscrit mais le lecteur intéressé pourra poursuivre cette lecture avec [GKLD11]. Nous reviendrons dans la section 2.1 sur un exemple d'attaque profilée appelée attaque *template*.

Un point commun de chacune de ces méthodes est qu'une fois l'attaque exécutée, l'attaquant n'a, *a priori*, aucune garantie que la valeur de l'octet de clé qu'il a récupérée est la bonne. Il a donc besoin d'un moyen d'évaluer la confiance qu'il peut avoir dans ces différentes attaques.

Évaluation des attaques Afin d'évaluer les performances d'une attaque, toutes les clés candidates sont classées dans un vecteur de taille $|\mathcal{K}|$, noté $\mathbf{g} = (g_1, g_2, ..., g_{|\mathcal{K}|})$, contenant leurs probabilités respectives. g_1 est considérée comme la clé la plus probable et $g_{|\mathcal{K}|}$ comme la moins probable. La position de l'octet numéro b de la clé dans $\mathbf{g}$ est dénotée $\mathbf{g}(\mathbf{k}^*[b])$ et est appelée *rang* de la clé [SMY09a]. La *guessing entropy* [SMY09b] est définie comme étant le rang moyen de l'octet b de la clé $\mathbf{k}^*$, noté $\mathbf{k}^*[b]$, parmi toutes les hypothèses de clé. C'est la métrique d'évaluation des attaques par canaux auxiliaires la plus répandue puisqu'elle permet d'estimer, une fois l'attaque finie, le nombre moyen de clés à essayer avant de retrouver la bonne clé. Autrement dit, une attaque est considérée réussie en utilisant N_a traces si la guessing entropy est égale à 1.

Une autre métrique liée à la guessing entropy est le *taux de succès*. Il est définie comme la probabilité qu'une attaque réussisse, en N_a traces, à retrouver le bon octet de clé $\mathbf{k}^*[b]$ parmi toutes les valeurs possibles. Un taux de succès de p correspond au fait de retrouver p fois la bonne clé sur 100 attaques. Par exemple, un taux de succès de 90% en 1000 traces signifie que, sur cent attaques différentes utilisant 1000 traces, 90 attaques ont réussi à retrouver la bonne clé. À savoir que les 1000 traces utilisées pour les attaques ne sont jamais exactement les mêmes. Dans leur article [SMY09b], Standaert *et al.* proposent d'étendre la notion de taux de succès à un ordre quelconque d. Cela correspond au fait que, étant

donnée une attaque, la bonne valeur de clé soit parmi les d valeurs les plus probables dans $\mathbf{g}$. Plus formellement, soit $A_{E_k,L}$ un adversaire essayant d'attaquer un calcul cryptographique E_k en utilisant un modèle de fuite L. L'attaquant réalise l'expérience dénotée $Exp^d_{A_{E_k,L}}$ plusieurs fois afin d'exploiter la fuite d'information. L'attaque donne le vecteur $\mathbf{g}$, de longueur d, qui est composé des d candidats les plus probables triés selon les résultats de l'expérience. Si $\mathbf{k}^*[b] \in \mathbf{g}$, l'attaque est considérée comme un succès et $Exp^d_{A_{E_k,L}} = 1$. On peut donc considérer le taux de succès d'ordre d comme :

$$\mathrm{SR}^d_{A_{E_k,L}} = \Pr[Exp^d_{A_{E_k,L}} = 1].$$

Dans le reste du manuscrit, le taux de succès d'ordre d sera noté SR^d.

Nous allons maintenant voir où et pourquoi le machine learning s'intègre dans les analyses par canaux auxiliaires profilées.

1.2.3 Intégration du machine learning aux attaques profilées

Le machine learning est depuis peu considéré comme une alternative à la création des modèles F_k, qui servent à prédire la valeur de la variable intermédiaire ciblée, grâce à certains avantages qu'il présente. Dans le reste du manuscrit, nous nous concentrerons sur les algorithmes d'apprentissage profond, connu aussi sous le nom de *deep learning*, qui consistent en l'entraînement de réseaux de neurones. Nous verrons également leur application aux attaques profilées et leur comparaison à l'attaque profilée considéré comme la plus puissante, l'attaque *template*. Cette dernière attaque est décrite au début du chapitre suivant dans la Section 2.1. Le reste de ce chapitre est dédiée à l'explication du fonctionnement de ces algorithmes de machine learning mais il est déjà intéressant de voir comment ils s'intègrent aux attaques profilées.

Utilisation des algorithmes de Deep Learning Il faut tout d'abord décrire brièvement ce que sont ces algorithmes pour expliquer leur utilisation. Les algorithmes de Deep Learning sont des algorithmes d'apprentissage automatique, c'est-à-dire qu'on leur fournit un ensemble d'exemples résolus de la tâche qu'on cherche à leur faire apprendre et, à partir des exemples et de leurs solutions, les algorithmes apprennent à résoudre cette tâche pour de nouveaux exemples inconnus. Nous verrons dans le chapitre suivant la théorie derrière cet apprentissage. Leur utilisation dans les attaques par canaux auxiliaires est donc la suivante : ils apprennent, à partir des traces de consommation de courant du composant ouvert $\mathcal{T}_{train}$, un modèle F_θ, de paramètres θ, pouvant prédire la valeur de la variable intermédiaire utilisée Z. Une fois l'apprentissage terminé, ils peuvent être utilisés pour prédire les valeurs de cette même variable intermédiaire pour les traces de l'ensemble d'attaque $\mathcal{T}_a$. La récupération de ces valeurs permettent ainsi la récupération de la valeur de la clé de la cible de l'attaque.

En remplacement du choix des points d'intérêt Grâce à leur capacité à gérer des données en grande dimension, les algorithmes de deep learning utilisés peuvent se passer de la phase de choix des points d'intérêt. Il est possible de leur fournir soit les traces entières

ou soit la partie de la trace où l'opération ciblée a lieu même sans connaître les points précis où l'information fuite. Cela permet de laisser la liberté aux algorithmes d'apprendre où se situent les points d'intérêt via la phase d'entraînement. Le réseau entraîné n'a aucune information, *a priori*, sur la localisation de ces points, mais il les détecte automatiquement étant donné qu'il en a besoin pour améliorer ses prédictions lors de l'entraînement. En effet, ce sont les seuls points corrélés à la valeur intermédiaire ciblée. On peut donc se passer de la phase de prétraitement des traces qui s'intègre automatiquement à la phase d'apprentissage des algorithmes.

En remplacement de la construction des modèles Un des autres avantages des algorithmes d'apprentissage profond est qu'ils permettent aussi de se passer de la phase de construction des modèles puisqu'ils servent essentiellement à les remplacer. En effet, l'attaquant apprend à l'algorithme à effectuer des prédictions sur la valeur de la variable intermédiaire recherchée. En ce faisant, l'algorithme fourni, pour chaque trace donnée en entrée, les valeurs de $Pr[Z = z|\mathbf{T}]$ pour toutes les valeurs possibles z. Ces valeurs peuvent ensuite être utilisées directement dans la formation du vecteur $\mathbf{g}$ et donc le calcul de récupération de la clé.

Le deep learning permet de regrouper les phases de choix des points d'intérêt et de construction des modèles en une seule qui sera la phase d'apprentis-sage. Maintenant que l'utilité générale de ces algorithmes a été présentée, nous allons voir qu'elles sont les contributions apportées dans ce manuscrit et comment elles s'inscrivent dans la recherche en cours dans ce domaine.

1.3 Contributions

Les travaux réalisés lors de cette thèse se situent au croisement de deux domaines différents. D'un côté, celui de la sécurité, dans lequel la compréhension des attaques est vitale pour lutter contre celles-ci. De l'autre, celui du deep learning, où les réseaux de neurones utilisées sont encore largement perçus comme des boîtes noires. Il a donc fallu mélanger ces deux aspects ce qui a été fait grâce à une collaboration, la première, entre l'équipe SESAM (Systèmes Embarqués Sécurisés et Architectures Matérielles) et l'équipe Data Intelligence du laboratoire Hubert Curien. Cette collaboration a mené aux contributions décrites ci-après.

Le premier besoin identifié lors de l'étude des algorithmes de deep learning appliqué aux attaques par canaux auxiliaires a été la nécessité d'une métrique d'évaluation des performances de ces algorithmes. De nombreuses métriques existent déjà pour les diverses applications qui sont faites du machine learning mais aucune ne rendait compte des problématiques des analyses par canaux auxiliaires. Celle qui s'en rapprochait le plus est l'*accuracy*, qui correspond au pourcentage de fois où l'algorithme réussit sa prédiction en attribuant la plus forte valeur à la bonne hypothèse. Cette métrique peut se traduire au niveau des attaques par canaux auxiliaires comme le résultat d'une analyse simple de la consommation, ou *SPA* pour *Simple Power Analysis*, qui correspond à une attaque n'uti-

lisant qu'une seule trace pour retrouver la clé, là où les attaques profilées se basent sur une accumulation d'information pour récupérer la clé. **Partant de ce constat et à partir de la base de donnée *ASCAD* mise au point par l'ANSSI, nous avons mis au point une métrique, basée sur les métriques existantes en analyse par canaux auxiliaires, permettant d'évaluer les performances futures du réseau pendant la phase d'apprentissage.** Les détails de cette méthode d'évaluation se trouve dans le **chapitre 3**.

Grâce à cette métrique, nous avons ensuite pu mettre en avant l'utilité de pratiquer l'*early stopping*, ou *arrêt anticipé*, de l'entraînement. Cette méthode consiste à surveiller les métriques d'entraînement des algorithmes afin d'arrêter l'entraînement avant qu'ils subissent les effets négatifs du surapprentissage, qui apparaissent lorsqu'un algorithme commence à apprendre par cœur ses données d'apprentissage. L'early stopping est une technique très répandue en machine learning mais elle nécessite une métrique adaptée pour en tirer des bénéfices intéressants. **Nous avons donc montré qu'à l'aide de cette seule méthode, les performances des algorithmes de deep learning mis à disposition par l'ANSSI pouvaient être améliorées de 30%.** Ces observations se trouvent dans la seconde moitié du **Chapitre 3**.

Pour continuer dans l'exploration de l'application des techniques de machine learning aux algorithmes dédiés aux attaques par canaux auxiliaires et leurs effets en termes de performance, nous nous sommes penchés sur l'utilisation de *batch normalization* et de régularisation avec les techniques de *dropout* et *weight decay*. L'intérêt principal de ces méthodes est qu'elles permettent d'améliorer les performances d'un réseau sans pour autant avoir à changer sa structure. En les adaptant correctement, on peut donc obtenir un réseau relativement performant sur différentes cibles sans avoir à modifier ses paramètres d'apprentissage ce qui peut se révéler intéressant dans le cadre d'évaluation puisque cela résulte en un gain de temps significatif. Concernant les techniques, la batch normalization consiste à normaliser les données à l'intérieur du réseau afin que l'apprentissage se fasse plus rapidement. Le dropout lui va périodiquement "geler" certains neurones pendant l'apprentissage. Ces neurones ne sont plus pris en compte dans les calculs et ne sont plus mis à jour durant cette période dans le but d'empêcher le réseau d'apprendre par cœur. Quant au weight decay, il sert à contrôler les valeurs de poids associées aux neurones pour éviter qu'elles ne deviennent trop grandes. L'impact de ces techniques sur l'apprentissage et les performances des réseaux est détaillé dans le **Chapitre 4**. **Dans ce chapitre, nous montrons en effet qu'en utilisant ces techniques, un réseau qui semble peu performant à la base peut égaler les meilleurs réseaux.**

Le Chapitre 5 se concentre lui sur l'exploration d'une nouvelle méthode d'attaque se reposant sur le principe qu'un réseau peut être efficace sur certaines traces mais relativement peu performant sur d'autres. C'est-à-dire que certaines traces peuvent être bien prédites alors que pour d'autres, l'hypothèse qui aurait dû être prédite avec la valeur la plus forte se retrouve parmi les dernières. En parvenant à étudier et quantifier ce phénomène, il est donc possible d'améliorer des attaques qui, auparavant, n'étaient pas possibles en ordonnant les traces et en sélectionnant les mieux prédites. Cette sélection n'étant pas

évidente à faire, nous avons donc entraîné un réseau de neurones afin qu'il détecte les raisons sous-jacentes de ce phénomène. Pour réaliser cet entraînement, il a d'abord fallu déterminer la fonction de perte pour mesurer les erreurs du réseau et pour se faire, nous avons adapté une métrique nouvellement introduite : la *ranking loss*. Cette métrique sert de base pour l'entraînement de réseaux dans l'objectif de faire des attaques par canaux auxiliaires et met une priorité sur le fait de bien faire ressortir la bonne hypothèse en première lors des prédictions. En modifiant certains termes de cette fonction, on obtient la *scoring loss* qui permet d'entraîner un réseau qui attribuera un score à chaque trace de l'ensemble d'attaque afin de savoir lesquelles sont à utiliser en priorité. Cette nouvelle méthode d'attaque présente un intérêt certain dans un contexte d'évaluation d'un composant puisqu'elle peut remettre en cause la non-faisabilité de certaines attaques. De plus, elle ne s'applique pas uniquement aux attaques utilisant du machine learning mais aussi aux attaques templates où des effets similaires ont pu être constatés.

Chapitre 2

Attaques par canaux auxiliaires profilées et Deep Learning

Ce chapitre commence par une description détaillée de l'attaque template. Cette attaque profilée est considérée comme la plus puissante mais elle n'est pas toujours évidente à mettre en place. Les notions de machine learning et de deep learning nécessaires pour la compréhension de leur application aux attaques par canaux auxiliaires sont également introduites par la suite. Ces principes sont importants pour comprendre l'influence que vont avoir certaines méthodes sur l'entraînement des algorithmes et par conséquent sur leurs performances lors des attaques.

2.1 Description de l'attaque Template

Une des premières attaques à réussir à exploiter la fuite d'information pour récupérer des valeurs sensibles a été l'attaque dite *Template*. Cette attaque profilée est aussi une des plus puissantes si appliquée efficacement [CRR03]. En contrepartie, étant une attaque profilée, elle demande des prérequis importants, comme vu dans la section 1.2.2, afin d'être mise en place. Nous allons ici détailler l'exécution d'une attaque contre la valeur en sortie de l'opération SubBytes d'un chiffrement AES.

Avant d'expliquer pourquoi il est plus simple de récupérer la valeur en sortie de l'opération SubBytes, il nous faut introduire la notion de modèle de fuite qui décrit la façon dont les informations fuitent durant l'exécution de l'algorithme

2.1.1 Choisir le modèle de fuite

Afin de réaliser une attaque par canaux auxiliaires, il faut choisir le modèle de fuite que l'on va considérer, c'est-à-dire sous quelle forme l'information est présente dans la consommation de courant. Il existe une multitude de modèles possibles mais trois sont plus couramment utilisés que les autres : le modèle de l'identité (ID), le modèle du poids de Hamming (HW) et le modèle de la distance de Hamming (HD).

Le premier modèle part du principe que la valeur de la variable intermédiaire que l'on cherche à retrouver a un impact direct sur la consommation de courant. C'est probablement le moins réaliste des trois étant donné que les valeurs sont principalement manipulées sous leur forme binaire mais sa simplicité en fait un modèle relativement efficace. Il a aussi le potentiel de récupérer la valeur de clé avec une seule trace grâce à la correspon-

dance un-à-un entre la valeur intermédiaire et la valeur de l'octet de clé. Dans le modèle du poids de Hamming, ou *Hamming Weight* en anglais, on considère que la fuite dépend du poids de Hamming de la variable intermédiaire, c'est-à-dire du nombre de 1 présent dans l'écriture binaire de cette valeur. Ce modèle est plus réaliste que le précédent dû au fait qu'il dépend de l'écriture binaire de la variable ciblée. Pour finir, on peut partir de l'hypothèse que la fuite se fait en suivant un modèle de distance de Hamming, ou *Hamming Distance* en anglais, qui correspond au nombre de transitions $0 \rightarrow 1$ et $1 \rightarrow 0$ entre deux mots binaires. C'est le modèle le plus réaliste car les canaux auxiliaires que sont la consommation de courant et l'émanation électromagnétique ou photonique, sont sensibles uniquement aux transitions à cause de la technologie CMOS sous-jacente. C'est pour cela que le nombre de transitions, ou changements de valeur de bit, représente au mieux ce qui se passe lors des calculs. Toutefois, c'est aussi le modèle le plus difficile à considérer étant donné que l'accès à la valeur présente dans le registre avant la valeur ciblée est une chose difficile à garantir.

Soit $\mathcal{M}$ le modèle choisi, on peut représenter la fuite $\mathcal{L}$ en fonction de la valeur de la variable intermédiaire v et d'une valeur de bruit B :

$$\mathcal{L}(v) = \mathcal{M}(v) + B,$$

où le bruit B suit généralement une loi normale $\mathcal{N}(\mu, \sigma)$ de moyenne μ et d'écart type σ. En effet, l'hypothèse retenue pour modéliser les fuites est celle d'un bruit Gaussien, c'est-à-dire que le bruit présent dans le signal est supposé suivre une loi normale [MOP08].

Une fois ce choix fait, il reste à savoir pourquoi l'attaquant cherche à retrouver la valeur en sortie de l'opération SubBytes et non seulement la valeur en entrée puisqu'elle permet elle aussi de récupérer la clé.

2.1.2 Choisir la cible : l'avantage de la sortie de l'opération SubBytes

La sortie de l'opération SubBytes a rapidement été identifiée comme l'endroit le plus propice à la récupération de la valeur de la variable intermédiaire. En effet l'opération SubBytes se doit d'être inversible pour permettre le déchiffrement, la connaissance de la valeur en sortie se traduit alors directement par la récupération de la valeur d'entrée. Une fois cette valeur récupérée et à l'aide de la connaissance du message clair, l'opération de *XOR* peut être inversée et la clé retrouvée. Un schéma se trouve sur la Figure 2.1 et illustre cette suite d'opération à inverser où, grâce à la connaissance de m_i^j et à la récupération de s_i^j, il est possible de retrouver k^j.

De plus, une propriété importante de la sortie de l'opération SubBytes est illustrée dans la Figure 2.1. Elle représente l'évolution de la consommation de courant en fonction des valeurs manipulées dans l'algorithme. La première consommation de courant correspond au message clair en entrée de l'AES. Ici, le modèle de fuite est l'identité et on suppose qu'il n'y a pas de bruit présent. On obtient donc une valeur de consommation de courant qui s'exprime seulement à l'aide d'une fonction linéaire en m, la valeur d'un octet du message d'entrée. La deuxième représentation montre la consommation de courant à la sortie de la première opération AddRoundKey après l'application du ou-exclusif avec la clé. Sur la

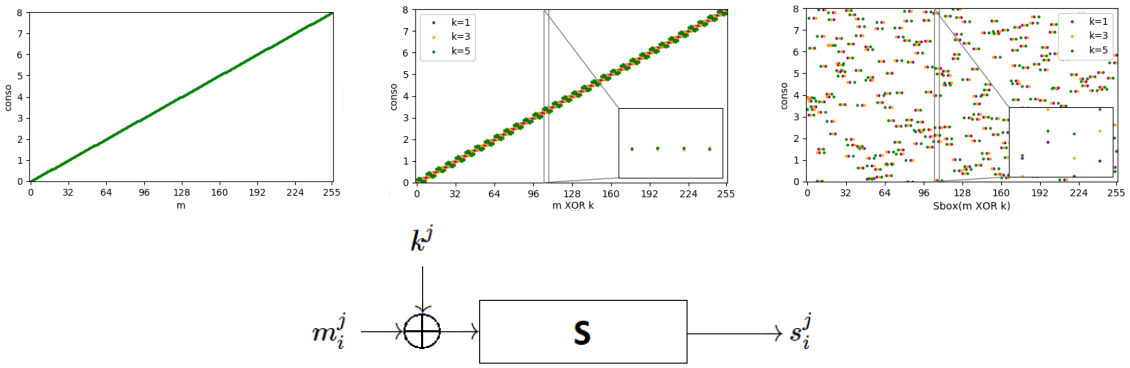


FIGURE 2.1: Exemple de l'impact des opérations AddRoundKey et SubBytes sur la consommation de courant dans le modèle de l'identité sans ajout de bruit. *À gauche* : consommation de courant représentée en tant que fonction de la valeur d'un octet du message d'entrée m. *Au centre* : Consommation de courant représentée en tant que fonction de la valeur de la sortie de l'opération AddRoundKey pour différentes valeurs proches de l'octet de clé k. *À droite* : Consommation de courant représentée en tant que fonction de la valeur de la sortie de l'opération SubBytes pour différentes valeurs proches de l'octet de clé k.

Figure 2.1, plusieurs hypothèses de clés proches sont considérées et on peut observer que pour des valeurs de clés proches, les consommations de courant en sortie du XOR sont elles-aussi proches. Il est possible de les distinguer même si cela devient difficile. Sur la droite, l'application de la boîte de substitution a pour effet de séparer les valeurs proches en entrée. Le résultat est donc, pour des clés proches, une consommation de courant facilement distinguable. C'est là que réside l'intérêt d'attaquer la sortie de l'opération SubBytes : la distingabilité des valeurs en sortie permet une récupération plus sûre de la valeur manipulée et de fait une récupération plus rapide de la clé.

La cible de l'attaque est maintenant clairement identifiée mais il reste un point important à voir avant la construction des templates : le choix des points d'intérêt qui permettront de caractériser au mieux la trace de consommation de courant.

2.1.3 Choix des points d'intérêt

Un point d'intérêt, abrévié *PoI* pour *Point of Interest*, est défini comme un point de la trace contenant de l'information sur la valeur recherchée. La détection des PoI, et leur choix, est donc une méthode similaire à une réduction de dimension. Elle se base sur le principe que tous les points d'une trace ne contiennent pas nécessairement de l'information sur la valeur de la variable intermédiaire et que l'information intéressante se situe sur un nombre restreint de points. Cette étape de réduction de la dimension des traces est obligatoire pour réaliser l'attaque puisque, sans elle, les calculs des templates deviendraient trop complexes. Il existe plusieurs méthodes pour choisir les points d'intérêt mais la plus répandue reste l'utilisation de la corrélation de Pearson, appelée simplement corrélation par la suite pour alléger le texte et estimée de la manière suivante :

$$\hat{\rho}_{\mathbf{x},\mathbf{y}} = \frac{\sum_{i=1}^{N}(\mathbf{x}[i]-\bar{\mathbf{x}})(\mathbf{y}[i]-\bar{\mathbf{y}})}{\sqrt{\sum_{i=1}^{N}(\mathbf{x}[i]-\bar{\mathbf{x}})^2}\sqrt{\sum_{i=1}^{N}(\mathbf{y}[i]-\bar{\mathbf{y}})^2}}, \tag{2.1}$$

où $\mathbf{x}$ et $\mathbf{y}$ sont des vecteurs de taille N et $\bar{\mathbf{x}}$ et $\bar{\mathbf{y}}$ représentent les moyennes de $\mathbf{x}$ et $\mathbf{y}$.

Cette méthode consiste à calculer la corrélation entre chaque point des traces de profilage et les valeurs de la variable intermédiaire de chaque trace. Il en résulte un vecteur de la taille d'une trace composé de toutes les valeurs de corrélation. Il suffit alors de sélectionner les points où les corrélations sont les plus fortes en valeur absolue et qui serviront à caractériser les traces. Le nombre de points sélectionnés a une influence sur la qualité des templates générés. Plus il y a de points, plus le template sera représentatif mais en contrepartie la complexité des calculs augmentera.

2.1.4 Construction des templates

Maintenant que les points d'intérêt $(j_1, ..., j_n)$ ont été choisis vient l'étape de construction des templates. Cette étape revient à construire les modèles F_k vu précédemment pour toutes les valeurs possibles de k. Pour construire un modèle F_k, l'attaquant sépare les traces de son ensemble de profilage $\mathcal{T}$ qui partagent la même valeur de k et ces traces forment l'ensemble $\mathcal{T}_k$. En utilisant les traces de cet ensemble, il va calculer un vecteur composé de la moyenne des traces à chaque point d'intérêt :

$$\boldsymbol{\mu}_k = \begin{bmatrix} \mu_{j_1,k} \\ \mu_{j_2,k} \\ \vdots \\ \mu_{j_n,k} \end{bmatrix},$$

où :

$$\mu_{j_i,k} = \frac{1}{|\mathcal{T}_k|}\sum_{\mathbf{t}\in\mathcal{T}_k}\mathbf{t}[j_i],$$

est la moyenne des traces $\mathbf{t}$ au point d'intérêt j_i.

Il aura aussi besoin d'une matrice de covariance :

$$\Sigma_k = \begin{bmatrix} v_{j_1} & c_{j_1,j_2} & \cdots & c_{j_1,j_n} \\ c_{j_2,j_1} & v_{j_2} & \cdots & c_{j_2,j_n} \\ \vdots & \vdots & \vdots & \vdots \\ c_{j_n,j_1} & c_{j_n,j_2} & \cdots & v_{j_n} \end{bmatrix},$$

où v_j représente la variance des traces $\mathbf{t} \in \mathcal{T}_k$ au point j et c_{j_1,j_2} la covariance des traces $\mathbf{t} \in \mathcal{T}_k$ aux points (j_1, j_2). Ces valeurs sont calculées à l'aide des formules :

$$v_j = \frac{1}{|\mathcal{T}_k|} \sum_{\mathbf{t} \in \mathcal{T}_k} (\mathbf{t}[j] - \boldsymbol{\mu}_k[j])^2, \tag{2.2}$$

$$c_{j_1,j_2} = \sum_{\mathbf{t} \in \mathcal{T}_k} (\mathbf{t}[j_1] - \boldsymbol{\mu}_k[j_1])(\mathbf{t}[j_2] - \boldsymbol{\mu}_k[j_2]). \tag{2.3}$$

À l'aide de ces valeurs, l'attaquant peut ensuite calculer la fonction de distribution des probabilités, notée *pdf* pour *probability distribution function*, en utilisant la formule :

$$F_k(\mathbf{t}') = \frac{1}{\sqrt{(2\pi)^n |\Sigma_k|}} e^{-((\mathbf{t}'-\boldsymbol{\mu}_k)^{\mathrm{T}} \Sigma_k^{-1} (\mathbf{t}'-\boldsymbol{\mu}_k)/2)}, \tag{2.4}$$

où $\mathbf{t}' = (\mathbf{t}[j_1], \mathbf{t}[j_2], ..., \mathbf{t}[j_n])$ représente le vecteur formé par les valeurs de $\mathbf{t}$ aux points d'intérêt.

2.1.5 Réalisation de l'attaque et récupération de la clé

Une fois l'ensemble des templates calculé pour chaque valeur de k possible, l'attaquant est prêt pour les utiliser sur les traces de l'ensemble d'attaque pour retrouver la clé. Il peut donc accumuler les probabilités de la façon suivante :

$$\mathbf{d}_{N_a}[k] = \prod_{i=1}^{N_a} \Pr[(P,K) = (p_i,k) | \mathbf{L} = \mathbf{l}_i] \tag{2.5}$$

$$= \prod_{i=1}^{N_a} \frac{\Pr[\mathbf{L} = \mathbf{l}_i | (P,K) = (p_i,k)]}{\Pr[\mathbf{L} = \mathbf{l}_i]} \Pr[(P,K) = (p_i,k)] \tag{2.6}$$

$$\propto \prod_{i=1}^{N_a} \frac{F_k(\mathbf{t'}_i)}{\Pr[\mathbf{L} = \mathbf{l}_i]} \tag{2.7}$$

$$\propto \prod_{i=1}^{N_a} F_k(\mathbf{t'}_i) \tag{2.8}$$

$$\propto \sum_{i=1}^{N_a} \log(F_k(\mathbf{t'}_i)), \tag{2.9}$$

où $\mathbf{d}_{N_a}[k]$ représente l'accumulation des probabilités pour l'hypothèse de clé k. Les différentes valeurs de $\mathbf{d}_{N_a}$ sont ensuite comparées entre elles pour former le vecteur $\mathbf{g}$, constitué de l'ensemble des scores associés aux différentes hypothèses de clé classées par ordre décroissant. Cela signifie que la première valeur de ce vecteur est l'hypothèse de clé la plus probable. Soient $\mathbf{k}^*$ est la valeur de la clé utilisée et b l'octet ciblé, si $\mathbf{k}^*[b] = \mathbf{g}[0]$ alors l'attaque est réussite et la clé est récupérée.

Les valeurs de $\mathbf{d}_{N_a}$ sont calculées en utilisant des équivalences étant donné que les valeurs $\Pr[\mathbf{L} = \mathbf{l}_i]$ et $\Pr[(P,K) = (p_i,k)]$ n'ont pas d'influence sur le résultat. En effet, si l'on suppose que les valeurs du message d'entré m sont choisies de manière uniforme, ce qui est le cas étant donné que l'attaquant a le contrôle de la valeur du message dans le contexte d'attaques par message choisi, alors la valeur de $\Pr[(P,K) = (p_i,k)]$ est la même pour tous les p_i : $\Pr[(P,K) = (p_i,k)] = 1/256$ dans le cas d'un modèle de fuite identité. Il

en résulte que ce terme est juste un facteur multiplicatif qui n'a pas d'influence sur l'ordre des valeurs de $\mathbf{d}_{N_a}$. Il en va de même pour le terme $\Pr[\mathbf{L} = \mathbf{l}_i]$ qui sert à normaliser la valeur de probabilité et ne dépend pas de la clé. Étant donné que l'adversaire est intéressé par l'ordre des éléments de $\mathbf{d}_{N_a}$, ce terme peut être omis lors des calculs pour les simplifier. La valeur de $\Pr[\mathbf{L} = \mathbf{l}_i|(P, K) = (p_i, k)]$ est elle remplacée par son estimation faite à l'aide des templates : $F_k(\mathbf{t}_i)$. De plus, il est aussi possible de remplacer le produit des estimations pour une somme de leur logarithme. Cette opération ne change pas l'ordre des valeurs de $\mathbf{d}_{N_a}$ mais a pour avantage de faciliter le calcul et de prévenir de problèmes calculatoires liés à la manipulation de petites valeurs de probabilités.

2.1.6 Adaptation de cette attaque à l'utilisation de Deep Learning

La Figure 2.2 illustre les trois grandes étapes des attaques par canaux auxiliaires profilées ainsi que les endroits où intervient le Deep Learning. L'entraînement du réseau vient ainsi remplacer les étapes de détection des points d'intérêt et de création des templates. Ces deux étapes sont remplacées par la création et l'entraînement d'un réseau de neurones qui apprend à détecter les points d'intérêt et intègre dans la fonction qui représente les templates de consommation de courant. Il ne reste plus qu'à appliquer le réseau sur les traces de consommation de courant du composant ciblé pour en obtenir un ensemble de vecteurs de prédictions qui jouent le rôle des *pdf* calculées avec les templates, c'est-à-dire les valeurs $F_k(\mathbf{t'}_i)$ pour l'ensemble des traces $\mathbf{t'}_i$ et l'ensemble des valeurs de k.

2.2 Machine Learning

Dans cette section, nous allons décrire les principes de base qui régissent le machine learning afin de préparer la description du deep learning et son utilisation dans les attaques par canaux auxiliaires. Un algorithme de machine learning est un algorithme ayant pour objectif d'apprendre à résoudre une tâche à l'aide d'un ensemble de données d'entraînement, ou *ensemble d'entraînement*. Pour ce faire, il construit un modèle qui prend en entrée un exemple du problème à résoudre et qui ressort la solution liée à cette entrée. Par exemple, dans un problème de classification, l'objectif est de déterminer la classe d'appartenance de l'exemple. L'algorithme doit donc apprendre, à partir d'exemples déjà classifiés dans le cas d'un apprentissage supervisé, à prédire, parmi les classes possibles vues lors de l'apprentissage, la classe de nouveaux exemples pour lesquels elle est inconnue. La construction de ce modèle se fait en plusieurs étapes qui sont détaillées au long de cette section. Chacune de ces étapes est essentielle à l'obtention d'un modèle efficace, nous allons par la suite voir comment elles sont traitées.

2.2.1 Concepts de base du Machine Learning

Le machine learning repose sur un principe central duquel découle la majeure partie des algorithmes et techniques utilisés lors de l'apprentissage. Ce principe est le suivant : un algorithme de machine learning doit être capable d'être performant sur des données qu'il n'a jamais vu. C'est le principe de *généralisation*. En partant d'un ensemble d'entraînement

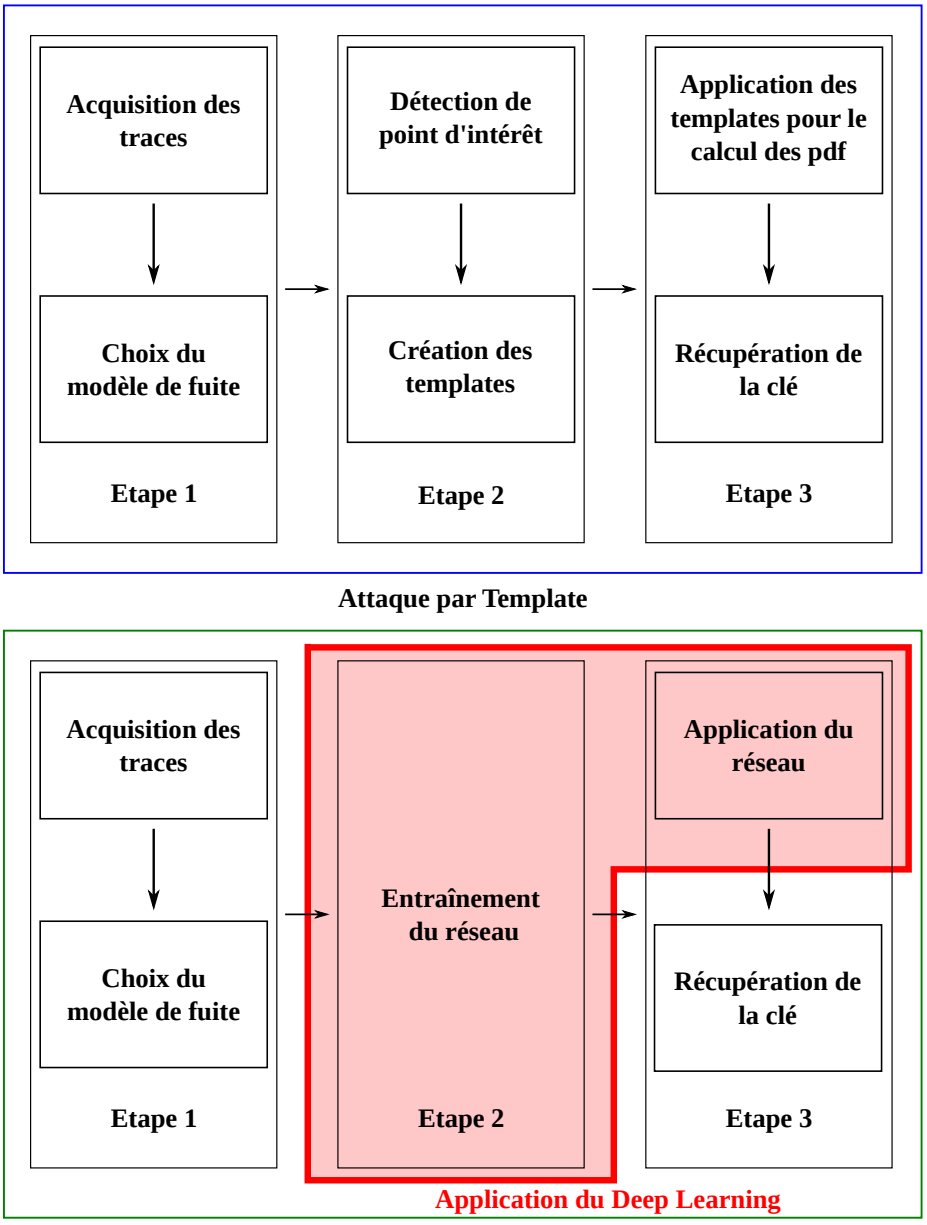


FIGURE 2.2: Étapes d'une attaque template et la correspondance d'une attaque basée sur le Deep Learning.

sur lequel il apprend, l'algorithme de machine learning doit être efficace sur des exemples qui ne se trouve pas dans cette ensemble. Ces exemples sont appelés exemples de test et forment l'*ensemble de test*. S'ils ne sont pas obtenus en même temps que les exemples d'entraînement, ils sont quand même supposés provenir d'une distribution similaire. Sans cela, il serait impossible d'avoir de la généralisation puisque les données d'entraînement et de test ne présenteraient aucuns liens.

Afin d'obtenir cette généralisation, l'algorithme cherche à minimiser l'erreur qu'il fait sur l'ensemble d'entraînement, aussi appelée *erreur d'entraînement*. Toutefois, et contrairement aux algorithmes classiques d'optimisation, l'objectif final n'est pas la minimisation de l'erreur d'entraînement mais la minimisation de l'*erreur de test*, c'est-à-dire l'erreur faite par l'algorithme sur les exemples de test qui lui sont inconnus. Cette erreur est aussi connue sous le nom d'*erreur de généralisation* et consiste en la moyenne des erreurs faites sur les données de test. Nous allons maintenant formaliser ce problème en introduisant de nouvelles notations pour décrire les moyens d'obtenir la généralisation du modèle.

Le but de l'algorithme de machine learning est d'approximer une fonction que l'on appelle communément *fonction cible*, que l'on note f^*, et qui représente le problème à résoudre. Dans un problème de classification, par exemple, on supposera les exemples issus d'un espace d'entrée $\mathcal{X} \subseteq \mathbb{R}^d$, et les classes d'un ensemble de labels $\mathcal{Y} \subseteq \mathbb{R}^k$. La fonction f^* définit le lien entre $\mathcal{X}$ et $\mathcal{Y}$. La méthode utilisée pour l'approximation de f^* consiste à apprendre une fonction f_θ, composée de paramètres θ et issue d'un ensemble d'hypothèses H défini a priori. Cet ensemble peut par exemple correspondre aux classifieurs linéaires en dimension d ou à l'ensemble des réseaux neurones avec certaines limitations sur la structure du réseau comme le nombre de couches ou de neurones. Les fonctions de H sont des fonctions paramétrées et l'objectif est de trouver, par un apprentissage à partir d'exemples, une fonction $f_\theta \in H$ qui approxime bien f^*, ce qui revient à estimer par apprentissage les paramètres θ permettant d'obtenir cette bonne approximation de f^*. Dans ce contexte, on suppose que l'ensemble $\mathcal{X} \times \mathcal{Y}$ est équipé d'une distribution de probabilité D et que l'on peut obtenir un échantillon d'exemples $\mathcal{T} = (\mathbf{x}_i, \mathbf{y}_i = f^*(\mathbf{x}_i))_{i=1}^n$ tirés selon cette distribution sans avoir la connaissance de la distribution D.

L'approximation f_θ correspond à une fonction définie au sein d'un ensemble de fonctions hypothèses H fixé a priori. Cette fonction sera appelée modèle par la suite. On part du principe que les exemples à résoudre lors de l'application du modèle suivent une distribution similaire à celle des exemples d'entraînement issus de $\mathcal{T}$ et donc que le modèle sera par la suite capable de généraliser ce qu'il a appris. Nous utiliserons la Figure 2.3 pour illustrer les différents concepts et les impacts qu'ils ont sur le résultat de l'entraînement. Cette figure représente, à l'aide de bulles, les ensembles de fonctions théoriquement représentables par le modèle. La plus grande bulle représente H, la famille de fonction de laquelle est tirée notre approximation f, soit par exemple l'ensemble des fonctions $h : \mathbb{R}^d \to \mathbb{R}^k$ qui représente l'ensemble des classifieurs linéaire en dimension d à valeur dans un espace de dimension k. À des fins de simplification, nous supposerons ici que la fonction cible f^* est inclue dans l'espace H. Cette hypothèse est relativement forte et de manière générale, il n'existe aucune garantie qu'elle soit vérifiée. Elle nous sert ici principalement à avoir un schéma simple quitte à perdre en réalisme. La fonction cible est donc

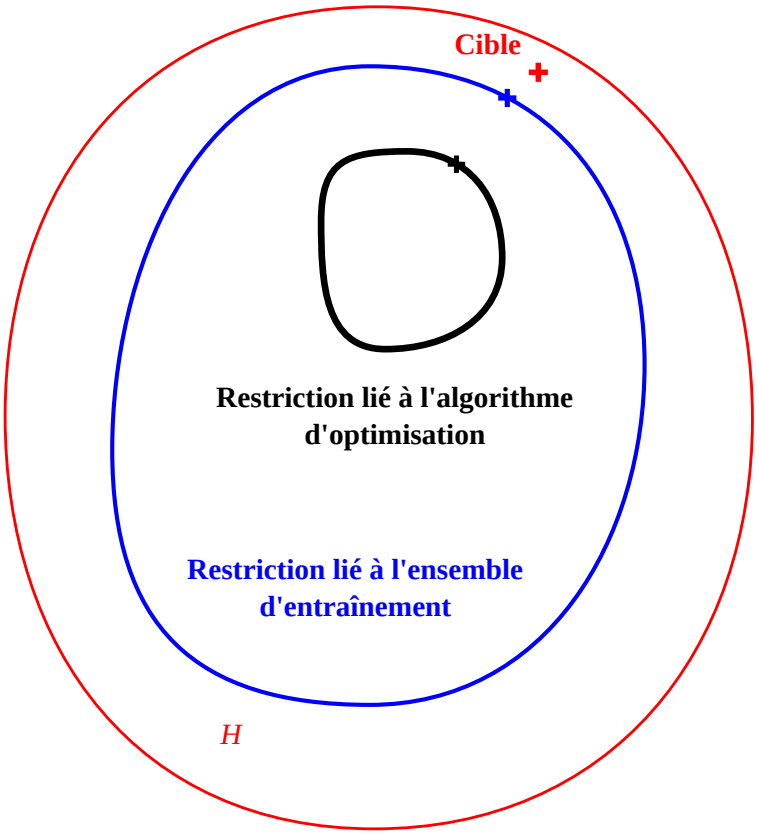


FIGURE 2.3: Représentation des biais.

ici représentée par la croix rouge. Le but de notre modèle f est de s'approcher le plus près possible de cette fonction. La première étape est maintenant le choix de la famille de fonctions H qui détermine l'ensemble des approximations possibles.

2.2.2 Choix de la famille de fonctions

Le choix de la famille de fonctions H est un aspect très important en machine learning. Comme on peut le voir sur la Figure 2.3, Il introduit un biais et détermine l'ensemble des fonctions approximables par l'algorithme. Ce choix est très lié à la *capacité* du modèle, c'est-à-dire son potentiel d'approximer un ensemble divers de fonctions. Pour illustrer ce concept, nous allons utiliser la Figure 2.4. Cette figure représente un exemple de régression linéaire où notre but est de retrouver la fonction polynomial de laquelle sont extraits les exemples d'entraînement. Pour ce faire, on peut jouer sur le degré des polynômes considérés. Soit H_k la famille de polynômes de degré k, il nous faut choisir avant l'entraînement de quel famille H_k la fonction f_θ sera tirée.

Sur la Figure 2.4, deux fonctions différentes f_{θ_1} et f_{θ_5} sont prises respectivement dans H_1 et H_5. On peut les décrire de la façon suivante :

$$
\begin{aligned}
f_{\theta_1}(x) &= w_1^{(1)} x + w_0^{(1)}, \\
f_{\theta_5}(x) &= w_5^{(5)} x^5 + w_4^{(5)} x^4 + w_3^{(5)} x^3 + w_2^{(5)} x^2 + w_1^{(5)} x + w_0^{(5)},
\end{aligned}
$$

où $\theta_j = w_i^{(j)}$ sont les paramètres des fonctions f_{θ_1} et f_{θ_5}, ici les coefficients des puissances de x, qui sont appris lors de l'entraînement. On en déduit que la capacité d'approximation de f_{θ_1} est plus faible que celle de f_{θ_5} notamment dû au fait que $H_1 \subset H_5$. Toutefois cela ne veut pas nécessairement dire que la fonction f_{θ_5} résultante de l'entraînement est plus efficace que la fonction f_{θ_1}. Comme on peut le voir dans cet exemple, aucune des fonctions ne réussit à approximer correctement la fonction cible qui, elle, est de degré 3. Cela illustre aussi les problèmes d'*underfitting* et d'*overfitting* qui correspondent à un sous-entraînement et à un sur-entraînement. Dans le premier cas, la capacité du modèle n'est pas suffisante pour intégrer correctement les exemples d'entraînement et dans le deuxième, la capacité du modèle est trop élevée ce qui mène à des prédictions parfaites des données d'entraînement mais une mauvaise généralisation. Nous reviendrons plus en détails sur ces phénomènes dans la Section 2.3.2. Dans l'exemple de la Figure 2.4, la fonction f_{θ_1} rencontre un problème d'underfitting puisqu'elle n'a pas assez de capacité et n'est donc pas capable de passer par tous les exemples d'entraînement. La limitation de sa famille de fonction est la cause de ce manque de capacité, il faut donc l'agrandir afin d'améliorer les performances de la fonction. La fonction f_{θ_5}, elle, est en overfitting, c'est-à-dire qu'elle a des résultats parfaits sur les exemples d'entraînement mais que cela ne se traduit pas par une bonne approximation de la fonction cible. Le problème rencontré ici est en partie lié à la famille de fonction H_5 qui laisse trop de liberté dans la capacité d'approximation. On peut toutefois imaginer qu'avec plus d'exemples d'entraînement, la fonction arriverait à une meilleure approximation de la fonction cible.

Cette exemple reflète notamment le fait que la fonction cible n'est pas toujours dans la famille de fonction choisie. Ici la connaissance de la fonction cible nous permet de mettre cela en évidence mais cette fonction n'est que très rarement connue puisque le but de l'algorithme de machine learning est de l'approximer. Il est donc essentiel de bien choisir la famille de fonction H.

2.2.3 Création de l'ensemble d'entraînement

La création de l'ensemble d'entraînement est également déterminante afin d'obtenir les meilleures performances possibles avec le modèle final. Les exemples d'entraînement sont obtenus à partir de la distribution D. Il est nécessaire que les données soient les plus représentatives possibles de la tâche à résoudre. En effet, si les données d'entraînement sont trop différentes des données sur lesquelles le modèle sera appliqué par la suite, il risque de ne pas être capable de résoudre le problème correctement. On peut donc voir sur la Figure 2.3 que les exemples de l'ensemble d'entraînement apportent aussi un biais qui va restreindre les fonctions atteignables par le modèle. Cependant, dans les cas où l'ensemble d'entraînement est trop petit, il peut aussi laisser trop de liberté aux fonctions complexes. C'est visible sur la Figure 2.4 où la fonction f_{θ_5}, même en ayant la capacité à approximer la fonction cible de degré 3, n'est pas capable de l'atteindre. Cela est lié au fait que les exemples d'entraînement disponibles ne sont pas assez représentatifs de la fonction cible. En effet, en théorie, avec un nombre croissant d'exemples et une capacité suffisante, il est possible d'approximer de manière très précise la fonction cible mais cette

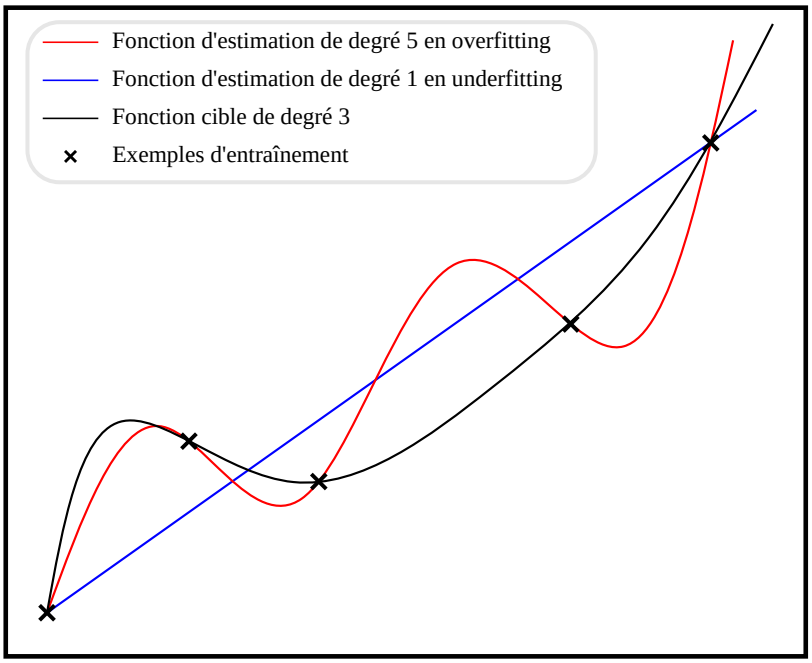


FIGURE 2.4: Représentation du potentiel d'approximation de différentes familles de fonction.

hypothèse n'est pas réaliste en pratique. Une fois cette restriction faite, on peut situer la croix bleue qui correspond à la valeur théorique du modèle la plus proche de la fonction objectif étant donné un ensemble d'entraînement donné. Pour minimiser la distance entre le modèle et cette valeur, il faut définir un moyen d'évaluer cette distance. Cela est fait à l'aide de la *fonction de perte*, ou *loss function*.

2.2.4 Fonction de perte

La fonction de perte, aussi connue sous le nom de *fonction de coût*, notée $\mathcal{L}$, a pour but de mesurer l'erreur que fait le modèle lors de ses prédictions ou de quantifier le niveau de pénalisation des erreurs effectuées. Cette fonction permet, en mesurant les erreurs faites par le modèle sur les données d'entraînement, de les corriger en modifiant les paramètres du modèle. On souhaite donc minimiser la valeur de la fonction de perte en fonction des paramètres du modèle. Cela correspond à un problème d'optimisation qui va permettre de minimiser la distance entre la fonction estimée par le modèle et la fonction cible. Toutefois, étant donné qu'on ne connaît pas la fonction cible, cette minimisation est indirecte et est faite à l'aide de la fonction de perte. Le choix de cette fonction est donc très important. Il dépend principalement du problème à résoudre et il existe un très grand nombre de fonctions de perte. Nous nous concentrerons dans ce manuscrit sur la fonction appelée *categorical cross entropy* (CCE) ou *entropie croisée catégorique* sur laquelle nous reviendrons Section 2.3.1.

La fonction de perte est utile pour évaluer l'erreur du modèle sur un exemple donné et corriger le modèle afin d'améliorer ses performances sur cet exemple mais le but de l'apprentissage est l'entraînement d'un modèle qui soit performant sur le plus grand nombre d'exemples possibles et notamment sur les exemples de test. On cherche donc à minimi-

ser le *risque*, qui représente l'espérance de l'erreur du modèle sur les données issus de la distribution D. Cependant, on ne connaît pas cette distribution D, on ne peut donc pas se concentrer sur la minimisation du risque. On va alors considérer le risque restreint aux exemples connus c'est-à-dire les exemples d'entraînement. Cette valeur est appelée *risque empirique*. Il est définie de la façon suivante :

$$\mathbb{E}_{(\mathbf{x},\mathbf{y})\sim\hat{D}}[\mathcal{L}(f(\mathbf{x};\boldsymbol{\theta}),\mathbf{y})] = \frac{1}{n}\sum_{i=1}^{n}\mathcal{L}(f(\mathbf{x}_i;\boldsymbol{\theta}),\mathbf{y}_i),$$

où $\mathbb{E}$ est l'opérateur de l'espérance mathématique et $\hat{D}$ représente la distribution empirique des exemples d'entraînement. Pour simplifier les notations, le risque empirique lié aux paramètres $\boldsymbol{\theta}$ et à l'ensemble d'entraînement T sera noté $\mathcal{L}_T(\boldsymbol{\theta})$ dans la suite du manuscrit. Le processus d'entraînement lié à la minimisation de ce risque s'appelle *minimisation du risque empirique* et peut être décrit de la façon suivante :

$$\min_{\boldsymbol{\theta}\in\mathbb{R}^m}\mathcal{L}_T(\boldsymbol{\theta}), \tag{2.10}$$

où m correspond ici au nombre de paramètres du modèle. Nous allons maintenant voir le processus d'optimisation permettant la minimisation du risque empirique.

2.2.5 Processus d'optimisation

Un problème d'optimisation d'une fonction $f(x)$ quelconque consiste à modifier la valeur de x afin de minimiser la fonction f. En machine learning, le processus utilisé pour résoudre le problème d'optimisation est généralement la *descente de gradient*, ou *gradient descent* [C+47, GBC16]. C'est une technique de minimisation d'une fonction basée sur le calcul de la dérivée de cette fonction. Si l'on considère la fonction f, sa dérivée (partielle) $f'(x)$ au point x indique la pente de la tangente de f au point $(x, f(x))$. Il est donc possible, grâce à la connaissance de cette pente de savoir dans quelle direction modifier x afin de minimiser f.

Dans notre cas, la fonction à minimiser est plus complexe mais le procédé reste globalement le même. En effet, la fonction à minimiser est la fonction de perte $\mathcal{L}(f(\mathbf{x};\boldsymbol{\theta}),\mathbf{y})$. Pour ce faire, il est possible de calculer son gradient en fonction des paramètres $\boldsymbol{\theta}$ de f et pour un exemple $(\mathbf{x}_i,\mathbf{y}_i)$ soit $\frac{\partial\mathcal{L}(f(\mathbf{x}_i;\boldsymbol{\theta}),\mathbf{y}_i)}{\partial\boldsymbol{\theta}}$ que l'on notera également $\nabla_{\boldsymbol{\theta}}\mathcal{L}(f(\mathbf{x}_i;\boldsymbol{\theta}),\mathbf{y}_i)$. Le gradient correspond à l'ensemble des dérivées partielles par rapport aux différentes coordonnées d'un vecteur, ici le vecteur $\boldsymbol{\theta}$ représentant les paramètres du modèle. Cette méthode repose donc sur la différentiabilité de la fonction de perte. En calculant les gradients de cette fonction pour chacun des paramètres du modèle, il est possible de déterminer quelles modifications aboutiraient à une réduction de la valeur de la fonction de perte. On peut poser :

$$\mathbf{g} = \frac{1}{n}\nabla_{\boldsymbol{\theta}}\sum_{i=1}^{n}\mathcal{L}(f(\mathbf{x}_i;\boldsymbol{\theta}),\mathbf{y}_i),$$

qui représente le changement à apporter aux paramètres pour améliorer les performances du modèle étant donné l'ensemble des exemples d'entraînement. Les paramètres $\boldsymbol{\theta}$ sont mis à jour de la manière suivante :

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon \mathbf{g},$$

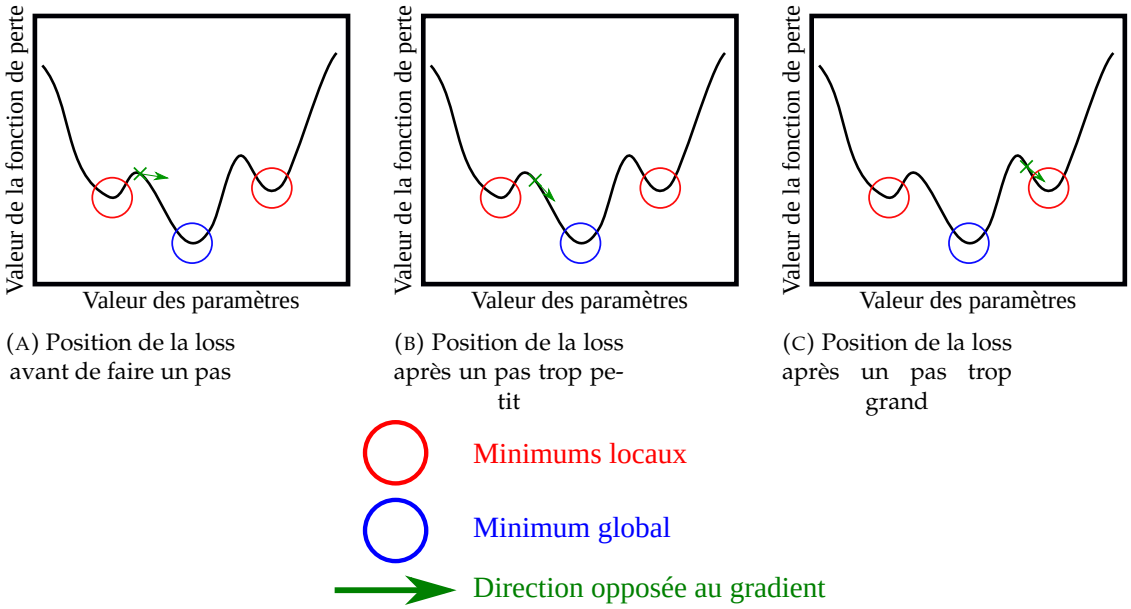
où ϵ représente le *learning rate* qui pondère les modifications apportées au modèle pour éviter des changements trop importants. Il sert à ajuster la taille des pas qui seront pris dans la direction qui minimise le risque empirique. La Figure 2.5 illustre un exemple de la forme d'une fonction de perte à minimiser où la Figure 2.5a est la position de $\mathcal{L}(f(\mathbf{x};\boldsymbol{\theta}),\mathbf{y})$ avant la mis à jour des paramètres. Comme on peut le voir dans la Figure 2.5b, un learning rate trop bas implique une convergence lente vers un minimum et peut aussi coincer l'optimisation dans une minimum local. Au contraire, un learning rate trop grand va engendrer des pas trop grands qui créeront de l'instabilité dans la convergence du réseau. Cela peut aussi conduire à rater certains minimums, comme illustré dans la Figure 2.5c. Il est donc très important de bien prendre en compte cet hyperparamètre lors de la création du réseau. On obtient alors l'Algorithme 1.

Algorithme 1: Descente de gradient

Données : Paramètres $\boldsymbol{\theta}$, learning rate ϵ;
tant que Condition de performance non remplie **faire**
  Calcul du gradient : $\mathbf{g} \leftarrow \frac{1}{n}\nabla_{\boldsymbol{\theta}} \sum_{i=1}^{n} \mathcal{L}(f(\mathbf{x}_i;\boldsymbol{\theta}),\mathbf{y}_i)$;
  Mis à jour des paramètres : $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon \mathbf{g}$.
fin

La mise à jour du modèle en utilisant la totalité des exemples d'entraînement est appelé une *epoch*. La méthode de descente de gradient est répétée à chaque epoch jusqu'à obtenir de bonnes performances et une réduction de l'erreur d'entraînement petit à petit. L'entraînement d'un modèle est généralement mesuré avec le nombre d'epoch d'entraînement qu'il fait. Le nombre d'epoch peut être considéré comme un hyperparamètre mais il existe également une méthode permettant de déterminer, au cours de l'entraînement, si un modèle a besoin de plus d'epoch ou non. Cette méthode, appelée l'*early stopping*, peut permettre un gain de temps et de performances significatif par rapport à un entraînement qui n'en aurait pas. Nous reviendrons sur l'application de l'early stopping en détails dans le Chapitre 3 avec notamment un exemple de son efficacité.

L'algorithme d'optimisation n'est toutefois pas parfait et il impose indirectement une nouvelle restriction représentée par la plus petite bulle sur la Figure 2.3. L'application de la descente de gradient permet d'obtenir la croix noire qui correspond à la meilleure approximation atteignable après l'ajout de tous les biais. Il est généralement difficile d'approximer parfaitement la fonction cible mais l'entraînement du modèle et les diverses techniques que nous verrons par la suite ont pour but de réduire au maximum la distance entre la croix noire et la croix rouge.



(A) Position de la loss avant de faire un pas

(B) Position de la loss après un pas trop petit

(C) Position de la loss après un pas trop grand

FIGURE 2.5: Exemple d'une fonction de perte à minimiser.

2.2.6 Hyperparamètres à optimiser

Il reste un dernier point à aborder qui s'applique de manière générale à tous les algorithmes de machine learning : les *hyperparamètres*. Contrairement aux paramètres du modèle qui sont modifiés au cours de l'entraînement par la processus d'optimisation, les valeurs des hyperparamètres sont choisies à la création du modèle et déterminent son architecture. Ils peuvent être décrits comme tous les paramètres que le réseau n'apprend pas lors de son entraînement mais ils sont quand même amenés à évoluer selon les performances du réseau. En effet, le choix des hyperparamètres joue sur la capacité du modèle et, pour reprendre l'exemple de la Figure 2.4, le choix du degré du polynôme qui sert d'approximation est un hyperparamètre. Si l'on constate que les performances du modèle ne sont pas bonnes, on peut alors jouer sur ses hyperparamètres pour améliorer le modèle. Avec le choix du modèle, ce sont eux qui influencent la seconde bulle de la Figure 2.3. Leur optimisation permet d'affiner l'ensemble des fonctions approximables par le modèle en réduisant la distance entre la fonction approximée et la fonction cible due au choix du modèle.

Des hyperparamètres importants et présents dans la majorité des modèles sont les hyperparamètres de régularisation. La régularisation permet de contrôler la complexité des modèles et de pénaliser les modèles trop complexes afin d'éviter l'overfitting en influant sur la valeur des poids. On peut ainsi l'inclure dans notre problème de minimisation de la manière suivante :

$$\min_{\boldsymbol{\theta}\in\mathbb{R}^m} \mathcal{L}_T(\boldsymbol{\theta}) + \lambda\Omega(f(\boldsymbol{\theta})),$$

où λ est un hyperparamètre et $\Omega(f(\boldsymbol{\theta}))$ est une fonction mesurant la complexité de la fonction $f(\boldsymbol{\theta})$, par exemple la norme $\|\cdot\|_2$, aussi connue sous le nom *weight decay* ou norme L_2. Cette norme mesure les différences entre les valeurs des poids et aura tendance à être

importante lorsque que seulement certains poids sont larges et les autres sont proches de zéro. Elle a pour effet de pénaliser les poids du réseau si leurs valeurs deviennent trop grandes, c'est-à-dire que la norme L_2 contrôlera les modifications que les poids subissent afin de les empêcher de prendre trop d'importance. Cela a tendance à forcer les poids à prendre des valeurs relativement faibles sans atteindre zéro. Cette norme et d'autres méthodes de régularisation sont discutées plus en détails dans le Chapitre 4.

Il est aussi possible de mettre en place un méta-algorithme qui effectue des itérations sur les hyperparamètres afin de trouver les meilleurs. On peut notamment citer les techniques de *grid search*, ou *grille de recherche*, et de *random search* [BB12], ou *recherche aléatoire*. La première consiste à établir une grille de tous les hyperparamètres et de leurs valeurs possibles puis de la parcourir en entraînant un modèle pour chaque possibilité de la grille. Cette méthode est intéressante tant que le nombre d'hyperparamètres reste bas puisqu'elle peut être très coûteuse en temps de calculs. La seconde méthode, la recherche aléatoire, définit des lois de distributions parmi lesquels chaque hyperparamètre est choisi à chaque itération. Cette technique se révèle généralement plus efficace que la grille de recherche en évitant de garder des hyperparamètres constants entre deux itérations.

Ces algorithmes sont généralement utilisés en conjugaison avec un *ensemble de validation*. Cet ensemble contient des exemples retirés de l'ensemble d'entraînement qui vont permettre de s'assurer que l'apprentissage se passe bien et que les hyperparamètres sont bien réglés. Les exemples de validation ne sont pas utilisés par le modèle pour l'ajustement de ses paramètres et lui sont donc inconnus. Durant l'apprentissage, différentes architectures du modèle sont entraînées puis évaluées sur l'ensemble de validation pour déterminer l'architecture qui fonctionne le mieux sur ces exemples inconnus. Cet ensemble peut aussi être utilisé pour évaluer l'évolution des performances des réseaux au cours de leurs entraînements. Pour finir, il est bon de noter que pour des problèmes similaires, les choix d'hyperparamètres peuvent généralement être réutilisés tout en conservant des bonnes performances.

2.3 Passage au Deep Learning

Les algorithmes de deep learning, ou apprentissage profond, se sont récemment imposés comme très performants dans de nombreux domaines, de la reconnaissance d'image [KSH12] au traitement sonore [HDY+12] en passant par le traitement du langage [SVL14]. Si l'explosion de leur popularité est récente, la théorie qui les encadre, elle, trouve déjà ses origines à partir des années 1940. Leur utilisation de nos jours est principalement due aux progrès récents qui viennent d'un ensemble de facteurs combinant des avancées sur les modèles, les algorithmes d'entraînement et d'optimisation [HOT06] et l'augmentation du nombre de données [LBBH98, KH+09], notamment dans les domaines de l'image et du texte. En conjugaison avec ces avancées, la disponibilité de toujours plus de données et la technologie pour les traiter ont achevé de placer les algorithmes de deep learning au centre de l'attention. L'aspect séparant le plus ces algorithmes des algorithmes classiques de machine learning étant leur capacité à traiter les données en grande dimension.

La démocratisation des cartes *GPU*, pour *Graphics Processing Unit*, à la base pensées pour le rendement graphique des jeux vidéos et spécialisées dans le calcul parallèle, a fortement joué en faveur de l'efficacité des algorithmes de deep learning [RMN09]. En effet, ces processeurs permettent de fortement accélérer l'entraînement de ces algorithmes ce qui a augmenté leur accessibilité. Il faut ajouter à ça la plus grande flexibilité des réseaux de neurones quant à l'ensemble de fonctions qu'ils peuvent approximer. Le théorème d'approximation universelle, ou *universal approximation theorem*, aussi connu comme théorème de Cybenko-Hornik-Funahashi [HSW89, Cyb89], indique qu'un réseau d'une couche est capable d'approximer de manière aussi proche que souhaité n'importe quelle fonction définie sur des sous-ensembles compacts de $\mathbb{R}^d$ à condition que la taille de la couche soit suffisamment grande. C'est là que la force de ces réseaux réside et c'est ce qui motive leur utilisation dans le contexte des attaques par canaux auxiliaires. Leur capacité à approximer des fonctions complexes et à traiter des données de grandes dimensions vient combler une lacune des attaques template. Nous allons maintenant discuter des principales caractéristiques de deux types de réseaux de neurones qui leur donnent cette capacité : les *perceptrons multicouches* et les *réseaux de neurones convolutionnels*.

2.3.1 Réseaux de neurones

Description d'un perceptron

Un perceptron est la plus petite unité présente dans les réseaux de neurones. Il est souvent appelé neurone mais le fonctionnement reste le même. La Figure 2.6 présente un exemple de perceptron sur lequel nous allons nous baser pour décrire les caractéristiques importantes des perceptrons que l'on retrouve dans la plupart des autres réseaux de neurones. En tant qu'algorithme d'apprentissage, le perceptron possède des paramètres à apprendre. Dans cette exemple, ces paramètres sont des *poids*, ou *weights*, représentés par w_1 et w_2 et un *biais* b. Les poids sont des coefficients appliqués aux données d'entrée du perceptron donc si l'entrée du perceptron est de dimension n, il possédera n poids, un pour chaque valeur. Le biais est lui une valeur globale associée à un perceptron, c'est-à-dire que chaque perceptron apprend un unique biais qui lui est propre et qui ne dépend pas de la dimension d'entrée du perceptron. La sortie du perceptron est ensuite calculée de la manière suivante :

$$s = \sigma(x_1 w_1 + x_2 w_2 + b),$$

où σ est une *fonction d'activation*. La fonction d'activation est la partie du perceptron qui apporte généralement de la non-linéarité à l'algorithme. Elle est nécessaire pour obtenir l'expressivité du perceptron, *i.e.* sa capacité d'approximation. Cette fonction est un hyperparamètre du modèle, elle est donc choisie lors de la construction du perceptron. Un choix classique est la fonction *sigmoid* basique définie par :

$$\text{sigmoid}(x) = \frac{1}{1 + \exp(-x)},$$

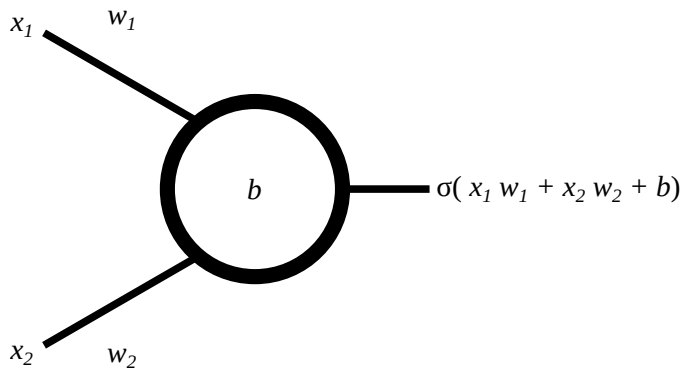


FIGURE 2.6: Exemple de perceptron.

mais il existe un grand nombre de fonction d'activation qui sont utilisés selon les caractéristiques des problèmes à résoudre.

Description d'un réseau de neurones

Un réseau de neurones consiste en un assemblage de perceptrons qui prend la forme de couches. Une couche de perceptrons correspond à un ensemble de perceptrons qui partagent la même entrée. En combinant la sortie de la fonction d'activation de chaque perceptron, on obtient la sortie de la couche. Dans le cas d'un réseau multicouches, la première couche prend en entrée les valeurs d'un exemple puis chacune des couches suivantes prend en entrée la sortie de la couche précédente et ce jusqu'à la sortie de la dernière couche qui correspond à la sortie du réseau. Les réseaux abordés dans ce manuscrit sont dits à *propagation en avant*, ou *feedforward*, c'est-à-dire que le flux de l'information se fait dans une seule direction, de l'entrée du réseau à sa sortie. Les valeurs en sortie du réseau vont dépendre du problème sur lequel il est appliqué, de la classification ou de la régression par exemple. Le nombre de couches ainsi que le nombre de neurones qui les composent font partie des hyperparamètres des réseaux de neurones. Les poids des neurones, qui sont indépendants pour chaque neurone, et les biais de chaque couche, qui s'appliquent aux neurones d'une couche, sont généralement représentés sous forme de matrices $\mathbf{W}$ et de vecteurs $\mathbf{b}$ pour les réseaux de neurones.

Description des Multi-Layer Perceptron : perceptrons multicouches

Le premier type de réseau que l'on va aborder est le perceptron multicouche, ou *Multi-Layer Perceptron* (MLP). Il est la plus simple version d'un réseau de neurones. Comme son nom l'indique, un MLP se compose de plusieurs couches elles-même constituées de neurones. La particularité de ces réseaux est que les couches qui les composent sont *entièrement connectées*, ou *fully-connected*. Cela veut dire que chaque neurone est connecté à tous les neurones qui composent la couche suivante. Un exemple de cette propriété est illustré dans la Figure 2.7. On y voit un réseau à trois couches, la couche d'entrée, une couche intermédiaire, aussi appelé *couche cachée* ou *hidden layer*, et une de sortie. L'intérêt d'avoir des couches entièrement connectées réside dans le fait qu'elles permettent de propager l'information à travers l'entièreté du réseau. Ce partage des informations fait que ce type de ré-

seau est particulièrement utile pour recouper des informations et les traiter afin répondre au problème donné.

L'utilisation des poids et des biais a déjà été mentionnée mais nous allons maintenant voir comment ils sont combinés avec l'entrée du réseau pour obtenir une sortie qui réponde au problème posé. Soit $\mathbf{x}$ le vecteur d'entrée d'une couche du réseau, une première transformation linéaire lui est appliquée par chaque neurone de la couche suivante :

$$u_i = (\mathbf{x}\mathbf{W}_i) + \mathbf{b}_i,$$

où u_i est la sortie de la couche i, $\mathbf{W}_i$ est la matrice de poids de la couche i et $\mathbf{b}_i$ est le biais associé à cette couche. Comme précédemment, une fonction d'activation est appliqué à chaque élément de la sortie de la couche. Dans tous les réseaux étudiés dans ce manuscrit, sauf mention contraire, cette fonction sera la fonction d'*unité linéaire réctifiée* ou *ReLU* (Rectified Linear Unit). La fonction ReLU a été montrée comme étant plus efficace que les autres fonctions d'activation dans le cadre des analyses par canaux auxiliaires [BPS+19]. Elle est définie de la manière suivante :

$$\mathrm{ReLU}(x) = \begin{cases} 0, \text{ if } x < 0 \\ x, \text{ if } x \geq 0 \end{cases}.$$

Dans le cadre de l'application de réseaux de neurones aux attaques par canaux auxiliaires, on cherche à faire une classification de la trace d'entrée pour pouvoir prédire quelle est la valeur de la variable intermédiaire liée à cette trace. On souhaite donc obtenir une valeur de probabilité sur chacune des valeurs possibles qui nous indiquera la vraisemblance de chaque hypothèse. Afin d'obtenir cette distribution de probabilités faite par le réseau, la fonction d'activation de la dernière couche est une fonction *softmax*. Cela permet de transcrire l'estimation du réseau sous forme de pseudo-probabilité étant donné que la fonction softmax projette les valeurs d'entrée sur des valeurs comprises entre 0 et 1 en s'assurant que ces valeurs somment à 1. La définition de cette fonction est la suivante : étant donné un vecteur $\mathbf{x}$ de taille n, le résultat du softmax de $\mathbf{x}$ est $\mathrm{Softmax}(\mathbf{x}) = (s_i)_{1 \leq i \leq n}$ où :

$$s_i = \frac{e^{\mathbf{x}[i]}}{\sum_{j=1}^{n} e^{\mathbf{x}[j]}}.$$

Maintenant que ces points ont été abordés, nous allons pouvoir passer au deuxième type de réseau traité dans ce manuscrit, les réseaux de neurones convolutionnels. Les expériences présentent dans les chapitres suivantes seront principalement faites en utilisant ce type de réseau.

Réseau de neurones convolutionnels

La spécificité des réseaux de neurones convolutionnels, ou *Convolutional Neural Networks* (CNN) en anglais, est l'utilisation d'opérations de convolution comme illustré dans la Figure 2.8a. L'opération de convolution consiste à appliquer un filtre $\mathbf{w}$ sur les données d'entrée $\mathbf{x}$ d'une couche, c'est-à-dire effectuer des multiplications élément par élé-

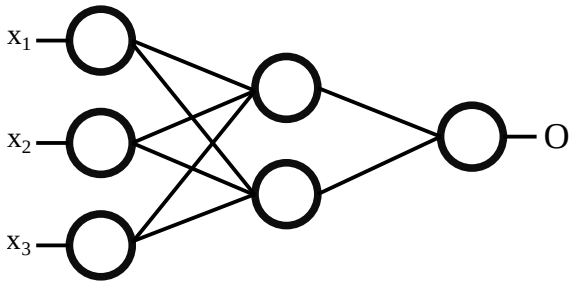


FIGURE 2.7: Exemple de MLP à trois couches.

ment puis sommer le tout. L'opération de convolution discrète peut être décrite de la façon suivante :

$$(\mathbf{x} * \mathbf{w}) = \sum_{i=1}^{n} \sum_{j=1}^{m} \mathbf{x}[i]\mathbf{w}[j],$$

où $\mathbf{x}$ est un vecteur de dimension n et $\mathbf{w}$ un vecteur de dimension m. La taille d'un filtre correspond aux nombres de poids qu'il contient et le nombre de filtres utilisés est fixé indépendamment pour chaque couche. Chaque filtre parcourt l'entièreté des données d'entrée en se décalant, à chaque application, d'une valeur fixe appelée *stride*. La taille et le nombre des filtres ainsi que la stride sont des hyperparamètres à fixer avant le début de l'apprentissage. Quand la taille du filtre n'est pas un diviseur de la taille des données d'entrée, ces données sont complétés en rajoutant des valeurs à ses extrémités autant que nécessaire. Cette pratique s'appelle le *padding* des données et il en existe plusieurs types comme le *same padding* qui correspond à répéter la valeur extrême ou le *zero padding* qui consiste à rajouter des 0. Dans le reste du manuscrit, nous utiliserons un *same padding* lors de l'entraînement des réseaux.

Tout ceci constitue la partie linéaire du réseau de neurones mais comme précédemment, elle n'est pas suffisante pour permettre d'approximer une fonction arbitraire. L'ajout de non-linéarités se fait de la même façon que pour les MLP à l'aide des fonctions d'activation. Elles sont appliquées cette fois au résultat de la partie convolutionnelle. On peut donc résumer la sortie du neurone n de la couche l à l'aide de la formule suivante :

$$\mathbf{x}^{(l)}[n] = \mathrm{ReLU}\big(\mathbf{W}_n^{(l)} * \mathbf{x}^{(l-1)} + \mathbf{b}^{(l)}\big),$$

où $\mathbf{W}_n^{(l)}$ est la matrice contenant les poids des filtres appliqués au neurone n de la couche l, $\mathbf{b}^{(l)}$ est le vecteur de biais de cette même couche, $*$ représente l'opération de convolution et $\mathbf{x}^{(l-1)}$ et $\mathbf{x}^{(l)}$ sont des vecteurs de dimension la taille des couches et les sorties respectives des couches $l-1$ et l.

La Figure 2.8a illustre les opérations linéaires. Dans cet exemple, la taille et le nombre de filtres sont de 2 et la valeur de stride est de 1 donc la dimension de la sortie est le double de celle d'entrée. Il faut faire attention à cette augmentation de dimension puisqu'elle augmente aussi le nombre de neurones de chacune des couches qui va suivre. Si elle n'est pas contrôlée, l'augmentation de la dimension va entraîner des calculs trop importants et fortement ralentir l'entraînement et l'utilisation du réseau. Pour prévenir ce phénomène, on

utilise en général des *pooling layers*, ou couches de pooling. Ces couches ont pour but de réduire la dimension des données en les combinant par paquet à l'aide d'une opération choisie, comme la moyenne ou le maximum, à la création du réseau. Cette opération cherche à résumer les données en perdant le moins d'information possible, elle varie donc selon le contexte d'utilisation du réseau. Dans la Figure 2.8b, l'opération de pooling a une taille de 2 et une stride de 2 aussi ce qui résulte en une réduction de dimension par un facteur 2. L'opération utilisée est une moyenne mais il est aussi courant d'utiliser le maximum ou le minimum à la place. Dans le reste du manuscrit, nous utiliserons l'opération *average pooling* suivant les recommandations faites par l'ANSII [BPS+19], qui calcule la moyenne des valeurs dans sa fenêtre.

Lors de l'utilisation de réseaux convolutionnels, il est courant de laisser une partie entièrement connectée en fin de réseau après la partie convolutionnelle. L'idée derrière cette méthode est que les convolutions du réseau vont servir à extraire les informations utiles des données d'entrée puis la partie connectée va utiliser ces informations pour conclure sur la solution au problème. Cette dernière partie représente donc les quelques dernières couches du réseau. Elle se conclut par une fonction *softmax* comme introduit dans la partie précédente.

Pour résumer l'utilisation des réseaux du point de vue d'un attaquant, ce dernier fournit la trace de consommation de courant $\mathbf{t}$ en entrée au réseau qui va lui donner un ensemble de prédictions représentés par la sortie du softmax et qui dépend du modèle de fuite utilisé. Par exemple, pour le modèle de fuite identité d'un octet, il y aura 256 valeurs en sortie du réseau qui constitueront la prédiction du réseau. Chacune de ces valeurs est associée à ce qu'on appelle une *classe*. Les prédictions en sortie du réseau peuvent être interprétées comme les valeurs des templates $F_k(\mathbf{t})$, qui représentent la probabilité que la clé utilisée pour obtenir la trace $\mathbf{t}$ est k, calculées pour toutes les valeurs possibles de k.

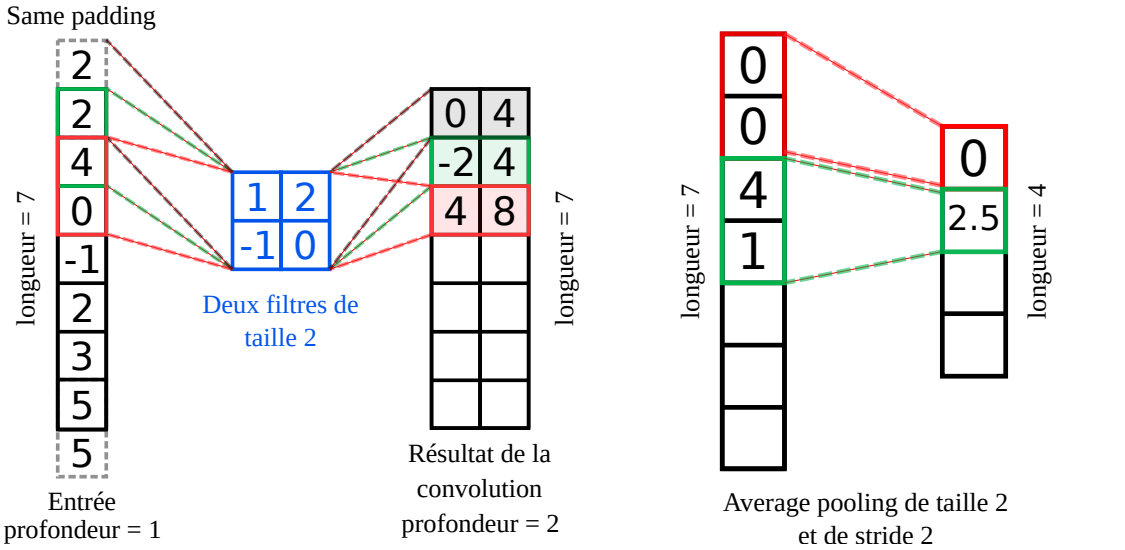
Nous allons maintenant discuter des méthodes utilisées lors de l'entraînement d'un réseau.

Description de la phase d'entraînement

Comme dit précédemment, les réseaux de neurones sont composés de deux types d'opérations, les opérations linaires et les opérations non-linéaires. Elles sont le résultat de la phase d'entraînement du réseau qui va ajuster les poids de chaque neurone ou filtre et les biais de chaque couche afin d'obtenir un réseau qui approxime le mieux la fonction objectif f^*. Dans cette partie, les détails de l'entraînement d'un réseau de neurones sont décrits plus précisément.

Le processus d'apprentissage reprend les concepts introduits dans la Section 2.2 avec quelques ajouts. Comme évoqué précédemment, une fonction de perte est choisie afin de représenter au mieux le problème à résoudre. Lors de l'application des réseaux de neurones aux attaques par analyse de canaux auxiliaires, la fonction de perte est généralement l'*entropie croisée catégorique* [BPS+19] définie par :

$$CCE(p, q) = -\sum_x p(x) log(q(x)),$$



(A) Exemple d'une opération de convolution (B) Exemple d'une opération de average pooling

FIGURE 2.8: Exemple d'une opération de convolution utilisant deux fitres de taille 2, un same padding et une stride de 1, et d'une couche de average pooling de taille 2 et de stride 2.

où p est la distribution attendue, q la distribution prédite par le réseau, c'est-à-dire la sortie du Softmax, et x représente les valeurs des classes ou labels possibles. La CCE permet de comparer deux distributions de données et d'évaluer leur ressemblance, ce qui répond bien à notre problème puisque l'on cherche à apprendre au réseau à prédire une distribution de probabilité. Le gradient de la CCE en fonction des paramètres du réseau est donc calculé afin de déterminer les modifications à appliquer aux paramètres pour réduire les erreurs du modèle sur les données d'entraînement. Nous n'avons cependant pas encore vu la méthode de calcul de ce gradient. Cette méthode, appelée *rétro-propagation* ou *backpropagation*, consiste à faire remonter l'information issue du calcul de la fonction de perte à travers le réseau afin de calculer les gradients en fonction de chacun des paramètres. Nous ne rentrerons pas dans les détails des calculs impliqués dans l'utilisation de retro-propagation. Nous indiquerons simplement qu'ils se basent en grande partie sur des calculs de dérivées basés sur la règle de dérivation en chaîne qui permet d'obtenir le gradient de la fonction de perte en fonction de chacun des paramètres du réseau. La retro-propagation est donc utilisée pour pouvoir appliquer l'algorithme de descente de gradient.

Cet algorithme est généralement légèrement modifié quand il est utilisé pour les réseaux de neurones. En effet, les réseaux de neurones utilisent généralement des ensembles d'entraînement très large, il est donc parfois compliqué d'appliquer efficacement la descente de gradient pour tous les exemples de ces ensembles. On peut donc découper l'ensemble d'entraînement en sous-ensembles appelés *batch*, ou *mini-batch*, composés d'exemples choisis aléatoirement. Les modifications à apporter au réseau sont moyennées pour chaque batch et la descente de gradient est ensuite appliquée pour chacun de ces batchs et on parle alors de *descente de gradient stochastique*, ou *stochastic gradient descent*. Les batchs sont composés d'exemples choisis aléatoirement au début de chaque epoch et les exemples ne peuvent être présents que dans un seul batch. On peut adapter l'Algorithme 1 pour ob-

tenir un nouvel algorithme prenant en compte les mini-batch décrit dans l'Algorithme 2. Cette fois-ci, ce n'est plus le gradient $\mathbf{g}$ qui est calculé à chaque batch mais une estimation $\hat{\mathbf{g}}$ dû au faite que seule une sous-partie de l'ensemble d'entraînement est utilisé dans le calcul.

Algorithme 2: Descente de gradient stochastique

Données : Paramètres $\boldsymbol{\theta}$, learning rate ϵ;
tant que Condition de performance non remplie **faire**
  Création des batchs de taille m à partir d'exemples aléatoires de l'ensemble d'entraînement T;
  Calcul de l'estimation du gradient : $\hat{\mathbf{g}} \leftarrow +\frac{1}{m}\nabla_{\boldsymbol{\theta}} \sum_{i=1}^{m} \mathcal{L}(f(\mathbf{x}_i;\boldsymbol{\theta}), \mathbf{y}_i)$;
  Mis à jour des paramètres : $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon\hat{\mathbf{g}}$.
fin

La taille des batchs est un hyperparamètre du modèle qui est choisi en fonction de facteurs multiples qui incluent par exemple l'utilisation ou non de GPUs, la quantité de mémoire disponible, la présence d'une architecture multi-cœurs [GBC16]. Il est bon de retenir principalement que le choix de cet hyperparamètre est le résultat de la recherche d'un compromis entre temps de calculs de l'entraînement et performances du réseau.

Une fois les hyperparamètres choisis, l'entraînement peut commencer. L'initialisation des poids du réseau peut se faire de plusieurs façons selon les problèmes à résoudre. Quelques exemples inclus des initialisations à valeurs constantes telles que tous les poids à 0 ou à 1 mais plus généralement les premières valeurs des poids suivent une distribution aléatoire, soit uniforme, soit avec une loi prédéfinie [GB10, SMG13, M$^+$10]. Dans les réseaux que nous verrons par la suite, la loi suivie est une loi normale de moyenne 0 et d'écart-type 1. Le nombre d'epochs d'entraînement nécessaires pour obtenir un réseau efficace dépend de beaucoup de paramètres : le réseau en lui-même et son architecture, l'ensemble d'entraînement et notamment le nombre d'exemples qu'il contient, et la fonction de perte choisie entre autres.

Une fois démarré, il est nécessaire de pouvoir évaluer correctement l'entraînement afin de d'analyser les potentiels problèmes rencontrés comme l'overfitting et l'underfitting par exemple. Nous allons donc discuter des façons d'évaluer un modèle durant son entraînement et de le régulariser afin de mieux contrôler l'entraînement.

2.3.2 Évaluer et régulariser

Nous allons maintenant voir le processus de décision responsable de la continuation ou de l'arrêt de l'entraînement.

Évaluer Parmi les éléments permettant d'évaluer un réseau lors de son entraînement, la valeur moyenne de la fonction de perte sur l'ensemble des exemples d'entraînement, *i.e* le risque empirique, est l'un des plus importantes. En suivant l'évolution du risque empirique, il est possible de voir si le réseau continue de s'améliorer et s'il est encore capable de réduire ses erreurs. Il faut toutefois faire attention lorsque le risque empirique

approche zéro. Cela peut en effet mener à de l'overfitting, qui correspond au fait qu'un réseau, après avoir passer trop de temps à s'entraîner sur l'ensemble d'entraînement, finisse par perdre en performance sur l'ensemble de validation et *a fortiori* sur l'ensemble de test. L'overfitting peut venir de plusieurs sources différentes et son effet dépend fortement du type de réseau utilisé mais il apparaît le plus souvent sur des réseaux aux architectures très, voir trop, complexes par rapport à la difficulté du problème à résoudre. Cette complexité permet au réseau d'apprendre à prédire les exemples d'entraînement parfaitement et donc d'avoir un risque empirique proche de zéro mais cela demande un nombre important d'epochs. Le problème d'apprendre ces exemples par cœur est que cela implique généralement une forte perte dans la capacité de généralisation du réseau qui ne saura plus prédire d'exemples dès qu'ils différeront légèrement de ce qu'il a vu à l'entraînement. Ce comportement est illustré dans l'exemple de la Figure 2.4.

Il existe aussi le problème inverse à l'overfitting, l'*underfitting*, qui cette fois-ci correspond à un sous-apprentissage du réseau. Cela signifie que, soit le réseau n'a pas vu assez de fois les exemples d'entraînement pour en extraire une information suffisante pour pouvoir prédire des nouveaux exemples, soit il n'est pas assez complexe pour apprendre à résoudre ce problème. Dans le premier cas, il faut augmenter le nombre d'epochs d'entraînement et dans le second, augmenter la complexité du réseau, tel que le nombre de couches par exemple. Ces deux phénomènes sont importants à prendre en compte puisqu'ils impactent directement les performances du réseau lors des tests post-entraînement, il est donc nécessaire d'être capable d'évaluer l'entraînement du réseau pendant qu'il a lieu afin de déterminer le meilleur moment pour l'arrêter.

Comme vu avec l'overfitting, il n'est donc pas recommandé de se fier exclusivement à la valeur de la fonction de perte évaluée sur les exemples d'entraînement pour estimer de la qualité de l'apprentissage. C'est là qu'intervient l'ensemble de validation mentionné précédemment. En effet, les exemples présents dans cet ensemble n'ont jamais été vus par le réseau, ils sont donc plus représentatifs de ce à quoi le réseau sera confronté dans ces utilisations futures, notamment des exemples de tests. Il est alors possible d'évaluer la fonction de perte sur cet ensemble afin de voir comment réagit le réseau sur des exemples inconnus. C'est cette évaluation qui sera déterminante dans le choix de l'arrêt ou non de l'entraînement. On peut aussi noter que d'autres métriques peuvent être intéressantes à utiliser pour mesurer différents aspects de la capacité de prédictions du réseau. Dans le cadre d'une classification à plusieurs classes, l'*accuracy* est souvent utilisée pour mesurer le nombre de fois, en pourcentage, où le réseau prédit la bonne valeur. Si l'on constate que la valeur de ces métriques commence à baisser régulièrement après avoir augmenté grâce à l'apprentissage, ou inversement, on peut en déduire que le phénomène d'overfitting commence. Cette stratégie d'analyse est la même pour toutes les métriques d'évaluation et indique le moment idéal pour arrêter l'entraînement. Le choix des métriques utilisées dépend du problème à résoudre et est fait afin qu'elles apportent des informations sur l'efficacité du réseau. Nous verrons par la suite une métrique d'évaluation créée spécifiquement pour évaluer les performances du réseau dans le cadre d'attaques par canaux auxiliaires, là où les métriques plus classiques se révèlent insuffisantes.

Régulariser Une fois l'évaluation du réseau mise en place, il est possible d'appliquer de nouvelles techniques qui se chargeront de *régulariser* le réseau. La régularisation d'un réseau correspond à toutes modifications faites à l'entraînement dans le but de réduire l'erreur de généralisation sans prendre en compte l'erreur d'entraînement. Ces techniques cherchent à améliorer les performances du réseau sur les données de validation et de test quitte à ce que les erreurs sur les données d'entraînement augmentent. En effet, il ne faut pas perdre de vue l'objectif final de l'apprentissage qui est d'être performant sur le plus grand nombre d'exemples possibles. Il est donc intéressant de trouver un compromis entre les performances sur les données d'entraînement et celles sur les données de test.

Une des techniques de régularisation les plus communes est l'*early stopping*. Elle consiste à arrêter l'apprentissage lorsque les performances sont les meilleures sur les données de validation. Les informations sur ces performances sont fournies par les métriques d'évaluation. Ce type de régularisation agit seulement sur l'entraînement du réseau sans appliquer de pénalisation sur les poids, contrairement à la norme L_2, et donc n'influence pas le réseau dans les valeurs possibles de ses paramètres. Il est alors possible de cumuler l'early stopping avec d'autres types de régularisation comme le *weight decay*, ou *régularisation L2*, et le *dropout*. Ces dernières techniques influencent directement les valeurs des poids en leur appliquant une pénalisation pour restreindre l'importance qu'ils peuvent prendre dans la valeur de sortie du réseau. Nous reviendrons plus en détails sur ces méthodes dans un chapitre qui leur est consacré dans la suite du manuscrit.

2.4 Le Deep Learning appliqué aux attaques par canaux auxiliaires

L'utilisation du deep learning dans le cadre des attaques par canaux auxiliaires présente plusieurs avantages notables. Tout d'abord, le deep learning permet de se passer des phases de détections des PoI et de prendre en compte l'entièreté des traces lors de ses traitements. Cela veut dire qu'il n'y a plus de risques de pertes d'informations liés aux restrictions imposées par la réduction de dimension. L'exploitation des traces par le réseau est donc au moins aussi complète que lors de la création de template. On peut aussi noter que les besoins en resynchronisation sont bien moindre avec l'utilisation des CNN grâce à l'utilisation des filtres convolutionnels [CDP17, BPS+19]. Ils permettent de détecter une fuite d'information quelle que soit sa position dans la trace. En effet, durant les opérations de convolutions à base de filtre, seuls les points proches les uns des autres sont pris en compte. Cela signifie que, dans le cadre de traces de consommation de courant, les points temporellement proches sont utilisés dans les calculs. Cette nouvelle approche soulève donc des problématiques importantes dans le domaine de la sécurité. Nous allons dans un premier temps retracer les travaux majeurs publiés dans ce domaine, présents sur la fresque Figure 2.9, avant d'analyser les différentes directions prises.

2.4.1 Utilisations du Machine Learning et des réseaux de neurones pour des attaques par canaux auxiliaires

La proximité et l'influence de la cryptologie sur le machine learning et inversement sont des sujets loin d'être récents. En 1991, Rivest publie un article [Riv91] où il prend

soin de détailler la ressemblance des concepts et des problématiques traités. Cela permet de constater que les interactions entre ces deux sujets précèdent de plusieurs années les premières attaques par analyse de canaux auxiliaires et également l'utilisation du machine learning dans le cadre d'attaques.

Il faut attendre 2004 pour trouver la première utilisation de machine learning, sous forme de réseau de neurones, pour effectuer une attaque par analyse de canaux auxiliaires [AA04]. Dans ces premiers travaux, le canal utilisé est le canal acoustique dans le but de retrouver les mots tapés sur un clavier d'ordinateur seulement à partir du son émis. Il était donc logique de se tourner vers les réseaux de neurones qui commençaient à montrer leur efficacité dans le domaine de la reconnaissance sonore [PWRZ00]. Cette attaque est améliorée quelques années plus tard [ZZT09] et montre bien l'efficacité de cette approche. Elle est reprise en 2010 avec cette fois-ci une utilisation de modèle de machine learning plus classique, comme les *Hidden Markov Models*, pour retrouver les mots imprimés par un certain type d'imprimante [BDG+10]. En utilisant des algorithmes de machine learning, les auteurs arrivent à traiter les données sonores et établir un lien avec les mots imprimés. Cette utilisation est proche du concept de classification et permet de réaliser une attaque similaire à une attaque template contre les algorithmes de chiffrement.

L'année suivante, des premiers tests sont réalisés pour évaluer le potentiel des algorithmes de machine learning, dans ce cas les Support Vector Machine (SVM) [HGDM+11], pour effectuer des attaques par analyse de canaux auxiliaires contre un AES. Ils concluent sur une efficacité similaire aux attaques templates et encourage à pousser les tests plus loin afin de mieux comprendre l'apport possible de ces algorithmes. La même année, d'autres travaux [LBM11] se concentre sur d'autres utilisations possibles des algorithmes de machine learning. Ils réussissent à appliquer ces techniques pour faire de la réduction de dimension sur des traces de consommation de courant et utilisent différentes combinaisons d'algorithmes de machine learning pour attaquer un triple DES. Ces travaux ont été complétés par la suite avec une attaque contre un AES utilisant un SVM pour prédire les poids de Hamming des variables ciblées [HZ12, BLR12]. Une autre approche prenant en compte l'aspect temporelle des traces [LBTM13] réussie à attaquer un DES d'une manière adaptable à d'autres algorithmes de chiffrement symétrique. Une attaque SPA est aussi montée contre le DES à l'aide de SVM et arrive à récupérer une grande partie de la clé avec une bonne précision [HJZ12].

Quelques années plus tard, des chercheurs parviennent également à utiliser un algorithme de machine learning afin de faire du démasquage [LBM15b], c'est-à-dire trouver la valeur du masque utilisée lors du chiffrement pour pouvoir attaquer la partie masquée et ainsi retrouver la valeur de la variable intermédiaire. Cela leur permet ensuite d'appliquer une attaque par canaux auxiliaires classique. D'autres travaux [LBM14] utilisent eux une combinaison d'algorithme de réduction de dimension et de classification pour attaquer une clé de triple DES et une de RSA, un algorithme de chiffrement asymétrique. Ils obtiennent des performances similaires à une attaque template faite sur les mêmes cibles et soulignent l'avantage du machine learning pour traiter de données en grande dimension ainsi que le bénéfice obtenu par la non-paramétrisation des algorithmes de machine learning par rapport aux attaques templates. Cela signifie que le fait de ne pas donner

d'apriori au modèle sur la distribution des fuites lui offre plus de liberté que lors de la création des templates qui se base sur l'hypothèse d'un bruit gaussien. La recherche sur le machine learning appliqué aux attaques par analyse de canaux auxiliaires continue notamment avec une attaque montée contre un AES sur des variables non gaussienne [PB14], *i.e.* dont la distribution du bruit ne suit pas une distribution normale, et des tests concernant les problèmes de dimensionnalité des traces [LPB+15, BOW15]. Ces derniers introduisent également la notion de décomposition biais-variance de l'erreur des modèles [LBM15a]. Les attaques basées sur le machine learning les plus récentes se concentrent elles sur l'optimisation des hyperparamètres des modèles [PHJ+17].

La première utilisation notable de deep learning [MZ13] est faite contre un AES à l'aide d'un MLP et même si les résultats se révèlent insuffisants, la méthode est jugée prometteuse. Après quelques raffinements, une nouvelle attaque utilisant un MLP est montée contre un AES masqué [MZVT15, GHO15]. Elle est faite en deux temps sur l'ensemble de données du DPAContest v4.1 [BBD+14], un concours ouvert proposant de soumettre des attaques contre une implémentation protégée d'AES. La première partie de cette attaque utilise un MLP pour récupérer des informations sur le masque utilisé lors du chiffrement, puis un deuxième MLP sert à la récupération de la clé. Ce même principe est réutilisé contre la version 4.2 du DPAContest mais cette fois-ci c'est la valeur de désynchronisation qui est récupéré par le premier MLP et l'attaque est alors possible à l'aide du second [MDM16]. Ces attaques marquent le début de l'utilisation de modèles de deep learning à des fins d'attaques par canaux auxiliaires. A partir de ce moment, la recherche dans ce domaine se concentre principalement sur les réseaux de neurones et ce qu'ils peuvent apporter aux attaques contre les implémentations protégées.

C'est notamment le cas en 2017 avec deux publications par Cagli *et al.* [CDP17] et Maghrebi *et al.* [MPP16]. La première explore les techniques d'augmentation des données, ou *Data Augmentation*, qui consistent en une augmentation de la taille de l'ensemble d'entraînement par l'ajout de nouveaux exemples créés à partir d'exemples d'entraînement via l'application d'une transformation, *e.g.* ajout de désynchronisation ou de *jitter* qui simule des dispersions temporelles. Cela mène à l'amélioration des performances de plusieurs réseaux CNN et MLP. La deuxième publication se concentre sur une comparaison détaillée entre les résultats d'une attaque template et ceux d'attaques à base de CNN et MLP. Ils concluent que ces derniers, lorsqu'ils sont bien choisis, obtiennent de meilleures performances que l'attaque template. L'année suivante voit elle aussi de nombreux articles s'intéressant à cette utilisation nouvelle du deep learning. Benadjila *et al.* [BPS+19] et l'ANSSI mettent en place une base de données publique, ASCAD, contenant à la fois un ensemble de traces provenant d'un AES maqué, des réseaux de neurones déjà entraînés issus d'une recherche d'hyperparamètres, décrite dans leur article, et des scripts modifiables permettant l'entraînement de MLP et de CNN. Outre l'intérêt de la publication de réseaux et de scripts utilisables par les chercheurs du domaine, la mise en place de la base de données permet d'établir un point de comparaison sur lequel tester les différentes architectures. Ce concept est central dans le domaine du machine learning où la détermination des meilleurs réseaux se fait généralement de manière empirique. Deux autres articles, cette même année, commencent à explorer non plus l'amélioration des réseaux mais leur compréhension.

Picek *et al.* [PHJ+19] arrivent à la conclusion que les métriques traditionnellement efficaces en machine learning ne sont pas adaptées pour une application aux attaques profilées basées sur les réseaux de neurones. Masure *el al.* [MDP19], eux, utilisent des techniques de visualisations des gradients afin de faire de la localisation de points d'intérêts *a posteriori*. C'est-à-dire qu'à partir d'un réseau déjà entraîné, ils arrivent à localiser les points de la trace d'entrée sur lesquels le réseau récupère son information. Même si l'intérêt est limité, c'est un premier pas vers une compréhension des mécaniques qui permettent aux réseaux d'apprendre à prédire les valeurs intermédiaires.

Cette tendance continue en 2020 avec divers travaux qui ne se concentre plus seulement sur l'amélioration des performances des réseaux mais aussi sur leur compréhension. Zaid *et al.* [ZBHV20a] mettent au point une méthode de construction de réseaux efficaces en se basant sur des techniques de visualisations des poids et des gradients. Zhang *et al.* [ZZN+20] proposent le *Cross Entropy Ratio*, qui peut servir de métrique d'évaluation du réseau mais principalement de fonction de perte spécialisée dans le contexte de données déséquilibrées. Cela permet notamment un apprentissage plus efficace lors de l'utilisation des modèles de fuite en poids de Hamming ou distance de Hamming. Par la suite, Perin *et al.* [PBP] proposent eux aussi une métrique d'évaluation de l'apprentissage se basant sur la notion de *chemin d'information*, qui cherche à évaluer comment l'information se propage dans le réseau en passant d'une couche à une autre. Parmi les travaux récents, on peut enfin cité ceux Zaid *et al.* [ZBD+21] qui mettent au point une fonction de perte dédiée à l'entraînement des réseaux utilisés pour les attaques par canaux auxiliaires. Cette fonction privilégie l'attribution d'une plus forte valeur de probabilité à la bonne classe que le réseau s'entraîne à prédire. Cela se traduit lors de la phase d'attaque par la possibilité d'utiliser moins de traces pour récupérer la valeur de la clé.

2.4.2 Conclusion

La première étape lorsqu'on traite de nouvelles attaques est de bien comprendre leur fonctionnement. Dans le cas des attaques par canaux auxiliaires profilées basées sur des réseaux de neurones, le principe d'attaque reste le même que pour l'attaque template qui a déjà été étudiée en profondeur. Il reste alors à se concentrer sur les nouveautés apportées, ici l'utilisation de réseaux de neurones pour la caractérisation des consommations de courant.

Comme nous avons pu le voir, l'utilisation de techniques de machine learning dans le but de réaliser des attaques par canaux auxiliaires n'est pas si récente. Toutefois, la majeure partie des premiers travaux portant sur ce sujet se concentre sur la faisabilité et l'efficacité de telles attaques, la première étape étant de voir si cette méthode peut effectivement remplacer l'attaque template. Il est aussi important de mentionner l'utilisation de ces techniques à des fins de pré-traitement des traces qui est aussi un domaine très important des analyses par canaux auxiliaires. Au fil du temps et avec le début de l'utilisation d'algorithmes de deep learning, le doute quant à l'efficacité de ce type d'attaque a disparu. Il a cependant laissé place à un autre point tout aussi important qui est le besoin de compréhension et d'interprétation de ces algorithmes. C'est là que la difficulté réside puisque, malgré leur application de plus en plus importante dans de nombreux domaines, le fonctionne-

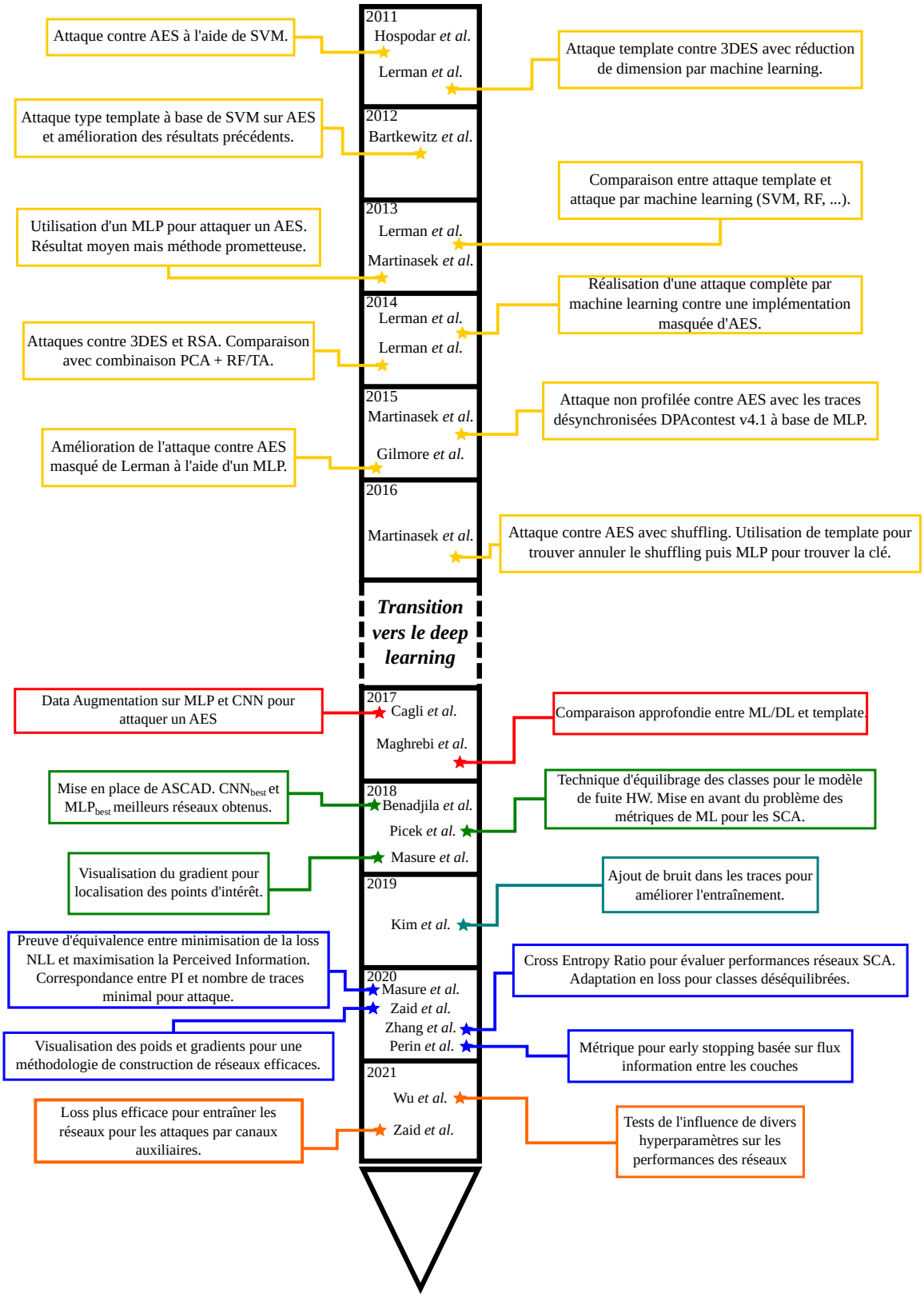


FIGURE 2.9: Fresque résumant les articles majeurs portant sur l'application des techniques de deep learning aux attaques par canaux auxiliaires.

ment des réseaux de neurones reste obscure. En effet, la théorie explique comment ces réseaux arrivent à apprendre à résoudre les tâches demandées mais elle ne donne aucune garantie sur une possible interprétation de leur fonctionnement. Une partie de la recherche en Deep Learning se concentre justement sur l'interprétabilité du comportement des réseaux. Pour certaines applications, il est possible, en analysant les poids et les couches de comprendre le "raisonnement" et la prise de décision des réseaux. Ce besoin a fait l'objet de travaux récents apportant des débuts de réponses.

On voit donc que coexiste en ce moment deux directions de la recherche sur ce sujet : l'amélioration des performances des réseaux via l'application de techniques utilisées en machine learning et la compréhension de l'apprentissage et des prédictions du réseau afin de mieux caractériser la menace que représentent ces attaques. Nous allons explorer dans cette thèse ces deux axes, d'une part avec une nouvelle métrique d'évaluation de l'apprentissage du réseau permettant de mieux évaluer l'application de techniques de machine learning et d'autre part en explorant une nouvelle méthode d'attaques par canaux auxiliaires se basant en partie sur la confiance des réseaux dans leurs prédictions et permettant d'amélioration les résultats d'attaques classiques.

Partie II

Contributions

Chapitre 3

Comprendre et évaluer les performances

Après avoir introduit les attaques par canaux auxiliaires profilées et l'utilisation des réseaux de neurones pour celles-ci, nous allons maintenant nous pencher sur la problématique importante de l'évaluation des réseaux. Comme nous avons pu le voir dans la Section 2.3.2, cette évaluation est nécessaire à la fois pour le processus d'entraînement mais aussi pour être capable de distinguer quelle architecture est la plus performante. Dans ce chapitre, nous allons dans un premier temps recontextualiser les différentes évaluations existantes des réseaux de neurones avant de discuter des travaux apportant des débuts de réponses quant à l'évaluation des réseaux de neurones utilisés pour les attaques par canaux auxiliaires profilées. Une fois ce contexte établi, nous décrirons les travaux qui ont conduit à la création d'une nouvelle métrique d'apprentissage et d'évaluation afin de répondre à cette problématique et présenterons les résultats obtenus sur la base de données ASCAD.

3.1 L'importance de l'évaluation des réseaux de neurones

Si l'évaluation des performances des réseaux de neurones est importante pour s'assurer que l'entraînement se passe bien et que les performances des réseaux sont bonnes, elle est aussi importante au niveau des évaluations de sécurité. La caractérisation de l'efficacité des attaques et de la sécurité des systèmes est une question essentielle et cette caractérisation n'est pas toujours facile à obtenir avec l'utilisation de réseaux de neurones. Pour cela, l'idéal est d'avoir une métrique d'évaluation des réseaux afin de juger de la menace qu'ils représentent, c'est-à-dire leur capacité à réussir les attaques et retrouver la clé.

De plus, comme on a pu le voir au cours du Chapitre 2, les performances d'un réseau sont liées à un ensemble de paramètres et d'hyperparamètres très variés. De manière générale, il est impossible de déterminer si un réseau entraîné est optimal et donc si l'attaque qui se base sur ce réseau est la meilleure possible. Pour d'autres types d'attaques par canaux auxiliaires, comme les attaques templates [CRR03], il est possible de déterminer si l'exploitation des fuites d'information est optimale mais ces attaques reposent sur une théorie bien comprise ce qui n'est pas le cas quand un réseau de neurones est utilisé. Les métriques ont alors une importance particulière puisqu'elles nous permettent d'évaluer certains aspects des réseaux entraînés, comme le taux de bonnes prédictions ou la

quantité d'erreur. Il faut les construire avec attention afin d'en retirer le plus d'information possible sur l'application qui sera faite du réseau.

L'évaluation des réseaux de neurones est également importante en deep learning pour s'assurer que les paramètres appris sont pertinents pour la tâche visée. Il est donc nécessaire d'avoir un moyen de juger de la qualité des paramètres entraînés. Ce jugement se fait en comparant deux états de paramètres différents vis-à-vis d'une grandeur mesurable à l'aide d'*une métrique*. Cela rend possible deux types de comparaisons différentes : une première qui a lieu durant la phase d'entraînement et qui compare la valeur de cette métrique entre différentes instances du réseau, par exemple entre deux epochs consécutives, et une deuxième qui, elle, a lieu après la fin de l'apprentissage et qui consiste à comparer la métrique pour deux architectures entraînées. Le but de la première évaluation est de vérifier que l'entraînement se déroule de manière appropriée. Elle permet aussi d'appliquer la méthode d'early stopping, discutée Section 2.3.2 et sur laquelle nous reviendrons dans ce chapitre, qui permet d'optimiser les performances du réseau en sortie d'apprentissage. La seconde évaluation sert généralement à déterminer les meilleurs hyperparamètres, c'est-à-dire les paramètres qui sont fixés avant l'entraînement et ne changent pas durant celui-ci, en comparant la métrique pour les deux réseaux. Cela permet de conclure sur l'impact qu'ont les différents hyperparamètres sur l'entraînement et les performances du réseau.

Il est important de noter que les descriptions faites ici se concentrent sur l'utilisation d'une seule métrique pour comparer deux réseaux mais, dans la pratique, il existe une multitude de métriques différentes qui évaluent différents aspects de l'apprentissage. Il est donc commun de monitorer plusieurs métriques, choisies pour leurs caractéristiques, durant l'entraînement afin que l'évaluation soit la plus représentative possible de l'objectif de l'apprentissage. Nous reviendrons en détails sur certaines de ces métriques par la suite pour établir un point de comparaison avec la nouvelle métrique introduite dans nos travaux. Nous allons commencer par décrire l'utilisation de ces métriques à des fins d'évaluation des réseaux tout en utilisant deux métriques classiques de machine learning qui serviront d'exemples.

3.1.1 Utilisation des métriques pour l'évaluation des réseaux

Les métriques utilisées pour l'évaluation des réseaux de neurones et plus généralement des modèles de machine learning sont appelées métriques de performance ou métriques d'erreur. Cela vient du fait qu'elles servent essentiellement à caractériser le modèle vis-à-vis d'une problématique particulière, on peut alors évaluer le degré de performance ou d'erreur du modèle sur ce problème. Les étapes suivantes [GBC16] sont généralement suivies afin de s'assurer de la bonne utilisation des métriques :

- La première étape est la détermination des objectifs de l'entraînement. Cela consiste à bien définir le problème à résoudre grâce au modèle entraîné et passe par la sélection d'une métrique d'erreur. Cette métrique permet de comparer les performances actuelles du modèle par rapport aux performances parfaites théoriques qui sont rarement atteignables en pratique.
- La deuxième étape est le choix des métriques de performances qui vont caractériser les performances du réseau dans différents domaines ou par rapport à différentes

problématiques. Ces métriques donnent une idée de l'efficacité du modèles à différents niveaux. Une hiérarchie peut être établie selon quels aspects du problème sont considérés comme les plus importants.

- La dernière étape consiste à observer l'impact des changements appliqués au modèle sur les métriques de performances choisies. Les changements peuvent être liés aux mises à jour du modèle par la remonté de gradient ou bien des modifications de certains hyperparamètres. Il est alors possible de déterminer quels changements sont bénéfiques ou non au modèle et par rapport à quels aspects du problème.

Maintenant que ces étapes sont définies, nous pouvons discuter d'exemples de métriques classiques de machine learning qui sont utilisables pour l'entraînement de modèles aux applications assez diverses.

3.1.2 Métriques classiques utilisées pour le Machine Learning

Même si ces métriques sont efficaces dans de nombreux problèmes, elles ont également des faiblesses qui sont généralement compensées par l'utilisation de métriques complémentaires. Nous nous concentrons ici sur la valeur de la fonction de perte et sur l'accuracy.

Fonction de perte La fonction de perte peut être vue comme une métrique fournissant une mesure de l'adéquation d'une prédiction prédiction du réseau avec la valeur cible à prédire, c'est-à-dire le label d'entraînement. Elle permet entre autres de définir la perte, c'est-à-dire la pénalisation attribuée au modèle pour chaque erreur ou imprécision dans la prédiction. Cette fonction est souvent associée au risque empirique qui mesure les pertes observées sur un échantillon d'exemples d'entraînement ou de validation. C'est également la valeur de la fonction de perte qui est utilisée pour calculer les gradients qui effectuent les changements des valeurs des paramètres du modèle. Étant donné que cette fonction mesure la quantité d'erreur faite par le modèle, on cherche généralement à ce que cette fonction atteigne sa valeur minimale théorique. C'est par ailleurs cette valeur que l'on cherche à atteindre en résolvant le problème de minimisation décrit dans l'équation 2.10. Il n'est toutefois pas réaliste d'atteindre ce minimum, on souhaite alors juste minimiser la fonction de perte au maximum.

La fonction de perte la plus couramment utilisée dans l'application des réseaux de neurones aux attaques par canaux auxiliaires est la Categorical Cross-Entropy (CCE) décrite Section 2.3.1. Cette fonction calcule de manière générale la différence qu'il existe entre deux distributions de probabilités. Dans notre cas, la différence est faite entre les prédictions du réseau et la valeur du label sous la forme d'une vecteur *one-hot encoded*, c'est-à-dire un vecteur composé de uniquement de 0 sauf à la position de la valeur encodée qui est un 1. Si l'on reprend la définition de la CCE dans le cas où l'on a 256 classes, on obtient :

$$\begin{aligned} CCE(p,q) &= -\sum_{x=1}^{256} p(x) \cdot \log(q(x)), \\ &= -\log(q(z)), \end{aligned}$$

où $z = f(p, k)$ est la valeur de la variable intermédiaire autrement dit la valeur du bon label et f est la primitive cryptographique utilisée. Cela vient du fait que $p(x) = 0$ pour tout $x \neq z$ dû à l'encodage *one-hot*. La valeur de cette fonction de perte ne prend alors en compte que la valeur de prédiction du bon label. Ce point de vue ne se traduit pas bien au niveau des attaques par canaux auxiliaires où l'on est plus intéressé par la valeur du bon label par rapport aux valeurs des autres labels. C'est donc plus la position du bon label par rapport aux autres qui est intéressante pour déduire des futures performances du réseau. Cela signifie que même si la CCE est utile dans son rôle de fonction de perte, elle apporte peu d'information sur l'objectif que l'on cherche à atteindre.

Accuracy L'accuracy mesure le pourcentage de bonnes prédictions faites par le réseau sur un ensemble d'exemples. C'est une métrique intéressante souvent utilisée dans les problèmes de classification puisqu'elle établit de manière assez précise les performances qu'on l'on peut attendre du modèle. Il est assez clair que la valeur que l'on souhaiterait atteindre est 100% d'accuracy mais il est rare que cela soit possible. On cherche donc le plus souvent à atteindre des niveaux d'accuracy meilleurs que ceux existants déjà dans la littérature pour montrer une amélioration par rapport aux anciens modèles.

Toutefois, l'accuracy présente un inconvénient majeur si les classes des exemples sont déséquilibrées. Les classes déséquilibrées correspondent à des problèmes où certaines classes doivent être prédites plus souvent que d'autres. Pour illustrer ce problème, nous pouvons considérer le problème de détection de *spam* pour les boîtes mails. Suivant l'hypothèse que les emails échangés sont 99% du temps légitimes, il est alors facile pour un modèle d'atteindre ce qui peut sembler être de très bons niveaux d'accuracy autour des 99%. Mais en réalité, un tel niveau peut être atteint simplement en utilisant un modèle dont la prédiction est toujours que l'email est légitime. Un autre souci vient de la nature même du problème qu'il faut analyser. En effet, dans ce cas, il est bien plus grave de prédire un email légitime comme spam que de laisser passer un spam. Il existe donc deux types d'erreur n'ayant pas le même poids. C'est pour cela que d'autres métriques comme la *precision* ou le *recall* sont utilisées dans ce type de problèmes [GBC16].

Les métriques choisies peuvent donc être monitorées lors de l'entraînement d'un modèle et doivent être bien adaptées, chose qui est déterminée en avance. Elles permettent alors d'interpréter la progression du modèle dans son processus de généralisation et la qualité de l'entraînement en se basant sur l'évolution de leurs valeurs. Ce sont aussi ces métriques qui sont utilisées lors de l'entraînement pour déterminer si le réseau est dans la phase d'underfitting ou d'overfitting. Nous allons maintenant voir comment cette détermination a lieu.

3.1.3 Lien entre l'évolution des métriques et les phénomènes d'underfitting et d'overfitting

Les phénomènes d'underfitting et d'overfitting ont déjà été abordés dans ce manuscrit mais nous allons cette fois-ci nous concentrer sur leurs détections au moyen des métriques de performances. Cette détection se fait le plus souvent en comparant la valeur d'une

métrique sur l'ensemble d'entraînement et sa valeur sur l'ensemble de validation. Cela permet de comparer les performances du modèle sur les exemples qu'il connaît via son entraînement et des exemples qui lui sont inconnus. On peut ainsi évaluer l'état interne du réseau pour détecter l'underfitting et l'overfitting [GBC16].

Underfitting L'underfitting se caractérise par de mauvaises performances à la fois sur les données d'entraînement et de validation. Cela se traduit au niveau des métriques par des valeurs qui sont encore éloignées des valeurs qu'on pourrait attendre d'un réseau bien entraîné, par exemple une accuracy qui reste faible. Ces mauvaises valeurs sont généralement liées à un manque d'apprentissage ou à une architecture de réseau non adaptée. Il est donc nécessaire d'être capable de le détecter efficacement et pour cela de choisir les bonnes métriques de performances. Dans le cas de l'underfitting, on peut constater que la différence entre les valeurs des métriques d'évaluation sur les exemples d'entraînement et sur ceux de validation n'est pas nécessairement grande mais plutôt que ces valeurs sont mauvaises quelque soit l'ensemble.

Lors de l'utilisation des réseaux de neurones pour les attaques par canaux auxiliaires, on peut interpréter le phénomène d'underfitting comme une phase de l'entraînement où le réseau n'est pas encore capable de bien détecter les fuites d'information. Il n'est alors pas en mesure de retrouver la valeur intermédiaire de manière consistante ce qui entraîne de mauvaises performances en termes d'attaques également.

Overfitting Il se caractérise par une grande différence entre les valeurs des métriques sur l'ensemble d'entraînement et leurs valeurs sur l'ensemble de validation. Cette différence vient du fait que les valeurs des métriques sur l'entraînement sont bonnes, contrairement à leurs valeurs sur la validation qui sont plus faibles. Ce phénomène est souvent lié à une architecture trop complexe pour le problème à résoudre.

Dans le cas des attaques par canaux auxiliaires, cette phase commence lorsque le réseau apprend à utiliser des points des traces de consommation de courant qui n'ont aucun lien avec la variable intermédiaire ciblée Z. Ces points ne correspondent à aucune fuite en lien avec la variable intermédiaire et ne devraient pas impacter les prédictions du réseau. Les informations apprises de ces points ne sont donc utiles que pour prédire des exemples d'entraînement spécifiques et seront au mieux inutiles lors de la prédiction des exemples de validation et de test ou au pire nuisibles aux bonnes prédictions du réseau. Il est encore une fois nécessaire de bien choisir les métriques de performances afin de pouvoir interpréter ces valeurs puisque cela détermine la détection ou non de l'overfitting.

Nous verrons dans la suite de ce chapitre l'étude de la métrique accuracy utilisée pour évaluer les performances des réseaux de neurones appliqués aux attaques par canaux auxiliaires et comment cette métrique ne permet pas à elle seule de conclure sur l'état du réseau. Mais tout d'abord nous allons voir la méthodologie à suivre pour l'utilisation des métriques au cours de l'apprentissage et l'application de la technique d'early stopping.

3.1.4 Technique d'early stopping et optimisation des performances

La technique d'early stopping consiste à arrêter l'entraînement d'un réseau lorsque l'on détecte que l'overfitting commence. Elle permet de déterminer l'epoch offrant le meilleur compromis entre les performances sur les ensembles d'entraînement et de validation durant l'apprentissage comme illustré sur la Figure 3.1. Cette epoch correspond généralement au réseau qui aura les meilleurs performances lors de son application sur des nouveaux exemples inconnus. La détermination de cette epoch passe par l'utilisation d'une métrique de performance qui est calculée à chaque epoch et qui indique le moment où le réseau obtient les meilleures performances, selon cette métrique. Ce moment précède généralement le début de la phase d'overfitting et sa détection permet d'éviter la perte de généralisation liée à l'overfitting.

La Figure 3.1 représente la méthodologie standard d'évaluation des réseaux au cours de leur entraînement. L'ensemble d'entraînement est séparé en deux, d'une part l'ensemble des exemples utilisés pour entraîner le réseau $\mathcal{T}_{train}$ et d'autre part l'ensemble des exemples de validation $\mathcal{T}_{val}$. Une fois l'entraînement démarré, l'état actuel du réseau CNN_e est régulièrement sauvegardé et utilisé pour calculer une ou plusieurs métriques sur les exemples d'entraînement et sur ceux de validation, $\text{Metrique}_{\text{train}}$ et $\text{Metrique}_{\text{val}}$. Ces valeurs sont ensuite comparées pour déterminer si l'entraînement doit continuer ou non. Elles sont également comparées aux valeurs des epochs précédentes afin de prendre en compte leur évolution au fil de l'entraînement. On souhaite donc que l'apprentissage, en plus de réduire l'erreur sur les exemples d'entraînement, réduise l'écart entre les valeurs des métriques sur les exemples d'entraînement et sur ceux de validation.

Ce procédé fait partie de l'ensemble des techniques d'optimisation des réseaux puisqu'il se charge d'optimiser l'hyperparamètre qui correspond au nombre d'epochs d'entraînement. Le nombre d'epoch d'entraînement devient donc un paramètre du réseau qui est déterminé lors de l'entraînement à l'aide des métriques.

La technique d'early stopping possède également un effet de régularisation [GBC16] puisqu'elle va restreindre l'ensemble des fonctions que le modèle peut atteindre en lui imposant d'arrêter son entraînement à un instant donné. Cette régularisation est intéressante puisqu'elle ne pénalise pas les poids du réseau et est donc utilisable en combinaison avec des types de régularisation explicite comme la régularisation L_2 par exemple. Nous verrons les effets de ces combinaisons dans le Chapitre 4.

Nous allons maintenant discuter de l'accuracy et de ses limitations lors de son application aux réseaux utilisés pour les attaques par canaux auxiliaires. Nous verrons aussi quelques pistes qui ont été explorées récemment pour créer une métrique efficace pour l'évaluation de ce type de réseau.

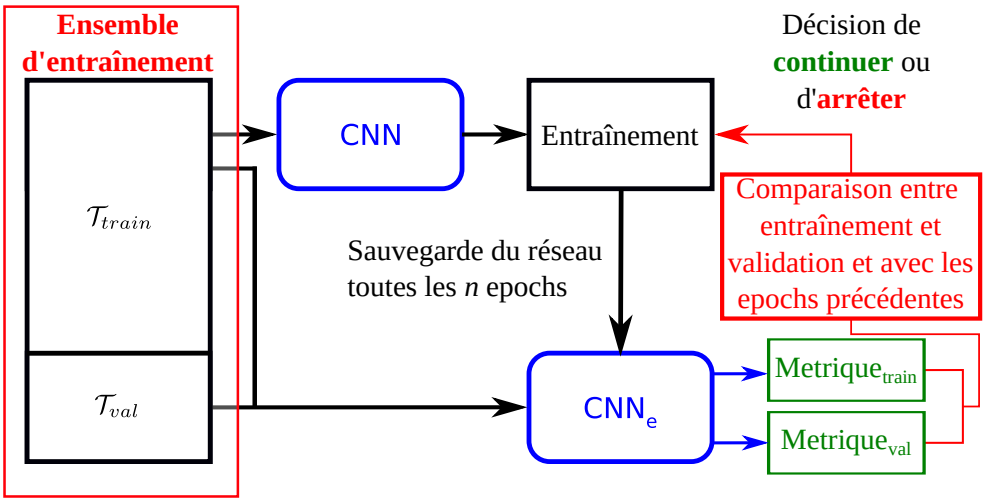


FIGURE 3.1: Description du processus d'early stopping qui passe par l'évaluation du réseau au cours de son entraînement et détermine l'arrêt ou non de l'apprentissage.

3.2 Métrique existante peu efficace et exploration des possibilités

L'accuracy est la métrique que l'on retrouve le plus dans les premiers travaux portant sur l'utilisation du deep learning pour les attaques par canaux auxiliaires [GHO15, KPH+19]. Toutefois, l'efficacité de cette métrique s'avère assez limitée lors de son application aux réseaux de neurones utilisés pour les attaques par canaux auxiliaires. Après ce premier constat effectué lors des travaux de Picek *et al.* [PHJ+19], des recherches pour trouver une métrique efficace dans ce contexte ont commencé et nous discuterons de certains pistes explorées.

3.2.1 Le problème de l'accuracy

Nous allons dans un premier montrer à l'aide d'une figure en quoi l'accuracy n'est pas une bonne métrique pour estimer les performances des réseaux lors de leur application aux attaques par canaux auxiliaires. Ensuite dans un second temps, nous donnerons des explications sur l'origine des limites de cette métrique pour cette application.

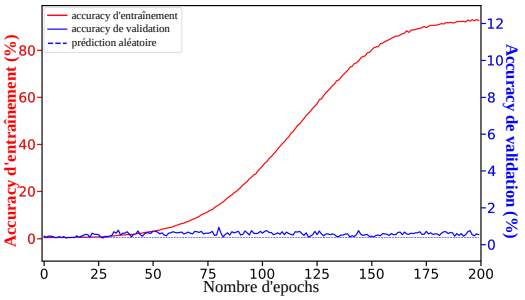
Afin de montrer les limitations de l'accuracy, il est possible de comparer l'évolution de cette métrique au cours de l'entraînement et à l'évolution du rang de la bonne clé qui est une métrique utilisée pour mesurer les performances des attaques par canaux auxiliaires. Le calcul du rang est ici fait à la fin de chaque epoch en utilisant 5000 traces d'entraînement et 5000 traces de validation. La limitation à 5000 traces vient du fait que la taille de l'ensemble de validation est fixée à 5000 exemples afin de ne pas trop réduire la taille de l'ensemble d'entraînement. La Figure 3.2 illustre l'évolution de l'accuracy d'un réseau à la fois sur l'ensemble d'entraînement et sur l'ensemble de validation, et le rang de la bonne clé après l'utilisation de 5000 traces d'attaque. Sur la Figure 3.2a, on peut voir que l'accuracy à l'entraînement augmente fortement au cours de l'apprentissage et atteint presque 100% au bout de 200 epochs, contrairement à l'accuracy de validation qui se situe à peine

au dessus de $0.39\% = 1/256$, qui est équivalent à la prédiction d'une valeur aléatoire. La Figure 3.2b illustre les évolutions du rang de la bonne clé calculé à partir de 5000 traces d'attaques issues de l'ensemble d'entraînement et de l'ensemble de validation au cours de l'entraînement du même réseau. On constate que la valeur du rang pour les données d'entraînement atteint 1 au bout de seulement quelques epochs et qu'il faut environ 30 epochs avant que le rang pour les exemples de validation fasse de même.

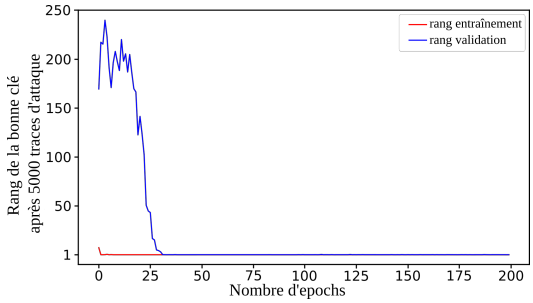
Quand on compare les valeurs d'accuracy à celles des rangs moyens de la clé après 5000 traces, on constate que, même si une différence existe entre la convergence du rang pour les attaques sur l'ensemble d'entraînement et celles sur l'ensemble de validation, dans les deux cas on obtient un rang moyen de 1 après l'epoch 30, ce qui veut dire que la bonne clé est retrouvée. Ici, le simple fait de regarder l'accuracy ne permet pas de déduire les performances futures du réseau en termes d'attaques par canaux auxiliaires. En effet, une accuracy élevée est liée à de bonnes performances puisque, si le réseau est capable de prédire les bonnes valeurs intermédiaires, il est alors capable de retrouver la clé rapidement. Toutefois, si l'accuracy d'un réseau est faible, il est difficile de tirer des conclusions sur ces performances en termes d'attaques.

Cela est dû au fait que lors d'une attaque de type template, c'est l'accumulation d'information sur la bonne clé grâce à plusieurs traces qui permet de la retrouver. Cela n'est pas pris en compte dans le calcul de l'accuracy. Elle se calcule en considérant les exemples indépendamment les uns des autres, c'est-à-dire qu'elle mesure les performances du réseau indépendamment pour chaque trace. Ceci correspond en analyse par canaux auxiliaires à une attaque de type *Simple Power Analysis* (SPA), ou *analyse simple de courant*. Ce type d'attaques est un cas particulier des attaques par canaux auxiliaires où le but est de retrouver la clé à l'aide d'une seule trace. Ces attaques ciblent généralement les algorithmes et architectures non protégées mais elles sont relativement faciles à contrer. On peut donc interpréter l'accuracy d'un réseau comme étant son efficacité pour effectuer une SPA mais cela ne se traduit pas nécessairement bien au niveau d'une attaque de type template qui demandent le plus souvent plusieurs traces pour retrouver la clé. Il peut donc être intéressant de monitorer l'accuracy lors de l'entraînement mais ce n'est pas sur cette métrique qu'il faut se baser pour estimer les performances futures du réseau pour les attaques par canaux auxiliaires profilées.

Un autre point qui limite l'accuracy est sa faiblesse vis-à-vis des problèmes ayant des classes déséquilibrées, par exemple lors du l'utilisation du modèle de fuites en poids de Hamming. Pour la valeur d'un octet, la distribution des valeurs du poids de Hamming n'est pas uniforme. Soit $H_k = \{i | i \in [\![0, 255]\!], \mathrm{HW}(i) = k\}$ l'ensemble des valeurs d'un octet ayant pour poids de Hamming k et HW l'opérateur calculant le poids de Hamming d'un nombre i. On a alors que $\mathrm{Card}(H_0) = 1$ et $\mathrm{Card}(H_4) = 70$ où Card est l'opérateur donnant le cardinal d'un ensemble, *i.e.* son nombre d'élément. Cela met bien en avant la différence de représentativité qu'il existe entre les différentes classes dans ce cas de figure. En effet, la classe correspondant au poids de Hamming 4 représente 27% des exemples si les valeurs de clé et de message d'entrée sont choisies uniformément. Un modèle peut donc atteindre une accuracy de 27% en attribuant toujours la valeur la plus élevé à la classe 4 tout en étant inutile pour réaliser les attaques. Ce problème rend l'utilisation de



(A) Accuracy.



(B) Rang de la bonne clé calculé à partir de 5000 traces d'attaque.

FIGURE 3.2: Évolution de l'accuracy et du rang de la bonne clé après l'utilisation de 5000 traces d'attaque évaluées sur l'ensemble d'entraînement et de validation au cours de l'entraînement de CNN_{best}.

l'accuracy pour les modèles de fuites du poids de Hamming et de la distance de Hamming très peu informative. Il est toutefois possible de compenser ce problème en regardant l'accuracy par classe par exemple mais cette métrique reste malgré tout inadaptée dans ce contexte.

Discussion sur la top-*k* accuracy L'accuracy n'est pas la seule métrique de précision utilisée, il existe des dérivées de cette métrique appelés top-*k* accuracy. Elles mesurent le nombre de fois où le bon label se situe parmi les k premières valeurs les plus probables prédites par le réseau. Ces métriques englobent donc plus de valeurs ce qui les rapproche de la problématique des attaques par canaux auxiliaires où même si un bon label arrive en position k, cela ne signifie pas forcément une mauvaise performance lors des attaques. Il serait donc intéressant d'étudier la top-*k* accuracy pour différentes valeurs de k afin de déterminer si une de ces valeurs permet à cette métrique d'être représentative des performances du réseau en termes d'attaques par canaux auxiliaires. Il reste cependant le problème que, même si le bon label est classé parmi les k premières valeurs, cela ne nous donne pas d'information sur les autres labels qui sont mieux classés que lui. Si ces labels correspondent à une autre valeur de clé, alors l'attaque ne pourra pas réussir.

3.2.2 Utilisation de la théorie de l'information

Une fois le constat fait que l'accuracy n'était pas efficace pour évaluer l'entraînement des réseaux de neurones dans le contexte des attaques par canaux auxiliaires, d'autres métriques ont été explorées. Parmi les plus notables on trouve une métrique se basant sur la théorie de l'information.

Dans leur article, Perin *et al.* [PBP] partent du concept de chemin d'information pour étudier comment l'information se propage entre les couches du réseau. Le chemin d'information est calculé à partir de l'information mutuelle entre l'entrée du réseau et l'activation de ses différentes couches ainsi qu'entre les couches intermédiaires et la sortie du réseau. Le début de l'évolution de l'information mutuelle décrit alors la phase d'apprentissage du réseau pendant laquelle le réseau commence son entraînement et l'information mutuelle est faible à travers le réseau. Plus l'entraînement avance, plus l'information mutuelle aug-

mente jusqu'à atteindre un maximum pour l'information mutuelle entre l'activation des couches intermédiaires et la sortie du réseau qui doit correspondre au maximum de généralisation. Une fois ce maximum passé, l'information mutuelle évolue différemment selon les exemples considérés. Lorsqu'elle est évaluée sur les exemples d'entraînement, l'information mutuelle entre les couches intermédiaires et la sortie reste haute puisque que le réseau entre dans la phase d'overfitting et donc il commence à apprendre par cœur les exemples d'entraînement. Cette phase est aussi appelée phase de compression car le réseau apprend à retenir une version compressée des exemples afin d'être capable de les prédire parfaitement. Pour les exemples de validation, une fois le maximum d'information mutuelle entre couches intermédiaires et sortie passé, il décroît dû au fait que plus la phase de compression avance plus la généralisation du réseau baisse.

En mesurant ces valeurs d'information mutuelle à chaque epoch, on obtient donc une métrique capable de prédire quand arrêter l'entraînement. Perin *et al.* effectuent donc des expériences d'early stopping comparant différentes métriques, dont la leur, pour déterminer laquelle obtient les meilleurs résultats. Les métriques testées inclues l'accuracy, le rang de la bonne clé en validation et la valeur de la fonction de perte. Leurs tests semblent indiquer que leur métrique est la plus consistante pour trouver la meilleure epoch pour arrêter l'entraînement. Cependant ces tests sont réalisés sur des métriques déjà considérées comme peu efficaces ou sur des critères peu précis comme l'utilisation du rang moyen de la bonne clé lors des attaques sur l'ensemble de validation. Malgré tout, cette approche reste intéressante et apporte un point de vue venant de la théorie de l'information pour résoudre un problème important dans l'entraînement des réseaux de neurones pour les attaques par canaux auxiliaires.

3.2.3 Cross-Entropy Ratio

Une autre approche a été utilisée récemment par Zhang *et al.* [ZZN+20]. Ils ont mis au point un ratio, appelé Cross Entropy Ratio (CER), qui a pour but de lier la cross entropy et les métriques d'attaques par canaux auxiliaires guessing entropy et success rate vu Section 1.2.2. Ils montrent que le ratio entre la valeur moyenne du score de la bonne clé et la moyenne de tous les autres scores permet de déterminer la convergence de l'attaque. Toutefois, cette convergence reste théorique puisque le ratio ne donne pas d'information sur le nombre de traces nécessaires pour la réussite des attaques. Ce nombre doit être calculé après coup à l'aide de la guessing entropy et du success rate. Zhang *et al.* constatent aussi que la valeur du ratio n'est pas nécessairement consistante dans le sens où, pour différents nombres d'epochs, des valeurs similaires du CER ne mènent pas à des performances similaires en termes d'attaques. Cela peut poser problème pour l'utilisation d'une telle métrique à des fins d'early stopping mais cette métrique reste pertinente pour l'ajustement des hyperparamètres du réseau.

Autre point important de ce ratio est qu'il peut être adapté en fonction de perte pour remplacer la cross entropy. Cette nouvelle fonction de perte est alors plus efficace dans le cadre d'entraînement avec des classes déséquilibrées. Ce cas de figure apparaît en attaques par canaux auxiliaires lors d'attaques basées sur un modèle de fuite en poids ou distance

de Hamming.

Dans la section suivante, nous décrirons notre propre apport dans la recherche d'une métrique d'évaluation des réseaux de neurones pour les attaques par canaux auxiliaires. Notre approche est différente des deux précédentes puisque nous nous basons directement sur les métriques d'évaluation des attaques par canaux auxiliaires pour dériver la métrique d'évaluation des réseaux.

3.3 $\Delta^d_{train,val}$: métrique d'évaluation des algorithmes de deep learning appliqués aux attaques par canaux auxiliaires

La métrique $\Delta^d_{train,val}$ se base sur une métrique classique d'évaluation des attaques par canaux auxiliaires, le taux de succès, pour évaluer les performances d'un réseau sur l'ensemble d'entraînement et l'ensemble de validation. Cela permet de tirer directement des conclusions sur l'état de l'entraînement du réseau, à savoir le risque d'un potentiel underfitting ou overfitting. Nous allons maintenant voir comment calculer cette métrique.

3.3.1 $\Delta^d_{train,val}$: détection de l'état de l'apprentissage

Considérons de nouveau f comme la fonction apprise par le réseau pour estimer la fonction cible f^* dans le cadre d'attaques par canaux auxiliaires. Nous pouvons définir deux quantités associées à f : $N^d_{train}(f)$ et $N^d_{val}(f)$ qui représentent les nombres minimaux de traces nécessaires, en utilisant le modèle f, pour atteindre un taux de succès à l'ordre d de 90% sur respectivement l'ensemble d'entraînement et celui de validation. On peut décrire ces valeurs de la façon suivante :

$$N^d_{train}(f) = min\{n_{train} \mid \forall n \geq n_{train},\ SR^d_{train}(f(n)) = 90\%\}$$

et,

$$N^d_{val}(f) = min\{n_{val} \mid \forall n \geq n_{val},\ SR^d_{val}(f(n)) = 90\%\}.$$

L'ordre du taux de succès représente ici le nombre d'hypothèses de clé à essayer avant de trouver la bonne, en les tirant dans l'ordre du vecteur $\mathbf{g}$ [1] résultant de l'attaque, qui contient l'ensemble des hypothèses classées de la plus probable à la moins probable.

En calculant ces valeurs liées au taux de succès à la fois sur l'ensemble d'entraînement et de validation, il est possible de voir la capacité de généralisation du réseau. Un réseau présentant une bonne généralisation aura des valeurs de N^d_{train} et N^d_{val} proches et, dans le cas contraire, ces valeurs seront éloignées. Le but de la métrique $\Delta^d_{train,val}$ est de fournir un indicateur de ce comportement, elle est calculée de la façon suivante :

$$\Delta^d_{train,val} = |N^d_{val} - N^d_{train}|.$$

1. Le vecteur $\mathbf{g}$ contient toutes les valeurs possibles de la clé rangées de la plus probable à la moins probable.

Le choix de la distance euclidienne entre les valeurs N^d_{train} et N^d_{val} à des fins de comparaison s'est imposé comme le plus naturel pour comparer des valeurs représentant des nombres de traces. Le calcul de N^d_{train} et N^d_{val} se base sur une métrique d'analyse par canaux auxiliaires connue qui est le taux de succès et la combine avec une approche venant du machine learning afin d'être capable d'évaluer n'importe quel réseau utilisé pour une attaque par canaux auxiliaires.

Cette proposition de métrique a l'avantage d'évaluer l'entraînement pendant qu'il a lieu et elle peut être visualisée après coup pour une analyse plus détaillée. Cela permet de détecter si le réseau est dans un état d'underfitting, d'overfitting ou si un bon compromis entre les performances sur les données d'entraînement et les performances sur les données de validation a été atteint. Le choix d'un taux de succès à 90% a été fait pour apporter plus de stabilité dans les valeurs de N^d_{train} et N^d_{val}. En effet, si un taux de succès à 100% est considéré, il est possible qu'une petite partie des attaques mettent plus de temps à converger que la majorité ce qui aurait tendance à artificiellement augmenter les valeurs N^d_{train} et N^d_{val} et les rendre plus instables. Nous allons maintenant voir comment se fait la détection de l'overfitting et de l'underfitting.

3.3.2 Détection de l'overfitting et l'underfitting

La détection de l'overfitting et de l'underfitting se fait en observant l'évolution de la métrique $\Delta^d_{train,val}$ au cours des epochs. Un exemple d'une telle évolution est donné dans la Figure 3.3. Elle contient à la fois une courbe représentant les valeurs de $\Delta^1_{train,val}$ et les valeurs de la moyenne glissante de $\Delta^1_{train,val}$ avec une fenêtre de taille 10. Cette inclusion est faite afin de mieux voir la tendance globale de la courbe. On peut identifier trois zones différentes sur cette figure qui correspondent aux trois états possibles du réseau pendant son entraînement.

- **Underfitting** : comme discuté dans la Section 2.3.2, l'underfitting est généralement lié soit à un manque de capacité du réseau, soit à un manque d'entraînement. Cela implique que les attaques utilisant ce réseau ne parviennent pas à retrouver la bonne clé et que les valeurs N^1_{train} et N^1_{val} sont soit très hautes, soit non-définies. On constate donc que quand le nombre d'epochs d'entraînement est faible, $\Delta^1_{train,val}$ n'est pas définie ou elle possède une valeur très grande. La partie sur la gauche de la Figure 3.3, en rouge, représente ce cas de figure puisqu'on y voit que pour moins de 80 epochs, $\Delta^1_{train,val}$ n'est d'abord pas calculable puis sa valeur décroît très rapidement. Cette décroissance correspond à la partie de l'entraînement où le réseau commence à généraliser ses connaissances et à être performant sur l'ensemble de validation. Malgré cette amélioration, il est nécessaire de continuer l'entraînement puisqu'une forte valeur de $\Delta^1_{train,val}$ signifie une forte différence entre la performance sur l'ensemble d'entraînement et la performance sur l'ensemble de validation.
- **Bon compromis** : Cet état est atteint lorsque le réseau a appris assez d'information à partir de l'ensemble d'entraînement pour généraliser correctement sur l'ensemble de validation. La valeur de $\Delta^1_{train,val}$ converge alors vers une valeur théorique $N_{bias}(\mathcal{T}_{train})$ qui représente la différence minimale entre N^1_{train} et N^1_{val} étant

donné l'ensemble d'entraînement $\mathcal{T}_{train}$. Soit e l'epoch qui minimise la distance entre $\Delta^1_{train,val}$ et $N_{bias}(\mathcal{T}_{train})$, on a alors :

$$\Delta^1_{train,val} \xrightarrow[epoch \to e]{} N_{bias}(\mathcal{T}_{train}). \tag{3.1}$$

Le bon compromis représente donc le nombre d'epochs pour lequel la valeur de $\Delta^1_{train,val}$ est la plus proche de $N_{bias}(\mathcal{T}_{train})$. Ce compromis apparaît dans la zone en vert de la Figure 3.3 où l'on voit que $\Delta^1_{train,val}$ atteint sa valeur minimale. Pour cette valeur, le réseau atteint son meilleur niveau de généralisation. Par la suite, nous parlerons de meilleur compromis pour décrire les meilleurs réseaux obtenus par l'entraînement sans garantir qu'ils soient optimaux.

- **Overfitting** : l'overfitting provient généralement de la combinaison d'une capacité trop grande du réseau et de trop d'entraînement. Cela résulte en un réseau qui commence à apprendre par cœur des exemples d'entraînement ce qui entraîne une perte de généralisation. On obtient alors un réseau qui perd en performance sur l'ensemble de validation mais en gagne sur l'ensemble d'entraînement. En effet, la valeur de N^1_{train} tend vers 1 si la capacité du réseau le permet ou plus généralement vers une valeur faible alors que, dans le même temps, N^1_{val} augmente vers une valeur théorique $N_{max}(\mathcal{T}_{train})$. Cette valeur représente le nombre maximum de traces dont aura besoin le réseau pour effectuer une attaque à partir des traces de validation une fois l'entraînement stabilisé, c'est-à-dire que l'erreur d'entraînement ne peut plus être réduite davantage et que les modifications du réseau ne changent plus significativement ses prédictions. On a alors :

$$\Delta^1_{train,val} \xrightarrow[epoch \to \infty]{} N_{max}(\mathcal{T}_{train}). \tag{3.2}$$

Ce phénomène est représenté par la zone de droite de la Figure 3.3 en bleu pour laquelle $\Delta^1_{train,val}$ croît régulièrement. Cette croissance indique une réduction des performances du réseau sur l'ensemble de validation et donc une perte de performance lors des attaques.

Dans le cas de la Figure 3.3, on peut remarquer que la croissance de $\Delta^1_{train,val}$ durant l'overfitting est plus douce que la décroissance de $\Delta^1_{train,val}$ durant l'underfitting. Pour ce réseau, l'effet de l'overfitting est moins important que celui de l'underfitting mais il est important d'éviter ces deux phénomènes. C'est ce que permet la métrique $\Delta^1_{train,val}$ lorsqu'elle est utilisée pour monitorer l'entraînement du réseau. Nous allons maintenant voir comment appliquée de la technique d'early stopping à partir de cette métrique.

3.3.3 Technique d'early stopping basée sur la métrique $\Delta^d_{train,val}$

La Figure 3.4 illustre la méthodologie de la Figure 3.1 en y incluant le calcul de la métrique $\Delta^d_{train,val}$ afin d'appliquer la technique d'early stopping. On voit sur cette figure que, à la fin de chaque epoch d'entraînement, le réseau est utilisé pour calculer les valeurs de N^d_{train} et N^d_{val}. Ces valeurs représentent ici les performances du réseau de neurones par rapport aux attaques par canaux auxiliaires. Il faut les comparer afin de déterminer quand se

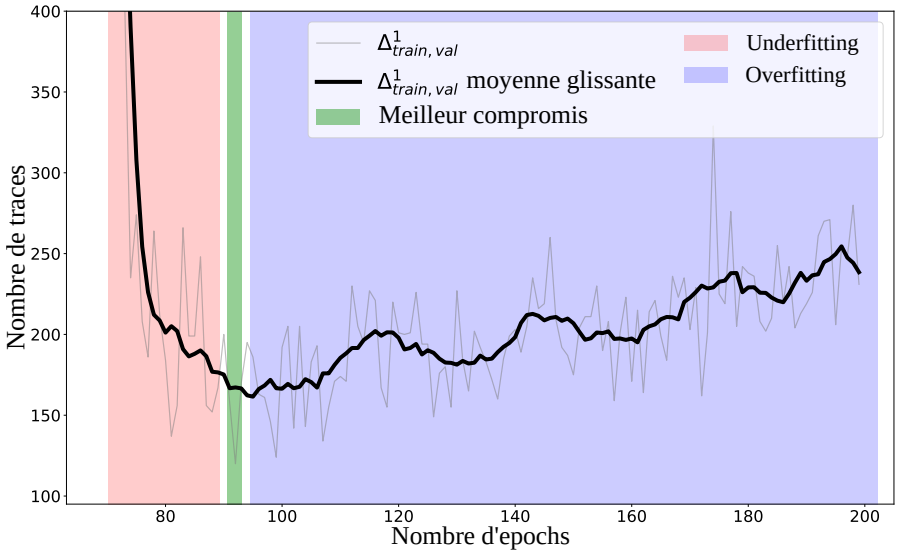


FIGURE 3.3: Exemple d'évolution de $\Delta^1_{train,val}$ au cours de l'entraînement d'un réseau. La courbe représentant $\Delta^1_{train,val}$ est faite à l'aide d'une moyenne glissante de fenêtre de taille 10.

situe le meilleur compromis entre ces performances. Dans le cas de $\Delta^d_{train,val}$, la comparaison se fait en calculant la différence entre les deux valeurs. La minimisation de cette différence est équivalente à la détermination du meilleur compromis. La méthodologie à suivre est de comparer les valeurs de $\Delta^d_{train,val}$ pour plusieurs epochs consécutives. Lorsque que l'overfitting du réseau commence, l'écart entre les performances sur les traces d'entraînement et de validation se creuse ce qui résulte en une augmentation de $\Delta^d_{train,val}$. La détection de cette augmentation durant plusieurs epochs permet de conclure que l'entraînement peut être arrêté. Il est important de noter qu'une augmentation ponctuelle n'est pas suffisante pour atteindre cette conclusion puisque chaque nouvelle epoch d'entraînement n'améliore pas nécessairement les performances du réseau sur l'entraînement ou la validation. Cela résulte en une valeur de $\Delta^d_{train,val}$ qui peut être instable comme illustrée dans la Figure 3.3 pour la courbe grise représentant la vraie valeur de la métrique. Au final, le réseau issu d'un entraînement utilisant la technique d'early stopping avec la métrique $\Delta^d_{train,val}$ sera celui minimisant la valeur de cette métrique et présentant le meilleur taux de généralisation.

3.3.4 Parallélisation des calculs sur plusieurs epochs

La complexité du calcul de $\Delta^d_{train,val}$ varie grandement en fonction du réseau utilisé et de la base de données. En effet, un réseau peu performant va nécessiter plus de traces pour réussir ses attaques tout comme lors de l'utilisation d'un ensemble de traces plus complexes où l'information présente est plus difficile à extraire. Plus les nombres de traces pour réussir les attaques sont importants, plus le calcul de $\Delta^d_{train,val}$ sera long mais une solution à ce problème est la parallélisation de l'entraînement et du calcul de la métrique. Une telle parallélisation est illustrée Figure 3.5 et Figure 3.6.

Dans la première figure, le temps de calcul de $\Delta^d_{train,val}$ est inférieur au temps d'entraînement d'une epoch, il est alors possible de calculer la valeur de $\Delta^d_{train,val}$ de l'epoch

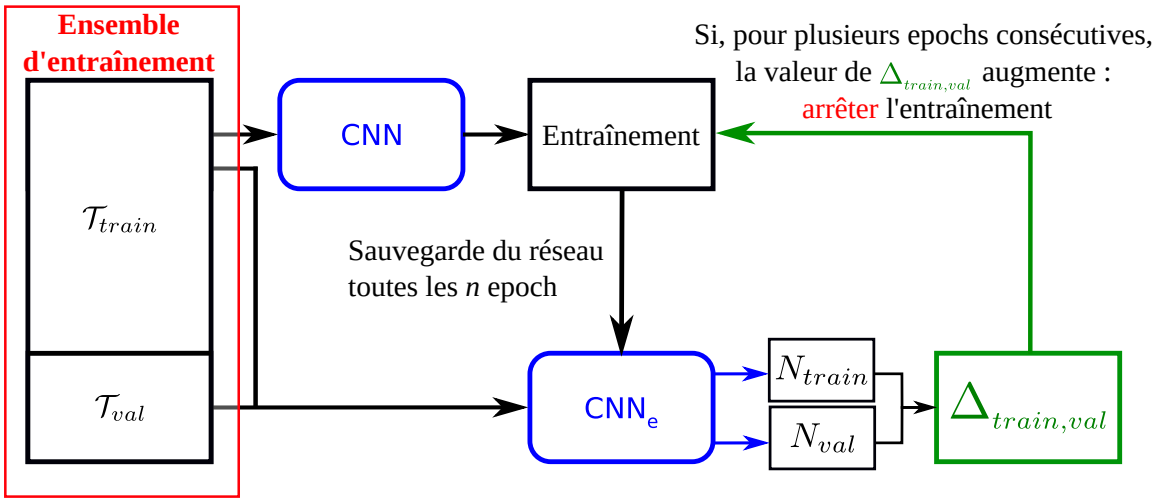


FIGURE 3.4: Description du processus d'early stopping utilisant la métrique $\Delta^d_{train,val}$ pour déterminer l'arrêt ou non de l'apprentissage.

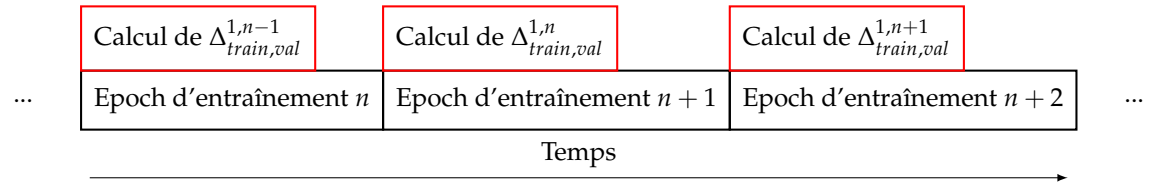


FIGURE 3.5: Calcul de $\Delta^1_{train,val}$ pour des epochs consécutives lorsque le calcul est plus court que l'entraînement d'une epoch

$n-1$ en même temps que l'entraînement de l'epoch n. En faisant de cette manière, la complexité en temps reste la même et seules les complexités en calcul et en mémoire augmentent. Ainsi, si la valeur de $\Delta^d_{train,val}$ pour l'epoch $n-1$ nous indique que l'entraînement peut être arrêté, seule une epoch supplémentaire aura été effectuée.

La deuxième figure représente le cas où le calcul de $\Delta^d_{train,val}$ est plus long que l'entraînement d'une epoch, cela peut arriver par exemple si l'ensemble d'entraînement est petit ou si le réseau lui même est peu complexe. Dans un tel cas, il est possible de ne pas calculer la métrique $\Delta^d_{train,val}$ à toutes les epochs mais seulement toutes les k epochs où k est déterminé selon les caractéristiques de l'entraînement. La Figure 3.6 montre un cas où $k = 2$ étant donné que le calcul de $\Delta^d_{train,val}$ à l'epoch $n-1$ est plus long que l'entraînement de la n-ième epoch. Une telle méthode réduit légèrement la précision de la technique d'early stopping mais n'est utilisée que dans des cas spécifiques.

3.4 Résultats expérimentaux sur CNN_{best}

3.4.1 Description du réseau et de la base de données utilisée

Toutes les expériences décrites dans cette section utilisent la base de données ASCAD [2] [BPS+19] pour l'entraînement et le test des réseaux de neurones pour les attaques par canaux auxiliaires. Cette base de données, mise en place par l'ANSSI, a pour but de définir un point de départ commun pour les chercheurs dans ce domaine en plus de fournir un

2. https://github.com/ANSSI-FR/ASCAD

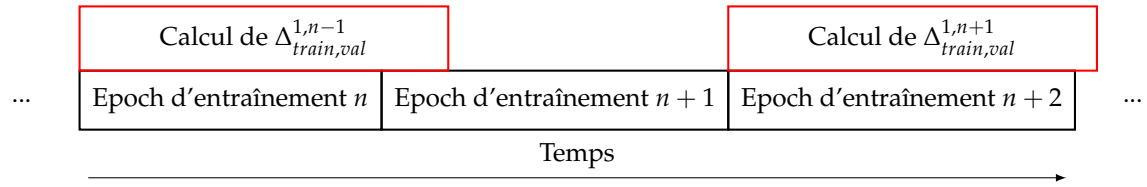


FIGURE 3.6: Calcul de $\Delta^1_{train,val}$ pour des epochs consécutives lorsque le calcul est plus long que l'entraînement d'une epoch

moyen de comparer les résultats obtenus avec l'apprentissage de diverses architectures. Nous nous concentrerons ici sur la partie clé fixe de cette base de données, c'est-à-dire que l'ensemble des traces d'entraînement ont été acquises avec une clé fixe utilisée pour les chiffrements.

Cet ensemble, que l'on appellera ASCAD clé fixe, est composé de 60 000 traces de consommations de courant obtenues à l'aide d'un microcontrôleur AVR ATMega8515 8-bit implémentant un AES sécurisé contre les attaques de premier ordre, *i.e.* une contremesure de masquage de premier ordre est présente dans l'algorithme de chiffrement. Cette contremesure consiste à masquer les valeurs intermédiaires des calculs par un ou-exclusif avec une valeur appelée *masque*. Le masque est une valeur aléatoire tirée pour chaque chiffrement qui permet de décorréler en partie la consommation de courant des valeurs intermédiaires de l'algorithme. Pour réussir à attaquer une telle implémentation, il faut alors mener une attaque dite du second ordre qui consiste à retrouver séparément la valeur du masque et la valeur masquée afin de la démasquer et de récupérer la valeur intermédiaire.

L'ensemble des traces est séparé en deux parties : 50 000 traces forment l'ensemble d'entraînement et 10 000 traces sont réservées pour l'ensemble de test. L'ensemble de validation est lui constitué de 5000 traces d'entraînement qui ne sont pas utilisées pour l'apprentissage. Les traces sont constituées de 700 points représentant la consommation de courant au cours du temps. Ces points ont été choisis parmi les 100 000 points que composent les traces originales parce qu'ils contiennent les fuites d'information sur la valeur du troisième octet de la sortie de la première opération SubBytes masquée ainsi que la valeur du masque associé. Le modèle de fuite associé aux traces est celui de l'identité et on obtient donc les labels Y de la façon suivante :

$$Y(\mathbf{k}^*) = SubBytes(\mathbf{p}[3] \oplus \mathbf{k}^*[3]),$$

où p représente le message d'entrée l'algorithme et k^* est la clé utilisée dans le chiffrement. La sortie du réseau est donc une prédiction de 256 valeurs, une pour chaque valeur possible de la sortie de l'opération SubBytes. Le taux de succès est calculé à partir du résultat de 100 attaques pour déterminer la valeur de $\Delta^1_{train,val}$. Chaque attaque a une limite de 5000 traces et les attaques ne réussissant pas à retrouver la clé en moins de 5000 traces ne sont pas considérées comme réussites dans le calcul de $\Delta^1_{train,val}$. Cela est fait pour éviter au maximum la redondance dans les traces utilisées pour la validation tout en gardant un ensemble d'entraînement le plus large possible.

Les valeurs de $\Delta^1_{train,val}$ qui apparaissent dans les figures de cette section sont calculées en utilisant une moyenne glissante de fenêtre de taille 3 afin de lisser les courbes pour vi-

sualiser leur forme globale. Cela permet de mieux faire apparaître les zones d'underfitting et d'overfitting et explique pourquoi certains minimums de ces courbes ne correspondent pas au minimum visuel. Nous allons maintenant présenter le réseau de neurones utilisé pour les différentes expériences.

Présentation du réseau CNN_{best}

Le réseau de départ utilisé est le réseau CNN_{best} présenté dans l'article d'introduction de la base de données ASCAD [BPS+19]. Ce réseau est issu d'une recherche d'hyperparamètres dont le meilleur résultat a été appelé CNN_{best}. C'est donc un bon point de départ pour tester de nouvelles techniques et essayer d'améliorer les performances du réseau. Une description générale du réseau est faite en Annexe A Tableau A.1. Il est composé de 5 couches convolutionnelles et de 2 couches complètement connectées utilisant une fonction d'activation ReLU décrite Section 2.3.1. Les filtres convolutionnels sont de tailles 11 et leur quantité double à chaque nouvelle couche allant de 64 à un maximum de 512. Un average pooling est utilisé après chaque couche convolutionnelle pour limiter l'augmentation de la dimension des données. L'entraînement de ce réseau utilise comme fonction de perte l'entropie croisée catégorique (CCE) décrite Section 2.3.1 et la fonction d'optimisation RMSprop avec un learning rate de 10^{-5}.

3.4.2 Application de la métrique $\Delta^1_{train,val}$ et de la technique d'early stopping sur CNN_{best}

Dans les travaux qui ont mené à la création de CNN_{best}, le nombre d'epochs d'entraînement était un hyperparamètre à tester et donc seuls trois nombres d'epochs différents ont été testés : 50, 75 et 100 epochs. L'entraînement ayant donné les meilleurs résultats étant 75 epochs, c'est celui-ci qui fut gardé. Toutefois, grâce à l'utilisation de la métrique $\Delta^1_{train,val}$, il est maintenant possible de valider ce choix ou de trouver le nombre d'epochs donnant les meilleurs performances en appliquant la technique d'early stopping.

L'entraînement du réseau CNN_{best} est fait en calculant la valeur de $\Delta^1_{train,val}$ à la fin de chaque époque en laissant le réseau s'entraîner sur 200 epochs. L'évolution de la métrique $\Delta^1_{train,val}$ est illustrée sur la Figure 3.7 avec en plus l'évolution de la valeur de N^1_{train}. On y voit également une croix noire indiquant le nombre d'epoch choisi dans la référence [BPS+19] et un trait en pointillés verts indiquant l'emplacement du minimum de $\Delta^1_{train,val}$. La valeur N^1_{train} est montrée afin de clairement expliquer que la remontée de $\Delta^1_{train,val}$ est uniquement liée à une augmentation de la valeur de N^1_{val} et est donc bien un signe d'overfitting. On peut voir sur la Figure 3.7 qu'après environ 30 epochs d'entraînement, le réseau est capable de généraliser assez ses connaissances pour réussir des attaques sur l'ensemble de validation en moins de 5000 traces avec un taux de succès de 90%. C'est le signe du passage d'un état d'underfitting, où le réseau est seulement capable d'attaquer les traces d'entraînement, à un état de bon compromis lié à la généralisation du réseau. Après cela, la valeur de $\Delta^1_{train,val}$ décroît pour atteindre son minimum à l'epoch 47. Pour ce nombre d'epoch, le réseau atteint un taux de succès de 90% avec seulement 800 traces. Une fois ce stade passée, $\Delta^1_{train,val}$ passe par une période plus ou moins stable durant une dizaine

d'epochs avant d'augmenter. Durant cette période d'augmentation, la grande capacité du réseau lui permet de commencer à apprendre les exemples d'entraînement par cœur ce qui le conduit à atteindre un régime instable autour d'une valeur de 3000 traces. Cela correspond à un état d'overfitting avancé où N^1_{val} converge vers une valeur $N_{max}(\mathcal{T}_{train}) \approx 3000$ comme décrit dans l'Equation 3.2. Dans l'article de présentation de ce réseau, la valeur recommandée pour le nombre d'epochs d'entraînement est de 75 epochs mais il apparaît sur cette figure que le réseau se trouve déjà dans un état d'overfitting à ce moment là. En effet, il doit utiliser en moyenne 1150 traces de validation pour atteindre un taux de succès de 90%. Arrêter l'entraînement à l'epoch 47 nous permet donc d'obtenir un gain de performance de 30% par rapport au réseau original. On peut aussi noter un gain de temps significatif puisque le réseau s'entraîne pendant environ 33% d'epochs en moins. La mesure de l'évolution de $\Delta^1_{train,val}$ nous permet donc de conclure après l'epoch 60 qu'il n'est pas nécessaire de continuer l'apprentissage et que le réseau de l'epoch 47 peut être conservé comme le meilleur issu de cet entraînement.

La présence de N^1_{train} sur cette figure nous montre également que même si l'epoch 47 obtient les meilleures résultats, les performances du réseau sur l'ensemble de validation sont encore loin de celles sur l'ensemble d'entraînement. La métrique $\Delta^1_{train,val}$ nous permet bien de déterminer la meilleure epoch pour arrêter l'entraînement et donc de réduire l'overfitting venant du sur-apprentissage mais cela ne suffit pas pour l'empêcher complètement. En effet, une partie de l'overfitting présent dans le réseau à l'epoch 47 est due à la trop grande capacité du réseau CNN_{best}. Cette grande capacité fait qu'il commence à apprendre par cœur les exemples d'entraînement avant même d'avoir atteint son meilleur niveau de généralisation. Cela explique que les attaques utilisant les traces d'entraînement atteignent un taux de succès de 90% en une dizaine de traces à ce point de l'apprentissage ce qui est bien plus faible que sur les traces de validation. Cet overfitting n'empêche pas le réseau d'atteindre une bonne généralisation, mais il réduit potentiellement les performances du réseau sur l'ensemble de validation et l'ensemble de test. Une discussion sur la manière d'utiliser $\Delta^1_{train,val}$ pour détecter cet overfitting se trouve dans la Section 3.5. Il ne peut pas être réglé grâce à de l'early stopping mais il est possible de le minimiser en abaissant la complexité du réseau ou à l'aide de techniques de régularisation [GBC16] comme celles mentionnées Section 2.3.2. Nous reviendrons en détails sur ces techniques dans le prochain chapitre de ce manuscrit.

3.4.3 Application de la métrique $\Delta^1_{train,val}$ sur d'autres réseaux

Nous allons appliquer la métrique $\Delta^1_{train,val}$ sur d'autres réseaux que CNN_{best} afin de voir son comportement et le gain de performances obtenu en utilisant l'early stopping. Le prochain réseau, appelé CNN_{BV}, est décrit Annexe A Table A.2 et a été introduit par Van der Valk *et al.* [vdVP19]. Dans leur article, ils étudient la décomposition de la guessing entropy en une composante variance et une composante biais. Pour leur étude, ils utilisent des réseaux de neurones avec différentes architectures afin de déterminer la meilleure. La fonction de perte utilisée est la *mean square error* (MSE) qui se calcul de la façon suivante :

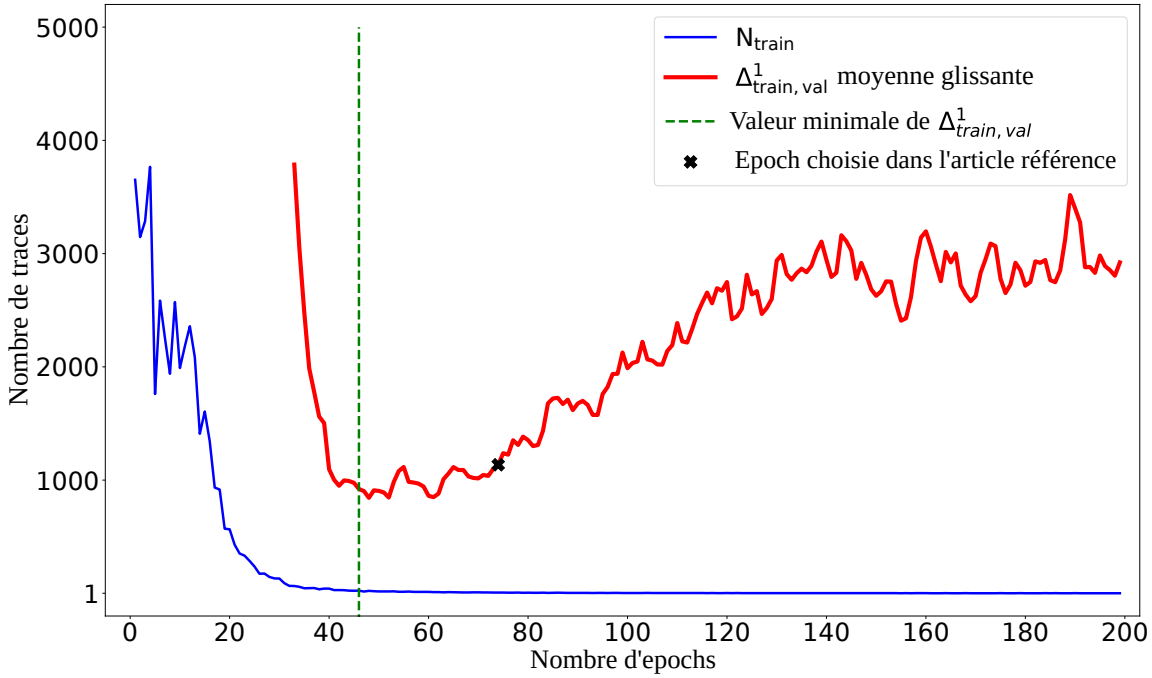


FIGURE 3.7: Évolution de $\Delta^1_{train,val}$ et N^1_{train} durant l'entraînement du réseau CNN_{best} et comparaison avec les résultats présentés dans [BPS+19].

$$\text{MSE}(Y, \hat{Y}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2, \tag{3.3}$$

où Y correspond aux labels des exemples et $\hat{Y}$ à leurs prédictions. La fonction d'optimisation est le *Adam optimizer* [KB17] avec un learning rate de 10^{-4} et le nombre d'epochs recommandé est de 50. La complexité des réseaux varie avec le nombre de couches de convolution utilisées, nous nous concentrerons ici sur les réseaux sans couche de convolution (0CONV) et avec seulement une seule couche (1CONV).

L'ensemble de données d'entraînement est également la base de données ASCAD mais le modèle de fuite est différent. Pour cette application, le modèle de fuite en poids de Hamming est utilisé, les labels deviennent donc :

$$Y(\mathbf{k}^*) = HW(Sbox(\mathbf{p}[3] \oplus \mathbf{k}^*[3])),$$

où $\mathbf{p}$ et $\mathbf{k}^*$ sont les mêmes que précédemment. Les prédictions sont donc maintenant composées de 9 valeurs, une pour chaque poids de Hamming possible pour un octet. Comme pour CNN_{best}, l'entraînement des réseaux est accompagné du calcul de $\Delta^1_{train,val}$ afin de monitorer l'apprentissage et de choisir l'epoch offrant le meilleur compromis entre performance à l'entraînement et performance en validation. Trois entraînements différents sont fait : le premier avec CNN_{BV} 0CONV et la fonction de perte MSE, décrite Équation 3.3, le deuxième avec CNN_{BV} 0CONV également mais cette fois-ci la fonction de perte CCE et le troisième avec CNN_{BV} 1CONV et la fonction de perte CCE. Cela nous permet de voir l'impact de la fonction de perte sur l'entraînement ainsi que l'effet de l'ajout d'une couche

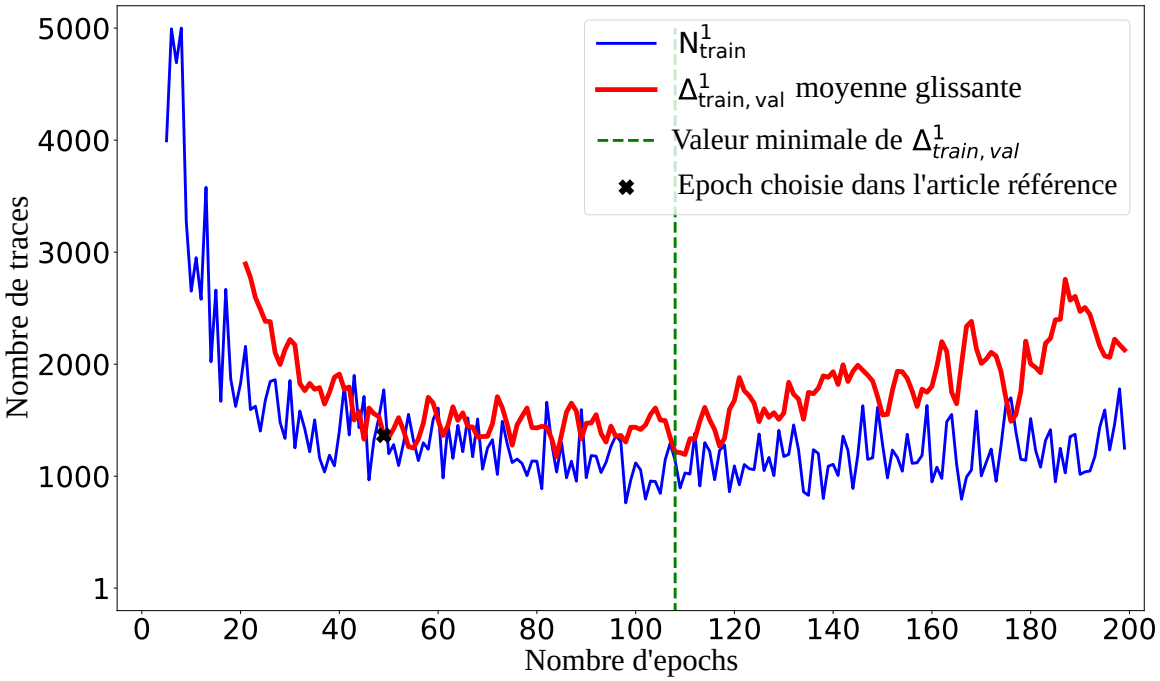


FIGURE 3.8: Évolution de $\Delta^1_{train,val}$ et N^1_{train} durant l'entraînement du réseau CNN_{BV} 0CONV en utilisant la fonction de perte MSE et comparaison avec les résultats présentés dans [vdVP19].

convolutionnelle.

La Figure 3.8 illustre l'évolution de la métrique $\Delta^1_{train,val}$ et de la valeur de N^1_{train} pour le premier réseau, c'est-à-dire CNN_{BV} 0CONV avec pour fonction de perte la MSE. On peut y voir que le réseau est capable d'atteindre un taux de succès de 90% sur l'ensemble de validation à partir de l'epoch 20 mais il continue sa phase de généralisation jusqu'à l'epoch 40 environ. Il entre après dans une phase d'instabilité, de l'epoch 40 à l'epoch 100 environ, où les changements appliqués par la descente de gradient font grandement varier les performances du réseau d'une epoch à une autre, à la fois sur l'ensemble d'entraînement et sur l'ensemble de validation. Cela veut aussi dire qu'il est plus difficile de localiser le meilleur compromis entre les deux. Le minimum de $\Delta^1_{train,val}$ est atteint à l'epoch 109 où le réseau a besoin d'environ 2100 traces en moyenne pour atteindre un taux de succès de 90% sur l'ensemble de validation. Cela représente une amélioration de 25% du nombre de traces requis pour l'attaque par rapport à l'état du réseau après 50 epochs d'entraînement, qui était le nombre d'epochs recommandé dans l'article [vdVP19].

En regardant l'évolution de N^1_{train}, on peut en conclure que ce comportement instable du réseau est dû à de l'underfitting lié à un manque de complexité. En effet, le fait que le réseau ne soit pas capable de faire converger la valeur de N^1_{train} vers 1 montre qu'il a du mal à apprendre à prédire les exemples d'entraînement. Il n'est pas assez complexe pour retrouver les informations nécessaires à de bonnes prédictions à l'entraînement. Cela se traduit aussi sur les performances en validation qui, si elles permettent de retrouver la clé, peuvent être améliorées. Toutefois la complexité du réseau n'est pas nécessairement la seule raison derrière ce comportement. Il peut aussi être lié au choix de la fonction de perte et du learning rate. Cette dernière hypothèse est testée lors de l'entraînement de CNN_{BV}

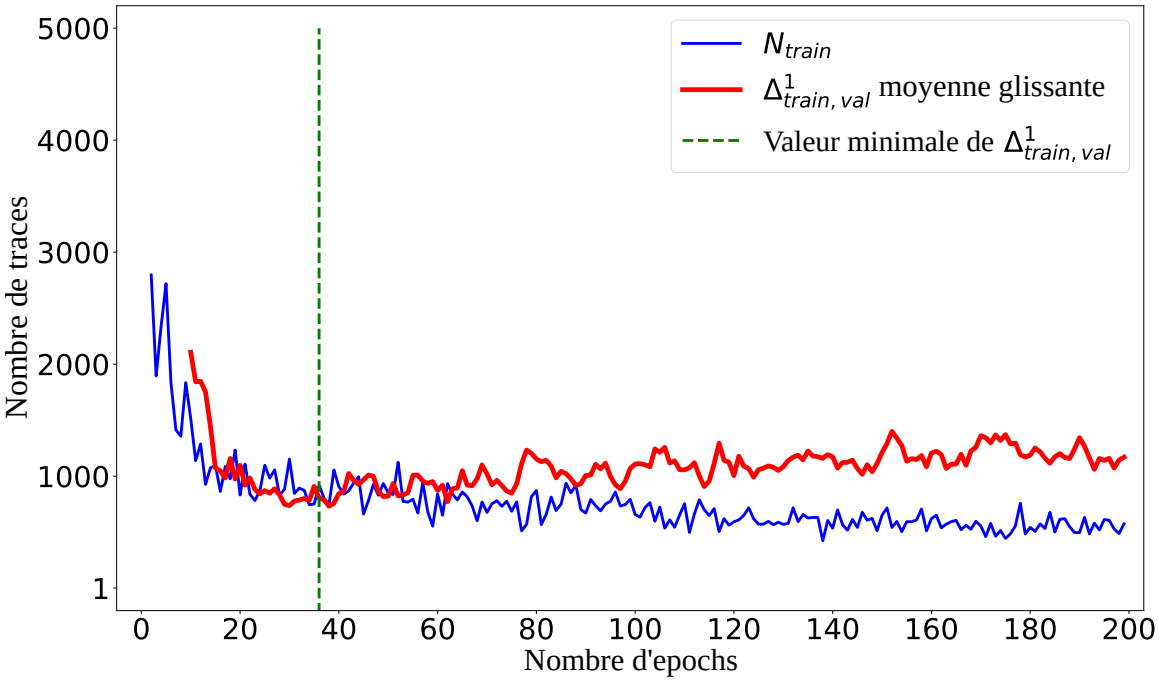


FIGURE 3.9: Évolution de $\Delta^1_{train,val}$ et N^1_{train} durant l'entraînement du réseau CNN_{BV} 0CONV en utilisant la fonction de perte CCE.

0CONV avec pour fonction de perte la CCE.

L'évolution de $\Delta^1_{train,val}$ et de N^1_{train} pour cet entraînement est illustrée Figure 3.9. On y voit que la convergence de $\Delta^1_{train,val}$ se fait plus rapidement puisqu'il est déjà possible d'obtenir un taux de succès de 90% sur la validation à partir de l'epoch 10 environ. On peut aussi remarquer que l'instabilité présente dans l'entraînement précédemment a en grande partie disparue, il est donc plus aisé d'identifier les périodes d'underfitting, de généralisation et d'overfitting. Le meilleur compromis est obtenu cette fois-ci à l'epoch 37 où le réseau est capable de réussir ses attaques sur l'ensemble de validation en environ 1300 traces. Une fois ce stade passé, on remarque une augmentation légère de la valeur de $\Delta^1_{train,val}$ jusqu'à la fin de l'entraînement causée en partie par la décroissance de N^1_{train}. Cela indique les premiers signes d'overfitting même si l'impact sur les performances de validation est encore léger. En effet, la capacité du réseau reste faible sans couche de convolution, cela veut dire qu'il aura moins tendance à sur-apprendre les exemples d'entraînement. L'application de la technique d'early stopping nous permet encore une fois de gagner en performance puisque arrêter le réseau au minimum de $\Delta^1_{train,val}$ réduit le nombre de traces nécessaires à l'attaque de près de 28% comparé à un entraînement de 50 epochs où le réseau a besoin de 1800 traces pour atteindre un taux de succès de 90%.

La dernière figure, Figure 3.10, représente l'évolution de $\Delta^1_{train,val}$ et de N^1_{train} au cours de l'entraînement de CNN_{BV} 1CONV avec la fonction de perte CCE. Sur cette figure, on retrouve un comportement de $\Delta^1_{train,val}$ proche de l'entraînement précédent mais cette fois-ci, l'overfitting a plus d'impact sur les performances de validation. En effet, grâce à sa couche de convolution, le réseau est capable d'apprendre à prédire presque parfaitement les exemples de l'ensemble d'entraînement comme en témoigne l'évolution de N^1_{train}. On peut en déduire que l'augmentation de la métrique, passée l'epoch 42 qui correspond à

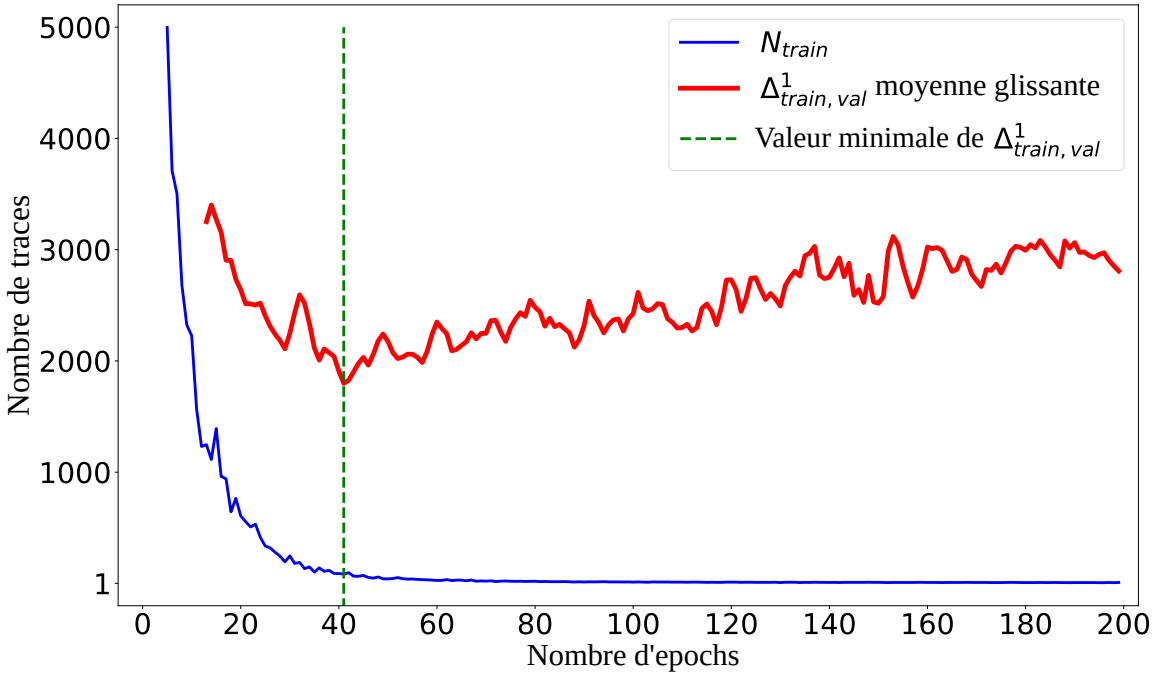


FIGURE 3.10: Évolution de $\Delta^1_{train,val}$ et N^1_{train} durant l'entraînement du réseau CNN_{BV} 1CONV en utilisant la fonction de perte CCE.

son minimum, est due à l'effet de l'overfitting sur les performances en validation. À ce minimum, le réseau est capable de réussir ses attaques en environ 1650 traces ce qui est meilleur qu'aux epochs 25 et 50 choisies dans l'article de van der Valk *et al.* où le réseau a besoin respectivement de 2700 et 2150 traces.

Pour en finir sur les expériences réalisées, un entraînement du réseau CNN_{BV} 1CONV a été effectué avec la fonction de perte MSE mais le réseau n'a été capable d'atteindre un taux de succès de 90% en moins de 5000 traces sur l'ensemble de validation pour aucune des epochs. Cela met en avant l'importance du choix de la fonction de perte dans l'apprentissage des réseaux. On peut aussi voir à partir de ces expériences qu'un réseau avec une faible capacité est moins victime du phénomène d'overfitting mais que cela ne le rend pas nécessairement meilleur. Toutefois, un réseau avec une grande capacité a tendance à subir l'overfitting plus rapidement et avec un impact plus fort, il faut donc être attentif à son entraînement pour limiter l'impact sur les performances du réseau. L'utilisation de régularisation est aussi fortement recommandée pour les réseaux plus complexe.

Pour conclure, on peut remarquer que grâce à l'utilisation de la métrique $\Delta^1_{train,val}$, la meilleure epoch d'entraînement est choisie de manière consistante. Cela permet parfois de réduire le temps d'entraînement par rapport à un nombre d'epoch fixe mais surtout d'éviter les zones d'underfitting et d'overfitting. Il en résulte un gain non-négligeable en termes de performance sur l'ensemble de validation ce qui se traduit aussi par un gain en performance sur l'ensemble de test. Les résultats des divers entraînements sont résumés dans le Tableau 3.1. Ces résultats confirment que l'utilisation de la CCE comme fonction de perte conduit à de meilleurs entraînements comme affirmer par Masure *et al.* [MDP20].

TABLE 3.1: Résumé des résultats obtenus avec en **gras** les choix fait grâce à $\Delta^1_{train,val}$

Réseaux	Nb de couches CONV	Loss	Référence	Nb epochs	N^1_{val}	$\Delta^1_{train,val}$	Différence en temps et performance entre références
CNN_{best}	5	CCE	[BPS+19]	75	1151	1145	Temps : -33.3%
			[RZC+20]*	**47**	**802**	**779**	N^1_{val} : -30.3%
CNN_{BV}	0	MSE	[vdVP19]	50	2960	1449	Temps : +98.5%
			[RZC+20]*	**109**	**2093**	**954**	N^1_{val} : -29.3%
		CCE	[vdVP19]	50	1849	915	Temps : -22.7%
			[RZC+20]*	**37**	**1331**	**413**	N^1_{val} : -28.6%
	1	CCE	[vdVP19]	50	2177	2136	Temps : -20.4%
			[RZC+20]*	**42**	**1659**	**1575**	N^1_{val} : -23.7%

** : publication issue de ces travaux de thèse*

3.5 Discussions sur la métrique $\Delta^d_{train,val}$

La métrique $\Delta^1_{train,val}$ s'est montrée efficace lors de nos tests mais nous pouvons malgré tout discuter de certains aspects qui n'ont pas pu être explorés.

Le premier concerne la valeur de l'ordre considéré pour le taux de succès des attaques. Lors de nos expériences, nous avons pu garder un taux de succès d'ordre 1 car le réseau et l'ensemble de données utilisés le permettaient. En effet, en partant d'un réseau déjà existant ayant été ajusté pour réussir à attaquer le dataset ASCAD, il a été naturel de considérer un taux de succès à l'ordre 1. Dans les autres cas, il est possible de modifier la valeur de l'ordre considérée pour le taux de succès afin d'obtenir une information sur $\Delta^d_{train,val}$ et les performances du réseaux même lorsque les attaques ne réussissent pas. Cela permet de procéder par étapes en améliorant le réseau grâce aux informations données par la métrique et en diminuant l'ordre du taux de succès en parallèle.

Un autre point intéressant à mentionner concerne la valeur de $\Delta^d_{train,val}$ en elle-même. En effet, en considérant seulement la valeur de $\Delta^d_{train,val}$, il n'est pas forcément possible d'en déduire les performances futures du réseau étant donné que la métrique est un mélange des performances sur les exemples d'entraînement et de validation. Par exemple, dans un cas où $\Delta^d_{train,val} = 10$, on ne peut pas savoir directement si cela vient du fait que $N^d_{train} = 10$ et $N^d_{val} = 20$ ou que $N^d_{train} = 2000$ et $N^d_{val} = 2010$. Cependant, et même si il n'est pas possible de généraliser cela, la nature des réseaux de neurones fait qu'il est bien plus probable de se trouver dans le premier cas de figure, c'est-à-dire une valeur faible de N^d_{train} et N^d_{val}, que dans le second cas. Cela vient du fait que des valeurs proches de N^d_{train} et N^d_{val} malgré le fait que leurs valeurs sont grandes indiqueraient une généralisation quasi parfaite des connaissances du réseau en même temps que des performances relativement faibles puisqu'il reste au réseau beaucoup de marge pour améliorer ses performances à l'entraînement. Pour conclure sur ce point, même si il n'existe pas de garanti qu'une faible valeur de $\Delta^d_{train,val}$ soit liée à de bonnes performances lors de l'application du réseau, ce cas de figure reste le plus probable de manière générale.

Le dernier point intéressant à discuter concerne le désavantage majeur de la métrique

$\Delta^d_{train,val}$ qui est le besoin d'exécuter les attaques sur les données d'entraînement et de validation. Ces attaques peuvent se montrer très coûteuses en termes de calculs selon les ensembles de données utilisés, notamment si l'information présente dans les traces est faible et qu'il faut un grand nombre de traces pour faire converger le rang. Dans ce genre de cas, il peut alors être intéressant d'adapter la façon de surveiller la valeur de $\Delta^d_{train,val}$ au cours de l'entraînement si son calcul classique est très coûteux. Une possible adaptation, autre que celle discutée Section 3.3.4, est qu'au lieu de calculer précisément le nombre de traces nécessaires pour une attaque, ce nombre peut être calculé par pallier, par exemple une fois toutes les 5 ou 10 traces voire plus si nécessaire. Cette modification fait perdre en précision mais la précision n'est pas un point important si les attaques elles-mêmes ont du mal à réussir. La métrique $\Delta^d_{train,val}$ possède donc plusieurs caractéristiques adaptables qui peuvent être modifiées en fonction des performances du réseau et de l'ensemble de données utilisé.

3.6 Conclusion

Au cours de ce chapitre, nous avons pu discuter de l'importance des métriques d'évaluation en machine learning et deep learning et par extension, l'importance de leur utilisation pour évaluer les réseaux de neurones utilisés pour les attaques par canaux auxiliaires. Nous avons également discuté de deux phénomènes pouvant grandement gêner l'apprentissage des réseaux : l'underfitting et l'overfitting. Ces états du réseau ont plusieurs origines liées principalement à un manque de capacité ou une capacité trop grande du réseau combinée avec un sous-entraînement ou un sur-entraînement. Être capable de les détecter est donc très important. Pour cela, une solution possible est de développer des métriques permettant de favoriser la détection de ces problèmes particuliers. De plus, elles sont essentielles à l'application d'un type de régularisation particulier appelé *early stopping* qui permet de déterminer le meilleur moment pour arrêter l'entraînement suivant des conditions spécifiques. Ces conditions sont faites pour obtenir, de manière générale, un bon compromis entre les performances du réseau sur les données d'entraînement et sur les données de validation.

Afin d'utiliser cette technique, nous avons mis en place une métrique qui compare les performances du réseau, en termes d'attaque par canaux auxiliaires, à la fois sur les traces d'entraînement et sur les traces de validation. Les performances sont déterminées par le nombre de traces nécessaires au réseau pour retrouver la clé avec un taux de succès de 90% pour l'entraînement ou la validation, notés respectivement N^1_{train} et N^1_{val}. Ces nombres de traces sont comparés à l'aide d'une distance euclidienne pour évaluer la différence, appelée $\Delta^1_{train,val}$, entre les performances sur les deux ensembles. En calculant cette valeur régulièrement durant l'entraînement, il est possible de déterminer l'état du réseau. Lorsque la courbe de $\Delta^1_{train,val}$ décroît régulièrement, cela indique que l'écart de performances du réseau se réduit et donc qu'il arrive à généraliser ses connaissances à l'ensemble de validation. $\Delta^1_{train,val}$ atteint ensuite un minimum qui correspond au moment où le compromis entre performance sur l'ensemble d'entraînement et l'ensemble de validation est le meilleur. C'est généralement ce réseau qui obtiendra les meilleurs performances une fois

appliqué sur l'ensemble de test. Il est toutefois nécessaire de continuer l'entraînement pour quelques epochs de plus pour s'assurer que le minimum a bien été trouvé. Si c'est le cas, la valeur de $\Delta^1_{train,val}$ augmente de manière régulière à partir de cette epoch. Cette augmentation peut être liée à au moins deux phénomènes : soit une augmentation de N^1_{val} due à l'impact de l'overfitting sur la généralisation du réseau, soit une baisse de la valeur de N^1_{train} causée par un sur-entraînement et une faible capacité qui n'entraîne pas d'augmentation significative de N^1_{val}, soit une combinaison des deux liée à un sur-entraînement et une grande capacité du réseau qui provoque à la fois un apprentissage par cœur des exemples d'entraînement et une baisse des performances sur l'ensemble de validation.

Ces scénarios sont explorés via l'entraînement de plusieurs réseaux sur la base de données ASCAD qui permet une reproductibilité simple des expériences. Les résultats de ces expériences confirment que les réseaux obtenus via la technique d'early stopping en utilisant la métrique $\Delta^1_{train,val}$ montrent les meilleures performances sur l'ensemble de validation et par extension l'ensemble de test. De plus, en examinant les performances du réseaux à son minimum, il est possible de déterminer le besoin en régularisation du réseau. En effet, pour le réseau CNN_{best}, il est clair, au vu de la valeur de N^1_{train} au minimum de $\Delta^1_{train,val}$, que le réseau présente des signes d'overfitting même si il est capable de bien généraliser. Cet overfitting est liée à la capacité trop grande du réseau et l'early stopping ne peut pas prévenir de son impact, il faut donc appliquer de la régularisation explicite pour essayer de contrer l'overfitting et ses effets négatifs.

Nous avons finalement discuté de certains points de la métrique $\Delta^d_{train,val}$ qui n'ont pas pu être explorés au cours de la thèse. Cette discussion répond aux potentielles faiblesses qui peuvent être relevées concernant la métrique telle que décrite dans sa forme originelle. Ces réponses restent de l'ordre théorique et il serait intéressant de les explorer plus en détails afin de renforcer l'intérêt de l'utilisation de $\Delta^d_{train,val}$ dans l'évaluation des réseaux de neurones entraînés pour les attaques par canaux auxiliaires.

L'impact de l'overfitting peut être réduit à l'aide de différentes techniques de régularisation comme par exemple l'augmentation de données, l'ajout de bruit ou le choix d'une architecture plus appropriée. Ces techniques utilisées pour réduire l'overfitting seront présentées dans le prochain chapitre et nous nous attarderons plus particulièrement sur une technique d'amélioration de performances, la *batch normalization*, et d'autres techniques de régularisations, le *weight decay* et le *dropout*, afin d'améliorer les performances du réseau.

Chapitre 4

Études de différentes techniques d'optimisation des réseaux de neurones

Après avoir vu les principes d'évaluations des réseaux de neurones et leur application à un exemple concret, nous allons nous concentrer sur l'amélioration des performances des réseaux. Les techniques que nous allons utiliser pour cela sont bien connues en machine learning : la *batch normalization* et les techniques de régularisation *weight decay* et *dropout*. La première a pour but d'aider le réseau de neurones lors de son apprentissage en réduisant la variance des distributions des données entre les couches du réseau après chaque mise à jour. Les deux autres appliquent une régularisation directe sur les valeurs des poids du réseau et vont ainsi participer à la réduction de l'overfitting du réseau. La première partie de ce chapitre servira de discussion sur certaines techniques déjà utilisées dans la littérature et les bénéfices qu'elles apportent. Ensuite, nous introduirons les techniques que nous avons étudiées en détails en expliquant leur fonctionnement avant de les appliquer au réseau CNN_{best} en conjugaison avec la métrique $\Delta^1_{train,val}$. L'objectif est de mettre en évidence les gains apportés par ces techniques ainsi que l'avantage de l'utilisation de $\Delta^1_{train,val}$ pour déterminer le potentiel d'amélioration du réseau.

4.1 Techniques d'amélioration de performance utilisées pour les applications des réseaux en analyses de canaux auxiliaires

Les techniques d'amélioration de performance sont très utilisées en machine learning et leurs effets sont généralement bien compris. C'est pour cela que la recherche récente en deep learning appliqué aux attaques par canaux auxiliaires a commencé par essayer de tirer partie de cette réserve de techniques existantes afin d'améliorer les performances des réseaux de neurones utilisés. Le Tableau 4.1 résume les différents travaux menés sur ces techniques et indique également comment elles ont été appliquées : une simple utilisation ou une plus profonde exploration. Effectivement, une partie des articles se contente d'utiliser les techniques sans rentrer dans le détail des améliorations qu'elles apportent, elles sont mentionnées comme "Utilisée" dans le Tableau 4.1. D'autres articles se concentrent sur la compréhension des gains de performance apportés par ces techniques et elles sont alors décrites comme "Explorée".

TABLE 4.1: Résumé des différentes méthodes utilisées et explorées pour l'amélioration des performances des réseaux de neurones dans le cadre d'attaques par canaux auxiliaires.

Référence \ Méthode d'amélioration	Ajustement d'architecture	Augmentation des données	Ajout de bruit	Batch Norm.	Dropout	Weight decay	Early Stopping
Zaid *et al.* [ZBHV20a] [ZBHV20b]	Explorée	-	-	-	-	-	-
Wouters *et al.* [WAGP20]	Explorée	-	-	-	-	-	-
Cagli *et al.* [CDP17]	-	Explorée	-	-	-	-	-
Picek *et al.* [PHJ+19]	-	-	Explorée	-	-	-	-
van der Valk *et al.* [vdVP19]	-	-	-	Utilisée	Utilisée	-	-
Masure *et al.* [MDP19]	-	-	-	Utilisée	-	-	-
Perin *et al.* [PBP]	-	-	-	-	Utilisée	-	Explorée
Li *et al.* [LKP20]	-	-	-	-	Utilisée	-	-
Weissbart *et al.* [WPB19]	-	-	-	-	-	Utilisée	-
Zhang *et al.* [ZZN+20]	Explorée	-	-	-	-	-	-
Robissout *et al.* [RBHG21]*	-	-	-	Explorée	Explorée	Explorée	Utilisée

** : publication issue de ces travaux de thèse.*

Un certain nombre des articles récents s'est concentré sur l'ajustement des architectures des réseaux de neurones utilisés [ZBHV20a, WAGP20, ZBHV20b, ZZN+20]. Par exemple, Zaid *et al.* [ZBHV20a] ont appliqué cette méthode au réseau CNN_{best} et à la base de donnée ASCAD. Ils ont réussi en réduisant de manière importante la taille du réseau à améliorer significativement ses performances en termes du nombre de traces d'attaque minimum pour retrouver la clé. Ils ont également mis au point une méthodologie permettant d'ajuster un réseau de neurones en fonction des caractéristiques de l'ensemble de traces utilisé. L'architecture résultante présente ainsi de bonnes performances lors des attaques utilisant les traces issues de cet ensemble. Cette méthodologie a été mise au point à l'aide de techniques de visualisation des poids du réseau de neurones permettant de déduire quels hyperparamètres sont efficaces contre certaines contremesures comme le masquage et la désynchronisation. Wouters *et al.* [WAGP20] ont continué d'étudier cette méthodologie et ils sont arrivés à la conclusion que l'utilisation de batch normalization pourrait permettre de se passer de l'étape de normalisation des traces faites avant l'entraînement. Zhang *et al.* [ZZN+20], eux, utilisent la métrique qu'ils ont mise au point, discutée dans la Section 3.2.3, pour valider le choix de leurs hyperparamètres. Contrairement à ces travaux, les techniques dont nous discuterons dans la suite de ce chapitre n'ont pas d'impact sur l'architecture du réseau.

Nous reviendrons plus en détails sur les travaux de Cagli *et al.* [CDP17] et Kim *et al.* [KPH+19] qui explorent certaines techniques de régularisation agissant au niveau de l'ensemble d'entraînement du réseau. Les autres articles utilisent des techniques d'amélioration de performance sans prendre le temps d'étudier ou de discuter de l'impact qu'elles ont sur leurs résultats. C'est pourquoi il nous est apparu nécessaire d'étudier en profondeur les techniques de batch normalization, dropout et régularisation par norme L2, aussi appelé *weight decay*, pour les réseaux utilisées dans le cadre des attaques par canaux auxiliaires.

4.2 Introduction de la batch normalization

La batch normalization est une technique de normalisation introduite par Ioffe *et al.* [IS15] en 2015. Son but est de normaliser les distributions des données en entrée de chaque couche du réseau afin de réduire l'impact des différentes mises à jour des poids du réseau sur ces distributions. Cela est formalisé par la notion de réduction du *internal covariate shift* ou ICS. L'ICS représente le changement dans la distribution des données de sortie des différentes couches durant l'apprentissage.

Pour donner une intuition de la motivation de Ioffe *et al.*, le principe est que les distributions des différents batch de données peuvent être très différentes. Par conséquent, les poids appris à une itération risquent d'être inadaptés pour bien prédire une autre itération et cela peut engendrer des valeurs de loss importantes. Ces fortes valeurs de la fonction de perte mènent à des gradients fortement différents d'un batch à l'autre qui provoque des difficultés de stabilisation du processus d'apprentissage. L'objectif de la batch normalisation est donc de réduire cet effet. Néanmoins, comme nous le verrons un peu plus tard, la justification de l'intérêt de la batch normalization comme une solution au problème de l'ICS a été remise en cause par un article récent qui argumente plutôt la batch normalisation comme une manière de lisser le problème d'optimisation de manière à rendre sa résolution plus simple.

La batch normalization est de nos jours utilisée dans la plupart des réseaux de neurones [BCN18] grâce au gain de performance qu'elle permet, même si l'origine de ce gain fait encore l'objet de recherche. Il est donc intéressant de voir les améliorations qu'elle peut apporter aux réseaux de neurones utilisés pour les attaques par canaux auxiliaires.

4.2.1 Processus de normalisation

Le processus de normalisation des données se fait à l'aide de couches spéciales dans le réseau appelées couches de batch normalization qui prennent en entrée la sortie d'une couche quelconque du réseau et normalise les données pour les fournir à la couche suivante. La notion de batch vient du fait que ce processus prend en compte tous les exemples d'un batch pour les calculs de normalisation. Il s'applique donc à chacun des batchs d'entraînement. La normalisation utilisée pour la batch normalization, consiste à prendre un ensemble de données et à modifier sa distribution afin qu'elle ait une moyenne de 0 et un écart-type de 1. La Figure 4.1 illustre l'effet de la normalisation sur les distributions de l'entrée d'une couche du réseau. Comme on peut le constater sur cette figure, cela permet d'homogénéiser les distributions ce qui conduit à des valeurs numériques plus proches les unes des autres tout en maintenant les écarts de proportions. L'impact des larges valeurs numériques des paramètres sur le gradient est ainsi moins important ce qui permet l'utilisation d'un learning rate plus grand et donc une aide à la convergence rapide vers une solution en limitant le risque de divergence du gradient [IS15]. La distribution des données de chaque couche est moins affectée par les mises à jour des poids et donc les couches suivantes n'ont pas à s'adapter aux nouvelles distributions qui résultent de ces changements de valeurs. L'apprentissage gagne alors en rapidité puisque de manière généralement, les calculs des gradients utilisés lors de la rétropropagation sont faits sous

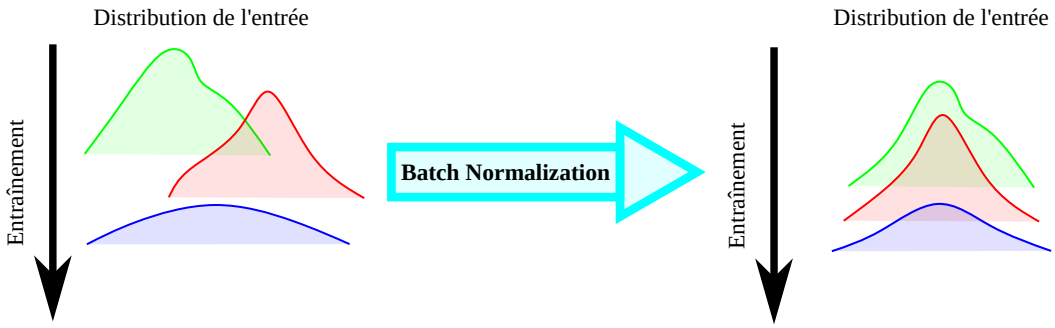


FIGURE 4.1: Effet de la batch normalization sur la distribution de l'entrée d'une couche.

l'hypothèse que les autres valeurs des poids du réseau reste fixe. Il en suit que limiter les changements possibles des distributions des valeurs des poids aide en partie à respecter cette hypothèse.

4.2.2 Couche de normalisation

La couche de normalisation est introduite par Ioffe *et al.* [IS15] afin d'appliquer la normalisation entre deux autres couches du réseau. Elle se situe généralement avant la fonction d'activation [STIM18] et elle est unique pour chaque couche et chaque batch d'exemples d'entraînement. Les paramètres de normalisation sont calculés à chaque nouveau batch pour chacune des couches afin que la distribution des données reste similaire au batch précédent. Le calcul de normalisation de l'entrée $x = (x^{(1)}, x^{(2)}, ..., x^{(d)})$ à d dimensions de la couche se fait de la manière suivante :

$$BN(x^{(k)}) = \gamma^{(k)} \frac{x^{(k)} - \mu^{(k)}}{\sigma^{(k)} + \epsilon} + \beta^{(k)}, \tag{4.1}$$

pour chaque dimension $k = 1, ..., d$ où $\mu^{(k)}$ est la moyenne de la dimension k, $\sigma^{(k)}$ est l'écart-type de la dimension k et où $\epsilon > 0$ est une constante ajoutée à des fins de stabilité numérique. En plus de $\mu^{(k)}$ et $\sigma^{(k)}$, des paramètres $\gamma^{(k)}$ et $\beta^{(k)}$ sont ajoutés au calcul pour réajuster la valeur de sortie si besoin. Ces paramètres $\gamma^{(k)}$ et $\beta^{(k)}$ sont entraînés et servent à éviter que la transformation de normalisation ne change ce que peut représenter la couche du réseau. À l'aide de ces paramètres, on s'assure que la couche de normalisation peut représenter la transformation identité si besoin.

Même si Ioffe *et al.* argumentent que le gain en performance des réseaux normalisés vient d'une réduction de l'ICS [IS15], Santurkar *et al.*, ont observé un autre effet : le lissage de la représentation de la fonction de perte [STIM18] que nous allons maintenant décrire.

4.2.3 Lissage de la fonction de perte

Santurkar *et al.* [STIM18] discutent dans leur article de ce qu'ils pensent être la vraie raison de l'efficacité des couches de normalisation : le lissage du paysage de la fonction de perte.

Dans un premier temps, ils démontrent, à partir d'une expérience, l'absence de connexion concrète entre la réduction de l'ICS et le gain en performance du réseau. Cette expérience consiste en l'ajout de bruit entre la couche de normalisation et la fonction d'activation. Le bruit étant choisi avec une moyenne et un écart type aléatoire, il est créé spécifiquement pour augmenter l'ICS. Grâce à cela, Santurkar *et al.* constatent qu'un réseau avec de la normalisation et qu'un réseau avec à la fois de la normalisation et du bruit obtiennent des performances similaires. Le gain de performance lié à la normalisation ne semble donc pas venir de la réduction de l'ICS. Ils remarquent également que, dans leurs expériences, même si les distributions en entrée des couches fluctuent, les changements restent légers et que leur impact n'est en conséquence pas si important sur le problème d'optimisation à résoudre.

Santurkar *et al.* continuent en étudiant la représentation de la fonction de perte et constatent que le fait d'utiliser des couches de normalisation lisse cette représentation. Ce lissage apparaît lorsqu'ils explorent le paysage de la fonction de perte. Cette exploration est illustrée Figure 4.2 où chaque flèche noire représente un pas de taille différente et les flèches rouges correspondent à la direction du pas suivant. À partir des poids du réseau $W^{(l)}$ et de la fonction de perte $\mathcal{L}(W^{(l)})$, on peut calculer la direction du gradient $-pas \times \nabla_{W^{(l)}}\mathcal{L}(W^{(l)})$ qu'il faut suivre pour minimiser $\mathcal{L}(W^{(l)})$. Le paramètre *pas* est là pour pondérer la taille du pas suivi dans cette direction, il est équivalent au learning rate que l'on a vu dans le Chapitre 2. L'exploration a lieu en faisant varier la valeur du paramètre *pas*. Chaque mise à jour des poids $W^{(l)}$ après l'application du gradient donne des nouveaux poids W'. On peut ensuite utiliser ces nouveaux poids pour calculer la direction du nouveau gradient $-\nabla_{W'}\mathcal{L}(W')$. En comparant ces deux valeurs, on obtient une idée de la forme du paysage de la fonction de perte. Si la différence entre les directions est grande, cela indique des changements de direction brusques qui vont rendre l'optimisation difficile et forcer l'utilisation d'un learning rate faible. Au contraire, si les différences sont faibles, cela signifie que le paysage de la fonction de perte est plus lisse et que la direction du gradient est prévisible. Un learning rate plus élevé peut donc être utilisé et l'optimisation se fera plus rapidement.

En réalisant cette expérience sur un réseau sans normalisation et un réseau avec normalisation, Santurkar *et al.* ont constaté que la présence de normalisation rendait la direction du gradient plus prévisible et avait un effet de lissage sur le paysage de la fonction de perte. C'est selon eux ce qui explique les améliorations apportées par les couches de batch normalization.

Ces couches permettent donc potentiellement aux réseaux d'apprendre plus rapidement à partir des exemples d'entraînement, mais, si elles appliquent une certaine forme de régularisation, elles ne sont pas forcément suffisantes pour limiter le phénomène d'overfitting. Il faut alors trouver d'autres moyens pour contrôler ce phénomène. Une solution est de travailler sur des approches de régularisation du réseau lors de l'apprentissage. La régularisation est un procédé général qui a pour but de contrôler l'évolution de la complexité des paramètres du modèle de manière à améliorer sa capacité de généralisation. Une approche que nous allons considérer dans le prochain chapitre consiste à contrôler

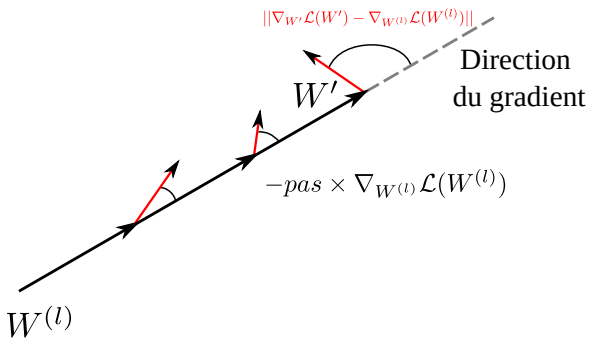


FIGURE 4.2: Changement dans la direction du gradient selon la taille du pas utilisé précédemment.

les paramètre en régulant leur évolution selon une norme. La combinaison de ce type de régularisation avec la batch normalisation permet de mieux maîtriser les effets de l'overfitting. Nous allons maintenant étudier dans la prochaine section quelques techniques de régularisation utilisables conjointement avec les couches de normalisation que nous venons de voir.

4.3 Introduction aux différentes formes de régularisation

Nous avons vu dans le Chapitre 3 la technique de l'early stopping qui est utilisée pour régulariser un réseau sans appliquer de contraintes directes sur les valeurs de ces paramètres. Dans cette section, nous discuterons dans un premier temps de techniques de régularisation appliquées aux réseaux de neurones pour les attaques par canaux auxiliaires présentes dans la littérature. Nous verrons ensuite plus en détails deux techniques différentes qui pénalisent directement les poids lors de l'entraînement.

4.3.1 Discussion sur les formes de régularisation déjà explorées

Augmentation des données ou *Data Augmentation*

L'augmentation des données [SK19], ou *Data Augmentation*, est une technique consistant à appliquer une transformation sur certains exemples d'entraînement afin d'en créer une copie modifiée qui est ajoutée à l'ensemble d'entraînement. Cette méthode applique de la régularisation au réseau en modifiant l'ensemble d'entraînement et en ajoutant des exemples généralement plus complexes à prédire.

L'augmentation des données peut être utile lorsque le nombre d'exemples d'entraînement est faible. Il est alors possible d'utiliser les exemples disponibles et de les modifier afin d'en créer de nouveaux qui permettront d'entraîner le réseau plus efficacement. Par exemple en reconnaissance d'image, des translations, des effets miroirs ou des réductions de la taille des images [KSH12, CMS12] peuvent être appliqués pour obtenir de nouveaux exemples. L'augmentation des données est aussi pratique dans les contextes de classes déséquilibrés [LKBS18] pour lesquels les nombres d'exemples disponibles pour les classes les moins courantes sont plus faibles rendant l'entraînement plus difficile. Il est alors possible d'utiliser l'augmentation des données pour créer de nouveaux exemples des classes

les moins représentées afin d'équilibrer toutes les classes. Cela permet d'entraîner le réseau de manière plus classique. Il faut toutefois faire attention à ce que les transformations modifient suffisamment les exemples pour ne pas biaiser le réseau tout en gardant la représentativité des exemples pour ne pas que le réseau apprenne de caractéristiques inutiles.

Ces applications ont pour but de générer un ensemble d'entraînement plus propice à l'obtention d'un réseau performant mais il existe une application plus ciblée. L'augmentation des données peut être utilisée pour augmenter la robustesse d'un réseau face aux perturbations qui peuvent se produire sur ses entrés lors de son utilisation.

Dans le cadre de l'utilisation de réseau de neurones pour les attaques par canaux auxiliaires, Cagli *et al.* [CDP17] ont appliqué la data augmentation afin d'augmenter la robustesse de leur réseau face à deux types de perturbations qui peuvent se produire lors de l'acquisition de traces via les canaux auxiliaires. Ces transformations sont nommées *Add-Remove Deformation* et *Shifting Deformation*. Le Add-Remove simule l'effet du jitter d'horloge sur les acquisitions des traces de consommation de courant, il retire et ajoute aléatoirement des points dans les traces. Les valeurs des points ajoutés sont calculées à partir d'une moyenne arithmétique des valeurs qui l'entourent. Le Shifting est une transformation de désynchronisation qui décale la fenêtre dans laquelle la trace est choisie au sein de l'acquisition globale de consommation de courant par exemple. Cela a pour effet que deux points correspondant à l'exécution d'une même opération ne se situent plus au même endroit sur la trace. L'ajout de nouveaux exemples transformés au sein de l'ensemble d'entraînement permet donc au réseau de se familiariser avec ces perturbations et d'apprendre à être plus robuste contre elles.

Un autre effet de l'augmentation des données dans ce contexte est la réduction de l'overfitting du réseau grâce à son effet de régularisation. L'ajout d'exemples différents des exemples classiques rend le problème d'apprentissage plus complexe et cela a comme impact une réduction de l'overfitting puisqu'il est plus difficile pour le réseau d'appendre les exemples par cœur. Cela permet à Cagli *et al.* [CDP17] d'utiliser une architecture complexe en limitant les effets de l'overfitting et ainsi d'obtenir de bonnes performances lors de leurs attaques.

Ajout de bruit dynamique dans l'entrée du réseau

Une autre méthode similaire à l'augmentation des données a été utilisée par Kim *et al.* [KPH+19] afin de lutter contre l'overfitting. Ils ont eux opté pour un ajout de bruit directement dans la sortie de la première couche de leur réseau. Ce bruit est issu d'une distribution normale de moyenne nulle avec un écart-type variable entre 0 (pas d'ajout de bruit) et 1 (ajout de bruit important). Les valeurs de bruit sont tirées indépendamment pour chaque batch d'exemples utilisé lors de l'entraînement. Le bruit n'est pas ajouté aux traces directement ce qui le différencie de l'augmentation de données classique. Il reste toutefois équivalent à une large augmentation des données où chaque exemple n'est utilisé qu'une seule fois et partage la même transformation que les autres exemples du batch. Selon les résultats de Kim *et al.*, l'ajout d'un bruit à l'intérieur du réseau permet bien d'améliorer les performances du réseau pour les attaques à condition que ce bruit ne soit pas trop impor-

tant.

Les deux techniques présentées ici agissent à différents niveaux de l'entraînement des réseaux de neurones : au niveau de l'ensemble d'entraînement pour l'augmentation des données et au niveau de la perception des exemples par le réseau pour l'ajout de bruit. Nous allons maintenant discuter de techniques qui, elles, ont un impact direct sur les valeurs des poids du réseau. Contrairement aux autres techniques vues jusque là, la méthodologie que nous allons suivre consiste à fixer l'architecture du réseau et l'ensemble d'entraînement et ensuite de régler la quantité de régularisation appliquée au réseau afin d'en tirer de meilleures performances.

4.3.2 Régularisation par norme L1 et par norme L2

Les régularisations L1 et L2, aussi connues sous le nom de régularisation par norme L1 et L2, sont un type de régularisation qui ajoutent une contrainte lors du calcul de la fonction de perte qui va dépendre de la complexité du réseau. La contrainte est calculée à l'aide des normes L1 et/ou L2 appliquées aux paramètres du réseau qui vont traduire deux types de complexités différentes que nous allons détailler.

Régularisation par norme L2 Cette régularisation évalue la complexité du réseau via les valeurs de ses poids et a été introduite en machine learning par Krogh *et al.* [KH92]. Plus spécifiquement, le terme ajouté lors du calcul de la fonction de perte représente la somme des valeurs des carrés des poids du réseau. La fonction de perte devient donc :

$$\begin{aligned}\mathcal{L}^*(\theta) &= \mathcal{L}(\theta) + \alpha||\theta||_2, \\ &= \mathcal{L}(\theta) + \alpha \sum_{\theta_i \in \theta} \theta_i^2,\end{aligned}$$

où θ représente l'ensemble des paramètres du réseau et α est un hyperparamètre contrôlant l'importance de la régularisation.

Le terme ajouté $\alpha||\theta||_2$ va donc varier en même temps que les valeurs des carrés des paramètres du réseau. En pratique, cela implique que les poids ayant de petites valeurs auront peu d'influence dans le calcul de la fonction de perte contrairement aux poids importants. La pénalisation se concentre donc sur ces derniers afin d'éviter qu'ils ne deviennent trop grands [KH92]. Dans l'ensemble, ce terme de régularisation fait en sorte que les poids du réseau soient proches de zéro sans les inciter à prendre la valeur zéro contrairement à la régularisation par norme L1 que nous verrons par la suite. Le but de cette application est d'empêcher que le réseau se repose sur une petite partie de ses poids pour prendre ses décisions ce qui risquerait de mener à de l'overfitting. La régularisation par norme L2 permet donc de réduire l'overfitting du réseau en pénalisant les neurones qui prendraient trop d'importance lors de l'apprentissage.

Il est notamment intéressant de combiner la régularisation norme L2 avec la batch normalisation car elle permet d'éviter que les paramètres du réseau augmentent de manière

non bornée pouvant nécessiter l'utilisation de pas d'apprentissage petits, et donc potentiellement ralentir la convergence [VL17]. C'est pour cela qu'il est intéressant d'utiliser la régularisation par norme L2 en conjugaison avec la batch normalization qui permet de compenser ce ralentissement. De plus, l'application de régularisation est particulièrement intéressante lorsqu'elle est utilisée sur un réseau qui présente de bonnes performances sur les données d'entraînement mais a du mal à généraliser sur les exemples de validation. En effet, en pénalisant le réseau lors de son entraînement, on va généralement réduire ses performances sur les exemples d'entraînement pour gagner en généralisation.

Régularisation par norme L1 Cette régularisation est similaire à la précédente mais elle change au niveau du calcul de la complexité des poids du réseau. La somme des carrés des poids est remplacée par une somme des valeurs absolues. La fonction de perte devient alors :

$$\begin{aligned}\mathcal{L}^*(\theta) &= \mathcal{L}(\theta) + \alpha||\theta||_1, \\ &= \mathcal{L}(\theta) + \alpha \sum_{\theta_i \in \theta} |\theta_i|.\end{aligned}$$

L'effet que cette norme a sur les valeurs des poids est différent puisqu'au lieu de contraindre tous les poids à prendre des petites valeurs, les poids faibles sont cette fois-ci forcés à prendre la valeur zéro. Les autres poids ne sont pas limités dans les valeurs qu'ils peuvent prendre. L'application de la régularisation par norme L1 résulte donc généralement en une couche qui possédera une partie de poids à zéro.

Choix de la régularisation pour le réseau de neurones $\mathbf{CNN_{best}}$ Dans le cas de CNN_{best}, sa forte complexité l'entraîne dans un état d'overfitting avec des performances sur les exemples d'entraînement qui convergent bien plus rapidement que les performances sur les exemples de validation. Cette grande complexité du réseau nous a poussé à privilégier la régularisation par norme L2 comme terme de régularisation afin de limiter l'overfitting basé sur le fait qu'un petit nombre de poids prenne trop d'importance dans la prédiction. Cette même réflexion a mené à l'exploration du *dropout* comme autre méthode de régularisation étant donné que le dropout permet de réduire l'interdépendance qu'il peut exister entre les neurones de couches successives dans un réseau.

4.3.3 Dropout

Contrairement aux techniques discutées jusqu'à présent qui agissent sur les valeurs que peuvent prendre les poids du réseau, le dropout a un effet différent. Il existe plusieurs types de dropout mais le plus commun sert à désactiver un pourcentage donné des neurones d'une couche à chaque nouveau batch [SHK+14]. Les neurones désactivés ne sont alors pas pris en compte lors des calculs de prédictions du réseau, l'effet est équivalent à fixer leur activation à zéro ce qui veut dire qu'ils n'ont pas d'impact sur la suite des calculs. Ceci est illustré par la Figure 4.3 qui montre un réseau avant et après application du dropout. Les neurones désactivés sont choisis aléatoirement pour chaque batch et on ob-

tient donc un réseau plus petit qui va faire les prédictions du batch associé. Les calculs de gradient sont eux aussi effectués sur ce réseau réduit, cela veut dire que seuls les poids actifs sont mis à jour. Une des interprétations possible du dropout est que le réseau final obtenu est équivalent à une moyenne effectuée sur l'ensemble des réseaux réduits [LSV19]. C'est-à-dire que l'entraînement se fait réellement sur un grand nombre de ces réseaux plus petits et que le réseau final peut être vu comme une moyenne de tous ces réseaux.

Un des bénéfices du dropout est de réduire temporairement (durant une epoch ou une itération) l'expressivité d'un réseau afin de limiter l'overfitting. Ceci a pour implication d'éviter d'avoir des neurones morts [PK16], c'est-à-dire d'avoir des neurones dont la fonction d'activation reste constante à 0 quelle que soit l'entrée et qui demeurent dans cet état pour le reste de l'entraînement. Le réseau a intuitivement besoin d'utiliser au maximum les neurones présents, il faut donc réduire le plus possible la présence de neurones morts. Cette situation se retrouve très souvent dans le cadre de fonctions d'activation type ReLu et le dropout permet d'en limiter fortement l'impact [PK16].

Un autre effet intéressant de l'inclusion de dropout dans le réseau est la réduction de la co-adaptation entre les neurones [HSK+12]. La co-adaptation est le résultat d'une forte dépendance des neurones les uns sur les autres pour le traitement de l'information au sein du réseau. Cette dépendance les rend moins robustes face à de nouvelles données et donc réduit la généralisation du réseau. Par exemple, certains neurones peuvent apprendre à annuler les sorties d'autres neurones ce qui résulte en une perte nette de capacité puisque les neurones annulés deviennent essentiellement inutiles. Le dropout réduit cet effet de co-adaptation puisqu'il rend incertain la présence des neurones lors du processus d'apprentissage, dû à l'aléa du choix des neurones désactivés, ce qui empêche globalement aux neurones d'apprendre un comportement en lien avec d'autres neurones. Ce faisant, on force les neurones à avoir des comportements indépendants les uns des autres et l'overfitting provenant de la co-adaptation est réduit.

Pour finir, les effets du dropout à l'intérieur des couches convolutionnelles des CNN ont été étudiés en détails par Park *et al.* [PK16]. Ils ont montré que la présence de dropout aide les filtres à apprendre à détecter des caractéristiques plus informatives et donc à améliorer la généralisation du réseau en ajoutant du bruit à la sortie des couches intermédiaires. Ils ont également constaté qu'augmenter la proportion de neurones désactivés pour les couches plus profondes du réseau menait à de meilleurs résultats et c'est cette méthodologie qui est suivie lors des expériences que nous avons faites sur l'application du dropout à CNN_{best}.

La section suivante contient la méthodologie et les résultats des expériences menées sur l'application de la batch normalization puis de la régularisation par norme L2 et dropout au réseau de neurones CNN_{best} [BPS+19]. Ces résultats sont également comparés aux autres résultats des travaux de l'état de l'art (listés dans le Tableau 4.1).

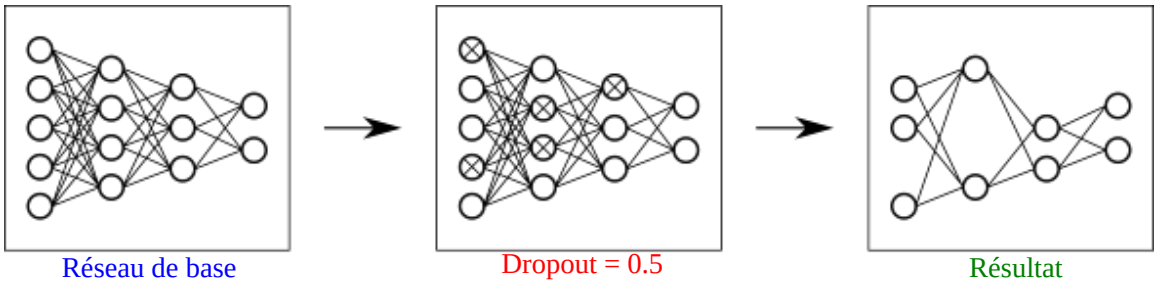


FIGURE 4.3: Exemple des effects du dropout sur un réseau complètement connecté.

4.4 Résultats expérimentaux

4.4.1 Description de l'ensemble de données ASCAD clé variable

Pour toutes les expériences présentées dans cette section, la métrique $\Delta^1_{train,val}$ est calculée durant l'entraînement du réseau afin de déterminer l'epoch présentant les meilleurs performances. Les réseaux sont entraînés à partir d'ensembles issus de l'ensemble de données ASCAD clé variable [BPS+19] qui diffère de l'ensemble ASCAD clé fixe introduit Section 3.4 au niveau des clés utilisées dans les traces d'entraînement. Cette fois-ci, la valeur de la clé utilisée lors des acquisitions est aléatoire et change pour chaque trace de l'ensemble d'entraînement. Il existe également un ensemble utilisant une clé fixe afin de pouvoir évaluer les attaques du réseau une fois l'entraînement terminé. Les détails de l'implémentation restent les mêmes que pour l'ensemble de données vu précédemment mais le nombre de traces présentes dans les différents ensembles varie. Il y a cette fois 200 000 traces d'entraînement, à clé variable, et 100 000 traces d'attaque, à clé fixe. Un autre changement concerne le nombre de points extraits des traces de base qui est cette fois de 1400. Ces points contiennent toujours les fuites de la valeur intermédiaire masquée et du masque. Le modèle de fuite reste donc le même et les labels des traces sont :

$$Y(\mathbf{k}^*) = Sbox(\mathbf{p}[3] \oplus \mathbf{k}^*[3]),$$

où $\mathbf{p}$ est le message d'entrée et $\mathbf{k}^*$ la bonne clé.

Le taux de succès est également calculé à partir de l'exécution de 100 attaques afin de déterminer la valeur de $\Delta^1_{train,val}$ et les attaques sont limitées à 5000 traces. Les attaques exécutées avec les traces à clé variable sont faites en suivant la méthode décrite dans la section suivante tandis que les attaques sur l'ensemble à clé fixe sont faites normalement. Les figures illustrant l'évolution de $\Delta^1_{train,val}$ montrent une courbe obtenue à l'aide d'une moyenne glissante de fenêtre de taille 3.

Présentation des ensembles de données

Les expériences présentées ici sont faites sur différents sous-ensembles de ASCAD clé variable. Ces trois sous-ensembles se nomment : Desync0, Desync50 et Desync100. Cela fait référence au fait qu'ils contiennent chacun différents niveaux de désynchronisation représentatifs des protections présentées en Section 1.2.2. Desync0 n'est composé que de

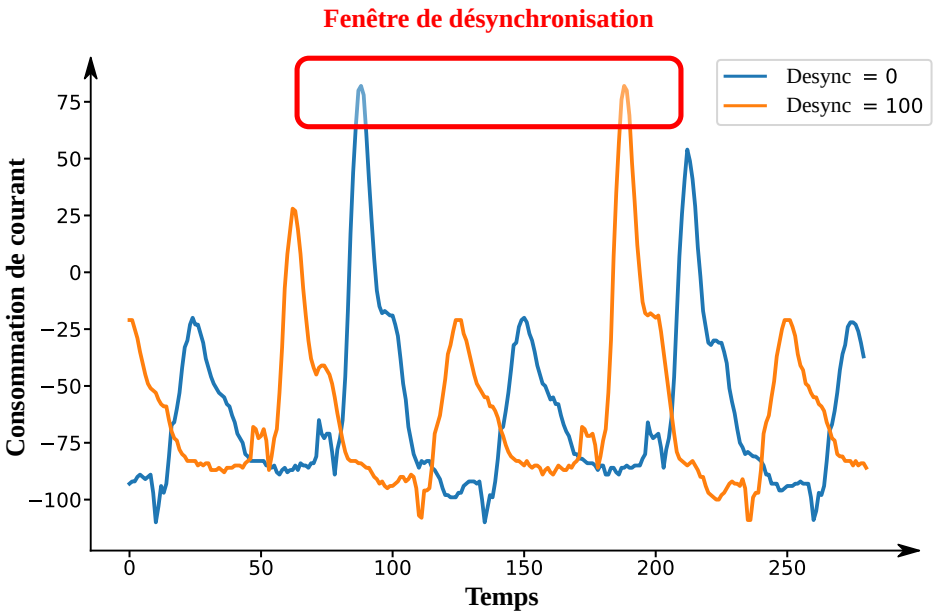


FIGURE 4.4: Traces de consommation de courant pour une desynchronisation $n = 0$ (en bleu) et $n = 100$ (en orange).

traces synchronisées, c'est-à-dire que les mêmes points de différentes traces correspondent aux fuites des mêmes opérations de l'algorithme. Les ensembles Desync50 et Desync100 contiennent des traces désynchronisées. La désynchronisation est appliquée sous la forme d'un décalage global des points comme pour la transformation Shifting de Cagli *et al.* [CDP17]. La fenêtre de sélection est décalée d'un nombre de points choisi de façon aléatoire et uniforme pour chacune des traces entre 0 et 50 pour Desync50 et entre 0 et 100 pour Desync100. Ce phénomène est illustré dans la Figure 4.4 où l'on voit la différence au niveau des pics de consommation de courant entre la courbe bleue synchronisée et la courbe orange ayant subi un décalage de 100 points. La présence de désynchronisation dans les traces de consommation de courant augmente la difficulté de la prédiction de la valeur intermédiaire manipulée puisque les fuites d'information ne sont plus présentes tout le temps à la même position dans la trace. Toutefois, Benadjila *et al.* [BPS+19] ont montré que les réseaux de neurones sont capables d'apprendre à prédire les valeurs intermédiaires malgré la présence de désynchronisation. Il est donc intéressant d'étudier l'impact de la batch normalization et de la régularisation sur ce type de protection.

4.4.2 Exécution d'attaques sur des traces à clé variable

Le but des attaques par canaux auxiliaires est de retrouver la clé utilisée par un composant donné, cette clé est donc normalement fixe. Toutefois, pour les attaques profilées, afin de caractériser au mieux la consommation de courant du composant ouvert, les clés utilisées pour les chiffrements sont choisies aléatoirement tout comme la valeur du message d'entrée. Cela est fait pour éviter un biais potentiel qui pourrait apparaître dans les templates réalisés. Les traces d'entraînement des réseaux de neurones suivent donc la même logique et utilisent une clé variable.

L'utilisation d'une clé aléatoire rend donc les attaques classiques impossibles puisqu'elles n'ont plus une cible fixe à retrouver. Cela crée un problème pour le calcul de

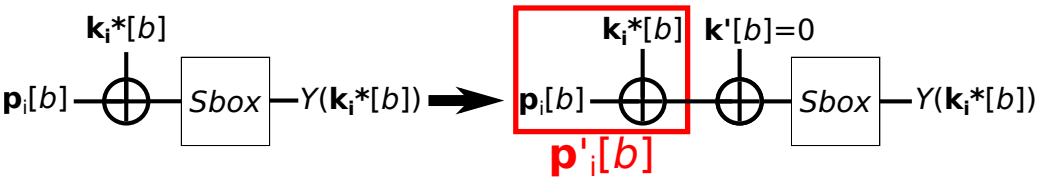


FIGURE 4.5: Illustration du ré-ordonnancement de la clé. La partie gauche de la figure montre un fonctionnement normal et la partie droite la méthode alternative qui ré-ordonne les clés.

la métrique $\Delta^d_{train,val}$ qui se base sur les nombres de traces nécessaires pour réussir les attaques sur les exemples d'entraînement et de validation. Cependant, ce problème peut être résolu en appliquant un ré-ordonnancement des valeurs des clés. Pour illustrer ce ré-ordonnancement, nous allons utiliser la Figure 4.5 qui montre un calcul classique d'AES et un calcul avec un réarrangement des valeurs des clés. Pour rappel, lors d'un AES, l'octet du message $\mathbf{p}_i[b]$ passe par une opération de ou-exclusif avec l'octet de la clé $\mathbf{k}^*_i[b]$ lors de l'opération AddRoundKey. Le résultat subit ensuite une substitution à l'aide de l'opération SubBytes qui utilise la Sbox et cela donne la valeur intermédiaire ciblée par le réseau $Y(\mathbf{k}^*_i[b])$. Ce calcul classique est illustré par la partie gauche de la Figure 4.5. Toutefois, durant la phase d'entraînement, la clé utilisée pour chaque trace est connue. Il est donc possible de calculer le ou-exclusif entre l'octet du message d'entrée et l'octet de clé pour ensuite considérer ce résultat comme la nouvelle valeur du message d'entrée $\mathbf{p}'_i[b]$. On a donc :

$$\mathbf{p}'_i[b] = \mathbf{p}_i[b] \bigoplus \mathbf{k}^*_i[b].$$

On peut ensuite simuler le ou-exclusif entre ce nouveau message $\mathbf{p}'_i[b]$ et une nouvelle valeur de clé qui serait fixée à zéro $\mathbf{k}'_i[b] = 0$. En répétant cette méthode pour chacune des traces, on obtient une nouvelle clé fixe simulée qui vaut 0.

De plus, étant donné que les valeurs d'entrée ne sont pas modifiées par ce procédé, on retrouve bien la même valeur $Y(\mathbf{k}^*_i[b])$ pour la variable intermédiaire. Les résultats des attaques sur les traces de l'ensemble d'entraînement et de l'ensemble d'attaque devraient être similaires étant donné que les fuites ciblés restent les mêmes. Seule la façon de calculer le rang de la bonne clé est modifiée. Pour éviter ce problème à l'avenir, il serait intéressant lors de la création de l'ensemble d'entraînement de garder un sous-ensemble de traces qui aurait une clé fixe et qui servirait d'ensemble de validation. Cela permettrait une comparaison plus directe entre performance sur l'ensemble de validation et performance sur celui d'attaque.

Nous avons utilisé cette méthode lors des entraînements des réseaux sur les ensembles à clés variables afin de calculer les valeurs N^1_{train} et N^1_{val}. Les résultats de ces tests sont présentés dans la section suivante.

4.4.3 Étude de l'évolution de $\Delta^1_{train,val}$ pour CNN_{best} en fonction des ensembles considérés et application de la batch normalization

Pour commencer cette étude, nous allons voir comment évolue la métrique $\Delta^1_{train,val}$ appliquée au réseau CNN_{best} pour les différents ensembles de données que nous venons d'introduire. En plus de la présence de désynchronisation, le nombre de traces d'entraînement utilisées va également varié afin d'étudier son impact sur l'overfitting du réseau.

CNN_{best}

La Figure 4.6 montre l'évolution de $\Delta^1_{train,val}$ pour l'entraînement de CNN_{best} sur Desync0 avec au choix 50 000, 100 000 ou 190 000 traces d'entraînement. Dans chacun des cas, 10 000 traces de l'ensemble d'entraînement sont mises de côté pour servir d'ensemble de validation. On constate que plus il y a de traces dans l'ensemble d'entraînement, plus la convergence de $\Delta^1_{train,val}$ est rapide et également que le minimum atteint pour la métrique est plus bas. En effet, le minimum pour l'entraînement à 50 000 traces se situe à l'epoch 67 à une valeur de $\Delta^1_{train,val}$ de 2329 traces. Pour 100 000 traces, le minimum est de 910 traces à l'epoch 49 et pour finir, $\Delta^1_{train,val}$ atteint une valeur minimale de 398 traces à l'epoch 44 pour l'entraînement à 190 000 traces. On peut voir sur la Figure 4.6 que l'ajout de traces dans l'ensemble d'entraînement a un effet régularisateur sur le réseau étant donné que l'évolution de $\Delta^1_{train,val}$ montre moins d'overfitting lors de l'utilisation de 100 000 et 190 000 traces d'entraînement. Le réseau est alors capable d'atteindre de bonnes performances sur les traces synchronisées avec 190 000 traces d'entraînement en parvenant à retrouver la clé à partir de 500 traces.

Par contre, dès que le problème se complexifie avec de la désynchronisation, les performances décroissent rapidement. La Figure 4.7 montre les évolutions au cours de l'entraînement du rang moyen de la bonne clé après 5000 traces de validation pour tous les réseaux sur les ensembles Desync50 et Desync100. Pour ces entraînements, la métrique $\Delta^1_{train,val}$ n'a pas pu être calculée car les attaques ne réussissent pas à atteindre un taux de succès de 90% en moins de 5000 traces de validation. D'un autre côté sur les traces d'entraînement, ce même taux de succès est atteint après seulement quelques epochs. L'ajout de difficulté liée à la désynchronisation mène donc à plus d'overfitting. Cela est due au fait que la capacité du réseau est suffisante pour apprendre par coeur les exemples d'entraînement et donc que la désynchronisation n'a pas d'impact important sur les performances sur l'ensemble d'entraînement. Malgré cet overfitting, on peut voir que le réseau est capable d'atteindre un rang correct avec les exemples de validation, surtout lors de l'utilisation de 190 000 traces. Le réseau parvient à atteindre un rang moyen de 1 aux epochs 35 et 37 pour respectivement Desync50 et Desync100 en s'entraînant avec 190 000 traces. Il apparaît également que les rangs pour les ensembles à 50 000 et 100 000 traces sont en décroissance à la fin des 200 epochs. Ces résultats montrent que l'ajout de traces d'entraînement permet de contrer légèrement le phénomène d'overfitting.

Les prochains tests se concentrent sur l'utilisation de la batch normalization afin d'améliorer les performances du réseau sur les ensembles de traces désynchronisées étant donné que l'évolution du rang montre que le réseau s'améliore encore à la fin des 200 epochs. Le

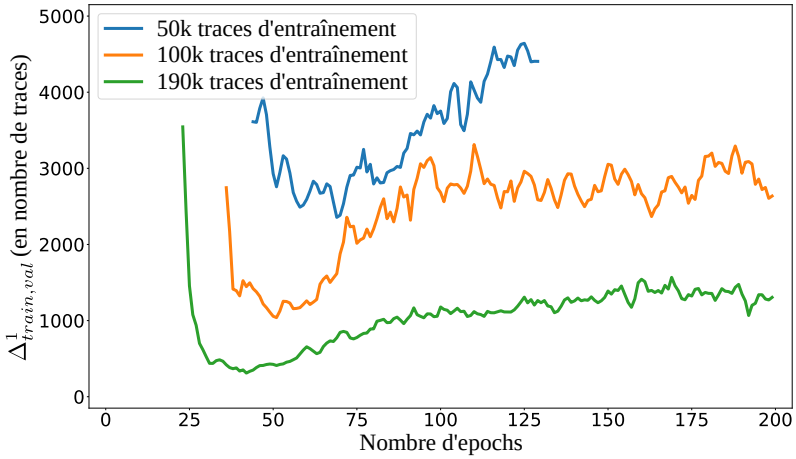
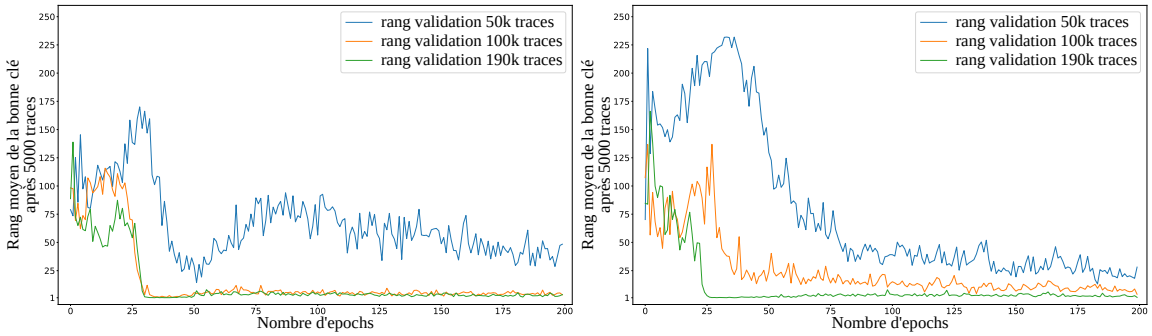


FIGURE 4.6: Évolution de $\Delta^1_{train,val}$ pour CNN_{best} durant l'entraînement pour différents nombres d'exemples d'entraînement sur Desync0.



(A) Desync50 (B) Desync100

FIGURE 4.7: Évolution du rang de la bonne clé après 5000 traces d'attaque durant l'entraînement du réseau CNN_{best} pour différents nombres d'exemples d'entraînement et différents niveaux de désynchronisation.

nouveau réseau formé avec l'utilisation de batch normalization se nomme CNN_{bn} et son architecture est décrite Annexe A Tableau A.3.

CNN_{bn}

La Figure 4.8 représente l'évolution de $\Delta^1_{train,val}$ pour le réseau CNN_{bn} sur les ensembles de données Desync0, Desync50 et Desync100 en utilisant à chaque fois un nombre différent de traces d'entraînement. Ce nouveau réseau est similaire à CNN_{best} mais contient en plus une couche de batch normalization entre la sortie de chaque fonction d'activation et la couche de pooling de chaque bloc convolutionnel. Comparé à CNN_{best}, on peut noter une amélioration significative des performances du réseau sur l'ensemble de validation. En effet, sur la Figure 4.8a, $\Delta^1_{train,val}$ converge plus rapidement et atteint des valeurs plus basses. La métrique est également plus stable pour le reste de l'entraînement. Avec 14 epochs d'entraînement, le réseau obtient des performances similaires sur l'ensemble d'entraînement et celui de validation comme indiquée par la valeur de $\Delta^1_{train,val}$ à 116 traces. La Figure 4.8b montre l'évolution de $\Delta^1_{train,val}$ pour l'ensemble Desync50. Cette fois-ci, dans ce contexte plus difficile, la métrique n'est calculable que pour les entraînements à 100

000 et 190 000 traces et seulement pour quelques epochs pour celui à 100 000 traces. Si la courbe d'évolution de $\Delta^1_{train,val}$ n'apparaît pas ou disparaît au cours de l'entraînement, cela signifie que le réseau n'était pas ou plus capable d'atteindre un taux de succès de 90% sur les attaques en validation. On constate que malgré la difficulté, l'entraînement à 190 000 converge et $\Delta^1_{train,val}$ arrive à une valeur minimale de 899 traces après 19 epochs. Toutefois, contrairement à l'entraînement sur Desync0, la forme de $\Delta^1_{train,val}$ indique que l'effet de l'overfitting est plus important étant donné que la valeur de N^1_{val} augmente rapidement une fois le minimum passé. Pour finir, les résultats de l'entraînement sur Desync100 sont illustrés Figure 4.8c. Dans ce cas, seul l'utilisation de l'entièreté des traces d'entraînement permet de calculer $\Delta^1_{train,val}$. Le minimum de la courbe est atteint après 19 epochs également à une valeur de 1793 traces. L'effet de l'overfitting est encore plus prononcé pour cet entraînement et la métrique $\Delta^1_{train,val}$ n'est plus calculable après l'epoch 125.

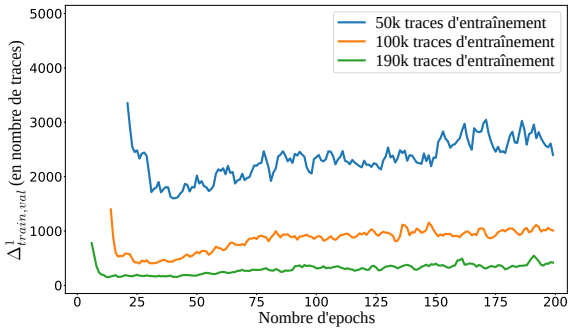
Malgré le contexte plus difficile des ensembles Desync50 et Desync100, la Figure 4.9, qui illustre l'évolution du rang après 5000 traces pour l'ensemble de validation, montre que les réseaux sont capables d'atteindre un rang proche de 1 quelque soit le nombre de traces d'entraînement utilisées. Cela signifie qu'ils sont tous capables de généraliser leurs connaissances même si ils n'arrivent pas nécessairement à faire les attaques en moins de 5000 traces. Au final, l'ajout de batch normalization améliore grandement la vitesse d'apprentissage des réseaux comme cela est montré par la métrique $\Delta^1_{train,val}$ qui converge plus rapidement. Toutefois, cette accélération implique également que l'overfitting se produit plus rapidement. L'addition de traces dans l'ensemble d'entraînement entraîne un léger effet de régularisation, comme pour CNN_{best}. Cet effet ralentit la convergence de N^1_{train}, ce qui aide à accélérer celle de $\Delta^1_{train,val}$, et il permet au réseau d'obtenir de meilleures performances sur les traces de validation en un nombre réduit d'epoch. De plus, la batch normalization améliore les performances du réseau dans son ensemble quelques soient les niveaux de désynchronisation même si il reste un écart large entre les performances à l'entraînement et en validation pour les ensembles Desync50 et Desync100. Le but de l'application de la régularisation est donc de réduire cet écart au maximum tout en maintenant de bonnes performances d'attaque.

4.4.4 Application des techniques de régularisation au réseau CNN_{bn}

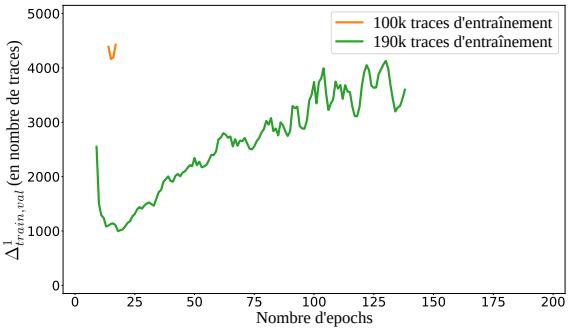
L'application de régularisation aux réseaux de neurones est un processus qui reste largement empirique. Les quantités de régularisation utilisées sont donc vues comme des hyperparamètres qu'il faut ajuster jusqu'à obtenir les meilleurs résultats. Il existe malgré tout des travaux qui apportent des conseils sur les quantités à appliquer selon les types de couches [PK16]. Nous verrons cela en même temps que nous discuterons de la méthodologie utilisée pour les tests des valeurs de régularisation.

Méthodologie d'application de la régularisation

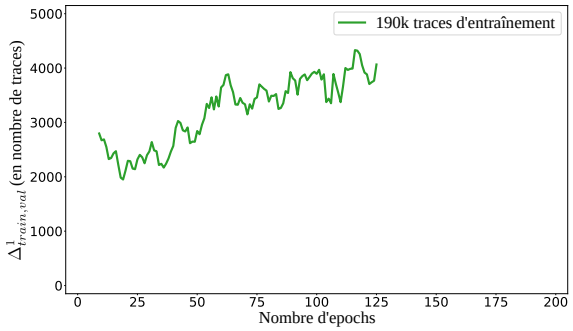
Le Tableau 4.2 récapitule les différentes quantités de régularisation testées sur le réseau CNN_{bn}. La valeur λ_D désigne le facteur de dropout et λ_{L_2} le facteur de weight decay. Les valeurs considérées pour le facteur λ_{L_2} ont été choisies entre 0 et 0.3 et testées par



(A) Desync0

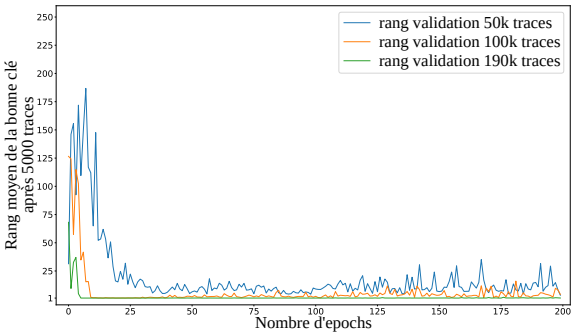


(B) Desync50

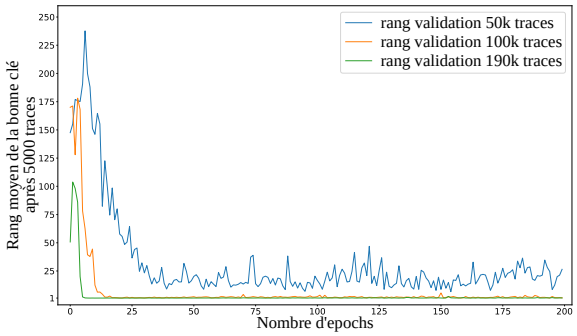


(C) Desync100

FIGURE 4.8: Évolution de $\Delta^1_{train,val}$ pour CNN_{bn} durant l'entraînement pour différents nombres d'exemples d'entraînement et différents niveaux de désynchronisation.



(A) Desync50



(B) Desync100

FIGURE 4.9: Évolution du rang de la bonne clé après 5000 traces d'attaque durant l'entraînement du réseau CNN_{bn} pour différents nombres d'exemples d'entraînement et différents niveaux de désynchronisation.

TABLE 4.2: Tableau récapitulatif des tests fait sur le dropout et la régularisation L2.

	Test ($pas = 0.1$)		Choix	
	λ_D	λ_{L_2}	λ_D	λ_{L_2}
$CONV1\&2$	$[0, ..., 0.3]$	$[0, ..., 0.3]$	0	0
$CONV3$	$[0, ..., 0.8]$	$[0, ..., 0.3]$	0.5	0.2
$CONV4$	$[0, ..., 0.8]$	$[0, ..., 0.3]$	0.6	0.3
$CONV5$	$[0, ..., 0.8]$	$[0, ..., 0.3]$	0.7	0.3
$FC1$	$[0, ..., 0.8]$	$[0, ..., 0.3]$	0	0
$FC2$	$[0, ..., 0.3]$	$[0, ..., 0.3]$	0	0

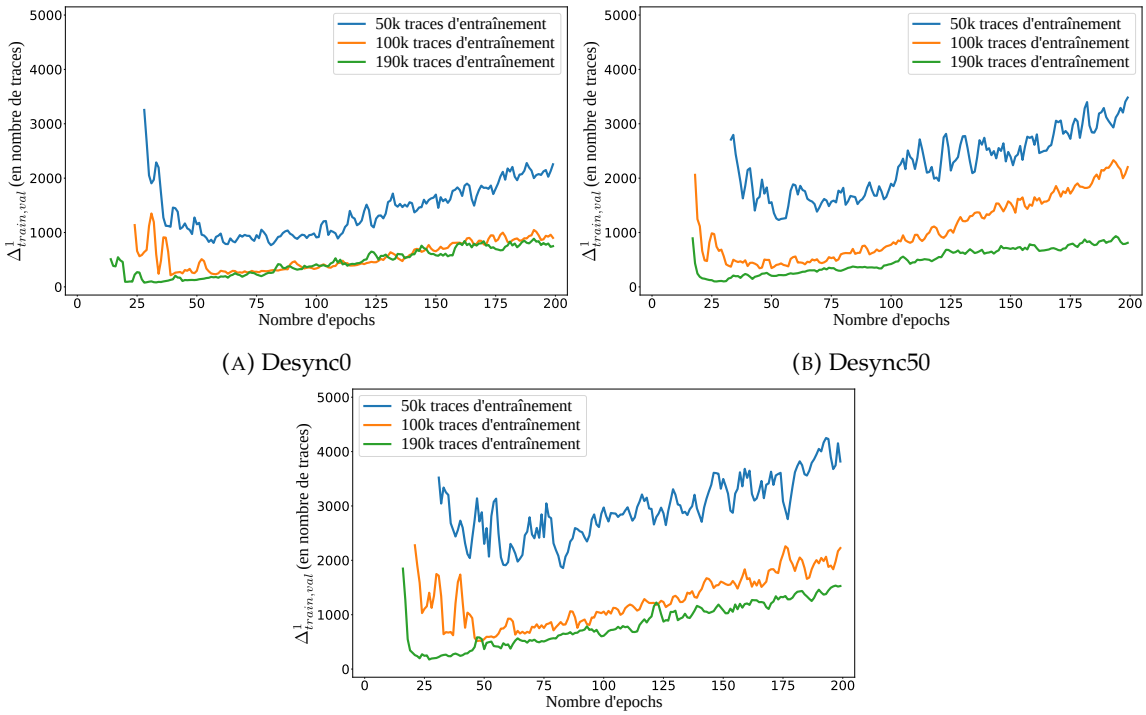
pas de 0.1 tandis que celles pour λ_D étaient comprises entre 0 et 0.8. Les premiers tests commençaient avec de faibles valeurs avant de les augmenter petit à petit. La limite à 0.3 pour λ_{L_2} a été choisie car elle impose déjà une forte pénalité aux poids des couches du réseau. Les valeurs donnant les meilleures performances d'attaque ont été gardées. Pour les deux premières couches convolutionnelles du réseau, aucune application de régularisation n'a mené à une amélioration des performances, les valeurs des facteurs sont donc fixées à 0. La même chose s'est produite pour les couches entièrement connectées qui elles non plus ne sont pas régularisées dans le réseau final. Pour les trois dernières couches du réseau, la régularisation s'est montrée très efficace. Ainsi, des valeurs de dropout de 0.5, 0.6 et 0.7 sont appliquées respectivement aux couches CONV3, CONV4 et CONV5. Cela correspond donc à 50%, 60% et 70% des neurones de ces couches qui sont désactivés aléatoirement à chaque nouveau batch d'entraînement. Ces quantités suivent bien les recommandations décrites par Park *et al.* [PK16] qui indiquent qu'une application de dropout de plus en plus forte pour les couches convolutionnelles plus profondes apporte les meilleurs résultats. En addition au dropout, des facteurs λ_{L_2} de 0.2, 0.3 et 0.3 sont appliqués à ces trois couches.

Le nouveau réseau ainsi formé par l'application de régularisation à CNN_{bn} se nomment $\text{CNN}_{\text{bnreg}}$. Nous allons maintenant décrire les résultats de $\text{CNN}_{\text{bnreg}}$ sur l'ensemble ASCAD clé variable.

$\text{CNN}_{\text{bnreg}}$

La Figure 4.10 représente l'évolution de la métrique $\Delta^1_{train,val}$ au cours de l'entraînement du réseau $\text{CNN}_{\text{bnreg}}$ pour des ensembles d'entraînement de tailles différentes et ayant différents niveaux de désynchronisation. Ce réseau contient des couches de batch normalization comme CNN_{bn} et également de la régularisation comme décrite dans la section précédente. Un résumé de l'architecture peut être trouvé dans le Tableau A.4 de l'Annexe A.

Sur la Figure 4.10a, on peut voir que les performances de $\text{CNN}_{\text{bnreg}}$ sur Desync0 sont à peu près les mêmes que celles de CNN_{bn}. La principale différence vient du fait que $\Delta^1_{train,val}$ met plus de temps à converger pour $\text{CNN}_{\text{bnreg}}$ en atteignant son minimum de 40 traces à l'epoch 29 quand 190 000 traces d'entraînement sont utilisées. Cependant, l'apport de la régularisation apparaît clairement sur la Figure 4.10b qui représente l'entraînement sur Desync50. Dans ce contexte désynchronisé, le réseau est capable d'atteindre un taux



(A) Desync0

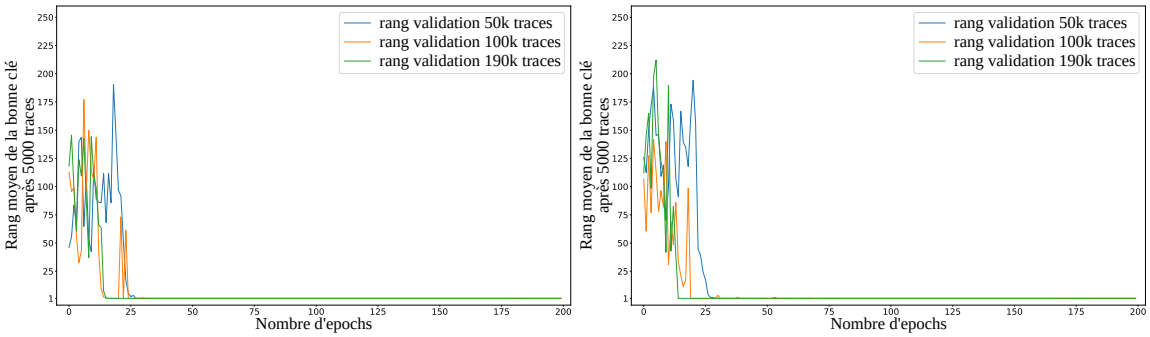
(B) Desync50

(C) Desync100

FIGURE 4.10: Évolution de $\Delta^1_{train,val}$ pour $\text{CNN}_{\text{bnreg}}$ durant l'entraînement pour différents nombres d'exemples d'entraînement et différents niveaux de désynchronisation.

de succès de 90% sur les traces d'entraînement et de validation quelque soit le nombre de traces utilisées lors de l'entraînement. On constate que même si les performances du réseau sont réduites par rapport à Desync0, la métrique $\Delta^1_{train,val}$ atteint quand même un minimum de 52 traces à l'epoch 26 grâce aux 190 000 traces d'entraînement. Finalement, la Figure 4.10c montre l'entraînement de $\text{CNN}_{\text{bnreg}}$ sur l'ensemble Desync100. Cette fois-ci le minimum de la métrique se trouve à l'epoch 25 à une valeur de 111 traces lors de l'utilisation de toutes les traces d'entraînement. Encore une fois, l'ajout de désynchronisation augmente les effets de l'overfitting mais cet overfitting reste bien moins prononcé que pour CNN_{bn} et il est surtout présent lors de l'emploi d'un nombre plus faible de traces d'entraînement. Toutefois, on constate que lorsque le réseau continue son apprentissage au-delà de son pic de généralisation, ses performances sur les traces de validation baissent rapidement comme indiqué par l'augmentation de $\Delta^1_{train,val}$. Il est donc très important de savoir quand arrêter l'entraînement pour obtenir le meilleur réseau possible.

Un autre effet de la régularisation, en plus d'améliorer les performances du réseau sur les ensembles Desync50 et Desync100, est que ces performances sont similaires quelques soient les niveaux de désynchronisation. Ainsi, la régularisation permet au réseau d'extraire plus d'informations des exemples d'entraînement afin de mieux généraliser. On peut constater que ce phénomène est plus difficile à voir sur la Figure 4.11 qui montre l'évolution du rang après 5000 traces de validation pour $\text{CNN}_{\text{bnreg}}$ sur Desync50 et Desync100 ce qui valide le besoin d'utilisation d'une métrique comme $\Delta^1_{train,val}$.



(A) Desync50 (B) Desync100

FIGURE 4.11: Évolution du rang de la bonne clé après 5000 traces d'attaque durant l'entraînement du réseau $\text{CNN}_{\text{bnreg}}$ pour différents nombres d'exemples d'entraînement et différents niveaux de désynchronisation.

Synthèse des résultats sur les trois réseaux

Un résumé des résultats obtenus à partir des différents réseaux s'entraînant sur les 190 000 traces de l'ensemble d'entraînement se trouve dans le Tableau 4.3. Il inclut également le nombre de traces minimal N_a^* nécessaire pour atteindre un taux de succès de 90% sur l'ensemble d'attaque donc l'ensemble à clé fixe. Comme mentionné dans la Section 4.4.2, le nombre N_a^* est proche de la valeur de N_{val}^1 pour la plupart des réseaux. Cela montre que N_{val}^1 est bien un bon indicateur des futures performances du réseau sur les traces d'attaque.

En utilisant la métrique $\Delta_{train,val}^1$ pour appliquer la technique d'early stopping, les performances du réseau CNN_{best} ont pu être augmentées de 42% sur l'ensemble Desync0 et le nombre d'epoch utilisé réduit de 41.3%. Toutefois, la technique d'early stopping n'apporte pas une aide suffisante pour que le réseau atteigne un taux de succès de 90% en moins de 5000 traces sur les ensembles Desync50 et Desync100 qui sont trop complexes.

Par contre, l'application de couches de batch normalization à travers le réseau CNN_{bn} a permis d'augmenter de manière significative les performances du réseau et réduit de 46.5% le nombre de traces nécessaires pour atteindre un taux de succès de 90%. De plus, l'amélioration de la vitesse d'entraînement a fait que le réseau atteint son pic de généralisation en nécessitant 68.2% d'epoch en moins. Cela a aussi conduit à la possibilité de réussir des attaques en moins de 5000 traces sur les ensembles désynchronisés Desync50 et Desync100.

Pour finir, l'utilisation de régularisation sur le réseau $\text{CNN}_{\text{bnreg}}$ a de nouveau permis une amélioration des performances sur ces ensembles de 70.6% et 81.7% pour respectivement Desync50 et Desync100 mais au prix d'une légère augmentation du nombre d'epochs d'entraînement.

Pour conclure, ces expériences confirment l'importance de l'utilisation de techniques d'amélioration de performances comme la batch normalization et de l'addition de régularisation ajustée au problème, comme le dropout et le weight decay. Sans cela, il est possible de surestimer la difficulté de certains ensembles de données. La batch normalisation permet au réseau d'apprendre plus vite, au risque de rentrer dans la phase d'overfitting plus rapidement, mais ce n'est pas toujours suffisant pour les problèmes plus complexes.

TABLE 4.3: Résumé des résultats de CNN_{best}, CNN_{bn} et CNN_{bnreg} en fonction des valeurs de $\Delta^1_{train,val}$, N^1_{val}, N^*_a et le nombre d'epochs d'entraînement pour l'entraînement utilisant 190 000 traces d'entraînement.

Réseaux	Référence	Nombre d'epochs	Desync	$\Delta^1_{train,val}$	N^1_{val}	N^*_a	Amélioration en termes de traces et d'epoch par rapport à CNN_{best}	Amélioration en termes de traces et d'epoch par rapport à [BPS+19]	Amélioration en termes de traces et d'epoch par rapport à CNN_{bn}
CNN_{best}	[BPS+19]	75	0	972	935	1275	-	-	-
			50	-	-	-	-	-	-
			100	-	-	-	-	-	-
	[RBHG21]*	44	0	398	542	589	-	N^1_{val} : **-42**% N^*_a : **-53**% Epoch : **-41**%	-
			50	-	-	-	-	-	-
			100	-	-	-	-	-	-
CNN_{bn}	[RBHG21]*	14	0	116	290	228	N^1_{val} : **-46**% N^*_a : **-61**% Epoch : **-68**%	N^1_{val} : -69% N^*_a : **-82**% Epoch : -81%	-
		19	50	899	927	964	-	N^1_{val} : - Epoch : -74%	-
		19	100	1793	1805	3333	-	N^1_{val} : - Epoch : -74%	-
CNN_{bnreg}	[RBHG21]*	29	0	40	244	150	N^1_{val} : **-55%** N^*_a : **-74**% Epoch : **-34%**	N^1_{val} : -74% N^*_a : **-88**% Epoch : -61%	N^1_{val} : -16% N^*_a : **-34**% Epoch : +107%
		26	50	52	273	301	-	N^1_{val} : - Epoch : -65%	N^1_{val} : **-70**% N^*_a : **-68**% Epoch : +36%
		25	100	111	330	347	-	N^1_{val} : - Epoch : -66%	N^1_{val} : **-81**% N^*_a : **-89**% Epoch : +31%

** : publication issue de ces travaux de thèse.*

Si l'on constate que les performances sur les données d'entraînement sont bonnes mais que celles de validation sont en deçà des attentes, il est alors fortement conseillé d'utiliser de la régularisation. En effet, on a pu voir que CNN_{bnreg} montre plus de résistance face à la désynchronisation et est également moins sensible au phénomène d'overfitting. Finalement, le contexte de traces désynchronisées met en avant le besoin d'évaluation des réseaux de neurones pendant leur entraînement étant donné qu'ils sont beaucoup plus enclins à subir l'overfitting. Si cet overfitting n'est pas contrôlé, il peut mener à de sérieuses pertes de performances lors des attaques.

4.4.5 Comparaison avec les autres techniques utilisées dans l'état de l'art

Le Tableau 4.4 compare les performances de différents réseaux sur la base de données ASCAD clé fixe et la base de données ASCAD clé variable. Ces performances sont évaluées par le nombre minimum de traces nécessaires pour réussir les attaques sur les ensembles d'attaque de chacune des bases de données. On peut également noter que la base de données ASCAD clé variable est considérée comme étant plus dure à attaquer d'après Wu *et al.* [WPP20]. Avant nos travaux, seulement deux autres travaux [PBP, WPP20] l'ont utilisée pour leurs tests et notre réseau CNN_{bnreg} obtient les meilleurs résultats.

Afin d'être complet, nous pouvons aussi discuter des résultats des différents travaux sur la base de données clé fixe. Même si les résultats obtenus sur les deux ensembles de

TABLE 4.4: Résumé des résultats d'autres travaux sur les bases de données ASCAD en fonction du nombre de traces nécessaires pour réussir les attaques.

Référence \ Base de données	ASCAD clé variable			ASCAD clé fixe		
	D0	D50	D100	D0	D50	D100
Wu *et al.* [WPP20]	1000	-	-	80	-	-
Perin *et al.* [PBP]	$\sim$ 180	-	-	-	-	-
Won *et al.* [WJB20]	-	-	-	-	-	190
Wouters *et al.* [WAGP20]	-	-	-	-	$\sim$ 200	$\sim$ 300
Zaid *et al.* [ZBHV20a]	-	-	-	191	244	270
Robissout *et al.* [RZC+20]	-	-	-	802	-	-
Kim *et al.* [KPH+19]	-	-	-	>500	-	-
Benadjila *et al.* [BPS+19]	1275	>5000	>5000	1151	>5000	>5000
Robissout *et al.* [RBHG21]*	150	301	347	-	-	-

** : publication issue de ces travaux de thèse.*

traces ne peuvent pas être comparés directement, il est quand même possible de faire quelques extrapolations. Premièrement, la majorité des travaux réalisés sur l'ensemble de données clé fixe [WPP20, ZBHV20a, WJB20] obtient de bien meilleurs résultats que ceux présents dans l'article original par Benadjila *et al.* [BPS+19]. Cela est normal étant donné que ce dernier article ne se concentrait pas exclusivement sur l'amélioration d'un réseau et il se contentait de tester quelques configurations d'hyperparamètres pour trouver la meilleure. On peut toutefois remarquer que l'ordre de magnitude des améliorations obtenues est le même que pour les améliorations apportées par CNN_{bnreg} sur CNN_{best} sur la base de données clé variable. Il est possible d'en conclure que notre approche permet d'obtenir des performances au moins similaire aux autres techniques étudiées tout en ne modifiant que très peu le réseau de neurones utilisé.

D'autres travaux [KPH+19, CDP17] mentionnés précédemment ont effectué des tests sur l'augmentation des données. Cagli *et al.* [CDP17] ont obtenu un réseau plus résistant aux transformations de Shifting et de Add-Remove à l'aide de la régularisation apportée par l'augmentation des données. Nos expériences montrent un gain en robustesse face à la désynchronisation grâce à l'application de régularisation explicite dépendant des valeurs des poids au niveau de la fonction de perte. Les deux méthodes utilisées appliquent donc la régularisation à différents niveau de l'entraînement. Les deux techniques d'augmentation des données et de régularisation sont complémentaires et leur combinaison pourrait donc mener à des résultats encore meilleurs.

L'autre technique similaire à de l'augmentation des données étudiée par Kim *et al.* [KPH+19] agit de manière encore différente. Leurs résultats montrent une amélioration des performances par rapport à CNN_{best} même si elle n'est pas au niveau des autres. Il serait intéressant de voir si une combinaison de cette technique avec les autres permettrait d'améliorer les résultats obtenus.

4.5 Conclusion

Ce chapitre se concentre sur l'étude et l'amélioration d'un réseau de neurones, CNN_{best}, au travers de l'utilisation de plusieurs techniques d'amélioration de performances issues du machine learning. Les entraînements sont également évalués à l'aide de la métrique $\Delta^1_{train,val}$ qui permet d'appliquer la technique d'early stopping. Les techniques choisies ont déjà fait leurs preuves, à la fois de manière théorique et empirique, sur les problèmes de machine learning. Nous nous sommes donc concentrés sur la batch normalization pour accélérer l'entraînement ainsi que sur le dropout et le weight decay pour régulariser le réseau et réduire l'overfitting. Nous avons également discuté des autres techniques utilisées ou explorées dans les travaux sur l'application de réseaux de neurones aux attaques par canaux auxiliaires. Notre approche a consisté à fixer une architecture de réseau et à étudier l'influence de chacune des techniques sur ce réseau.

Les résultats obtenus dans le Chapitre 3 nous ont poussé à nous demander si l'entraînement du réseau CNN_{best} était réellement complet étant donné que les résultats à sa meilleure epoch d'entraînement montraient qu'il restait un certain niveau d'overfitting. Après avoir entraîné ce même réseau sur un ensemble de données plus complet ASCAD clé variable, et des ensembles dérivés plus complexes comprenant de la désynchronisation, il est apparu que la complexité du réseau semblait ralentir son apprentissage. En effet, ses performances pour les ensembles comprenant moins de traces d'entraînement continuent de progresser même à la fin des 200 epochs fixées. Pour résoudre ce problème sans augmenter le nombre d'epoch d'entraînement, nous avons eu recours à la technique de normalisation : batch normalization. Cette technique consiste à normaliser les distributions de données en sortie de chaque couche du réseau afin de réduire le décalage entre les distributions avant et après mise à jour des poids. Un effet secondaire de cette normalisation est un lissage du paysage de la fonction de perte. Ce lissage réduit les changements brusques de direction du gradient et permet une minimisation plus rapide de la fonction de perte. Grâce à cette technique, nous avons formé CNN_{bn} qui consiste en l'architecture de CNN_{best} avec de la batch normalization à la sortie de chacune de ses couches. Ce nouveau réseau a montré une capacité d'apprentissage bien plus grande et plus rapide que CNN_{best}. En effet, il est capable d'atteindre son maximum de généralisation en 68% d'epochs de moins et arrive à réussir les attaques avec 61% de traces en moins dans le contexte de traces synchronisées. Pour les contextes plus complexes de traces désynchronisées, ce nouveau réseau est capable de réussir les attaques en s'entraînant avec l'entièreté de l'ensemble de traces disponibles mais les effets de l'overfitting interviennent plus rapidement. On constate donc que l'overfitting est toujours présent dans CNN_{bn} même à la meilleure epoch et qu'il faut un moyen de le réguler. C'est pour cela que nous avons choisi d'utiliser des méthodes de régularisation.

Les techniques choisies pour la régularisation du réseau sont le dropout et le weight decay, aussi connu sous le nom de régularisation par norme L2. Le dropout est une technique consistant à désactiver un pourcentage de neurones d'une couche du réseau choisis aléatoirement à chaque batch d'entraînement. Les neurones désactivés ne sont donc plus pris en compte dans les calculs du réseau et dans la remontée du gradient. Le but de

cette désactivation est de réduire à la fois le nombre de neurones *morts* et la co-adaptation des neurones entre eux. L'application de dropout a donc un effet de régularisation globale du réseau qui permet de réduire l'overfitting. Au contraire, le weight decay est une technique de régularisation qui s'applique directement au niveau de la fonction de perte et prend la forme d'un terme qui va pénaliser les poids en fonction de leurs valeurs. La pénalisation du weight decay est calculée à partir de la norme L2 des poids du réseau et a tendance à favoriser une distribution des poids vers des valeurs proches de 0 tout en évitant que certains poids ne prennent trop d'importance. Le réseau obtenu en combinant ces techniques et la batch normalization se nomme CNN_{bnreg}. Les performances de ce réseau sur les traces synchronisées restent similaires à celles de CNN_{bn} mais celles sur les ensembles désynchronisées sont bien meilleures. Le réseau est ainsi capable de réussir ses attaques quelque soit le niveau de désynchronisation et quelque soit le nombre de traces utilisées dans l'ensemble d'entraînement. Il est également beaucoup moins victime d'overfitting comme le montre l'évolution de la métrique $\Delta^1_{train,val}$ dans chacun des cas. En comparant nos résultats à ceux d'autres travaux du domaine, il est clair que les techniques que nous avons exploré permettent de passer d'un réseau ayant des performances relativement faibles à un réseau étant au niveau de l'état de l'art.

En conclusion, les techniques d'amélioration des performances sont des outils très importants à considérer lors de l'étude des réseaux de neurones appliqués aux attaques par canaux auxiliaires. Ils permettent d'obtenir des gains significatifs de performances et ne pas les prendre en compte pourrait mener à une sous-estimation de la puissance de ces attaques. De plus, la régularisation est une technique importante à utiliser lorsque l'on identifie le phénomène d'overfitting lors de l'entraînement de notre réseau. Dans le prochain chapitre, nous allons explorer une piste possible d'amélioration des attaques par canaux auxiliaires se basant sur l'ordonnancement des prédictions des réseaux afin de réduire le rang de la bonne clé.

Chapitre 5

Classement des traces à partir des prédictions

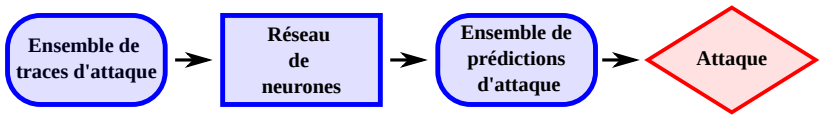
Ce chapitre se concentre sur un nouvel aspect des attaques par canaux auxiliaires profilées qui n'a pas encore été traité par la communauté : le classement des traces. Classiquement en analyse par canaux auxiliaires, les traces utilisées pour une attaque sont considérées comme équivalentes dans le sens où chacune des traces apportent de l'information aidant à retrouver la clé utilisée. Nous allons, dans ce chapitre, essayer d'adopter un nouveau point de vue sur ces attaques. Plutôt que de partir du principe que chaque trace apporte autant d'information à l'attaque sur la valeur intermédiaire, nous allons émettre l'hypothèse suivante : toutes les traces de l'ensemble d'attaques ne contiennent pas le même niveau d'information sur la variable intermédiaire et il est possible de déterminer les traces les mieux classifiées par le modèle utilisé pour l'attaque, *i.e.* le template ou le réseau de neurones. Cela pose un certain nombre de questions que nous allons détailler et auxquelles nous allons essayer de répondre au cours de ce chapitre.

5.1 Problématique soulevée

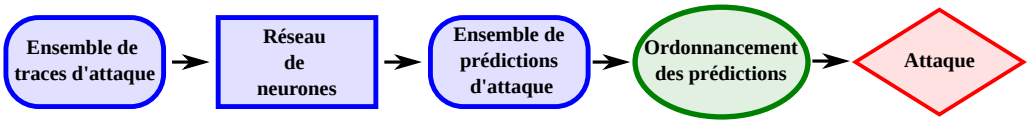
Tout d'abord, commençons par revenir sur la problématique soulevée en la reformulant légèrement : est-il possible que certains couples (trace, prédiction) aient un impact négatif sur l'attaque et la récupération de la clé ? Si oui, est-il possible de détecter ces traces et de les séparer des autres afin d'obtenir de meilleurs résultats ?

Ces questions se sont posées dans un premier temps vis-à-vis de l'utilisation du deep learning et de réseaux de neurones pour effectuer les prédictions sur la valeur de la variable intermédiaire ciblée dans les attaques profilées. En effet, les réseaux de neurones entraînés ne sont pas parfaits et sont biaisés par l'ensemble de traces utilisé pour leur entraînement. Cela veut dire qu'un tel réseau aura plus de facilités et de confiance lors de la prédiction de certaines traces qui sont proches de celles qu'il a vu lors de son entraînement. La question reste : est-il possible de déterminer quelles sont ces traces dans un ensemble de traces d'attaque pour lequel on ne connaît pas la clé ?

Un autre point sur lequel nous reviendrons concerne l'attaque template. Cette attaque est différente des attaques basées sur les réseaux de neurones au niveau du calcul des prédictions de chaque trace mais reste similaire sur le fait qu'il est nécessaire de créer un



(A) Méthode d'attaque classique



(B) Méthode d'attaque avec ordonnancement

FIGURE 5.1: Schéma illustrant la méthode d'attaque classique des attaques par canaux auxiliaires et les changements de la nouvelle méthode.

template à partir d'un ensemble de traces de profilage. Un biais peut donc être également retrouvé dans ce type d'attaque.

5.1.1 Modification du schéma d'attaque classique

Dans un premier temps, il nous semble intéressant de revenir sur la méthode classique d'exécution des attaques par canaux auxiliaires et sur les changements apportés par notre nouvelle méthode. La Figure 5.1 illustre la différence entre les deux façons de faire les attaques. La méthode classique correspond à celle vu au cours du Chapitre 2. A partir d'un ensemble de traces d'attaque et d'un réseau de neurones entraîné à prédire la valeur intermédiaire utilisée dans les calculs de l'algorithme cryptographique, un ensemble de prédictions est calculé. Cet ensemble est constitué des prédictions de chaque trace de l'ensemble d'attaque et sert à effectuer la récupération de la clé. Les prédictions utilisées lors de l'attaque sont extraites aléatoirement de cet ensemble.

C'est là, la principale différence entre les deux méthodes d'attaque. Dans la méthode avec ordonnancement, nous rajoutons une étape avant l'attaque qui consiste à faire un ordonnancement des prédictions qui seront utilisées pour récupérer la clé. L'attaque qui suit se base sur cet ordonnancement pour déterminer la prédiction suivante à utiliser lors de l'accumulation des probabilités.

Nous allons voir au cours de ce chapitre plusieurs façons d'effectuer l'ordonnancement des prédictions avant cela, il nous faut formaliser le problème.

5.1.2 Formalisation du problème

Pour formaliser ce problème, nous allons rappeler et définir quelques termes :

- $\mathcal{T}$: l'ensemble de traces d'attaque ;
- F_θ : un modèle de paramètres θ qui effectue des prédictions sur les traces $t \in \mathcal{T}$;
- $\mathcal{P}$: l'ensemble des prédictions de F_θ sur $\mathcal{T}$, *i.e.* $\mathcal{P} = F_\theta(\mathcal{T})$;
- $\mathrm{Adv}_{\mathcal{P}}(n)$: un adversaire utilisant n prédictions de $\mathcal{P}$ pour récupérer la clé k^* ;
- $\mathrm{Exp}_{\mathrm{Adv}_{\mathcal{P}}(n)} = r_n$: résultat de l'attaque de $\mathrm{Adv}_{\mathcal{P}}(n)$ exprimé par le rang $r_n \in [\![1, 256]\!]$ de la clé k^* dans le vecteur $\mathbf{g}$ contenant les hypothèses de clé classées selon les résultats de l'attaque.

À partir de ces définitions, si $\mathrm{Exp}_{\mathrm{Adv}_{\mathcal{P}}(n)} = 1$ alors la clé k^* est bien récupérée grâce aux n traces utilisées et si $\mathrm{Exp}_{\mathrm{Adv}_{\mathcal{P}}(n)} > 1$ alors la clé n'est pas directement récupérée. Dans le cas $\mathrm{Exp}_{\mathrm{Adv}_{\mathcal{P}}(n)} = r_n > 1$, il est possible de réduire la valeur de r_n en augmentant la valeur de n. Toutefois, cette valeur est parfois limitée par une valeur $n_{\max}$ qui représente le nombre maximal de traces d'attaque qu'il est possible d'acquérir. Cela pose problème notamment quand : $\mathrm{Exp}_{\mathrm{Adv}_{\mathcal{P}}(n_{\max})} = r_{n_{\max}} > 1$. C'est dans ce cas qu'interviennent nos travaux où nous posons le problème suivant.

Problème Trouver un ordonnancement o tel que $\mathcal{P}_o$ représente les prédictions des traces arrangées selon o de sorte qu'il existe $n_o \in \mathbb{N}^*$ avec $n_o < n_{\max}$ tel que :

$$\mathrm{Exp}_{\mathrm{Adv}_{\mathcal{P}_o}(n_o)} = r_{n_o} < r_{n_{\max}}.$$

Un exemple d'un tel phénomène est illustré Figure 5.2. On y voit deux évolutions de rang différentes basées sur le même ensemble de prédictions, une pour laquelle l'ordonnancement des prédictions utilisées est aléatoire et une pour laquelle un ordonnancement est déterminé au préalable. Le rang final pour ces deux attaques est donc le même étant donné que les mêmes prédictions sont utilisées mais les évolutions des rangs sont très différentes. Pour la courbe du rang avec un ordonnancement aléatoire, on constate que le rang moyen décroît de manière régulière avec l'utilisation de plus de prédictions alors que le rang avec un ordonnancement déterminé décroît très rapidement au début de l'attaque et atteint même une valeur de 1 avant d'augmenter à la fin pour atteindre le rang final $r_{n_{\max}}$. Dans ce cas de figure, l'utilisation de l'ensemble de prédictions ordonnées de façon aléatoire nous entraîne vers une fausse conclusion : *l'attaque n'aboutit pas avec n_{max} traces*. Toutefois, on voit ici que l'attaque est possible en n_o traces à condition que les $n_{\max}$ prédictions soient correctement ordonnées.

Remarque On peut remarquer qu'il existe une équivalence entre une trace et sa prédiction du point de vue de l'attaque. Étant donné que nous nous basons sur les prédictions pour établir les ordonnancements, nous parlerons dans la suite du chapitre d'ordonnancement des prédictions ou de prédictions ordonnées.

Notre but dans ce chapitre va être de déterminer et d'étudier diverses méthodes permettant d'établir l'ordonnancement o. En effet, comme nous pouvons le voir sur l'exemple, déterminer un tel ordonnancement ainsi que le bon nombre de prédictions à utiliser avec cet ordonnancement peut permettre de réduire le rang final de la clé k^* voire même dans certains cas de la retrouver. Au-delà de l'aspect amélioration des attaques, une telle possibilité peut aussi remettre en cause certaines évaluations de sécurité assurant la sûreté d'un système qui ne l'est pas nécessairement. Il faut toutefois relativiser le problème et mettre au clair ce qu'il n'est pas. Il n'est pas un moyen d'améliorer les performances d'attaques déjà réussies. En effet, si $\mathrm{Exp}_{\mathrm{Adv}_{\mathcal{P}}(n_{\max})} = 1$ et que l'on détermine un ordonnancement o et un nombre de traces n_o tel que $\mathrm{Exp}_{\mathrm{Adv}_{\mathcal{P}_o}(n_o)} = 1$ soit une attaque réussie, cela n'est pas équivalent à une amélioration de l'attaque. L'aspect important à considérer lors des attaques est le nombre de traces d'attaque à récupérer pour retrouver la clé. Si l'attaque

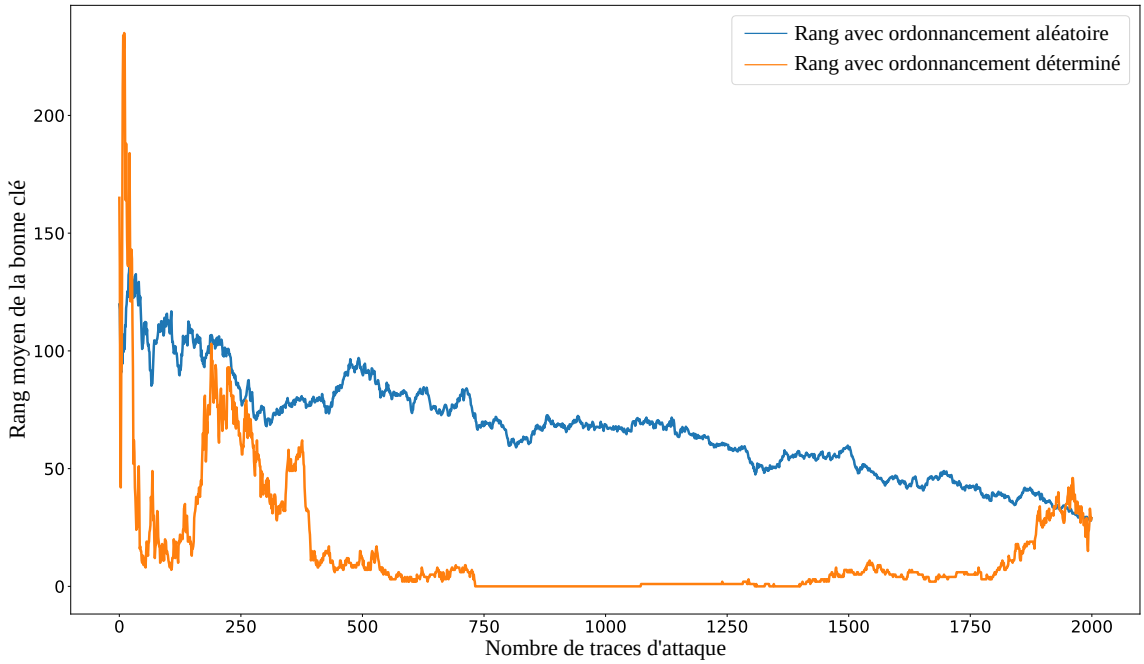


FIGURE 5.2: Évolution du rang moyen de la bonne clé du même ensemble de prédictions avec ou sans ordonnancement des prédictions.

réussit en utilisant $n_{\max}$ traces, alors une attaque qui réussit en n_o traces n'est pas une amélioration s'il faut dans tous les cas faire l'acquisition des $n_{\max}$ traces. Ce cas de figure est illustré dans la Figure 5.3 où l'on peut voir que la clé est récupérée de manière consistante en quelques traces pour l'attaque avec des prédictions ordonnées là où il en faut plusieurs centaines avec des prédictions choisies aléatoirement. Toutefois, étant donné que les deux attaques réussissent en moins de 500 traces, l'utilisation d'un ordonnancement n'améliore pas les performances de l'attaque en termes de nombre de traces d'attaque à acquérir.

Maintenant que ce point est clarifié, nous pouvons discuter de l'effet principal de cette nouvelle méthode d'attaque. En montrant la possibilité de faire converger des attaques via un ordonnancement des prédictions, nous pouvons redéfinir les cas de figure possibles pour une attaque dans le contexte où $\mathrm{Exp}_{\mathrm{Adv}_{\mathcal{P}}(n_{\max})} > 1$. Le meilleur cas devient le cas où l'ordonnancement permet de déterminer une valeur n_o tel que $\mathrm{Exp}_{\mathrm{Adv}_{\mathcal{P}_o}(n_o)} = 1$, c'est-à-dire que l'attaque ne parvenait pas à récupérer la clé précédemment mais y parvient maintenant. Le pire cas est dorénavant le cas où un tel ordonnancement n'est pas déterminable et donc l'attaque se déroule de manière classique. On constate donc que la façon classique de réaliser les attaques est le pire cas de figure possible ce qui signifie que la nouvelle méthode proposée ne peut qu'améliorer les résultats. Ceci a donc la capacité de remettre en question un certain nombre d'évaluations de sécurité dans lesquelles ce cas de figure n'était pas considéré.

Remarque Soient $\mathcal{P}$ un ensemble de $n_{\max}$ prédictions de traces d'attaque et o un ordonnancement quelconque des prédictions de $\mathcal{P}$. On a alors :

$$\mathrm{Exp}_{\mathrm{Adv}_{\mathcal{P}_o}(n_{\max})} = r_{n_{\max}},$$

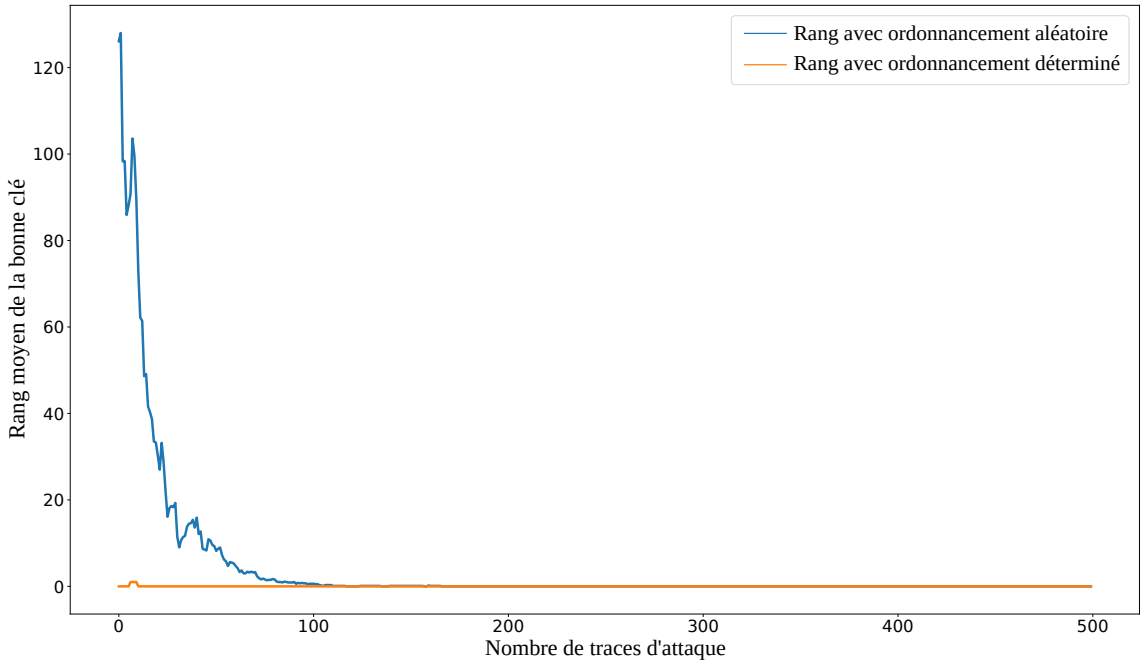


FIGURE 5.3: Évolution du rang moyen de la bonne clé pour deux ordonnancements différents lorsque l'attaque est réussie.

où $r_{n_{\max}}$ est le rang final atteint lorsque l'on utilise toutes les prédictions de $\mathcal{P}$ lors de l'attaque. Cette propriété vient du fait que l'accumulation des probabilités est déterministe donc le rang final est toujours le même pour un ensemble de prédictions fixe.

Nous nous concentrons donc dans la suite du chapitre sur l'analyse des attaques qui ne réussissent pas en utilisant toutes les traces de l'ensemble d'attaque afin de détecter un sous-ensemble de ces traces permettant de réduire le rang final de la bonne clé voire de la retrouver. Il nous faut un moyen de faire la distinction en deux types de prédictions : les prédictions aidant à la convergence du rang et celles qui ont un impact négatif sur cette convergence.

5.2 Distinction de la qualité des prédictions

L'origine de cette distinction provient des modèles utilisés pour faire les prédictions à partir des traces. Que ce soit pour les réseaux de neurones ou pour les templates, il est nécessaire d'utiliser des traces de profilage afin d'entraîner ou de créer le modèle qui fera les prédictions. Le modèle appris est biaisé et est plus précis sur les traces qui ont permis de le construire. La preuve en est que les attaques contre les ensembles d'entraînement nécessitent toujours moins de traces que les attaques sur les ensembles de validation et de test. Le modèle extrait plus d'information des traces qu'il connaît que de traces qu'il n'a jamais vu. Cela veut dire que les traces de l'ensemble d'attaque qui ressemblent le plus à celles d'entraînement seront probablement mieux prédites que les autres. On peut alors créer une distinction entre les traces bien perçues et celles mal perçues en fonction des caractéristiques qu'elles partagent avec les traces d'entraînement. Cependant, cette distinction n'est en pratique pas faisable seulement avec les traces de consommation de

courant. Il faudrait en effet être capable de caractériser entièrement la consommation de courant du composant, ce qui est très difficile à faire en pratique. Une solution est donc d'utiliser les prédictions du modèle pour essayer d'effectuer cette distinction.

En effet, ce n'est pas la trace en elle-même qui va nous intéresser mais la façon qu'a le modèle de prédire cette trace et cette façon est caractérisée par la distribution des prédictions. Dès lors, la recherche d'une distinction entre les traces passe par une distinction prédictions plus ou moins bien prédites. On peut alors garder l'hypothèse classiquement utilisée en attaque par canaux auxiliaires que toutes les traces contiennent un niveau d'information similaire mais que c'est le traitement de cette information par le modèle qui va varier. La différence entre une trace bien ou mal prédite réside dans la capacité du réseau à extraire l'information contenue dans la trace.

Il nous faut maintenant trouver un moyen de caractériser cette distinction au niveau de l'attaque. Pour la phase de test préliminaire, nous allons étudier les comportements des prédictions de traces venant de l'ensemble de validation et de l'ensemble de test pour lesquels la clé utilisée est connue. Cela veut dire que nous avons accès aux labels de chaque trace et donc nous pouvons identifier les valeurs des labels au sein des prédictions. Il est alors possible regarder leurs positions par rapport aux autres classes en fonction des prédictions. C'est équivalent à regarder le rang d'attaques n'utilisant qu'une seule trace. Nous utiliserons le terme de rang d'une prédiction pour parler de la position de son label parmi toutes les classes rangées de la plus probable à la moins probable. Il est alors possible de déterminer ce que l'on pourrait appeler un *ordonnancement parfait* où les prédictions sont classées par rapport à leurs rangs. Toutefois, cette méthode demande la connaissance de la clé pour connaître les labels et n'est donc applicable qu'au niveau des traces d'entraînement et de test. Il faut donc trouver d'autres moyens d'ordonner les prédictions qui ne nécessite pas de connaissance sur les valeurs intermédiaires du chiffrement. Dans la prochaine section, nous allons discuter de quelques méthodes possibles qui se basent sur des notions de confiance du modèle en ses prédictions.

5.2.1 Quelques méthodes pour distinguer la qualité des prédictions

Les premiers tests que nous avons réalisés pour déterminer un ordonnancement des prédictions d'attaque se sont concentrés sur l'utilisation de certaines valeurs particulières des prédictions. Ces analyses sont principalement valables pour l'utilisation d'un réseau de neurones utilisant un *SoftMax* comme fonction d'activation de sa dernière couche. Cette fonction force les valeurs des prédictions à se situer entre 0 et 1 et à se sommer à 1. Nous cherchons à analyser à partir de la distribution des valeurs le niveau de confiance du réseau dans ses prédictions. Il faut toutefois distinguer confiance et performance puisque le fait qu'un réseau montre une forte confiance dans sa prédiction ne veut pas dire qu'elle est correcte. Malgré cela, la confiance du réseau présente un intérêt certain pour distinguer les prédictions bien ou mal prédites.

Médiane La médiane d'une distribution représente la valeur qui permet de séparer cette distribution en deux parties ayant le même nombre d'éléments. Une partie contient les valeurs supérieures à la médiane et l'autre les valeurs inférieures. Dans le cas de notre

application, la médiane sépare les 128 valeurs de prédictions les plus fortes des 128 les plus basses de la prédiction. Elle permet de faire apparaître une certaine distinction entre les prédictions bien prédites et les prédictions mal prédites. Il est possible de lier cette valeur à la confiance du réseau dans sa prédiction. En effet, plus un réseau montrera de la confiance dans sa prédiction, plus il aura tendance à attribuer de fortes valeurs aux classes qu'il considère les plus probables et par conséquent de faibles valeurs aux classes restantes. Ceci est dû à l'utilisation d'une opération de Softmax en sortie du réseau qui impose une normalisation sur les valeurs de prédiction. L'impact sur la médiane est que plus un réseau attribuera un petit nombre de fortes valeurs dans sa prédiction, plus la médiane aura tendance à être faible. Au contraire, s'il est peu sûr de comment prédire une trace, il aura tendance à accorder des valeurs proches et relativement élevées à un plus grand nombre de classes. De cette façon, il est possible d'ordonner les prédictions depuis celle ayant la médiane la plus faible à celle ayant la plus forte. La médiane peut alors être un indicateur de cette confiance et être utilisée comme distingueur pour déterminer quelles prédictions sont à privilégier.

Minimum Le minimum de chaque prédiction est aussi une valeur intéressante à explorer. Même si sa connexion avec la confiance du réseau est moins évidente, elle reste présente. En effet, si un réseau associe une forte valeur à la classe qu'il considère la plus probable, cela signifie que les autres classes auront nécessairement des valeurs plus faibles et donc que le minimum de ces autres valeurs aura plus de chance d'être faible. Nous pouvons donc ordonner les prédictions de manière croissante par rapport au minimum des scores des prédictions de la sortie du softmax de chaque trace afin de représenter la confiance du réseau dans les diverses prédictions. De façon similaire, on peut utiliser le maximum des scores des prédictions associés à chaque trace. Cette fois-ci cependant, il faut utiliser un ordre décroissant pour obtenir une indication de la prédiction montrant le plus de confiance. Il est logique pour un réseau d'attribuer la valeur maximale de prédictions à la trace et la classe pour lequel il est le plus sûr de lui. Toutefois, lors de nos tests, même si le maximum permettait d'obtenir des ordonnancements intéressants, ceux-ci n'étaient pas aussi efficaces que ceux obtenus avec le minimum. Nous nous sommes donc concentrés sur l'utilisation du minimum dans la suite de ce chapitre.

L'utilisation de ces différents distingueurs est illustrée Figure 5.4. Cette figure montre le traitement de trois traces différentes par un réseau de neurones ainsi que les prédictions qui en résultent. Les valeurs des prédictions sont ensuite utilisées pour déterminer un ordonnancement à utiliser lors de l'attaque.

Autres indicateurs Nous avons également regardé comment se comportent d'autres indicateurs donnant une idée de la confiance du réseau. Parmi eux on peut noter la variance, l'entropie, la distance entre les classes les plus probables d'une prédiction, la distance entre la médiane et le minimum pour citer les plus importants. Nous ne parlerons pas en détails de chaque indicateur mais ils montrent tous un potentiel pour ordonner les prédictions du réseau même s'ils ne les classent pas toujours de la même façon. Ce potentiel a été mesuré

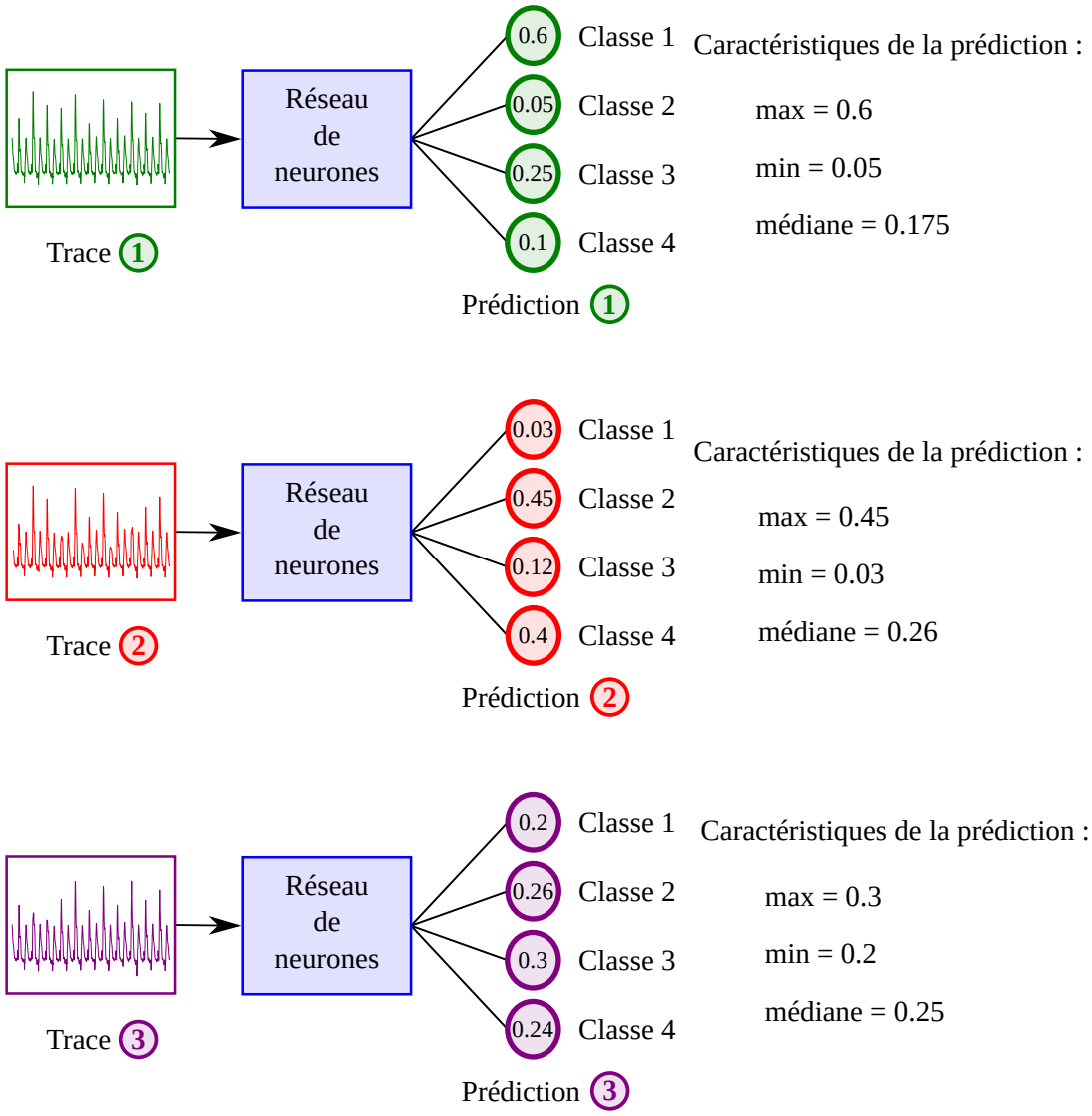
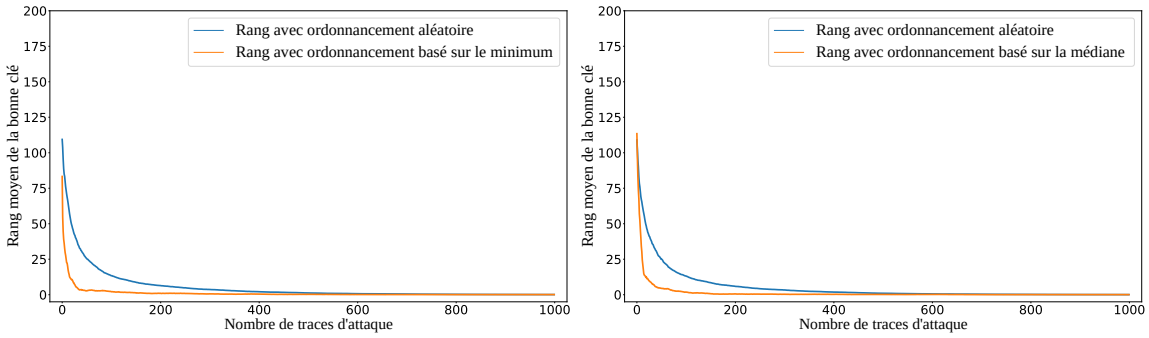


FIGURE 5.4: Exemple de l'utilisation de distingueurs (maximum, minimum, médiane) pour ordonner les prédictions de 4 classes possibles (1,2,3 et 4).



(A) Minimum (B) Médiane

FIGURE 5.5: Évolution du rang moyen de la bonne clé pour des attaques utilisant un ordonnancement aléatoire et un ordonnancement basé sur un distingueur (minimum et médiane) des prédictions du réseau CNN_{best} sur la base de données ASCAD clé fixe et l'ensemble Desync0.

de manière empirique via des tests réalisés sur l'architecture CNN_{best} et sur les ensembles de données ASCAD clé fixe et clé variable.

Nous allons maintenant voir quelques résultats obtenus avec des ordonnancements faits à partir de la médiane et du minimum.

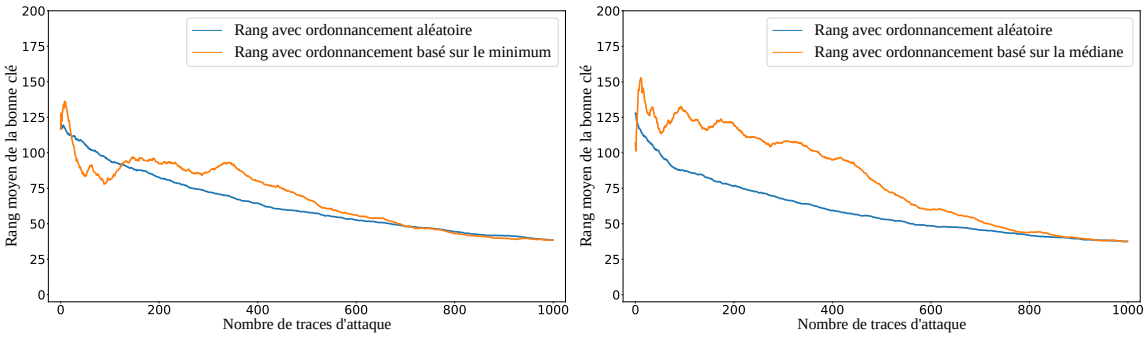
5.2.2 Résultats expérimentaux sur des réseaux de neurones

ASCAD clé fixe

Nous allons commencer par étudier le comportement des attaques utilisant le réseau CNN_{best} pour attaquer l'ensemble de traces d'attaque de la base de données ASCAD clé fixe décrite Section 3.4. Le réseau est le même que celui utilisé par Benadjila *et al.* [BPS+19] et a été entraîné pendant 75 epochs en utilisant 50 000 traces. Les courbes mentionnées dans cette section représentent l'évolution du rang moyen calculé à partir de 100 attaques utilisant des traces aléatoires choisies parmi les 10 000 traces de l'ensemble d'attaque.

La Figure 5.5 représente l'évolution du rang de la bonne clé pour des attaques utilisant un ordonnancement aléatoire des prédictions et des attaques utilisant un ordonnancement basé soit sur le minimum des prédictions, soit sur leur médiane. Les attaques sont faites sur des traces synchronisées ce qui explique qu'elles réussissent en moins de 1000 traces. On constate que, comme expliqué précédemment, utiliser un ordonnancement des prédictions lorsque le rang converge vers 1 n'apporte pas d'amélioration à l'attaque. On se situe dans le pire cas évoqué en Section 5.1.2 mais cela n'a pas d'impact étant donné que l'attaque est réussie. On peut tout de même noter que les attaques utilisant des prédictions ordonnées à partir du minimum et de la médiane convergent en moyenne plus rapidement que pour des prédictions ordonnées aléatoirement.

Desync50 Nous avons continué les tests sur un ensemble de données plus difficile : l'ensemble de traces désynchronisées Desync50. Cet ensemble contient des traces ayant subi un décalage de ses points d'une valeur comprise entre 0 et 50. Le même réseau CNN_{best}, entraîné sur les traces synchronisées, est utilisé pour exécuter des attaques sur Desync50. Ceci rend les attaques plus difficiles pour le réseau puisqu'il n'a jamais fait face à la désyn-



(A) Minimum (B) Médiane

FIGURE 5.6: Évolution du rang moyen de la bonne clé pour des attaques utilisant un ordonnancement aléatoire et un ordonnancement basé sur un distingueur (minimum et médiane) des prédictions du réseau CNN_{best} sur l'ensemble ASCAD clé fixe Desync50.

chronisation lors de son entraînement. On peut le constater sur la Figure 5.6, qui représente l'évolution du rang moyen des attaques pour un ordonnancement aléatoire des prédictions et pour un ordonnancement basé sur le minimum et la médiane des prédictions. La valeur finale du rang au bout des 1000 traces est de 34 en moyenne. De plus, les Figures 5.6a et 5.6b montrent que l'utilisation d'ordonnancements basés sur le calcul du minimum et de la médiane respectivement mènent à des attaques qui convergent plus lentement que les attaques classiques.

Étant acquis que les ordonnancements déterminés sont fixes pour un ensemble de prédictions donné, ceci signifie que la divergence que l'on observe par rapport à l'évolution du rang des attaques classiques peut être inversée si l'ordonnancement des prédictions est lui-même inversé. En reprenant l'exemple de la Figure 5.4, l'inversion de l'ordonnancement des prédictions se fait de la manière suivante.

Ordonnancements déterminés par les différents distingueurs :

max ① ② ③

min ② ① ③

médiane ① ③ ②

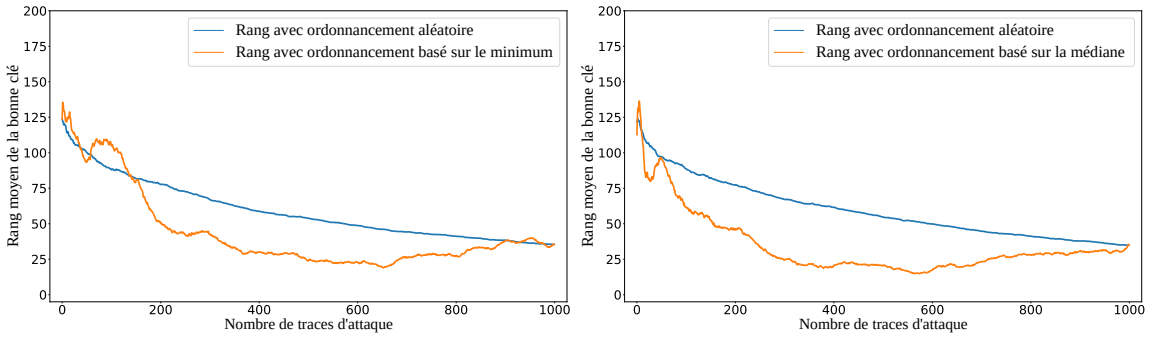
Ordonnancements **inversés** déterminés par les différents distingueurs :

max ③ ② ①

min ③ ① ②

médiane ② ③ ①

Les prédictions placées en premières se retrouvent utilisées en dernières lors de l'attaque et inversement pour les prédictions placées en dernières. La Figure 5.7 illustre ce



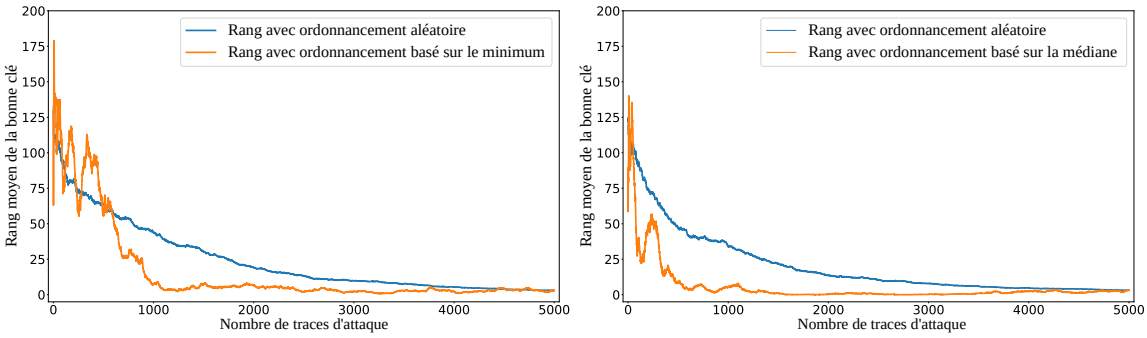
(A) Minimum inversé (B) Médiane inversé

FIGURE 5.7: Évolution du rang moyen de la bonne clé pour des attaques utilisant un ordonnancement aléatoire et un et un ordonnancement inversé basé sur un distingueur (minimum et médiane) des prédictions du réseau CNN_{best} sur l'ensemble ASCAD clé fixe Desync50.

phénomène en montrant les évolutions des rangs de la bonne clé pour des attaques utilisant des prédictions ordonnées avec le minimum et la médiane et en inversant les ordonnancements. Ceci signifie utiliser en premier les traces ayant les prédictions ayant les minimums les plus élevés et de même pour les médianes. On constate alors que l'on retrouve bien la divergence avec le rang des attaques utilisant des prédictions ordonnées aléatoirement mais cette fois-ci inversée. Les rangs minimaux atteints par les attaques ordonnées sont d'environ 10 pour la médiane et 19 pour le minimum. Ils sont inférieurs au rang final ce qui résulte en une amélioration de l'attaque.

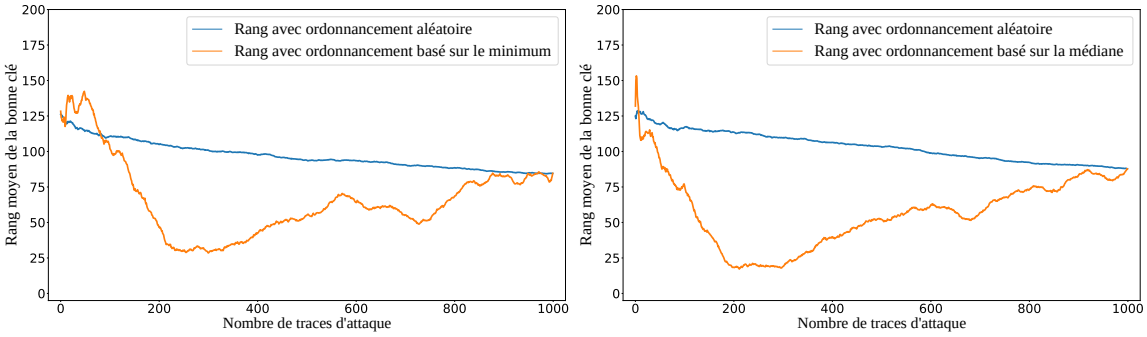
Pour finir les tests sur l'ensemble Desync50, nous avons augmenté le nombre de traces utilisées pour les attaques à 5000 traces. La Figure 5.8 montre les résultats de ces attaques. On peut y voir que l'utilisation de la médiane pour ordonner les prédictions permet d'obtenir des résultats légèrement meilleurs que pour le minimum. On constate aussi que le rang atteint une valeur de 1 pendant quelques milliers de traces sur la Figure 5.8b alors que le rang final est de 3. Cela indique que la bonne clé peut être récupérée directement avec cette méthode d'attaque, contrairement à la méthode classique, à condition d'être capable de déterminer le bon nombre de traces à utiliser. Pour pousser les tests plus loin, nous avons ensuite considéré l'ensemble de traces Desync100.

Desync100 Cet ensemble est encore plus difficile à attaque pour CNN_{best} étant donné que les traces peuvent avoir subi un décalage d'au plus 100 points. De nouveau, les ordonnancements basées sur les valeurs des minimums et des médianes des prédictions ont été inversés, c'est-à-dire que les premières prédictions de cette ordonnancement sont placées en dernières et les dernières en premières, afin d'obtenir de bons résultats. Les résultats sont présentés sur les Figures 5.9 et 5.10 qui montrent des attaques utilisant 1000 traces pour la première et 5000 traces pour la deuxième. Une fois de plus, on constate que les attaques utilisant des prédictions ordonnées convergent dans un premier temps vers la valeur 1 avant de réaugmenter pour atteindre le rang final. L'utilisation du minimum sur la Figure 5.9a permet d'obtenir un rang minimal d'environ 28 et l'utilisation de la médiane sur la Figure 5.9b permet d'atteindre le rang 20 au minimum de la courbe. Ces valeurs sont



(A) Minimum inversé (B) Médiane inversé

FIGURE 5.8: Évolution du rang moyen de la bonne clé pour des attaques utilisant 5000 traces et un ordonnancement aléatoire et un ordonnancement inversé basé sur un distingueur (minimum et médiane) des prédictions du réseau CNN_{best} sur l'ensemble ASCAD clé fixe Desync50.



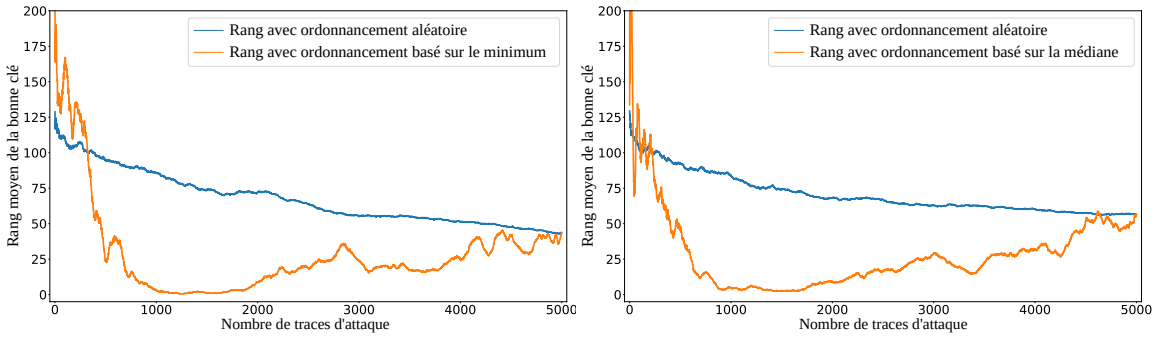
(A) Minimum inversé (B) Médiane inversé

FIGURE 5.9: Évolution du rang moyen de la bonne clé pour des attaques utilisant un ordonnancement aléatoire et un ordonnancement inversé basé sur un distingueur (minimum et médiane) des prédictions du réseau CNN_{best} sur l'ensemble ASCAD clé fixe Desync100.

une nette amélioration par rapport au rang final qui se situe autour de 90.

L'amélioration est également notable lors de l'utilisation de 5000 traces d'attaque sur les Figures 5.10a et 5.10b. Le rang minimum atteint lors de ces attaques pour un ordonnancement inversé basé sur le minimum est de 2 et celui pour un ordonnancement basé sur la médiane est de 3. La clé est donc presque récupérée grâce aux prédictions ordonnées là où elle est classée au rang environ 50 en moyenne à la fin des attaques. L'amélioration obtenue est très intéressante mais elle est aussi fortement liée aux ensembles de données et nous allons maintenant voir les raisons de ces convergences du rang et l'explication de l'inversement des ordonnancements des prédictions pour Desync50 et Deync100.

Explication des résultats Nous avons déjà donné des pistes d'explications basées sur la confiance du modèle pour expliquer que l'utilisation d'un ordonnancement permettait une convergence plus rapide du rang de la bonne clé. Dans le cadre des tests effectuées à l'aide du réseau CNN_{best}, nous pouvons pousser plus loin ces explications. Pour cela, nous pouvons regarder comment sont distribuées les valeurs des prédictions pour chacun des ensembles Desync0, Desync50 et Decync100. La Figure 5.11 représente les moyennes



(A) Minimum inversé (B) Médiane inversé

FIGURE 5.10: Évolution du rang moyen de la bonne clé pour des attaques utilisant 5000 traces et un ordonnancement aléatoire ou un ordonnancement inversé basé sur un distingueur (minimum et médiane) des prédictions du réseau CNN_{best} sur l'ensemble ASCAD clé fixe Desync100.

des distributions des prédictions ordonnées par leurs valeurs croissantes. C'est-à-dire que chaque prédiction est ordonnée de manière croissante donc de sa valeur la plus faible à la plus grande. Une moyenne est ensuite faite à partir des prédictions ordonnées. Les premières valeurs sont donc les moyennes des minimums des prédictions pour les ensembles Desync0, Desync50 et Desync100 et les dernières sont les moyennes des maximums. Cela nous permet de constater une grande différence dans ces valeurs entre chaque ensemble de données. Pour le minimum, les valeurs sont de $2 \cdot 10^{-5}$, $2 \cdot 10^{-6}$ et $1.5 \cdot 10^{-6}$ pour respectivement les ensembles Desync0, Desync50 et Desync100. Il y a donc un ordre de grandeur de 10^{-1} entre l'ensemble Desync0 et les ensembles désynchronisés et on remarque que plus la désynchronisation est élevée, plus la moyenne des minimums est basse. On retrouve également ce phénomène pour le maximum. Le maximum pour Desync0 est de 0.1 environ, 0.3 pour Desync50 et 0.6 pour Desync100. Cette fois, une valeur moyenne élevée de maximum est liée à une valeur élevée de désynchronisation.

Si l'on fait le lien avec la discussion sur le minimum et maximum de la Section 5.2.1, si le minimum et maximum sont respectivement suffisamment faibles et élevés, cela indique un bon niveau de confiance du réseau dans ses prédictions. Or, si l'on regarde les performances des attaques on se rend compte que le réseau prédit mal les traces désynchronisées. Ces deux phénomènes peuvent s'expliquer par le fait que le réseau est entraîné sur des traces synchronisées, il n'a donc jamais vu de traces dont les points sont décalés. Du point de vue du réseau, on peut séparer les ensembles Desync50 et Desync100 en deux sous-ensembles de traces : le sous-ensemble des traces synchronisées et le sous-ensemble des traces désynchronisées. Le réseau est alors capable de prédire correctement le premier sous-ensemble mais rencontre des difficultés pour prédire le deuxième. Les points qu'il utilise habituellement pour déduire la variable intermédiaire du chiffrement ne sont plus aux mêmes endroits. Cela veut dire qu'il se base sur des points de consommation de courant qui ne contiennent pas de fuite sur la valeur recherchée. Il interprète ces points comme des valeurs extrêmes et attribue à une des classes une valeur de prédiction très élevée. Cela impacte directement le minimum des prédictions comme on peut le voir sur la Figure 5.12 qui montre les valeurs des minimums classées en fonction de la valeur de

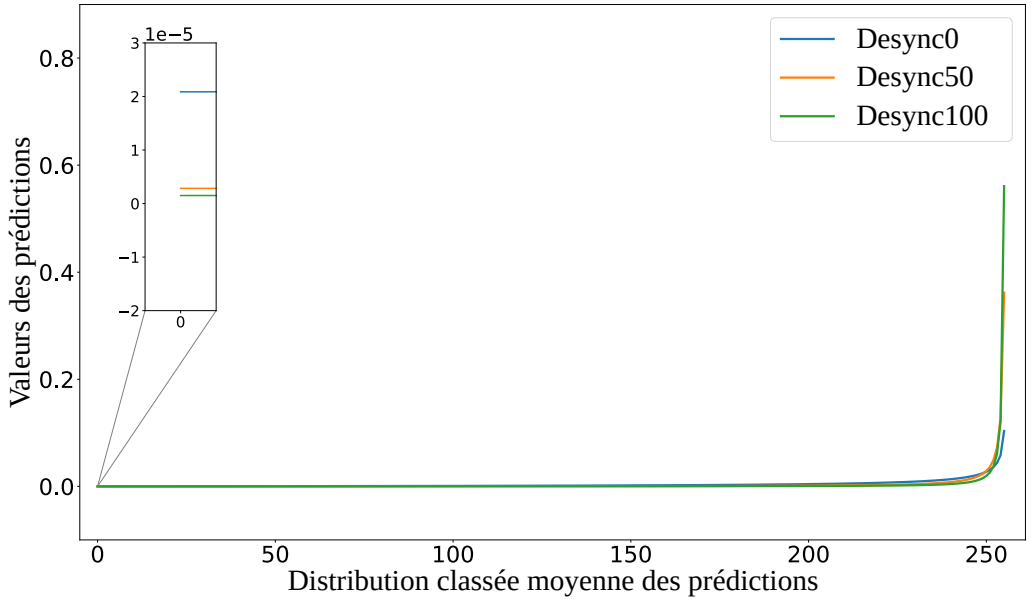


FIGURE 5.11: Évolution de la moyenne des prédictions ordonnées par ordre croissant de valeur pour les différentes niveaux de désynchronisation.

désynchronisation associée à la trace. On y voit que plus la désynchronisation est élevée, plus le minimum est bas.

Ces phénomènes expliquent à la fois les comportements des attaques utilisant des prédictions ordonnées et pourquoi il est nécessaire d'inverser l'ordonnancement pour faire converger le rang de ces attaques. Cela est dû au fait que les valeurs des minimums et des médianes ne servent plus nécessairement à ordonner les traces mais servent à discriminer les traces selon la valeur de désynchronisation.

ASCAD clé variable

La suite des tests est faite sur la base de données ASCAD clé variable composée de traces de 1400 points. Le réseau utilisé pour réaliser les attaques est CNN_{best} entraîné à partir de 50 000 traces d'entraînement synchronisées pendant 75 epochs. Les courbes représentent l'évolution du rang moyen calculé à partir de 100 attaques utilisant des traces aléatoires choisies parmi les 100 000 traces de l'ensemble d'attaque. Les attaques faites avec des prédictions ordonnées dans cette section sont toutes faites avec un ordonnancement normal c'est-à-dire que les traces sont rangées du minimum de la valeur considérée jusqu'à son maximum.

Desync0 La Figure 5.13 représente l'évolution de la valeur moyenne du rang pour des attaques sur Desync0 utilisant des prédictions ordonnées par le minimum et la médiane des prédictions. On peut voir sur les Figures 5.13a et 5.13b que les attaques ordonnées n'améliorent que peu le rang final, qui est de 4, obtenant des minimums du rang d'environ 2.5 et 3 pour respectivement l'ordonnancement par le minimum et celui par la médiane. Toutefois, il apparaît encore une fois que cette nouvelle méthode d'attaque arrive à faire

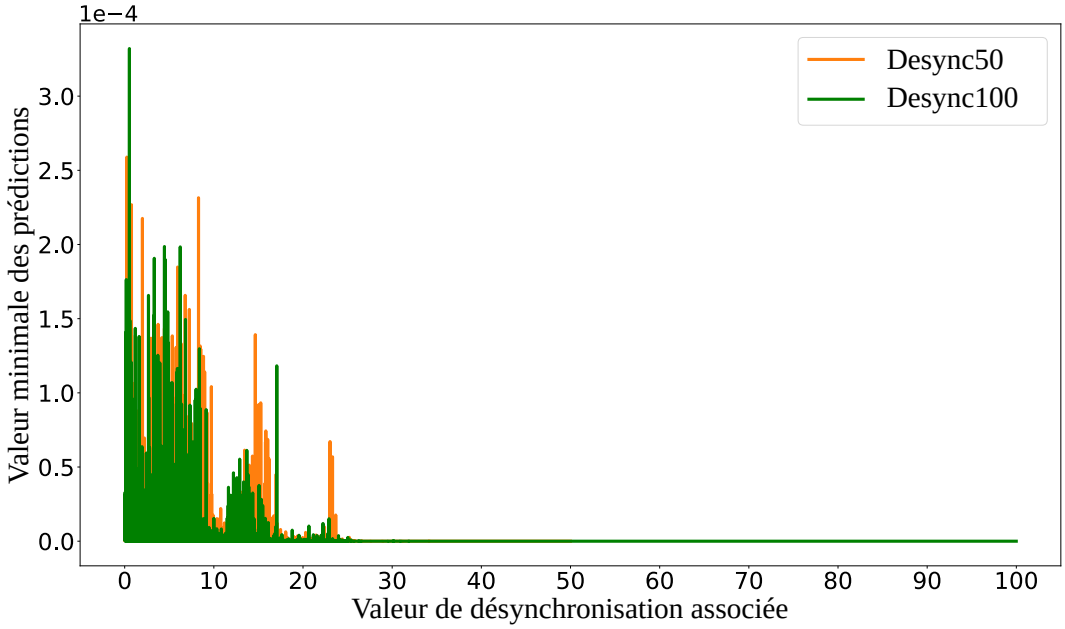
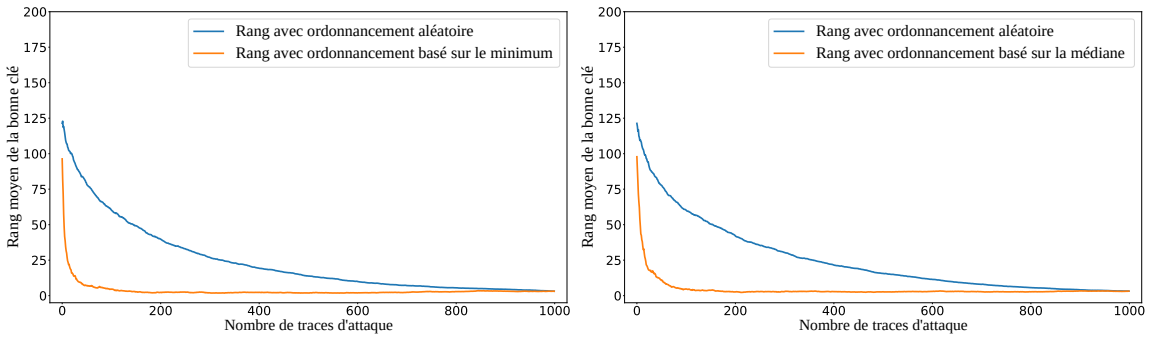


FIGURE 5.12: Valeurs minimales des prédictions pour Desync50 et Desync100 classées en fonction de la désynchronisation des traces.



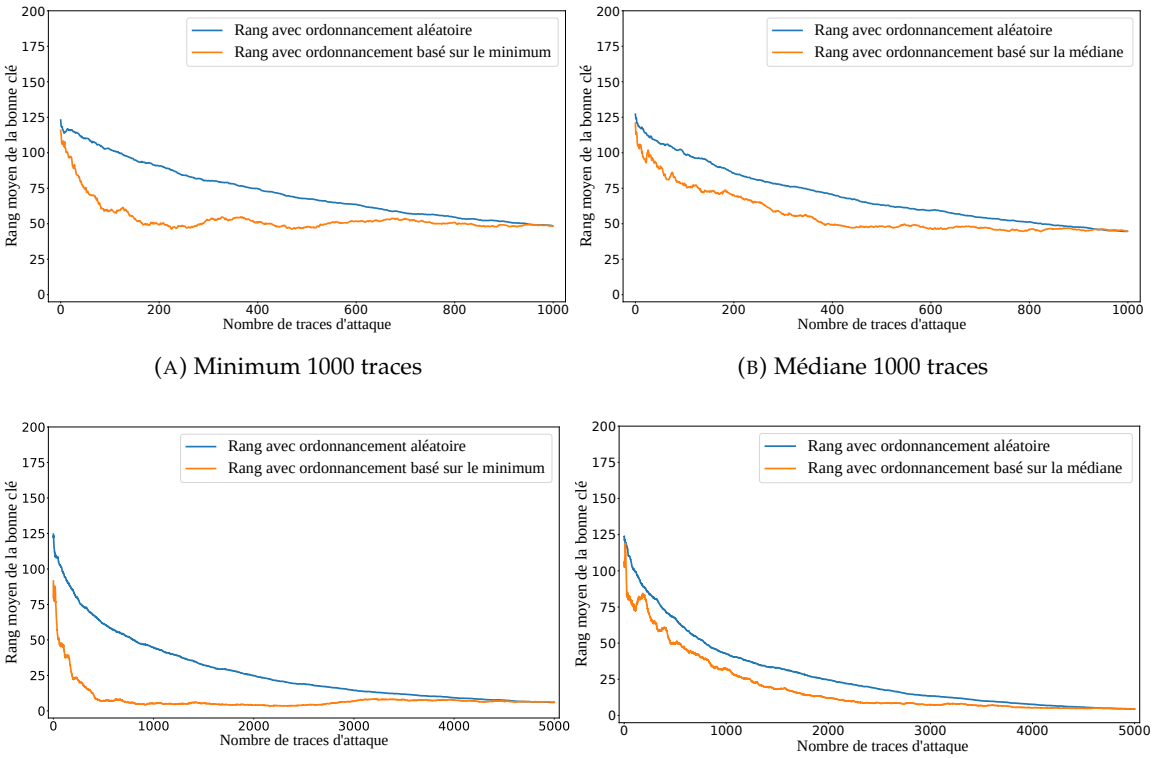
(A) Minimum (B) Médiane

FIGURE 5.13: Évolution du rang moyen de la bonne clé pour des attaques utilisant un ordonnancement aléatoire et un ordonnancement basé sur un distingueur (minimum et médiane) des prédictions du réseau CNN_{best} sur l'ensemble ASCAD clé variable Desync0.

converger le rang plus rapidement ce qui valide le positionnement des meilleurs traces en début d'attaque à l'aide des prédictions ordonnées.

Si ces résultats sont intéressants, ils n'apportent pas beaucoup d'information étant donné que ce réseau est capable de réussir les attaques en seulement un peu plus de 1000 traces. Nous allons donc une fois de plus regarder le comportement des attaques utilisant le même réseau sur les ensembles de données Desync50 et Desync100.

Desync50 La Figure 5.14 représente l'évolution des moyennes des rangs pour des attaques utilisant 1000 ou 5000 traces soit avec des prédictions ordonnées avec le minimum et la médiane des prédictions soit aléatoires. On peut voir sur ces figures qu'en utilisant seulement 1000 traces d'attaques, les effets de la nouvelle méthodes d'attaques sont légers. En effet, on ne constate qu'une faible amélioration en utilisant l'ordonnancement basé sur



(A) Minimum 1000 traces

(B) Médiane 1000 traces

(C) Minimum 5000 traces

(D) Médiane 5000 traces

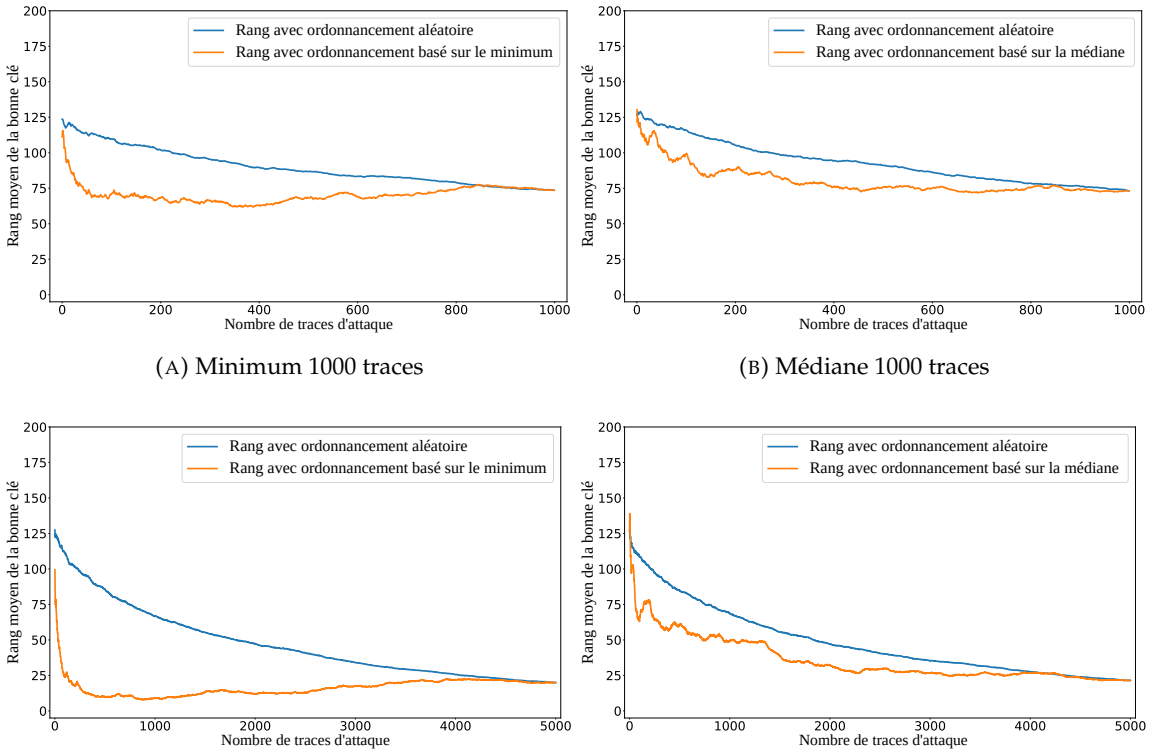
FIGURE 5.14: Évolution du rang moyen de la bonne clé pour des attaques utilisant différents nombres de traces et un ordonnancement aléatoire et un ordonnancement inversé basé sur un distingueur (minimum et médiane) des prédictions du réseau CNN_{best} sur l'ensemble ASCAD clé variable Desync50.

le minimum des prédictions avec un minimum du rang à 47 au lieu de 49 pour le rang final. L'utilisation de la médiane ici permet seulement une convergence légèrement plus rapide du rang mais pas d'amélioration du rang final.

Pour mieux observer l'effet de l'ordonnancement des traces, nous avons regardé le résultat d'attaques utilisant 5000 traces. Les Figures 5.14c et 5.14d représentent les évolutions des rangs moyens avec ou sans ordonnancement. On peut y voir que l'ordonnancement basé le minimum permet un amélioration de l'attaque avec un minimum du rang à environ 4.5 là où le rang final est de 7. L'utilisation de la médiane, elle, ne mène pas à l'obtention d'un rang inférieur au rang final mais seulement d'avoir une convergence en moyenne un peu plus rapide.

Cette fois encore la médiane n'apporte pas de bénéfice notable par rapport à l'attaque normale alors que l'utilisation du minimum permet de faire gagner plusieurs places à la valeur de la bonne clé. Nous allons finir les tests basés sur CNN_{best} en essayant d'attaquer l'ensemble désynchronisé Desync100 qui devrait être plus dur pour le réseau que Desync50.

Desync100 Les résultats des tests sur l'ensemble de données Desync100 sont illustrés Figure 5.15 pour des attaques utilisant 1000 et 5000 traces. Les Figures 5.15a et 5.15b montrent les différences entre l'ordonnancement utilisant la valeur du minimum pour la



(A) Minimum 1000 traces

(B) Médiane 1000 traces

(C) Minimum 5000 traces

(D) Médiane 5000 traces

FIGURE 5.15: Évolution du rang moyen de la bonne clé pour des attaques utilisant différents nombres de traces et un ordonnancement aléatoire et un ordonnancement inversé basé sur un distingueur (minimum et médiane) des prédictions du réseau CNN_{best} sur l'ensemble ASCAD clé variable Desync100.

première et de la médiane pour la deuxième. L'utilisation du minimum est plus efficace que l'utilisation de la médiane et permet d'obtenir un rang minimal à 63 ce qui est mieux que le rang final à 74. Toutefois l'amélioration reste faible sur les attaques basées sur 1000 traces.

Les Figures 5.15c et 5.15d illustrent les mêmes résultats mais pour des attaques utilisant 5000 traces. Cette fois-ci la réduction du rang minimum à l'aide de l'ordonnancement basé sur le minimum est plus significative puisque l'on passe d'un rang final de 22 à un rang minimum de 9. Le nombre de places gagnées reste donc à peu près le même mais il est plus intéressant étant donné qu'il réduit de plus de 50% la valeur du rang minimal de la bonne clé.

Nous allons maintenant discuter des différents résultats et interpréter les comportements observés lors des attaques sur les différents ensembles de données et en fonction de ordonnancements considérés.

Explications des résultats La première question qu'il semble important d'aborder est : pourquoi les ordonnancements ne sont pas inversés lors des attaques sur les ensembles de données issus de ASCAD clé variable ? Pour répondre à cette question, il faut regarder la Figure 5.16 qui illustre la distribution des prédictions lorsqu'elles sont ordonnées pour chaque trace puis moyennées. On retrouve sur la figure les distributions des prédictions

pour les ensembles Desync0, Desync50 et Desync100. On peut voir grâce à cette figure que les distributions pour les trois niveaux de désynchronisation sont bien plus proches que pour ASCAD clé fixe. En effet, la différence entre la valeur moyenne du maximum de chaque prédiction pour Desync0 et Desync100 passe d'environ 0.4 pour ASCAD clé fixe à moins de 0.06 environ pour ASCAD clé variable. Cette différence est significative puisqu'elle se répercute sur tout le reste des valeurs de la prédiction. Les minimums moyens des prédictions pour les différents ensembles sont également plus proches. Cela indique que le réseau prédit les traces synchronisées et les traces désynchronisées de manière plus similaire. Il n'y a donc plus besoin d'inverser les ordonnancements afin de discriminer les traces synchronisées et désynchronisées. Une autre raison pour expliquer ce rapprochement vient du fait que les traces ont 1400 points dans les ensembles issus de ASCAD clé variable comparé à 700 points pour ASCAD clé fixe. Une désynchronisation d'un même nombre de points a un effet plus faible sur les prédictions du premier ensemble que sur celles du second.

Nous allons maintenant répondre à une autre question : pourquoi l'utilisation de l'ordonnancement à l'aide du minimum semble avoir moins d'effet sur les traces de ASCAD clé variable que sur celles de ASCAD clé fixe ? Pour faciliter la réponse à cette question, nous allons utiliser la Figure 5.17. Cette figure reprend le principe de la Figure 5.12 mais pour les prédictions de ASCAD clé variable, c'est-à-dire que les minimums des prédictions sont ordonnés en fonction de la désynchronisation associée à chaque trace. De plus, étant donné que l'ensemble comprend 100 000 traces, les prédictions sont moyennées par fenêtre de 10 points pour Desync50 et 5 points pour Desync100. Cela permet d'aligner les points ayant les mêmes désynchronisations et explique pourquoi la courbe de Desync100 est plus grande que celle de Desync50. Des séparateurs sont également affichés afin de séparer les prédictions en blocs. Chaque bloc regroupe les prédictions ayant des valeurs de désynchronisation dans le même intervalle. Le premier bloc correspond à un intervalle de désynchronisation de 0 à 10, le deuxième de 10 à 20, etc... Sur la Figure 5.17, on peut voir que la valeur du minimum des prédictions dépend en effet de la valeur de désynchronisation de la trace. Toutefois, l'écart qui existe en les minimums pour des traces peu désynchronisées et des traces très désynchronisées est plus petit que pour la base de données ASCAD clé fixe. Cela est visible lorsque l'on compare les Figures 5.12 et 5.17. Sur la Figure 5.12, les valeurs minimales atteintes sont bien plus basses que sur la Figure 5.17, cela veut dire qu'il est plus simple de distinguer les traces fortement désynchronisées des traces peu désynchronisées et ainsi déterminer un bon ordonnancement.

Enfin nous allons discuter des raisons qui expliquent pourquoi la médiane n'est pas efficace pour déterminer un bon ordre. Il n'est pas facile de déterminer les raisons exactes des pauvres performances de la médiane mais la Figure 5.18, montrant l'évolution de la valeur médiane des prédictions en fonction du niveau de désynchronisation, peut nous y aider. Les courbes sont obtenues de la même manière que pour la Figure 5.17. On voit ainsi sur la courbe que pour les traces ayant un niveau de désynchronisation supérieur à 20, la valeur de la médiane se stabilise autour de $1.5 \cdot 10^{-3}$. Il devient donc difficile de distinguer les traces moyennement désynchronisées des traces fortement désynchronisées mais les traces étant peu désynchronisées restent facilement identifiables. Ces dernières traces

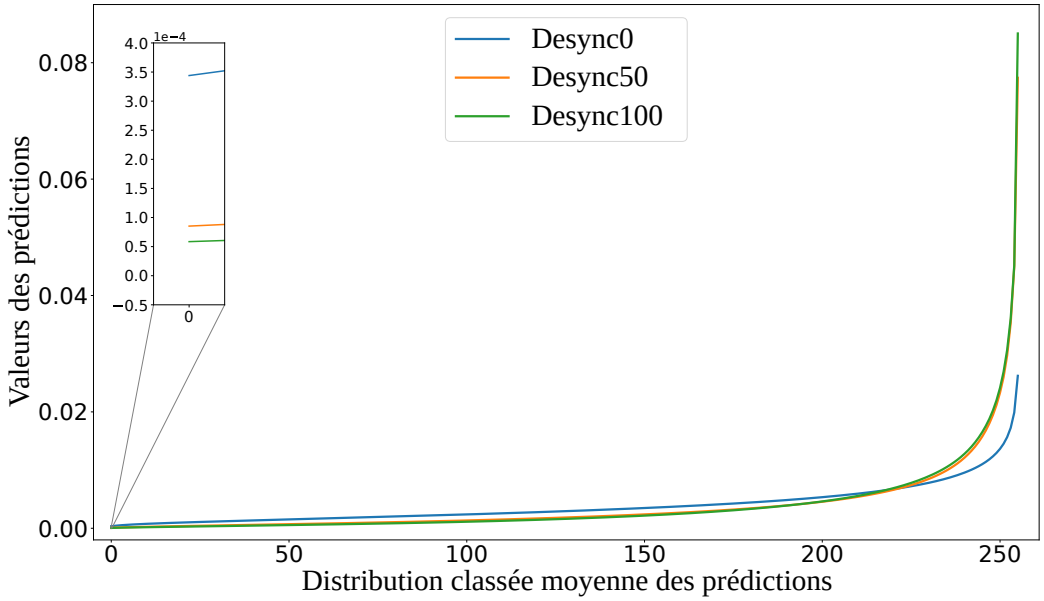


FIGURE 5.16: Évolution de la moyenne des prédictions ordonnées par ordre croissant de valeur pour différentes niveaux de désynchronisation.

expliquent le fait que les courbes utilisant un ordonnancement basé sur la médiane des prédictions convergent plus rapidement au début de l'attaque que lorsqu'un ordonnancement aléatoire est utilisé. Les traces peu désynchronisées sont placées en premières par l'ordonnancement mais les traces moyennement et fortement désynchronisées qui suivent ne sont pas ordonnées correctement car elles sont difficilement distinguables les unes des autres. Cela explique pourquoi il n'est pas possible d'atteindre un rang minimum inférieur au rang final.

Ces premières expériences montrent des comportements intéressants liés à cette nouvelle façon d'exécuter les attaques. Il est en effet possible d'atteindre des rangs minimums inférieurs aux rangs finaux des attaques de manière consistante. Toutefois, ces tests présentent l'inconvénient que le réseau CNN_{best} est appliqué à des datasets sur lesquels il n'est pas entraîné afin d'augmenter la difficulté des attaques. Cela permet de voir les effets de l'utilisation d'un ordonnancement mais n'est pas le contexte idéal. Nous allons donc effectuer une dernière série de tests sur des attaques template. Cette fois-ci, les templates sont créés avec différents nombres de traces afin d'établir plus en détails les gains obtenus avec cette méthode.

5.2.3 Résultats expérimentaux sur des attaques template

Dans cette section, nous avons utilisé des traces de consommation de courant issues d'un ChipWhisperer (XMEGA 8-bit) exécutant un AES 128-bit. Ces traces sont ensuite centrées sur 1000 points incluant le premier tour de l'AES[1]. L'ensemble de profilage est constitué de 50 000 traces utilisant des clés aléatoires et l'ensemble d'attaque se compose

1. La raison de ce ciblage est décrite Section 2.1.

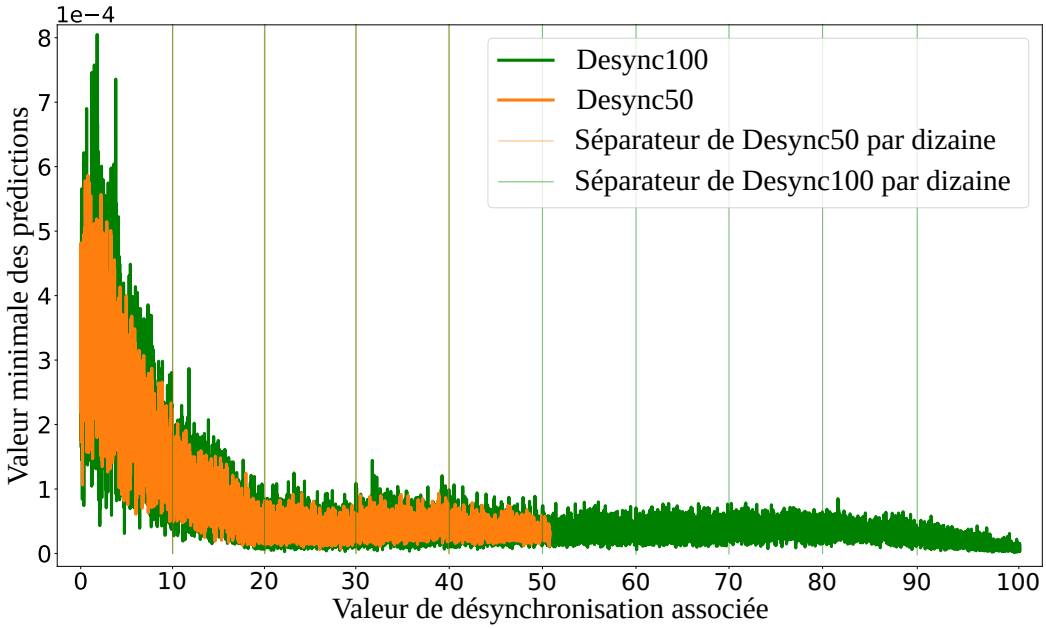


FIGURE 5.17: Valeurs minimales des prédictions pour Desync50 et Desync100 classées en fonction de la désynchronisation des traces.

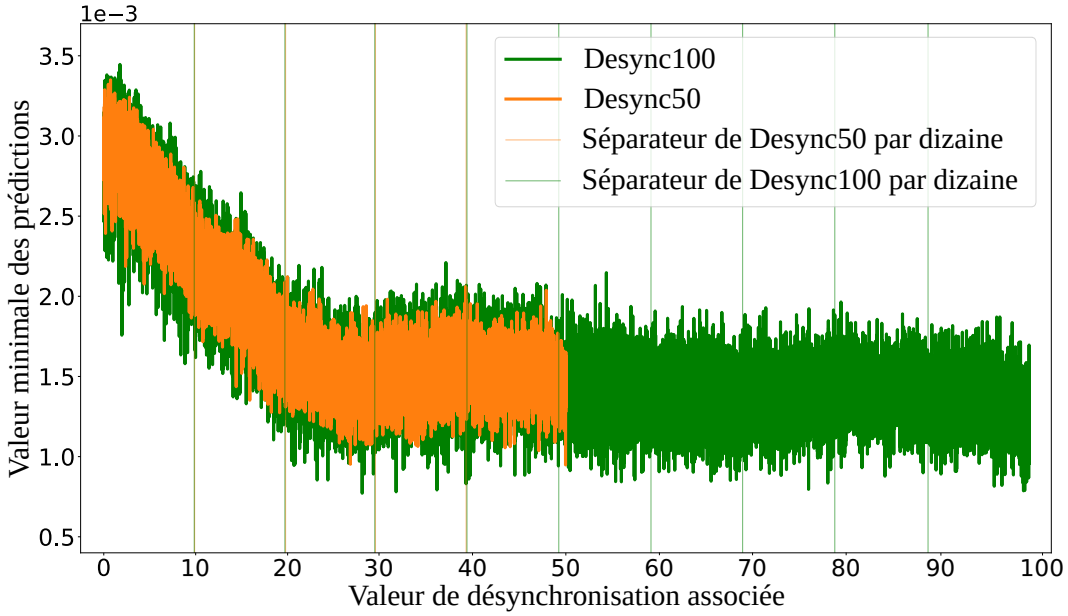


FIGURE 5.18: Valeurs médianes des prédictions pour Desync50 et Desync100 classées en fonction de la désynchronisation des traces.

de 1000 traces à clé fixe. L'octet choisi pour les attaques est le 16ème octet, la cible est donc :

$$Y(\mathbf{k}^*) = SubBytes(\mathbf{p}[16] \oplus \mathbf{k}^*[16]),$$

où $\mathbf{p}$ représente le message d'entrée de l'AES et $\mathbf{k}^*$ la clé utilisée. Cette implémentation d'AES n'est pas protégée et présente d'importantes fuites, il est donc très simple de réussir à récupérer la clé. Pour augmenter la difficulté des attaques, nous avons ajouté du bruit dans la consommation de courant afin de réduire la corrélation entre la valeur intermédiaire et les points des traces. Ce bruit est issu de la distribution normale $\mathcal{N}(0, 0.05)$ et chaque point de chaque trace reçoit une valeur différente de bruit. Les tests réalisés utilisent chacun un nombre de traces différent pour la construction du template afin d'observer comment se comporte les attaques ordonnées dans chacun des cas. Les templates sont réalisés à partir des 5 points des traces ayant le plus haut SNR.

Afin de se mettre dans le même contexte qu'avec les réseaux de neurones pour exécuter les attaques ordonnées, le template n'est plus appliqué à la volée au cours de l'attaque mais au préalable. Les templates de chaque valeur intermédiaire possible sont appliqués à chaque trace d'attaque afin d'obtenir un ensemble de prédictions similaire aux prédictions des réseaux de neurones. Les attaques sont faites à partir de ces prédictions qui servent également à déterminer les ordonnancements. Les résultats montrés par la suite proviennent de moyennes de 100 attaques réalisées sur 500 traces issues de l'ensemble d'attaque.

Template construit avec 10 000 traces Les résultats des attaques utilisant le template construit à partir de 10 000 traces de profilage sont illustrés Figure 5.19. Les attaques ordonnées utilisent un ordonnancement basé sur le minimum des prédictions de chacune des 500 traces de l'ensemble d'attaque. Les résultats montrent que le minimum moyen du rang obtenu en ordonnant les traces est de 47 là où le rang final des attaques est de 77. Même si ce rang améliore l'attaque, on constate que la forme de la courbe du rang utilisant les prédictions ordonnées est moins lisse que lors de l'utilisation de réseaux de neurones. Cela indique que l'ordonnancement basé sur le minimum des prédictions n'est pas optimal et qu'un meilleur ordre peut être trouvé.

Template construit avec 20 000 traces La Figure 5.20 illustre les résultats obtenus à partir du template crée avec 20 000 traces de profilage. Les Figures 5.20a et 5.20b montrent les évolutions du rang utilisant un ordonnancement normal et inversé. Cette fois-ci, l'ordonnancement normal basé sur le minimum, c'est-à-dire quand les traces sont classées du minimum du minimum au maximum du minimum, ne permet pas d'améliorer la valeur du rang minimum par rapport au rang final. Cependant l'ordonnancement inversé, lui, donne un rang minimal d'environ 10 ce qui améliore le rang final de 26. Nous reviendrons sur l'inversion de l'ordonnancement après avoir vu les résultats du template à 50 000 traces.

Template construit avec 50 000 traces La Figure 5.21 illustre les évolutions du rang pour les différentes méthodes d'attaque. On peut constater qu'encore une fois l'ordonnancement normal n'apporte pas d'amélioration à l'attaque et que la convergence du rang est

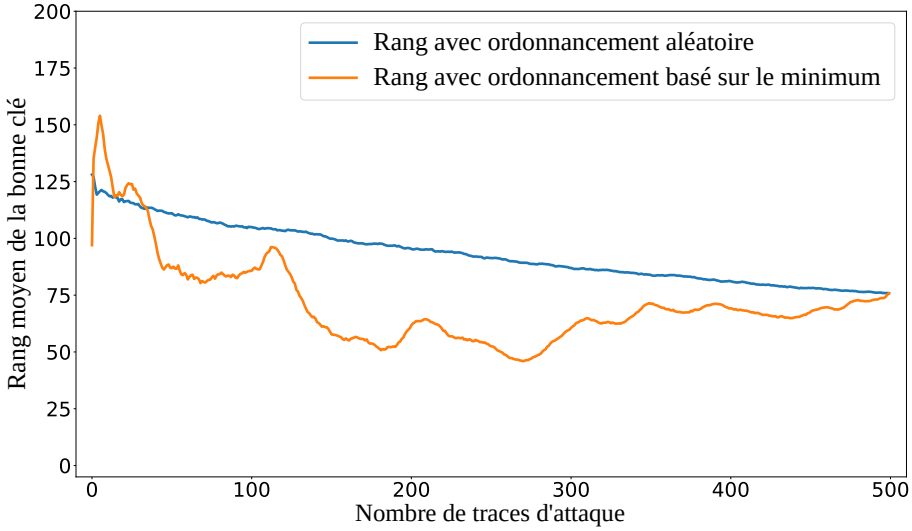
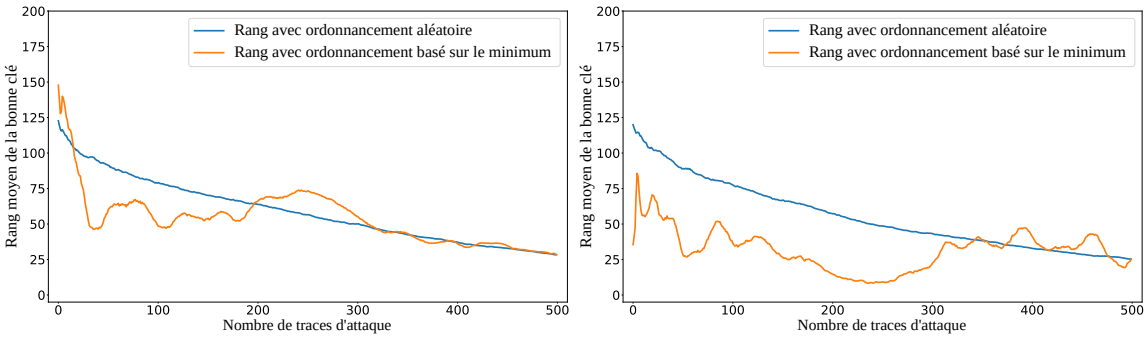
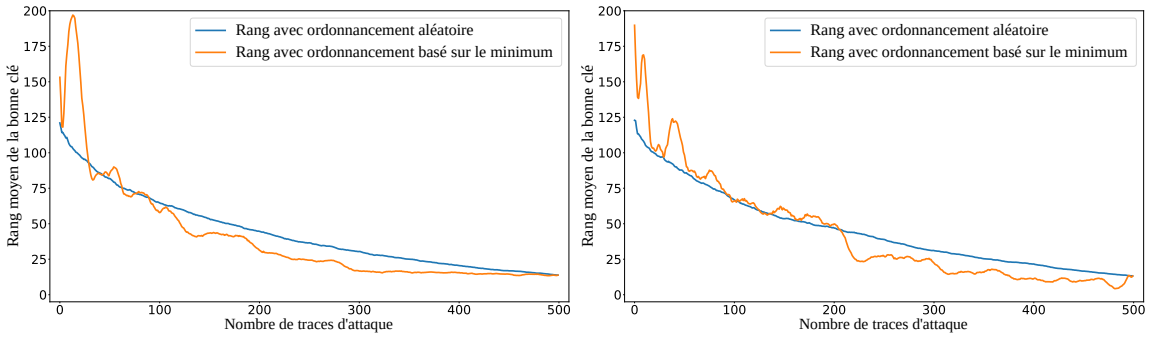


FIGURE 5.19: Évolution du rang moyen de la bonne clé pour des attaques utilisant un ordonnancement basé sur le minimum des prédictions d'un template réalisé avec 10 000 traces.



(A) Ordonnancement Normal (B) Ordonnancement Inversé

FIGURE 5.20: Évolution du rang moyen de la bonne clé pour des attaques utilisant un ordonnancement normal et un ordonnancement inversé basés sur le minimum des prédictions d'un template réalisé avec 20 000 traces.



(A) Ordonnancement Normal (B) Ordonnancement Inversé

FIGURE 5.21: Évolution du rang moyen de la bonne clé pour des attaques utilisant un ordonnancement normal et un ordonnancement inversé basés sur le minimum des prédictions d'un template réalisé avec 50 000 traces.

légèrement plus rapide puisque une valeur proche du rang final est atteinte au bout de 300 traces. L'ordonnancement inversé, quant à lui, permet une nette amélioration passant du rang final de 14 à un rang minimal de 7. Toutefois, ce rang minimal est atteint en utilisant environ 480 traces soit presque l'entièreté des traces d'attaque. Ce résultat contraste avec les résultats précédents qui montraient des minimums atteints avec généralement moins de 50% des traces de l'attaque.

Interprétations des résultats Les résultats obtenus sur les attaques templates sont plus complexes à interpréter. Contrairement aux attaques basées sur les réseaux de neurones pour lesquelles une architecture fixe était utilisée, les templates créés pour les attaques sont plus variables. Le nombre de traces utilisées pour les générer et le nombre de points d'intérêt considérés peuvent faire grandement varier les résultats des attaques. Un autre point important est que les valeurs des prédictions obtenues par l'application des templates ne se situent pas dans un intervalle fixe. Il arrive donc que les valeurs d'une prédiction soient supérieures à toutes les valeurs d'une autre. Il semble alors moins logique de se baser sur le minimum des prédictions pour ordonner les traces. Toutefois le minimum est la valeur la plus consistante dans nos expériences et permet d'obtenir des résultats intéressants malgré tout.

5.2.4 Conclusion des tests

Au cours des tests que nous avons réalisés sur les attaques basées sur les réseaux de neurones et sur les attaques templates, nous avons pu constater que la nouvelle méthode d'attaque basée sur l'ordonnancement des traces peut amener une amélioration des attaques. Cependant, ces tests ne nous permettent pas de conclure sur le meilleur ordonnancement possible. Les différents essais basés sur les valeurs comme le minimum et la médiane permettent l'obtention d'un rang minimal inférieur au rang final. L'instabilité qui apparaît dans certains résultats indique également qu'un autre ordonnancement serait optimal. Nous pouvons donc maintenant tenter de déterminer le meilleur ordonnancement. Pour cela, nous avons décidé d'explorer un pan du machine learning qui se concentre sur le classement d'exemples les uns par rapport aux autres et l'attribution de scores. Plus par-

ticulièrement, nous allons utiliser un réseau de neurones qui apprendra à classer les traces et déterminer un ordonnancement à partir de leurs prédictions.

5.3 Utilisation de réseaux de neurones pour résoudre le problème

Malgré des résultats prometteurs, il a été impossible de clairement identifier un indicateur permettant d'obtenir des résultats nettement au-dessus des autres. Nous avons donc décidé d'entraîner un réseau de neurones pour résoudre la tâche d'assigner un score à chacun des couples (trace,prédiction) de l'ensemble d'attaque. L'objectif est d'avoir un score qui fournisse un indicateur assez fiable de la qualité de la trace et de sa prédiction.

Le problème à résoudre dans ce contexte est un problème d'ordonnancement des traces. Cette fois-ci, il ne s'agit pas de classer les traces parmi un ensemble de labels mais de les classer les une par rapport aux autres. Pour cela, il faut attribuer un score à chaque trace afin de déterminer un ordre. La plupart des indicateurs utilisés dans nos travaux ont partiellement réussi à effectuer cette tâche. Toutefois, ils n'aident pas à déterminer le critère permettant d'ordonner les prédictions et il est clair qu'une amélioration est encore possible. Pour cela, nous allons entraîner un nouveau réseau de neurones afin qu'il fasse le classement des traces. Cette utilisation des réseaux de neurones fait partie de l'approche *Learning to rank* déjà bien étudiée en machine learning [BSR+05, BRL06, CLL+09].

5.3.1 L'approche Learning to rank

L'approche Learning to rank en machine learning englobe les techniques permettant de créer des modèles dont le but est de classer des données. Ces modèles sont utilisés dans de nombreux domaines comme les systèmes de recommandation [LMK+11, XJP+10, CAS16], la recherche d'information [WBSG10] ou le traitement automatique du langage naturel [Li14].

Un problème de classement peut être formulé de la manière suivante :

Problème Soient $\mathcal{Q}$ un ensemble de requêtes, $\mathcal{E}$ un ensemble d'entrée et $\mathcal{C}$ un classement des entrées de $\mathcal{E}$ par rapport aux requêtes de $\mathcal{Q}$: entraîner un modèle $F_\theta(q, e)$ de paramètres θ capable de classer une entrée $e \in \mathcal{E}$ en fonction d'une requête $q \in \mathcal{Q}$.

L'approche Learning to rank est vue principalement comme un problème d'apprentissage supervisé où le classement des données d'entraînement correspond aux labels d'entraînement. Ces labels représentent la pertinence des entrées entre elles vis-à-vis des requêtes. Le plus souvent, le modèle attribue un score à chaque entrée qui peut être utilisé pour déterminer l'entrée la plus pertinente.

Nous pouvons maintenant décrire notre problème sous la forme d'un problème Learning to rank. Soient $\mathcal{T}_{train}$ l'ensemble des traces d'entraînement, $\mathcal{T}_{val}$ l'ensemble des traces de validation et $\mathcal{T}_{att}$ l'ensemble des traces d'attaque. La première étape consiste en l'apprentissage d'un modèle $F_\theta(t)$ capable, étant donné une trace t, de fournir une prédiction sur la valeur intermédiaire correspondant à cette trace. Ce modèle est entraîné à partir de $\mathcal{T}_{train}$ et $\mathcal{T}_{val}$, utilisé seulement pour la validation, afin d'être performant sur $\mathcal{T}_{att}$. On

peut également considérer les ensembles $\mathcal{P}_{\mathcal{T}_{val}}$ et $\mathcal{P}_{\mathcal{T}_{att}}$ qui correspondent aux ensembles de prédictions effectuées par le modèle F_θ sur les traces de l'ensemble de validation et de l'ensemble d'attaque. L'ensemble $\mathcal{P}_{\mathcal{T}_{train}}$ n'est pas utilisé par la suite étant donné que les prédictions du modèle F_θ sur les traces d'entraînement ne sont pas représentatives des prédictions du même modèle sur l'ensemble d'attaque. Ceci est dû en grande partie au processus d'entraînement et à l'overfitting, que l'on retrouve dans la plupart des modèles utilisés pour les attaques, qui résultent en un modèle bien plus performant sur les traces d'entraînement que sur les traces d'attaque.

Le but de l'approche Learning to rank est l'apprentissage d'un second modèle $F'_{\theta'}(p)$ de paramètres θ' qui va attribuer un score à chaque prédiction $p \in \mathcal{P}$ en fonction d'un classement défini au préalable. Nous reviendrons par la suite sur la manière de définir ce classement. Ce second modèle s'entraîne à partir des prédictions $p \in \mathcal{P}_{\mathcal{T}_{val}}$ pour être capable de classer les prédictions de $\mathcal{P}_{\mathcal{T}_{att}}$ de façon à déterminer un ordre plus intéressant pour les attaques que l'ordre aléatoire.

À partir de notre application du problème Learning to rank, nous allons détailler les différentes manières d'approcher l'apprentissage.

- La *Pointwise approach*, ou approche point par point. Pour cette approche, on suppose qu'à chaque prédiction correspond un score. Le but du modèle est donc, étant donnée une prédiction p, de prédire un score $F'_{\theta'}(p) = s_p$ qui représente l'importance de la prédiction p. Il reçoit donc les exemples un à un et leur donne à chacun un score. Les scores sont ensuite comparés afin de déterminer un ordre dans les prédictions.
- La *Pairwise approach* [BSR+05, WBSG10], ou approche paire par paire, consiste à fournir au modèle une paire d'exemples (p_i, p_j) pour laquelle il doit prédire un ordre $F'_{\theta'}(p_i, p_j) = o_{p_i,p_j}$. Une valeur limite l est déterminée afin que si $o_{p_i,p_j} > l$, la prédiction p_i est considérée comme plus importante que la prédiction p_j et on note cette relation $p_i \triangleright p_j$. Il est ensuite possible de comparer les paires de prédictions entre elles afin d'établir un classement des prédictions de l'ensemble d'attaque.
- La *Listwise approach* [CQL+07, XLW+08], ou approche par liste, se concentre sur l'utilisation de listes d'exemples à ordonner. Le modèle reçoit donc en entrée une liste $(p_i)_{1 \leq i \leq n}$ et doit retourner un classement de chacun de ces éléments en fonction de leur pertinence. Cette approche nécessite d'être capable en amont de définir un classement complet de l'ensemble des exemples d'entraînement.

Pour la résolution de notre problème de classement, nous avons choisi de nous concentrer sur l'entraînement d'un réseau en suivant la Pairwise approach. Il est toutefois possible d'utiliser les deux autres approches en adaptant la méthode d'entraînement. L'approche Listwise semble être la plus pertinente étant donnée qu'elle prend en compte des listes de prédictions pour les classer ce qui se rapproche le plus de la méthodologie utilisée dans les attaques par canaux auxiliaires. Toutefois, elle est aussi plus complexe à mettre en place, c'est pourquoi nous nous sommes concentrés sur l'approche Pairwise. Nous allons maintenant détailler le réseau utilisé et la manière de résoudre le problème Learning to rank.

5.3.2 Utilisation d'un réseau siamois

Étant donné que nous avons choisi de nous concentrer sur l'approche Pairwise, il nous a semblé intéressant de considérer les réseaux siamois comme méthode d'apprentissage du réseau de classement. Un réseau siamois est constitué d'une architecture de réseau dupliquée donnant lieu à deux réseaux qui partagent leurs poids et qui font des prédictions en parallèle sur différents exemples [BBB+93]. Les résultats de ces prédictions sont ensuite comparés afin de donner la sortie du réseau. Les réseaux siamois sont généralement utilisés dans les problèmes de suivi d'objet en mouvement dans les vidéos [BVH+16, GFZ+17] ainsi que pour les problèmes de Learning to rank [GSC+19] entre autres.

La Figure 5.22 illustre un schéma de réseau siamois. On y voit en vert les deux réseaux qui partagent leurs architectures et leurs poids. Chacun prend en entrée une prédiction et lui associe un score, les scores sont ensuite comparés et le résultat de la comparaison est utilisé pour le calcul de la fonction de perte. Les poids de chaque réseau sont ensuite mis à jour de la même façon afin de garder la similarité. Dans notre application, nous gardons un des réseaux à la fin de l'entraînement afin de l'utiliser sur l'ensemble d'attaque. Ce faisant, on obtient un score pour chaque prédiction $p \in \mathcal{P}_{\mathcal{T}_{att}}$ ce qui permet de les ordonner pour l'attaque.

L'utilisation de la Pairwise approach se fait donc durant l'entraînement du réseau siamois où des paires de prédictions issues de $\mathcal{P}_{\mathcal{T}_{val}}$ sont choisies aléatoirement et fournies au réseau. Il reste maintenant à définir les labels de chaque paire de prédictions. Nous allons discuter de la méthode choisie pour évaluer quelles prédictions sont jugées plus intéressantes et voir comment d'autres méthodes pourraient également être considérés.

5.3.3 Choix des labels des paires de prédictions

Le choix des labels des paires n'est pas évident et plusieurs choix peuvent être valables. Le but de ce choix est de déterminer une relation $p_i \triangleright p_j$ entre les prédictions p_i et p_j qui indique laquelle devrait être positionner avant l'autre lors d'une attaque. Plusieurs pistes peuvent être suivies puisqu'il n'existe pas de travaux étudiant cette problématique. Nous discuterons ici de différentes méthodes : celle basée sur la comparaison du rang de la valeur intermédiaire, celle comparant directement la valeur de prédiction associée à la variable intermédiaire et finalement une combinant les deux précédentes.

Comparaison des rangs

La première méthode consiste en la comparaison des rangs des valeurs intermédiaires au sein de leur prédiction. Par rang, on désigne la position de la valeur du bon label au sein de toutes les valeurs de la prédiction classées par ordre décroissant. Le rang 1 indique que le bon label a bien été prédit par le réseau. On peut donc décrire la relation $p_i \triangleright p_j$ en utilisant les rangs des bons labels au sein de chaque prédiction. Soient r_i le rang du label de p_i et r_j celui de p_j, on obtient que :

$$p_i \triangleright p_j \Leftrightarrow r_i > r_j.$$

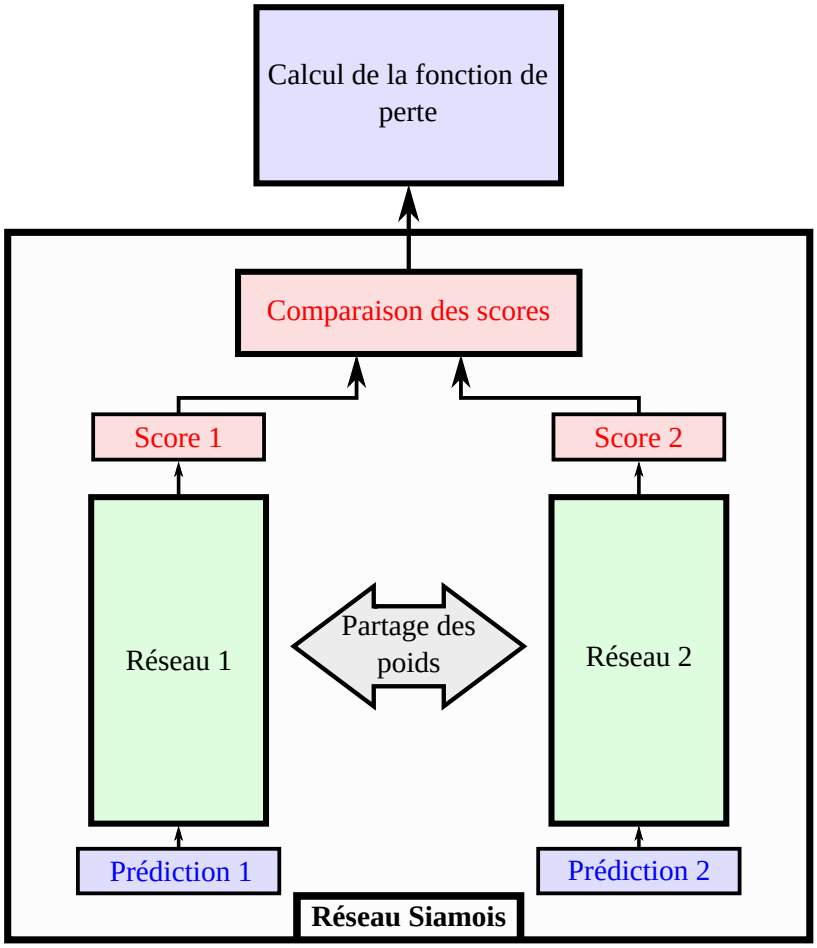


FIGURE 5.22: Schéma d'un réseau siamois. Les deux réseaux partagent les mêmes poids et reçoivent chacun une entrée différente. Les scores qu'ils déterminent sont ensuite comparés pour être utilisés dans la fonction de perte.

Cette méthode se concentre uniquement sur le classement des bons labels pour établir un ordre de classement pour les prédictions. La logique derrière cette méthode vient du fait que si le réseau est capable de prédire un ordre similaire pour les prédictions d'attaque, alors le résultat des attaques sera grandement amélioré.

Il existe un léger inconvénient à l'utilisation de cette méthode puisqu'elle ne prend pas en compte la valeur de prédiction du label. Il est donc possible d'avoir des prédictions jugées équivalentes car de rang égale mais qui ont des valeurs très éloignées. Or, une prédiction du bon label avec une valeur plus basse a moins d'impact sur la convergence du rang car elle fait progresser le rang plus doucement.

Comparaison des valeurs des bons labels

Une autre façon possible pour déterminer l'ordre dans lequel les prédictions devraient être est d'utiliser directement la valeur associée au bon label. On obtient donc l'équivalence suivante :

$$p_i \triangleright p_j \Leftrightarrow s_{p_i}(y_i) > s_{p_j}(y_j),$$

où $s_{p_i}(y_i)$ représente le score du bon label y_i de la prédiction p_i. En effet, plus cette valeur est élevée, plus elle va contribuer à la convergence du rang lors de l'accumulation des prédictions de la phase d'attaque. De plus, une forte valeur associée au bon label signifie que le rang du label est également élevé.

L'inconvénient est que lors de la comparaison de deux valeurs associées aux bons labels, les rangs des labels ne sont pas pris en compte. Cela veut dire qu'une trace ayant son bon label positionné rang 1 peut être jugée moins intéressante qu'une trace ayant un label de rang inférieur. Par exemple, si la distribution est proche d'une distribution uniforme, la valeur associée au bon label ne sera pas élevée même si son rang est haut. Au contraire, pour une distribution très diverse avec beaucoup de valeurs très faibles et le reste des valeurs très hautes, même si le label a un rang moyen, il peut avoir une valeur de prédiction élevée. Un autre problème est que ce type de comparaison ne prend que très peu en compte les autres valeurs de prédictions. En effet, seules les valeurs des bons labels sont utilisées dans le calcul de l'ordre des traces. On ne tient donc pas compte du nombre de labels possédant une valeur supérieure au bon label alors que c'est un aspect important pour la convergence du rang.

Combinaison des deux méthodes

La solution idéale semblerait donc être une combinaison des deux méthodes qui tiendrait compte à la fois de la position de la valeur du bon label dans la prédiction et de sa valeur en elle-même. Le problème est maintenant de déterminer comment combiner ces valeurs. Il faut aussi prendre en compte le fait que cette combinaison est à intégrer dans la fonction de perte utilisée pour l'apprentissage du réseau $F'_{\theta'}$. Cela peut donc poser des problèmes si le résultat de la combinaison est trop instable comme nous le verrons par la suite.

Nous allons clore cette section par une discussion sur l'évolution du rang et faire quelques remarques sur son fonctionnement.

Discussion sur l'évolution du rang

L'étude de la nouvelle méthode d'attaque décrite dans ce chapitre se concentre beaucoup sur l'évolution du rang de la bonne clé pour déterminer si les traces d'attaques sont classées dans un ordre intéressant ou non. Au cours de nos expériences nous avons constaté quelques comportements qui peuvent sembler contre-intuitifs et que nous voulions aborder.

Une prédiction donnée ayant un rang pour la bonne clé supérieur au rang actuel de la bonne clé dans l'accumulation des probabilités obtenues sur l'ensemble des traces observées jusqu'ici lors de l'attaque peut quand même faire progresser ce rang. En effet, ce n'est pas parce que le rang de la bonne clé dans une prédiction est supérieur à son rang actuel que cette prédiction n'aide pas à faire converger l'attaque. La Figure 5.23 illustre ce phénomène en montrant l'évolution du classement des valeurs possibles de la clé après l'ajout d'une prédiction supplémentaire. Ces valeurs sont classées de la plus probable en bas à la moins probable en haut. On y voit en rouge la bonne clé et en bleu les valeurs ayant un rang inférieur à la bonne clé après l'utilisation de n prédictions. Si l'on regarde la position de la bonne clé dans la nouvelle prédiction, celle-ci est placée avant les valeurs qui sont meilleures qu'elle dans le rang avec n traces mais son rang dans la prédiction $n+1$ est supérieur à son rang avec n traces. Toutefois, malgré cela, le nouveau rang de la bonne clé dans le classement avec $n+1$ traces est inférieur à son rang avec n traces. Pour bien comprendre ce phénomène, il faut voir que les valeurs avec un rang inférieur à la bonne clé ne sont pas toujours les mêmes pour chaque prédiction. Par conséquent, même si le rang de la bonne clé dans la prédiction $n+1$ est supérieur à son rang dans le classement des hypothèses avec n traces, le nouveau rang de la bonne clé dans le classement avec $n+1$ traces peut être inférieur au rang précédent.

Pour bien interpréter la progression de la bonne clé, il ne faut donc pas regarder son rang dans la nouvelle prédiction par rapport à toutes les autres valeurs possibles mais uniquement par rapport aux valeurs ayant un rang inférieur. Il faut aussi inclure dans ce calcul les potentielles valeurs qui vont passer devant la bonne clé dans le nouveau classement. Pour reprendre l'exemple de la Figure 5.23, le rang de la bonne clé dans la prédiction $n+1$ pourrait être considéré comme 1 étant donné que la bonne clé est positionnée avant toutes les valeurs en bleu dans cette prédiction et qu'aucune autre valeur ne lui passe devant dans le rang avec $n+1$ traces. Si considéré ainsi, le rang de la bonne clé dans chaque prédiction peut varier en fonction du moment où la prédiction est utilisée lors de l'attaque.

Le rang représente une accumulation de probabilité, décrite Section 2.1.5, ce qui veut dire que le rang de la bonne clé peut rester stable même quand les prédictions ajoutées à l'attaque sont mauvaises pour cette clé. C'est un phénomène qui peut être observé facilement si l'ordre utilisé pour réaliser l'attaque utilise directement le rang de la bonne clé dans chaque prédiction. Dans ce cas là, les premières prédictions positionnent la bonne clé en première et les dernières prédictions en dernière. Ceci est illustré par la Figure 5.24 qui représente un exemple d'évolution du rang de la bonne clé avec un ordre

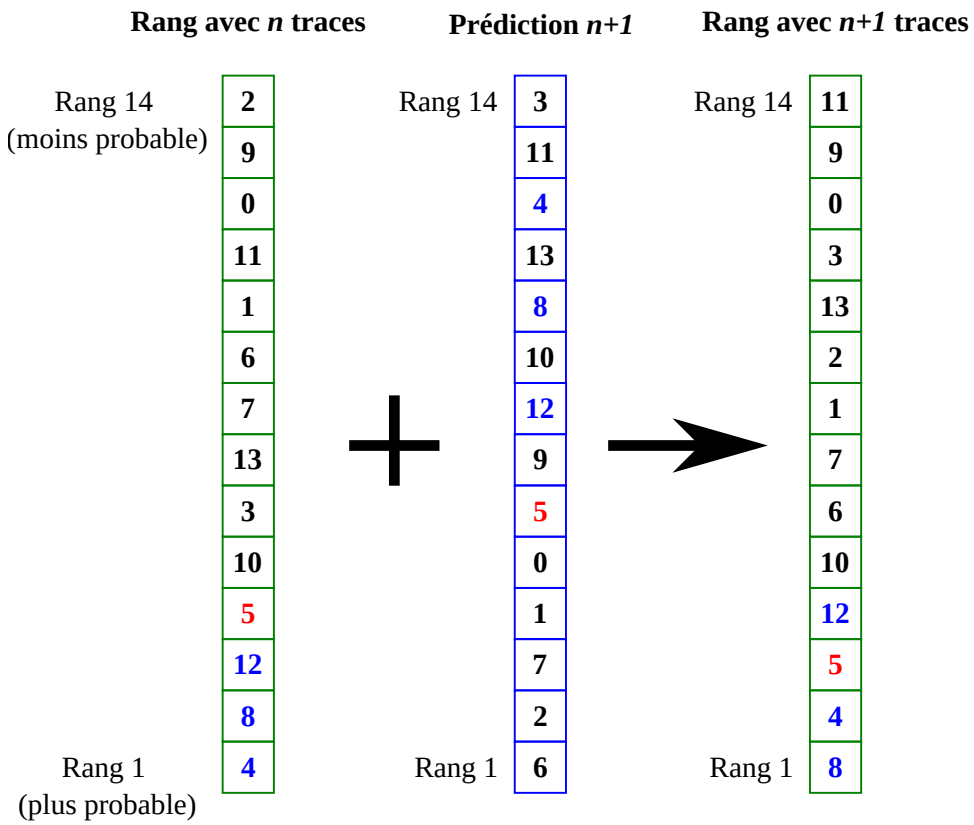


FIGURE 5.23: Schéma illustrant l'évolution du rang de différentes hypothèses de clés après l'ajout d'une nouvelle prédiction. La bonne clé est représentée en rouge et les hypothèses mieux placées sont représentées en bleu.

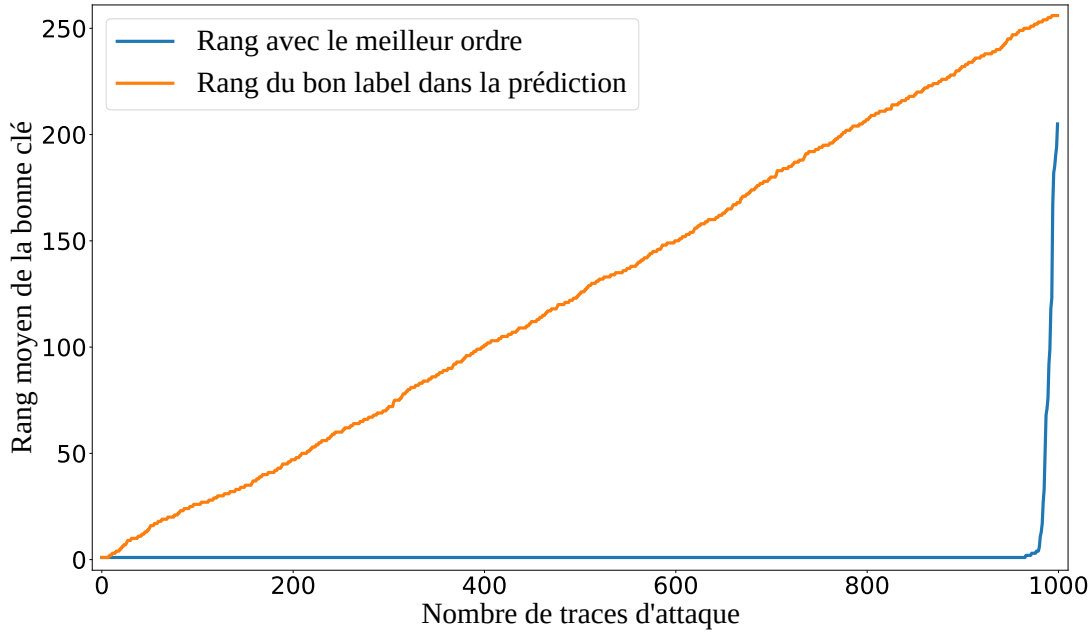


FIGURE 5.24: Exemple d'évolution du rang de la bonne clé en utilisant un ordre déterminé par le rang du bon label dans chaque prédiction. La valeur du rang du bon label est également affichée pour chaque trace.

déterminé par le rang du bon label dans les prédictions. De plus, la valeur du rang du bon label pour chaque trace est représentée au-dessus de chaque point correspondant dans l'évolution du rang de la bonne clé. On constate un rang égal à 1 stable pendant la quasi totalité de l'attaque sauf à la toute fin où le rang diverge vers sa valeur finale autour de 200. Cela est mis en relief par la présence du rang du bon label pour chaque trace qui permet de comparer l'évolution de la position du bon label avec rang actuel de la bonne clé. On constate alors que l'évolution du rang est très peu représentative du rang du bon label puisqu'il faut attendre les 20 dernières traces pour voir le rang de la bonne clé augmenter. À ce moment là, le rang du bon label est autour de 250 et donc très élevé. Cet exemple un peu extrême illustre que la stabilité d'un rang faible n'est pas nécessairement liée aux bons résultats des traces qui sont ajoutées à l'attaque mais repose sur les bons résultats des traces précédentes.

Nous allons maintenant discuter de la fonction de perte qui nous permettra d'entraîner le réseau siamois afin qu'il soit capable de correctement assigner un score à chaque prédiction de l'ensemble d'attaque.

5.3.4 Adaptation de la ranking loss

La *ranking loss* a été introduite par Zaid *et al.* [ZBD+21] comme une nouvelle fonction de perte dédiée à l'entraînement de réseaux de neurones utilisés dans les attaques par canaux auxiliaires. Dans leurs travaux, Zaid *et al.* adressent en partie les critiques faites à la Categorical Cross Entropy (CCE), Section 3.1.1, concernant le fait que cette fonction de perte se concentre sur la valeur associée au bon label et non la position du bon label par rapport aux autres. En partant de la problématique de *Learning to rank*, ils mettent au point une fonction de perte qui va activement prendre en compte la position du bon label dans

les prédictions du réseau pour pénaliser plus fortement lorsque ce label est mal positionné. Nous allons décrire la fonction de perte introduite par Zaid *et al.* avant de voir comment elle peut s'adapter à notre problématique.

Rappel des formules de la ranking loss :

Etant donné c_{k^*} et c_k les classes associées à k^* la bonne clé et k une hypothèse de clé, la probabilité de la relation $c_{k^*} \triangleright c_k$ établissant que c_{k^*} doit être classée plus haut que c_k est donnée par :

$$Pr(c_{k^*} \triangleright c_k) = \frac{1}{1 + e^{-\alpha(s_{N_a}(k^*) - s_{N_a}(k))}}, \tag{5.1}$$

où $s_{N_a}(k)$ représente le score en sortie de réseau de la clé k et α est un paramètre à régler. L'utilisation de la fonction logistique dans ce calcul de probabilité vient du fait que l'on cherche à approximer une fonction indicatrice [QLL10].

Cela mène à la définition de la fonction de perte partielle suivante :

$$l_{N_a}(c_{k^*}, c_k) = -\bar{P}_{k^*,k} \cdot \log_2(P_{k^*,k}) - (1 - \bar{P}_{k^*,k}) \cdot \log_2(1 - P_{k^*,k}), \tag{5.2}$$

avec $P_{k^*,k} = Pr(c_{k^*} \triangleright c_k)$ et $\bar{P}_{k^*,k} = \frac{1}{2}(1 + \text{rel}_{k^*,k})$ où $\text{rel}_{k^*,k} \in \{-1, 0, 1\}$. Cette définition de la fonction de perte partielle est dérivée des travaux de Burges *et al.* [BRL06]. On peut transformer cette expression car $\bar{P}_{k^*,k} = 1$ étant donné que la clé k^* est toujours plus pertinente qu'une autre hypothèse de clé k donc $\text{rel}_{k^*,k} = 1$. On obtient :

$$l_{N_a}(c_{k^*}, c_k) = \log_2\left(1 + e^{-\alpha(s_{N_a}(k^*) - s_{N_a}(k))}\right). \tag{5.3}$$

Cela mène à la définition de la *Ranking Loss* :

$$\mathcal{L}_{RkL}(F_\theta, \mathcal{T}, N_a) = \sum_{\substack{k \in \mathcal{K} \\ k \neq k^*}} \left(\log_2\left(1 + e^{-\alpha(s_{N_a}(k^*) - s_{N_a}(k))}\right)\right). \tag{5.4}$$

5.3.5 Adaptation au problème de classement des prédictions

Nous devons commencer par redéfinir le problème de ranking pour l'appliquer à notre problème de scoring. La relation entre les classes $c_{k^*} \triangleright c_k$ n'est plus importante, à la place nous allons considérer la relation $p_i \triangleright p_j$ qui représente le fait que la prédiction p_i soit plus importante pour l'attaque que la prédiction p_j. Pour cela il nous faut redéfinir la probabilité $Pr(c_{k^*} \triangleright c_k)$.

Dans la ranking loss, la différence $(s_{N_a}(k^*) - s_{N_a}(k))$ est utilisée pour représenter le problème de classement de la bonne clé au sein de chaque prédiction. Elle permet donc d'approximer la fonction indicatrice de ce problème. Pour adapter cette approximation au problème de classement des prédictions, nous avons remplacé la différence de l'Équation 5.1 par $(s_i - s_j)$ où s_i et s_j représentent les scores des prédictions p_i et p_j attribués par le réseau siamois. Cette valeur représente la différence de classement qui doit être faite entre deux prédictions selon le réseau siamois.

Ces modifications mènent à la définition suivante pour la probabilité de la relation $p_i \triangleright p_j$:

$$Pr(p_i \triangleright p_j) = \frac{1}{1 + e^{-\alpha \cdot (s_i - s_j)}}. \tag{5.5}$$

La fonction de perte d'entropie croisée binaire peut ensuite être utilisée étant donné que l'on reste dans un cas de figure binaire : soit la prédiction p_i est meilleure que la prédiction p_j, soit l'inverse. On obtient la fonction de perte partielle :

$$l_{N_a}(p_i, p_j) = -\bar{P}_{p_i,p_j} \cdot \log_2(P_{p_i,p_j}) - (1 - \bar{P}_{p_i,p_j}) \cdot \log_2(1 - P_{p_i,p_j}), \tag{5.6}$$

avec $P_{p_i,p_j} = Pr(p_i \triangleright p_j)$ et $\bar{P}_{p_i,p_j} = \frac{1}{2}(1 + \text{rel}_{p_i,p_j})$ où $\text{rel}_{p_i,p_j} \in \{-1, 0, 1\}$. Contrairement à l'application de Zaid *et al.*, la valeur de rel_{p_i,p_j} n'est plus fixe à 1 étant donné que selon la formation des paires, la prédiction p_i peut être moins importante que la prédiction p_j. Soient r_i et r_j les rangs des bons labels des traces t_i et t_j au sein des prédictions p_i et p_j. Afin de faciliter les calculs, les rangs sont ici calculés en rangeant les hypothèses par ordre croissant de probabilité. Cela signifie que le meilleur rang devient 256 et le moins bon est 1. On obtient alors :

- $\text{rel}_{p_i,p_j} = 1$ quand $r_i > r_j$;
- $\text{rel}_{p_i,p_j} = 0$ quand $r_i = r_j$;
- $\text{rel}_{p_i,p_j} = -1$ quand $r_i < r_j$.

Il faut alors distinguer les trois cas et observer les conséquences au niveau de la fonction de perte.

Cas 1 : $\text{rel}_{p_i,p_j} = 1$

Ce cas est similaire au cas de la ranking loss, on a $r_i > r_j$ et on obtient :

$$l_{N_a}(p_i, p_j) = \log_2\Big(1 + e^{-\alpha \cdot (s_i - s_j)}\Big). \tag{5.7}$$

Il y a trois possibilités dans ce cas :

- $s_i > s_j$: les scores données par le réseau siamois sont en accord avec la relation rel_{p_i,p_j} et on a $l_{N_a}(p_i, p_j) \approx 0$ donc on ne pénalise presque pas ;
- $s_i = s_j$: les scores données par le réseau siamois sont égaux ce qui n'est pas en accord avec la relation rel_{p_i,p_j}. On a alors $l_{N_a}(p_i, p_j) = 1$ et on pénalise légèrement le réseau ;
- $s_i < s_j$: les scores du réseau siamois sont en contradiction avec la relation rel_{p_i,p_j}, $l_{N_a}(p_i, p_j) > 1$ donc on pénalise fortement.

Cas 2 : $\text{rel}_{p_i,p_j} = 0$

Dans ce cas, le classement des deux prédictions devrait être similaire donc on ne veut pas pénaliser le réseau quand les scores qu'il retourne sont proches. On obtient $\bar{P}_{k^*,k} = \frac{1}{2}$

d'où :

$$\begin{aligned}
l_{N_a}(p_i, p_j) &= -\frac{1}{2}\log_2(P_{p_i,p_j}) - \frac{1}{2}\log_2(1 - P_{p_i,p_j}) \\
&= -\frac{1}{2}\log_2(\frac{1}{1+e^{-\alpha\cdot(s_i-s_j)}}) - \frac{1}{2}\log_2(1 - \frac{1}{1+e^{-\alpha\cdot(s_i-s_j)}}) \\
&= -\frac{1}{2}\Big(-\log_2(1+e^{-\alpha\cdot(s_i-s_j)}) \\
&\qquad + \log_2\Big(\frac{e^{-\alpha\cdot(s_i-s_j)}}{1+e^{-\alpha\cdot(s_i-s_j)}}\Big)\Big) \\
&= -\frac{1}{2}\Big(-\log_2(1+e^{-\alpha\cdot(s_i-s_j)}) \\
&\qquad + \log_2\big(e^{-\alpha\cdot(s_i-s_j)}\big) - \log_2\big(1+e^{-\alpha\cdot(s_i-s_j)}\big)\Big) \\
&= -\frac{1}{2}\Big(-2\log_2(1+e^{-\alpha\cdot(s_i-s_j)}) + \log_2\big(e^{-\alpha\cdot(s_i-s_j)}\big)\Big)
\end{aligned}$$

La valeur de $l_{N_a}(p_i, p_j)$ dépend donc de la différence $(s_i - s_j)$:

- Si $s_i > s_j$, alors $l_{N_a}(p_i, p_j) > 1$. On pénalise le réseau puisqu'il a mal prédit les scores des prédictions ;
- Si $s_i = s_j$, alors $l_{N_a}(p_i, p_j) = 1$. Le réseau est légèrement pénalisé mais peu par rapport aux autres situations ;
- Si $s_i < s_j$, alors $l_{N_a}(p_i, p_j) > 1$ et donc on pénalise de nouveau fortement le réseau.

Cas 3 : $\mathbf{rel}_{p_i,p_j} = -1$

Ce dernier cas représente les paires où le bon label de la prédiction p_i a un rang inférieur au bon label de la prédiction p_j soit $r_i < r_j$. On veut donc pénaliser le réseau lorsqu'il donne un score s_i supérieur au score s_j. Cette fois, $\bar{P}_{k^*,k} = 0$ donc :

$$\begin{aligned}
l_{N_a}(p_i, p_j) &= -\log_2(1 - P_{p_i,p_j}) \\
&= -\log_2(1 - \frac{1}{1+e^{-\alpha\cdot(s_i-s_j)}}) \\
&= -\log_2\Big(\frac{e^{-\alpha\cdot(s_i-s_j)}}{1+e^{-\alpha\cdot(s_i-s_j)}}\Big) \\
&= -\log_2\big(e^{-\alpha\cdot(s_i-s_j)}\big) + \log_2\big(1+e^{-\alpha\cdot(s_i-s_j)}\big)
\end{aligned}$$

Dans ce cas là :

- Si $s_i > s_j$, alors $l_{N_a}(p_i, p_j) > 1$. Le réseau se trompe en accordant plus d'importance à la prédiction p_i, il faut le pénaliser ;
- Si $s_i = s_j$, alors $l_{N_a}(p_i, p_j) = 1$ donc le réseau est légèrement pénalisé ;
- Si $s_i < s_j$, alors $l_{N_a}(p_i, p_j) \approx 0$. Le réseau prédit correctement l'ordre des prédictions donc on ne pénalise pas.

5.3.6 Équivalence au cas 1

Dans cette section, nous allons voir comment en modifiant légèrement la fonction de perte, on peut ramener les Cas 2 et 3 au Cas 1. Pour cela, il nous faut rajouter le terme $(r_i - r_j)$ à l'intérieur du calcul de la fonction de perte partielle. L'Équation 5.7 devient donc :

$$l_{N_a}(p_i, p_j) = \log_2\Big(1 + e^{-\alpha \cdot (r_i - r_j) \cdot (s_i - s_j)}\Big). \tag{5.8}$$

De cette façon, la relation rel_{p_i,p_j} est directement intégrée dans le calcul de la fonction de perte.

Cet ajout conserve également les caractéristiques de chacun des trois cas vu précédemment. Si on regarde les conditions de pénalisation, on constate trois cas différents :

- Soit $(r_i - r_j) = 0$ ou $(s_i - s_j) = 0$ d'où $l_{N_a}(p_i, p_j) = 1$, on pénalise légèrement le réseau ;
- Soit $(r_i - r_j) \cdot (s_i - s_j) < 0$ d'où $l_{N_a}(p_i, p_j) > 1$. Ici les signes des différences sont opposés, les scores du réseau siamois ne sont pas en accord avec l'ordre obtenu par le rang donc on pénalise fortement ;
- Soit $(r_i - r_j) \cdot (s_i - s_j) > 0$ d'où $l_{N_a}(p_i, p_j) < 1$, les signes des deux termes sont égaux ce qui indique que les scores prédits par le réseau siamois sont bien ordonnés donc on ne pénalise pas.

Remarque Le cas où $(r_i - r_j) = (s_i - s_j) = 0$ reste un cas marginal, ce n'est donc pas impactant de pénaliser légèrement le réseau dans ce cas là même s'il arrive que le réseau ait correctement prédit l'équivalence des prédictions p_i et p_j.

Conclusion

Maintenant que nous avons défini la fonction de perte partielle, nous pouvons l'utiliser pour définir la fonction de perte liée au score ou *Scoring Loss* de la manière suivante.

Définition 5.3.1. Soient F_θ un modèle de paramètres θ et $\mathcal{P}$ l'ensemble des prédictions des traces de $\mathcal{T}$, la fonction de perte *Scoring Loss* est définie de la manière suivante :

$$\mathcal{L}_{ScL}(F_\theta, \mathcal{P}) = \sum_{\substack{p_i, p_j \in \mathcal{P} \\ i \neq j}} \Big(\log_2\Big(1 + e^{-\alpha \cdot (r_i - r_j) \cdot (s_i - s_j)}\Big)\Big), \tag{5.9}$$

où r_i représente le rang du bon label de la trace i au sein de la prédiction p_i et s_i est le score attribué à la prédiction p_i par le réseau siamois.

Il ne reste plus qu'à définir l'architecture du réseau siamois et de l'entraîner afin de voir s'il est capable de déterminer un ordre intéressant pour les attaques.

5.4 Expériences sur l'application de la Scoring Loss

La première étape pour tester la Scoring Loss est de définir l'architecture du réseau utilisé pour obtenir les scores. Ce réseau est ensuite dupliqué pour former le réseau siamois

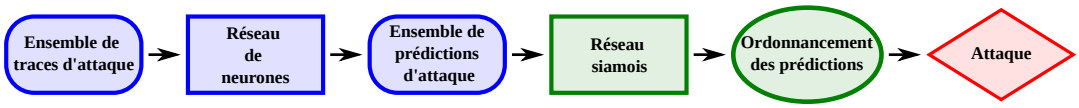


FIGURE 5.25: Schéma illustrant la nouvelle méthode d'attaque des attaques par canaux auxiliaires utilisant un réseau siamois pour déterminer un ordonnancement des prédictions.

entraîné. Une fois l'entraînement terminé, le réseau extrait du réseau siamois est prêt à être utilisé pour calculer le score associé aux traces d'attaque.

Description de la méthode d'attaque La Figure 5.25 illustre la nouvelle méthode d'attaque incluant l'utilisation du réseau siamois. Les premières étapes restent les mêmes jusqu'à la récupération des prédictions du réseau de neurones. Les prédictions des traces de validation sont utilisées pour entraîner le réseau siamois à attribuer un score aux prédictions. Ce réseau sert ensuite à établir un ordonnancement des prédictions d'attaque basé sur les scores de ces prédictions. Cet ordonnancement est conservé lors de la phase d'attaque pour améliorer le rang minimum atteint par la bonne clé.

Description du réseau Étant donné que le problème n'est en apparence plus aussi complexe que la prédiction des probabilités faite par le premier réseau, nous pouvons nous rabattre sur une architecture plus simple. De plus, étant donné que le nouveau réseau se base sur les prédictions du précédent pour déterminer un score, les valeurs d'entrée n'ont plus réellement de lien entre elles, il n'est donc plus nécessaire d'utiliser un réseau convolutionnel. Nous avons décidé pour ces raisons de nous servir d'un réseau MLP (Multi-Layer Perceptron), abordé Section 2.3.1, composé de deux couches de 1024 neurones. Ces couches utilisent une fonction d'activation ReLU et la sortie du réseau est composée d'un neurone qui se charge de calculer le score de la prédiction avec une activation linéaire. L'architecture de ce réseau peut être trouvée dans l'Annexe A Tableau A.5. Le réseau utilise l'optimiseur *Adam* [KB17] avec un learning rate de 0.001. La Scoring Loss utilise une valeur de base de $\alpha = 0.1$. L'impact de la valeur de α sur l'entraînement est discuté par Zaid *et al.* dans leurs travaux [ZBD+21] nous ne reviendrons donc pas dessus ici.

Description de l'entraînement L'entraînement du réseau se déroule de la manière suivante. Les prédictions des traces de validation sont utilisées comme données d'entraînement lorsqu'elles sont en nombre suffisant pour être exploitées. Si ce n'est pas le cas, les prédictions des traces d'entraînement sont utilisées. Il n'est pas clairement défini quel est le meilleur ensemble à utiliser pour l'apprentissage du réseau siamois. D'une part, les prédictions des traces de validation sont plus proches des prédictions d'attaque ce qui peut aider à obtenir une meilleure généralisation. D'autre part, utiliser les prédictions des traces d'entraînement peut permettre d'apprendre au réseau à distinguer les bonnes prédictions des mauvaises étant donné que presque toutes les prédictions d'entraînement peuvent être considérées comme bonnes. Cela vient du fait que le réseau utilisé pour obtenir les prédictions est capable de mieux prédire les traces d'entraînement que celles de validation.

Des paires de prédictions sont ensuite formées aléatoirement pour être fournies au réseau siamois durant l'apprentissage. Le nombre de paires utilisées lors de l'entraînement va varier selon l'ensemble de données et la quantité de prédictions disponibles. Chaque paire est fournie au réseau siamois qui va attribuer un score à chaque prédiction avant de comparer ces scores à l'aide d'une différence. Le signe de cette différence indique quelle prédiction est jugée meilleure par le réseau. La fonction de perte pénalise ensuite le réseau en fonction de sa façon d'ordonner la paire fournie.

Nous allons maintenant discuter des résultats obtenus à partir de l'utilisation des réseaux de scoring sur les différents ensembles de données vu précédemment.

5.4.1 ASCAD clé fixe

Le premier ensemble que nous allons voir est l'ensemble ASCAD clé fixe avec le réseau CNN_{best} entraîné pendant 75 epochs sur l'ensemble synchronisé Desync0. Les figures de cette section représentant les évolutions du rang moyen de la bonne clé au cours des attaques sont obtenues en réalisant des attaques utilisant des traces choisies aléatoirement parmi les 10 000 traces des ensembles d'attaques. Les attaques utilisant 1000 traces sont répétées 900 fois afin de lisser la moyenne et les attaques utilisant 5000 traces sont répétées 100 fois.

Desync0 La Figure 5.26 représente les évolutions des rangs moyens de la bonne clé pour des attaques réalisées sur Desync0. Ces attaques utilisent soit un ordonnancement basé sur la valeur du minimum des prédictions soit un ordonnancement basé sur les scores des prédictions. On peut voir que dans le cas d'attaques sur des traces synchronisées, le fait d'ordonner les prédictions à partir du minimum ou du score est équivalent. L'attaque réussit au bout des 1000 traces ce qui ne laissent pas réellement de place pour une amélioration en utilisant des ordonnancements. On remarque toutefois que la convergence du rang avec des prédictions ordonnées avec le score est similaire à celle qui se base sur le minimum, elle est donc plus rapide que la convergence d'attaques utilisant un ordonnancement aléatoire. Cela montre que le réseau est bien capable d'apprendre à ordonner les traces en leur attribuant un score.

Desync50 Les attaques suivantes utilisent toujours le réseau CNN_{best} décrit précédemment mais ciblent cette fois l'ensemble Desync50 pour augmenter la difficulté des attaques et voir l'impact des ordonnancements. Le réseau siamois utilisé pour obtenir le score est le même mais la valeur du paramètre α de la Scoring loss est modifiée à $\alpha = 0.001$ afin d'avoir un bon entraînement.

La Figure 5.27 illustre les évolutions des rangs de la bonne clé pour des attaques utilisant 1000 ou 5000 traces et des ordonnancements obtenus à partir du minimum et du score. Les ordonnancements basées sur le minimum sont inversés comme dans la Section 5.2.2 pour parvenir à ces résultats. Lors de l'utilisation de 1000 traces d'attaque, le rang final est de 35 mais le rang minimal atteint en utilisant un ordonnancement basé sur le minimum est de 21 et celui en utilisant un ordonnancement basé sur le score est de 16. On constate donc que l'ordonnancement basé sur le score est plus efficace pour classer les pré-

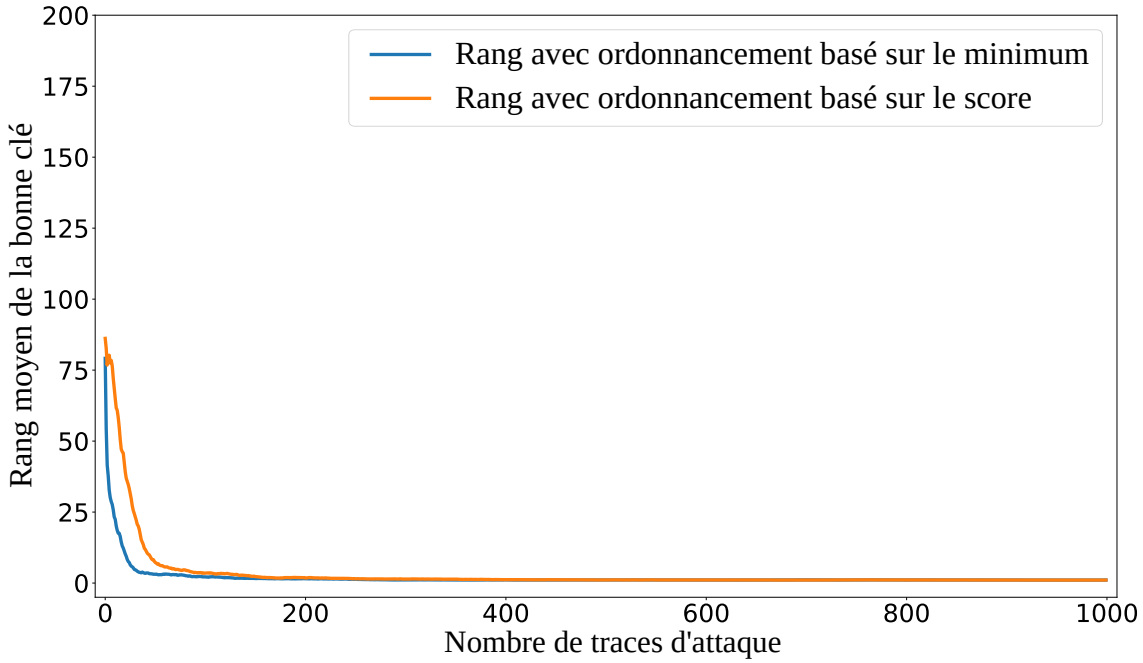
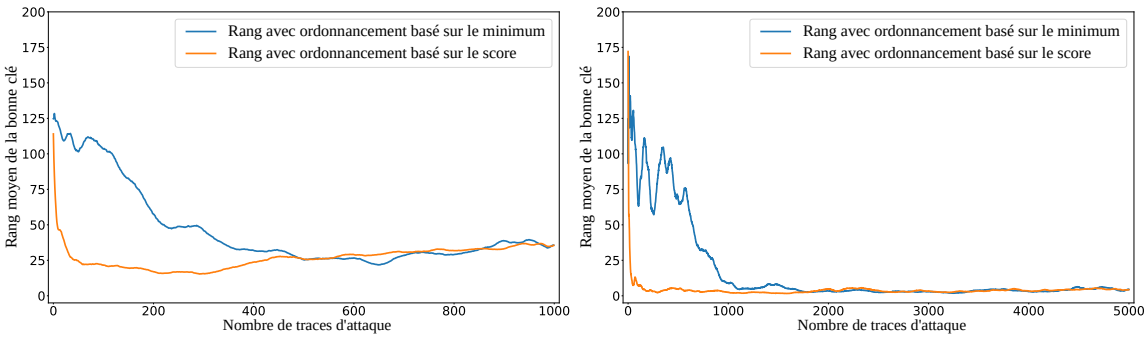


FIGURE 5.26: Évolution du rang moyen de la bonne clé pour des attaques utilisant un ordre normal déterminé par le minimum et un ordre déterminé par le score sur l'ensemble de données ASCAD clé fixe Desync0.

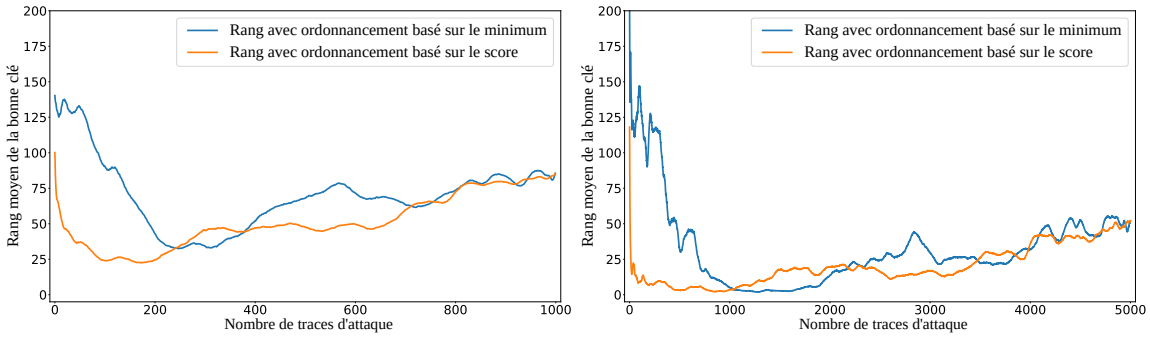
dictions des traces d'attaques. Cela se confirme également sur les attaques utilisant 5000 traces même si de manière moins prononcée. Pour ces attaques, le rang final est de 4.5 en moyenne là où le minimum du rang en utilisant un ordonnancement basé sur le minimum est de 1.8 et celui utilisant un ordonnancement basé sur le score est de 1.6. Cette différence minime s'explique notamment par le fait que les attaques sont presque réussies dans les deux cas ce qui laisse peu de place à une amélioration.

Desync100 Les attaques se concentrent maintenant sur l'ensemble de données Desync100 issu de ASCAD clé fixe. L'architecture du réseau siamois change de nouveau : le nombre d'epochs d'entraînement baisse à 5 et la valeur de α devient 0.0001. Cela permet d'obtenir de bons résultats via l'utilisation des scores du réseau.



(A) 1000 traces d'attaque (B) 5000 traces d'attaque

FIGURE 5.27: Évolution du rang moyen de la bonne clé pour des attaques utilisant un ordre inversé déterminé par le minimum et un ordre déterminé par le score sur l'ensemble de données ASCAD clé fixe Desync50.



(A) 1000 traces d'attaque (B) 5000 traces d'attaque

FIGURE 5.28: Évolution du rang moyen de la bonne clé pour des attaques utilisant un ordre inversé déterminé par le minimum et un ordre déterminé par le score sur l'ensemble de données ASCAD clé fixe Desync100.

La Figure 5.28 illustre ces résultats. Elle représente les évolutions des rangs moyens de la bonne clé avec des ordonnancements des prédictions basés sur le minimum et sur le score. Pour les attaques utilisant 1000 traces, le minimum du rang avec un ordonnancement basé sur le score est de 23 là où le minimum du rang utilisant un ordonnancement basé sur le minimum est de 33 et le rang final de 85. Dans le cas des attaques de 5000 traces, le rang final vaut 51 mais le minimum des rangs utilisant des ordonnancements est de 2. Sur ces figures, on voit que le rang des attaques avec un ordonnancement basé sur le score converge plus rapidement que lorsque les prédictions sont ordonnées en fonction de leurs minimums. Cela indique que les traces intéressantes pour l'attaque sont placées plus tôt dans l'ordonnancement basé sur le score que dans l'ordonnancement basé sur le minimum. Un autre avantage notable de l'utilisation du score pour ordonner les prédictions est qu'il n'y a plus de problème pour déterminer s'il faut inverser ou non l'ordonnancement contrairement à l'utilisation du minimum.

Au delà du gain potentiel apporté par cette nouvelle méthode d'attaque, ces résultats confirment l'intérêt de l'utilisation d'un réseau de neurones pour attribuer un score aux prédictions et ainsi les ordonner. Nous allons maintenant voir l'application du réseau siamois à l'ensemble de données ASCAD clé variable.

5.4.2 ASCAD clé variable

Comme lors des tests de la Section 5.2.2, nous allons utiliser la base de données ASCAD clé variable et ces sous-ensembles Desync0, Desync50 et Desync100 pour effectuer des tests sur différents niveaux de difficulté. L'ensemble d'entraînement de la base de données ASCAD clé variable est composé de 200 000 traces ce qui permet d'en utiliser 50 000 pour entraîner le réseau CNN_{best} qui va faire les prédictions et d'en conserver 150 000 pour l'ensemble de validation. Cela n'était pas possible dans le cas de ASCAD clé fixe étant donné que l'ensemble d'entraînement ne contient que 50 000 traces. Lors des entraînements des réseaux siamois, nous avons donc pu utiliser les prédictions des traces de l'ensemble de validation comme prédictions d'entraînement. Les 100 000 paires d'entraînement ont été générées à partir de 50 000 prédictions de l'ensemble de validation. Les réseaux s'entraînent pour 10 epochs avec une valeur $\alpha = 0.1$ pour la fonction de perte. De

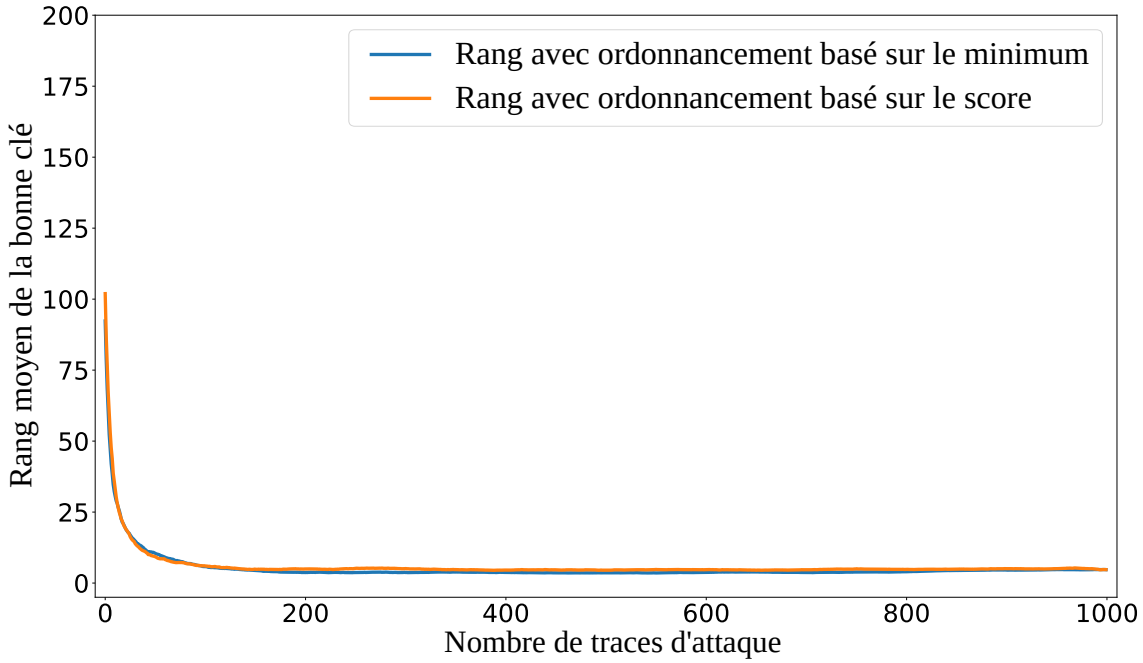


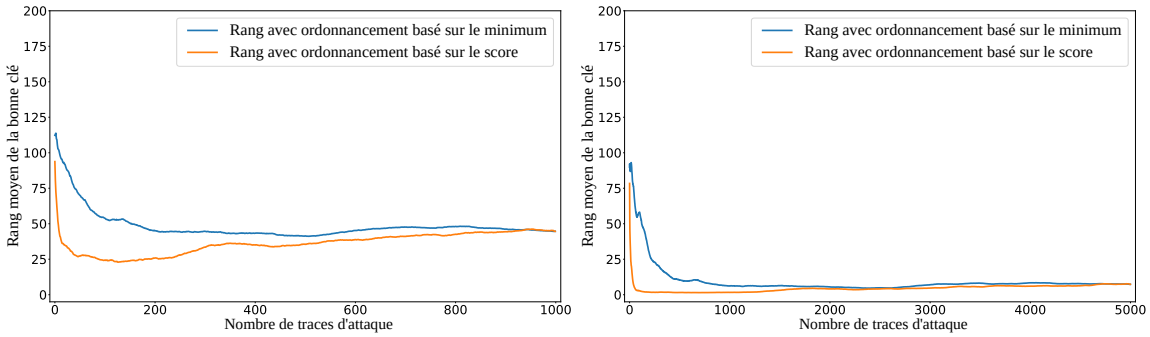
FIGURE 5.29: Évolution du rang moyen de la bonne clé pour des attaques utilisant un ordre normal déterminé par le minimum et un ordre déterminé par le score sur l'ensemble de données ASCAD clé variable Desync0.

plus, l'ensemble d'attaque est également plus grand avec 100 000 traces d'attaque ce qui permet de calculer les moyennes des rangs à partir de 900 attaques à la fois pour celles utilisant 1000 traces et 5000 traces.

Desync0 La Figure 5.29 illustre l'évolution des rangs utilisant le minimum des prédictions ou leurs scores pour les ordonner. Encore une fois, l'étude de l'évolution du rang avec des ordonnancements basés sur le minimum et le score pour l'ensemble synchronisé Desync0 nous permet de voir que cette méthode d'attaque n'apporte que peu d'amélioration lorsque les attaques réussissent déjà. Ici le rang final est d'environ 3 et les rangs minimaux atteints lors des attaques sont d'environ 2 pour chaque des ordres. On peut donc seulement constater que la convergence des rangs est très rapide mais qu'aucun ordonnancement ne parvient à faire atteindre le rang 1.

Nous allons maintenant voir les résultats des attaques sur les traces désynchronisées de Desync50.

Desync50 Les résultats des ordonnancements des prédictions sur Desync50 sont illustrés sur la Figure 5.30. La Figure 5.30a montre les évolutions du rang lors de l'utilisation de 1000 traces d'attaque. Le rang final moyen de ces attaques est de 45 et les minimums atteints sont de 41 et 24 pour respectivement l'ordonnancement inversé basé sur le minimum et l'ordonnancement basé sur le score. On constate que le rang converge plus rapidement vers une valeur plus basse lorsque l'on ordonne les traces en utilisant les scores des prédictions. L'ordonnancement basé sur le minimum lui améliore à peine la valeur du rang final.



(A) 1000 traces d'attaque (B) 5000 traces d'attaque

FIGURE 5.30: Évolution du rang moyen de la bonne clé pour des attaques utilisant un ordre inversé déterminé par le minimum et un ordre déterminé par le score sur l'ensemble de données ASCAD clé variable Desync50.

Ces résultats sont confirmés par les attaques à partir 5000 traces dont les résultats sont illustrés Figure 5.30b. On y voit l'évolution du rang avec un ordonnancement basé sur le score qui atteint un rang d'environ 1.5 là où le rang minimal avec l'ordonnancement inversé basé sur le minimum est 4.5 et le rang final est 7.5. L'utilisation de l'ordonnancement à partir du score permet donc une amélioration nette du rang quelque soit le nombre de traces utilisées dans l'attaque. Cela montre que les scores permettent bien d'identifier les bonnes prédictions des mauvaises.

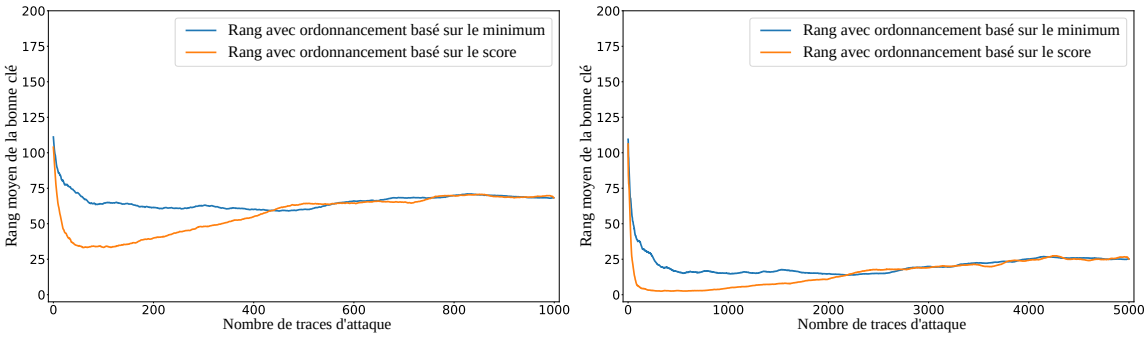
Nous allons maintenant regarder les résultats sur l'ensemble Desyn100 pour compléter cette étude.

Desync100 La Figure 5.31 montre les évolutions des rangs de la bonne clé avec des ordonnancements basés sur le minimum et le score. Dans le cas où 1000 traces d'attaque sont utilisées, le rang minimal en ordonnant avec le score est de 34, celui en ordonnant avec le minimum et en inversant les positions des prédictions est de 59 et le rang final est de 68. L'amélioration obtenue dans ce cas est similaire au cas de Desync50 et permet de réduire de moitié le rang de la bonne clé si le bon ordre est utilisé. Cette amélioration est encore meilleure lorsqu'on utilise 5000 traces d'attaque. On obtient un rang minimal avec l'ordonnancement basé sur le score de 3 là où le rang final est de 25.

Les diverses expériences faites sur les ensembles de données ASCAD montrent clairement l'avantage de l'utilisation du score pour ordonner les prédictions. D'une part, le score donné par le réseau siamois ne présente pas les inconvénients du minimum des prédictions présentés dans la Section 5.2.2. D'autre part, les ordonnancements basés sur le score permettent des améliorations significatives des résultats de cette nouvelle méthode d'attaque par rapport au minimum.

5.5 Conclusion du chapitre

Dans ce chapitre, nous avons abordé une toute nouvelle façon d'exploiter les traces lors des attaques par canaux auxiliaires. Plutôt que de seulement considérer les traces comme toutes équivalentes et choisir leur ordre aléatoirement, nous avons essayé de déterminer



(A) 1000 traces d'attaque (B) 5000 traces d'attaque

FIGURE 5.31: Évolution du rang moyen de la bonne clé pour des attaques utilisant un ordre inversé déterminé par le minimum et un ordre déterminé par le score sur l'ensemble de données ASCAD clé variable Desync100.

des méthodes pour les ordonner. Ces ordonnancements doivent indiquer quelles traces sont plus intéressantes à utiliser pour faire converger le rang plus rapidement. Si un tel ordonnancement ne permet pas d'améliorer le résultat final d'une attaque, il peut toutefois arriver à améliorer le rang minimal atteint par la bonne clé en se concentrant sur les meilleures traces. Ainsi, s'il est possible de déterminer un tel ordonnancement avec le nombre de traces à utiliser lors de l'attaque, le résultat de celle-ci peut être grandement amélioré. Cela peut même aller jusqu'à réussir une attaque qui en apparence ne converge pas vers le rang 1.

Nous avons, dans un premier temps, testé cette nouvelle approche de manière empirique sur les prédictions de réseaux de neurones. En effet, en nous basant sur des méthodes aidant à distinguer les bonnes des mauvaises prédictions, nous pouvons observer une amélioration du rang minimal atteint lors des attaques. Ces premiers résultats se basent sur certaines valeurs particulières des prédictions comme la médiane ou le minimum. Étant donné que les prédictions simulent une distribution de probabilité, il est possible d'utiliser ces valeurs pour déterminer un certain degré de confiance du réseau dans la prédiction. Après avoir présenté les résultats sur les bases de données ASCAD clé fixe et ASCAD clé variable et donné des débuts d'explication sur l'origine de ce phénomène, nous avons poursuivi les tests sur un autre type d'attaque : l'attaque template. Il est apparu qu'un phénomène similaire se produisait avec ces attaques même si il n'était pas consistant. En effet, selon le nombre de traces utilisées pour la création du template et le nombre de points d'intérêt considérés, l'ordonnancement à l'aide du minimum avait des effets plus ou moins forts sur la convergence du rang.

Après avoir fait le constat qu'aucune des valeurs remarquables essayées ne parvenaient à obtenir des résultats consistants, nous avons approché le problème du point de vue du machine learning. Nous avons mis au point un réseau de neurones ayant pour but d'attribuer un score à chaque prédiction afin de pouvoir par la suite les ordonner lors des attaques. Un réseau siamois a donc été choisi pour réaliser cette tâche étant donné que ce type de réseau est performant dans les problèmes *Learning to rank* qui consistent à classer des exemples selon leur pertinence. Pour entraîner efficacement ce réseau, nous avons adapté la fonction de perte *Ranking Loss*. Cela a permis d'entraîner le réseau de manière

à ce qu'il attribue des scores élevés aux traces importantes. La détermination de l'importance d'une trace lors d'une attaque a été discutée et plusieurs méthodes ont été proposées. Nous avons choisi de nous concentrer sur le rang du bon label au sein des prédictions comme indicateur de l'importance d'une trace et nous l'avons inclus dans le calcul de la nouvelle fonction de perte : la *Scoring Loss*. Une fois cette fonction bien définie, des réseaux ont été entraînés pour apprendre à ordonner les prédictions obtenues à partir de différents réseaux et différents ensembles de données. En utilisant les scores ainsi déterminés, nous avons pu constater une amélioration globale des résultats par rapport à l'utilisation de valeurs comme le minimum. Même si les classements générés ne sont pas parfaits, ces résultats confirment le fonctionnement de la Scoring Loss et valident l'utilisation d'un réseau siamois afin d'apprendre à classer les traces. Plus particulièrement, les améliorations obtenues par rapport à des attaques classiques montrent que cette nouvelle méthode d'attaque est pertinente et qu'elle doit être explorée plus en profondeur.

Pour conclure, nous avons introduit et expérimenté une nouvelle façon de faire des attaques par canaux auxiliaires. Cette méthode a le potentiel de faire réussir des attaques qui auparavant n'étaient pas capables de retrouver la clé. La nouvelle perspective qu'offre cette méthode peut remettre en question certaines évaluations de sécurité pour lesquelles l'ordonnancement des prédictions devrait devenir une nécessité. Il nous semble donc important qu'elle soit étudiée plus en profondeur afin d'être mieux caractérisée et comprise. La compréhension de cette méthode peut notamment passer par une étude des réseaux utilisés pour effectuer le classement étant donné qu'ils contiennent les informations nécessaires pour distinguer les bonnes des mauvaises prédictions.

Conclusion

Dans ce manuscrit nous avons étudié de multiples aspects des attaques par canaux auxiliaires basées sur les techniques d'apprentissage profond. Ce sujet, relativement nouveau au début de cette thèse, a pris beaucoup d'ampleur au cours de ces dernières années. Les réseaux de neurones et plus globalement les techniques d'apprentissage automatique montrent leur efficacité dans de plus en plus de domaines de nos jours y compris la sécurité. Il était donc naturel d'appliquer ces techniques aux attaques par canaux auxiliaires d'autant plus que les problématiques abordées dans les deux domaines sont similaires. Toutefois, s'il est rapidement apparu que l'efficacité des réseaux pour ce type d'attaque était indéniable, il était clair qu'il existait un problème de compréhension des mécanismes des réseaux. En sécurité peut être encore plus que dans d'autres domaines, la compréhension des moyens utilisés pour les attaques est essentielle afin de pouvoir préparer des contre-mesures efficaces et de se protéger contre ces attaques. Ces travaux de thèse s'inscrivent dans l'étude et la compréhension des réseaux de neurones utilisés pour les attaques par canaux auxiliaires.

Contributions

Les contributions apportées par ces travaux de thèse sont multiples et interviennent sur différents aspects des réseaux de neurones et des attaques.

Premièrement, au cours du Chapitre 3, nous nous sommes concentrés sur la compréhension des réseaux de neurones utilisés pour les attaques par canaux auxiliaires. Nous avons commencé par étudier les réseaux de neurones et leur apprentissage du point de vue des attaques par canaux auxiliaires ce qui nous a permis de mettre au point une nouvelle métrique d'apprentissage. Ce problème était important puisqu'il a rapidement été montré que les métriques classiques de machine learning n'étaient pas représentatives des performances des réseaux en termes d'attaques par canaux auxiliaires. Cette nouvelle métrique nous a permis de mettre en avant l'importance de bien monitorer l'apprentissage des réseaux, même dans le contexte de ces attaques, afin d'éviter le phénomène d'*overfitting* qui nuit gravement aux performances. Pour cela, nous avons discuté de la technique d'*early stopping*, qui consiste à arrêter l'entraînement avant que la généralisation du réseau ne baisse, et de comment la mettre en place grâce à notre métrique. La combinaison de cette technique avec une métrique appropriée permet un gain significatif de performance par rapport à un entraînement qui ne les utiliserait pas. Ces travaux ont été présentés lors du workshop WRAC'H 2019 puis ont été publiés et présentés à la conférence COSADE 2020 [RZC+20].

Dans le Chapitre 4, nous avons poursuivi l'étude du réseau utilisé précédemment en appliquant des techniques de régularisation pour améliorer les performances du réseau. En effet, lors de l'application de notre métrique à l'entraînement du réseau, nous avons pu constater que, même si l'early stopping permettait de réduire le phénomène d'overfitting, l'architecture complexe du réseau entraînait malgré tout un sur-apprentissage. Cela a également mené à l'étude de techniques permettant à la fois de réduire ce phénomène et d'améliorer les performances du réseau de neurones. En appliquant tout d'abord la technique de *batch normalization* puis les techniques de régularisation *weight decay* et *dropout*, il a été possible d'améliorer grandement les performances du réseau sans toucher à son architecture. Nous avons pu constater une baisse significative des effets de l'overfitting liée à l'utilisation de ces techniques grâce à la métrique introduite précédemment. Cela a permis d'atteindre des niveaux de performances égaux aux meilleurs réseaux de neurones sur les ensembles de données publiques de l'ANSSI. Les résultats de ce travail ont fait l'objet d'une publication dans le journal ACM JETC [RBHG21].

Le Chapitre 5 constitue un travail exploratoire d'une nouvelle façon de réaliser les attaques par canaux auxiliaires. Cette nouvelle méthode a été mise au point après avoir constaté que le réseau de neurones était plus robuste aux perturbations, comme l'ajout de bruit, ici par phénomène de désynchronisation, sur certaines traces spécifiques. Cela a donc posé la question de la provenance de cette robustesse et également de savoir si ces traces pouvaient être détectées parmi l'ensemble des traces d'attaque. Pour effectuer cette détection, nous avons cherché un moyen de classer les traces de consommation de courant en fonction de leurs prédictions afin de prioriser l'utilisation des meilleures prédictions. Différents tests empiriques ont été menés à la recherche d'un moyen de classer les prédictions et plusieurs pistes ont donné des résultats satisfaisants mais manquant de généralisation. Nous nous sommes tournés vers l'apprentissage automatique pour répondre à ce problème équivalent aux problèmes de *Learning to rank* communs dans ce domaine. Pour effectuer le classement des prédictions, nous avons utilisé un réseau siamois permettant d'entraîner un réseau à comparer deux entrées similaires et déterminer une préférence entre les deux. Nous avons également mis au point une fonction de perte permettant d'entraîner le réseau siamois : la *Scoring Loss*. Cette fonction est adaptée d'une autre fonction de perte, la *Ranking Loss*, également basée sur l'approche *Learning to Rank* et utilisée pour entraîner de manière plus efficace les réseaux de neurones pour les attaques par canaux auxiliaires. La nouvelle méthode d'attaque utilisant le réseau siamois a permis de montrer qu'il était possible d'obtenir des rangs pour la bonne clé inférieurs au rang final en utilisant un nombre plus restreint de traces d'attaque. Nous avons ainsi établi une notion de pire cas qui correspondrait à une attaque classique et les autres cas correspondraient aux attaques utilisant des ordonnancements des prédictions. Cela a le potentiel de remettre en question certaines évaluations de sécurité jugées jusqu'à présent sûres via les méthodes d'attaque normales. Un article basé sur ces travaux est en cours de rédaction.

Ouvertures et perspectives

L'application du machine learning aux attaques par canaux auxiliaires reste un sujet nouveau et si de nombreux travaux ont été menés sur ce sujet, il reste encore de nombreuses questions ouvertes. Nous nous concentrerons ici sur les questions qui découlent à la fois des travaux décrits dans ce manuscrit et de ceux qui n'ont pas pu être approfondis suffisamment pour y apparaître.

Premièrement, il nous semble intéressant de prendre le temps de revenir sur les diverses techniques et métriques utilisées dans les articles étudiant cette application du machine learning, y compris les techniques abordées dans ce manuscrit. En effet, de plus en plus de travaux apparaissent et utilisent une large variété de techniques provenant d'autres domaines du machine learning pour les appliquer aux attaques par canaux auxiliaires. S'il est intéressant de voir les résultats de ces applications, cela crée également une saturation de la recherche qui rend difficile la compréhension des effets de chaque technique. Il serait donc pertinent selon nous d'essayer de prendre du recul par rapport aux techniques utilisées afin de clairement établir l'apport de chaque technique et le meilleur contexte pour leurs utilisations. De cette façon, cela faciliterait le travail de construction et d'optimisation des réseaux de neurones et donc les évaluations de sécurité reposeraient sur des bases plus solides.

Une deuxième piste qui nous semble importante est l'étude de la nouvelle méthode d'attaque décrite dans ce manuscrit se basant sur l'ordonnancement des prédictions afin de réduire le rang de la bonne clé en utilisant un nombre restreint de traces. Il reste plusieurs points à explorer concernant l'utilisation du réseau siamois qui détermine un score pour chaque prédiction afin de pouvoir les ordonner. Un premier concerne les exemples d'entraînement de ce réseau : faut-il utiliser les prédictions du réseau de base sur les traces d'entraînement ou sur les traces de validation et quel impact cela a-t-il sur les scores attribués par le réseau. Un autre point intéressant concerne une étude sur le nombre de paires utilisées lors de l'entraînement et également de voir si la formation de paires composées de prédictions proches aide le réseau siamois dans son processus d'attribution des scores. Pour en finir sur les axes de recherche possibles issus de cette nouvelle méthode, il nous semble intéressant de considérer l'inclusion de la trace de consommation de courant en entrée du réseau siamois en plus de la prédiction. Cela pourrait donner au réseau des informations supplémentaires qui l'aideraient dans la détermination du score associé au couple (trace, prédiction).

Enfin, un dernier sujet qui semble prometteur est celui des exemples adversariaux [KGB17, GSS14, KW20], c'est-à-dire les exemples modifiés afin de tromper le réseau de neurones. Nous avons eu l'occasion d'expérimenter avec ces exemples mais sans avoir le temps d'étudier en profondeur les impacts qu'ils pourraient avoir sur les réseaux de neurones utilisés pour les attaques par canaux auxiliaires. Toutefois, il est apparu que ces exemples représentent une piste à la fois d'amélioration possible des attaques et de contre-mesure potentielle. En effet, l'utilisation d'exemple adversariaux dans l'ensemble d'entraînement d'un réseau lui permet d'acquérir une certaine robustesse face à des types de transformations donnés qui peuvent être appliquées sur les exemples. D'un autre côté,

il est aussi possible de créer des perturbations spécifiques qui, une fois ajoutées aux traces, rendent les prédictions du réseau inefficaces pour retrouver la clé. Il reste la question de savoir si ces perturbations pourront s'appliquer uniquement sur un réseau spécifique ou s'il est possible de les généraliser à tout un type de réseaux de neurones. L'implémentation matérielle de ce type de contre-mesure reste également un aspect à travailler si les exemples adversariaux arrivent à être développés comme contre-mesures efficaces.

Communications et publications

Publication dans un journal international

1 Damien Robissout, Lilian Bossuet, Amaury Habrard, Vincent Grosso. **"Improving deep learning networks for profiled side-channel analysis using performance improvement techniques."** *J. Emerg. Technol. Comput. Syst.*, July 2021 [RBHG21].

Conférence internationale avec comité de lecture

2 Damien Robissout, Gabriel Zaid, Lilian Bossuet, Amaury Habrard. **"Online performance evaluation of deep learning networks for profiled side-channel analysis."** *International Workshop on Constructive Side-Channel Analysis and Secure Design*, Oct. 2020 [RZC+20].

Présentations à un congrès international sans actes

3 Damien Robissout. **"Improved Deep Learning Side-Channel Attacks using Normalization Layers."** *Cryptographic Architectures embedded in logic devices (CryptArchi 2019)*, Prague, Czech Republic, June 2019.

Présentations à des congrès nationaux sans actes

4 Damien Robissout. **"Improved Deep Learning Side-Channel Attacks using a new performance metric, Normalization Layers and Regularization."** *Workshop on Randomness and Arithmetics for Cryptography on Hardware (WRAC'H 2019)*, Roscoff, France, April 2019.

5 Damien Robissout. **"Online performance evaluation of deep learning networks for profiled side-channel analysis."** *Journée Machine Learning et SCA des GDR Sécurité Information et SoC2*, Online, Dec. 2020.

Présentation dans une école d'été

6 Damien Robissout, Gabriel Zaid. **"Online Class and Hands on Session : Using deep learning techniques against threads by side channel analysis."** *Arqus 2021 Summer School in Cybersecurity*, Online, Sept. 2021.

Séminaire

7 Damien Robissout. **"Online performance evaluation and improvement of deep learning networks for profiled side-channel analysis."** *Instituto Madrileño de Estudios Avanzados (IMDEA)*, Madrid, Spain, May 2021.

Posters

8 Damien Robissout. **"Online performance evaluation of deep learning networks for profiled side-channel analysis."** *Évaluation HCERES, Laboratoire Hubert Curien*, Saint-Étienne, France, Feb 2020.

9 Damien Robissout. **"Online performance evaluation of deep learning networks for profiled side-channel analysis."** *Journée de la recherche de l'école doctorale EDSIS*, Saint-Étienne, France, June 2020.

Présentations de vulgarisation scientifique

10 Damien Robissout, Brice Colombier. **"Cyber-sécurité & Intelligence Artificielle."** *Pint of Science 2019*, Saint-Étienne, France, May 2019.

11 Damien Robissout. **"Sécurité & Chiffrement"** *Fête de la Science 2021*, Saint-Étienne, France, Oct. 2021.

Annexe A

Networks

TABLE A.1: Hyperparamètres du réseau CNN_{best} [BPS+19].

Type de couche	Hyperparamètres
Taille de l'entrée	1400
Convolution 1D	Nombre de filtres = 64, Taille des filtres = 11, Padding = Same, Activation = ReLU
Average Pooling	Taille du Pooling = 2
Convolution 1D	Nombre de filtres = 128, Taille des filtres = 11, Padding = Same, Activation = ReLU
Average Pooling	Taille du Pooling = 2
Convolution 1D	Nombre de filtres = 256, Taille des filtres = 11, Padding = Same, Activation = ReLU
Average Pooling	Taille du Pooling = 2
Convolution 1D	Nombre de filtres = 512, Taille des filtres = 11, Padding = Same, Activation = ReLU
Average Pooling	Taille du Pooling = 2
Convolution 1D	Nombre de filtres = 512, Taille des filtres = 11, Padding = Same, Activation = ReLU
Average Pooling	Taille du Pooling = 2
Flatten	-
Fully-connected	Nombre de neurones = 4096
Fully-connected	Nombre de neurones = 4096
Sortie	Softmax : 256 classes

TABLE A.2: Hyperparamètres du réseau CNN_{BV} [vdVP19]

Type de couche	Hyperparamètres
Taille de l'entrée	700
Convolution 1D	Nombre de filtres = 8, Taille des filtres = 3, Activation = ReLU
Batch Normalization	-
Max Pooling (Optionnel)	Taille du Pooling = 2
Convolution 1D	Nombre de filtres = 16, Taille des filtres = 3, Activation = ReLU
Batch Normalization	-
Max Pooling (Optionnel)	Taille du Pooling = 2
Convolution 1D	Nombre de filtres = 32, Taille des filtres = 3, Activation = ReLU
Batch Normalization	-
Max Pooling (Optionnel)	Taille du Pooling = 2
Convolution 1D	Nombre de filtres = 64, Taille des filtres = 3, Activation = ReLU
Batch Normalization	-
Max Pooling (Optionnel)	Taille du Pooling = 2
Convolution 1D	Nombre de filtres = 64, Taille des filtres = 3, Activation = ReLU
Batch Normalization	-
Max Pooling (Optionnel)	Taille du Pooling = 2
Flatten	-
Dropout	Coefficient = 0.5
Fully-connected	Nombre de neurones = 512
Dropout	Coefficient = 0.5
Sortie	Softmax : 9 or 256 classes

TABLE A.3: Hyperparamètres du réseau CNN_{bn}.

Type de couche	Hyperparamètres
Taille de l'entrée	1400
Convolution 1D	Nombre de filtres = 64, Taille des filtres = 11, Padding = Same, Activation = ReLU
Batch Normalization	-
Average Pooling	Taille du Pooling = 2
Convolution 1D	Nombre de filtres = 128, Taille des filtres = 11, Padding = Same, Activation = ReLU
Batch Normalization	-
Average Pooling	Taille du Pooling = 2
Convolution 1D	Nombre de filtres = 256, Taille des filtres = 11, Padding = Same, Activation = ReLU
Batch Normalization	-
Average Pooling	Taille du Pooling = 2
Convolution 1D	Nombre de filtres = 512, Taille des filtres = 11, Padding = Same, Activation = ReLU
Batch Normalization	-
Average Pooling	Taille du Pooling = 2
Convolution 1D	Nombre de filtres = 512, Taille des filtres = 11, Padding = Same, Activation = ReLU
Batch Normalization	-
Average Pooling	Taille du Pooling = 2
Flatten	-
Fully-connected	Nombre de neurones = 4096
Fully-connected	Nombre de neurones = 4096
Sortie	Softmax : 256 classes

TABLE A.4: Hyperparamètres du réseau CNN_{bnreg}.

Type de couche	Hyperparamètres
Taille de l'entrée	1400
Convolution 1D	Nombre de filtres = 64, Taille des filtres = 11, Padding = Same, Activation = ReLU
Batch Normalization	-
Average Pooling	Taille du Pooling = 2
Convolution 1D	Nombre de filtres = 128, Taille des filtres = 11, Padding = Same, Activation = ReLU
Batch Normalization	-
Average Pooling	Taille du Pooling = 2
Convolution 1D	Nombre de filtres = 256, Taille des filtres = 11, Padding = Same, Activation = ReLU, L_2 = 0.2, Dropout = 0.5
Batch Normalization	-
Average Pooling	Taille du Pooling = 2
Convolution 1D	Nombre de filtres = 512, Taille des filtres = 11, Padding = Same, Activation = ReLU, L_2 = 0.3, Dropout = 0.6
Batch Normalization	-
Average Pooling	Taille du Pooling = 2
Convolution 1D	Nombre de filtres = 512, Taille des filtres = 11, Padding = Same, Activation = ReLU, L_2 = 0.3, Dropout = 0.7
Batch Normalization	-
Average Pooling	Taille du Pooling = 2
Flatten	-
Fully-connected	Nombre de neurones = 4096
Fully-connected	Nombre de neurones = 4096
Sortie	Softmax : 256 classes

TABLE A.5: Hyperparamètres du réseau MLP_{score}.

Type de couche	Hyperparamètres
Prédiction d'entrée	Taille : 256
Fully-connected	Nombre de neurones = 1024, Activation = ReLU
Fully-connected	Nombre de neurones = 1024, Activation = ReLU
Sortie	Linéaire : 1 valeur de score

Bibliographie

[AA04] Dmitri Asonov and Rakesh Agrawal. Keyboard acoustic emanations. In IEEE Symposium on Security and Privacy, 2004. Proceedings. 2004, pages 3–11. IEEE, 2004.

[Age03] National Security Agency. Cnss policy no. 15, fact sheet no. 1 national policy on the use of the advanced encryption standard (aes) to protect national security systems and national security information. 2003.

[BB12] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. Journal of machine learning research, 13(2), 2012.

[BBB+93] Jane Bromley, James W Bentz, Léon Bottou, Isabelle Guyon, Yann LeCun, Cliff Moore, Eduard Säckinger, and Roopak Shah. Signature verification using a "siamese" time delay neural network. International Journal of Pattern Recognition and Artificial Intelligence, 7(04) :669–688, 1993.

[BBD+14] Shivam Bhasin, Nicolas Bruneau, Jean-Luc Danger, Sylvain Guilley, and Zakaria Najm. Analysis and improvements of the dpa contest v4 implementation. In Rajat Subhra Chakraborty, Vashek Matyas, and Patrick Schaumont, editors, Security, Privacy, and Applied Cryptography Engineering, pages 201–218, Cham, 2014. Springer International Publishing.

[BCN18] Léon Bottou, Frank E Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. Siam Review, 60(2) :223–311, 2018.

[BDG+10] Michael Backes, Markus Dürmuth, Sebastian Gerling, Manfred Pinkal, and Caroline Sporleder. Acoustic side-channel attacks on printers. In USENIX Security symposium, volume 10, pages 307–322, 2010.

[BLR12] Timo Bartkewitz and Kerstin Lemke-Rust. Efficient template attacks based on probabilistic multi-class support vector machines. In International Conference on Smart Card Research and Advanced Applications, pages 263–276. Springer, 2012.

[BOW15] Valentina Banciu, Elisabeth Oswald, and Carolyn Whitnall. Reliable information extraction for single trace attacks. In 2015 Design, Automation & Test in Europe Conference & Exhibition (DATE), pages 133–138. IEEE, 2015.

[BPS+19] Ryad Benadjila, Emmanuel Prouff, Rémi Strullu, Eleonora Cagli, and Cécile Dumas. Deep learning for side-channel analysis and introduction to ascad database. Journal of Cryptographic Engineering, 10 :163–188, 2019.

[BRL06] Christopher Burges, Robert Ragno, and Quoc Le. Learning to rank with nonsmooth cost functions. Advances in neural information processing systems, 19 :193–200, 2006.

[BSR+05] Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. Learning to rank using gradient descent. In Proceedings of the 22nd international conference on Machine learning, pages 89–96, 2005.

[BVH+16] Luca Bertinetto, Jack Valmadre, João F. Henriques, Andrea Vedaldi, and Philip H. S. Torr. Fully-convolutional siamese networks for object tracking. In Gang Hua and Hervé Jégou, editors, Computer Vision – ECCV 2016 Workshops, pages 850–865, Cham, 2016. Springer International Publishing.

[C+47] Augustin Cauchy et al. Méthode générale pour la résolution des systemes d'équations simultanées. Comp. Rend. Sci. Paris, 25(1847) :536–538, 1847.

[CAS16] Paul Covington, Jay Adams, and Emre Sargin. Deep neural networks for youtube recommendations. In Proceedings of the 10th ACM Conference on Recommender Systems, RecSys '16, page 191–198. Association for Computing Machinery, 2016.

[CCJ+16] Lily Chen, Lily Chen, Stephen Jordan, Yi-Kai Liu, Dustin Moody, Rene Peralta, Ray Perlner, and Daniel Smith-Tone. Report on post-quantum cryptography, volume 12. US Department of Commerce, National Institute of Standards and Technology, 2016.

[CDP17] Eleonora Cagli, Cécile Dumas, and Emmanuel Prouff. Convolutional neural networks with data augmentation against jitter-based countermeasures. In International Conference on Cryptographic Hardware and Embedded Systems, pages 45–68. Springer, 2017.

[CLL+09] Wei Chen, Tie-Yan Liu, Yanyan Lan, Zhi-Ming Ma, and Hang Li. Ranking measures and loss functions in learning to rank. Advances in Neural Information Processing Systems, 22 :315–323, 2009.

[CMS12] Dan Ciregan, Ueli Meier, and Jürgen Schmidhuber. Multi-column deep neural networks for image classification. In 2012 IEEE conference on computer vision and pattern recognition, pages 3642–3649. IEEE, 2012.

[CQL+07] Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. Learning to rank : from pairwise approach to listwise approach. In Proceedings of the 24th international conference on Machine learning, pages 129–136, 2007.

[CRR03] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template attacks. In Cryptographic Hardware and Embedded Systems - CHES 2002, pages 13–28, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[Cyb89] George Cybenko. Approximation by superpositions of a sigmoidal function.

Mathematics of control, signals and systems, 2(4) :303–314, 1989.

[DR99] Joan Daemen and Vincent Rijmen. Aes proposal : Rijndael. 1999.

[GB10] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Proceedings of the thirteenth international conference on artificial intelligence and statistics, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.

[GBC16] Ian J. Goodfellow, Yoshua Bengio, and Aaron C. Courville. Deep Learning. Adaptive computation and machine learning. MIT Press, 2016.

[GFZ+17] Qing Guo, Wei Feng, Ce Zhou, Rui Huang, Liang Wan, and Song Wang. Learning dynamic siamese network for visual object tracking. In Proceedings of the IEEE international conference on computer vision, pages 1763–1771, 2017.

[GHO15] Richard Gilmore, Neil Hanley, and Maire O'Neill. Neural network based attack on a masked implementation of aes. In 2015 IEEE International Symposium on Hardware Oriented Security and Trust (HOST), pages 106–111. IEEE, 2015.

[GKLD11] Sylvain Guilley, Karim Khalfallah, Victor Lomné, and Jean-Luc Danger. Formal framework for the evaluation of waveform resynchronization algorithms. In IFIP International Workshop on Information Security Theory and Practices, pages 100–115. Springer, 2011.

[GMO01] Karine Gandolfi, Christophe Mourtel, and Francis Olivier. Electromagnetic analysis : Concrete results. In International workshop on cryptographic hardware and embedded systems, pages 251–261. Springer, 2001.

[GSC+19] Martin Gleize, Eyal Shnarch, Leshem Choshen, Lena Dankin, Guy Moshkowich, Ranit Aharonov, and Noam Slonim. Are you convinced ? choosing the more convincing evidence with a siamese network. arXiv preprint arXiv :1907.08971, 2019.

[GSS14] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. arXiv preprint arXiv :1412.6572, 2014.

[HDY+12] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition : The shared views of four research groups. IEEE Signal processing magazine, 29(6) :82–97, 2012.

[HGDM+11] Gabriel Hospodar, Benedikt Gierlichs, Elke De Mulder, Ingrid Verbauwhede, and Joos Vandewalle. Machine learning in side-channel analysis : a first study. Journal of Cryptographic Engineering, 1(4) :293, 2011.

[HJZ12] Hera He, Josh Jaffe, and Long Zou. Side channel cryptanalysis using ma-

chine learning. CS229 Project, pages 1–392, 2012.

[HOT06] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. Neural computation, 18(7) :1527–1554, 2006.

[HSK+12] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. arXiv preprint arXiv :1207.0580, 2012.

[HSW89] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. Neural networks, 2(5) :359–366, 1989.

[HZ12] Annelie Heuser and Michael Zohner. Intelligent machine homicide. In International Workshop on Constructive Side-Channel Analysis and Secure Design, pages 249–264. Springer, 2012.

[IS15] Sergey Ioffe and Christian Szegedy. Batch normalization : Accelerating deep network training by reducing internal covariate shift. In International conference on machine learning, pages 448–456. PMLR, 2015.

[KB17] Diederik P. Kingma and Jimmy Ba. Adam : A method for stochastic optimization, 2017.

[KGB17] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. Adversarial machine learning at scale. In 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings. OpenReview.net, 2017.

[KH92] Anders Krogh and John A Hertz. A simple weight decay can improve generalization. In Advances in neural information processing systems, pages 950–957, 1992.

[KH+09] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[KJJ99] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Annual international cryptology conference, pages 388–397. Springer, 1999.

[Koc96] Paul C Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In Annual International Cryptology Conference, pages 104–113. Springer, 1996.

[KPH+19] Jaehun Kim, Stjepan Picek, Annelie Heuser, Shivam Bhasin, and Alan Hanjalic. Make some noise. unleashing the power of convolutional neural networks for profiled side-channel analysis. IACR Transactions on Cryptographic Hardware and Embedded Systems, pages 148–179, 2019.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. Advances in neural information processing systems, 25 :1097–1105, 2012.

[KW20] Mazaher Kianpour and Shao-Fang Wen. Timing attacks on machine learning : State of the art. In Yaxin Bi, Rahul Bhatia, and Supriya Kapoor, editors, Intelligent Systems and Applications, pages 111–125, Cham, 2020. Springer International Publishing.

[LBBH98] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11) :2278–2324, 1998.

[LBM11] Liran Lerman, Gianluca Bontempi, and Olivier Markowitch. Side channel attack : an approach based on machine learning. Center for Advanced Security Research Darmstadt, 29, 2011.

[LBM14] Liran Lerman, Gianluca Bontempi, and Olivier Markowitch. Power analysis attack : an approach based on machine learning. International Journal of Applied Cryptography, 3(2) :97–115, 2014.

[LBM15a] Liran Lerman, Gianluca Bontempi, and Olivier Markowitch. The bias–variance decomposition in profiled attacks. Journal of Cryptographic Engineering, 5(4) :255–267, 2015.

[LBM15b] Liran Lerman, Gianluca Bontempi, and Olivier Markowitch. A machine learning approach against a masked aes. Journal of Cryptographic Engineering, 5(2) :123–139, 2015.

[LBTM13] Liran Lerman, Gianluca Bontempi, Souhaib Ben Taieb, and Olivier Markowitch. A time series approach for profiling attack. In International Conference on Security, Privacy, and Applied Cryptography Engineering, pages 75–94. Springer, 2013.

[Li14] Hang Li. Learning to rank for information retrieval and natural language processing. Synthesis lectures on human language technologies, 7(3) :1–121, 2014.

[LKBS18] Joffrey L Leevy, Taghi M Khoshgoftaar, Richard A Bauder, and Naeem Seliya. A survey on addressing high-class imbalance in big data. Journal of Big Data, 5(1) :1–30, 2018.

[LKP20] Huimin Li, Marina Krček, and Guilherme Perin. A comparison of weight initializers in deep learning-based side-channel analysis. Cryptology ePrint Archive, Report 2020/904, 2020. https://eprint.iacr.org/2020/904.

[LMK+11] Yuanhua Lv, Taesup Moon, Pranam Kolari, Zhaohui Zheng, Xuanhui Wang, and Yi Chang. Learning to model relatedness for news recommendation. In Proceedings of the 20th international conference on World wide web, pages 57–66, 2011.

[LPB+15] Liran Lerman, Romain Poussier, Gianluca Bontempi, Olivier Markowitch, and François-Xavier Standaert. Template attacks vs. machine learning

revisited (and the curse of dimensionality in side-channel analysis). In International Workshop on Constructive Side-Channel Analysis and Secure Design, pages 20–33. Springer, 2015.

[LSV19] Alex Labach, Hojjat Salehinejad, and Shahrokh Valaee. Survey of dropout methods for deep neural networks. arXiv preprint arXiv :1904.13310, 2019.

[M+10] James Martens et al. Deep learning via hessian-free optimization. In ICML, volume 27, pages 735–742, 2010.

[MDM16] Zdenek Martinasek, Petr Dzurenda, and Lukas Malina. Profiling power analysis attack based on mlp in dpa contest v4. 2. In 2016 39th International Conference on Telecommunications and Signal Processing (TSP), pages 223–226. IEEE, 2016.

[MDP19] Loïc Masure, Cécile Dumas, and Emmanuel Prouff. Gradient visualization for general characterization in profiling attacks. In International Workshop on Constructive Side-Channel Analysis and Secure Design, pages 145–167. Springer, 2019.

[MDP20] Loïc Masure, Cécile Dumas, and Emmanuel Prouff. A comprehensive study of deep learning for side-channel analysis. IACR Transactions on Cryptographic Hardware and Embedded Systems, pages 348–375, 2020.

[MOP08] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. Power analysis attacks : Revealing the secrets of smart cards, volume 31. Springer Science & Business Media, 2008.

[MPP16] Houssem Maghrebi, Thibault Portigliatti, and Emmanuel Prouff. Breaking cryptographic implementations using deep learning techniques. In International Conference on Security, Privacy, and Applied Cryptography Engineering, pages 3–26. Springer, 2016.

[MZ13] Zdenek Martinasek and Vaclav Zeman. Innovative method of the power analysis. Radioengineering, 22(2) :586–594, 2013.

[MZVT15] Zdenek Martinasek, Ondrej Zapletal, Kamil Vrba, and Krisztina Trasy. Power analysis attack based on the mlp in dpa contest v4. In 2015 38th International Conference on Telecommunications and Signal Processing (TSP), pages 154–158. IEEE, 2015.

[PB14] Hiren Patel and Rusty O Baldwin. Random forest profiling attack on advanced encryption standard. International Journal of Applied Cryptography, 3(2) :181–194, 2014.

[PBP] Guilherme Perin, Ileana Buhan, and Stjepan Picek. Learning when to stop : a mutual information approach to prevent overfitting in profiled side-channel analysis.

[PHJ+17] Stjepan Picek, Annelie Heuser, Alan Jovic, Simone A Ludwig, Sylvain

Guilley, Domagoj Jakobovic, and Nele Mentens. Side-channel analysis and machine learning : A practical perspective. In 2017 International Joint Conference on Neural Networks (IJCNN), pages 4095–4102. IEEE, 2017.

[PHJ+19] Stjepan Picek, Annelie Heuser, Alan Jovic, Shivam Bhasin, and Francesco Regazzoni. The curse of class imbalance and conflicting metrics with machine learning for side-channel evaluations. IACR Transactions on Cryptographic Hardware and Embedded Systems, 2019(1) :1–29, 2019.

[PK16] Sungheon Park and Nojun Kwak. Analysis on the dropout effect in convolutional neural networks. In Asian conference on computer vision, pages 189–204. Springer, 2016.

[PWRZ00] Richard C Price, Jonathan P Willmore, William JJ Roberts, and KJ Zyga. Genetically optimised feedforward neural networks for speaker identification. In KES'2000. Fourth International Conference on Knowledge-Based Intelligent Engineering Systems and Allied Technologies. Proceedings (Cat. No. 00TH8516), volume 2, pages 479–482. IEEE, 2000.

[QLL10] Tao Qin, Tie-Yan Liu, and Hang Li. A general approximation framework for direct optimization of information retrieval measures. Information retrieval, 13(4) :375–397, 2010.

[RBHG21] Damien Robissout, Lilian Bossuet, Amaury Habrard, and Vincent Grosso. Improving deep learning networks for profiled side-channel analysis using performance improvement techniques. J. Emerg. Technol. Comput. Syst., 17(3), June 2021.

[Riv91] Ronald L Rivest. Cryptography and machine learning. In International Conference on the Theory and Application of Cryptology, pages 427–439. Springer, 1991.

[RMN09] Rajat Raina, Anand Madhavan, and Andrew Y Ng. Large-scale deep unsupervised learning using graphics processors. In Proceedings of the 26th annual international conference on machine learning, pages 873–880, 2009.

[RZC+20] Damien Robissout, Gabriel Zaid, Brice Colombier, Lilian Bossuet, and Amaury Habrard. Online performance evaluation of deep learning networks for profiled side-channel analysis. In International Workshop on Constructive Side-Channel Analysis and Secure Design, pages 200–218. Springer, 2020.

[SHK+14] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout : a simple way to prevent neural networks from overfitting. The journal of machine learning research, 15(1) :1929–1958, 2014.

[SK19] Connor Shorten and Taghi M Khoshgoftaar. A survey on image data augmentation for deep learning. Journal of Big Data, 6(1) :1–48, 2019.

[SMG13] Andrew M Saxe, James L McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. arXiv preprint arXiv :1312.6120, 2013.

[SMY09a] François-Xavier Standaert, Tal G. Malkin, and Moti Yung. A unified framework for the analysis of side-channel key recovery attacks. In Antoine Joux, editor, Advances in Cryptology - EUROCRYPT 2009, pages 443–461, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[SMY09b] François-Xavier Standaert, Tal G Malkin, and Moti Yung. A unified framework for the analysis of side-channel key recovery attacks. In Annual international conference on the theory and applications of cryptographic techniques, pages 443–461. Springer, 2009.

[STIM18] Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Mądry. How does batch normalization help optimization ? In Proceedings of the 32nd international conference on neural information processing systems, pages 2488–2498, 2018.

[SVL14] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In Advances in neural information processing systems, pages 3104–3112, 2014.

[vdVP19] Daan van der Valk and Stjepan Picek. Bias-variance decomposition in machine learning-based side-channel analysis. Cryptology ePrint Archive, Report 2019/570, 2019. https://eprint.iacr.org/2019/570.

[VL17] Twan Van Laarhoven. L2 regularization versus batch and weight normalization. arXiv preprint arXiv :1706.05350, 2017.

[WAGP20] Lennert Wouters, Victor Arribas, Benedikt Gierlichs, and Bart Preneel. Revisiting a methodology for efficient cnn architectures in profiling attacks. IACR Transactions on Cryptographic Hardware and Embedded Systems, 2020(3) :147–168, 2020.

[WBSG10] Qiang Wu, Christopher JC Burges, Krysta M Svore, and Jianfeng Gao. Adapting boosting for information retrieval measures. Information Retrieval, 13(3) :254–270, 2010.

[WJB20] Yoo-Seung Won, Dirmanto Jap, and Shivam Bhasin. Push for more : On comparison of data augmentation and smote with optimised deep learning architecture for side-channel. In International Conference on Information Security Applications, pages 227–241. Springer, 2020.

[WPB19] Leo Weissbart, Stjepan Picek, and Lejla Batina. On the performance of multilayer perceptron in profiling side-channel analysis. 2019. https://eprint.iacr.org/2019/1476.

[WPP20] Lichao Wu, Guilherme Perin, and Stjepan Picek. I choose you : Automa-

ted hyperparameter tuning for deep learning-based side-channel analysis. IACR Cryptol. ePrint Arch., 2020 :1293, 2020.

[XJP+10] Biao Xiang, Daxin Jiang, Jian Pei, Xiaohui Sun, Enhong Chen, and Hang Li. Context-aware ranking in web search. In Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval, pages 451–458, 2010.

[XLW+08] Fen Xia, Tie-Yan Liu, Jue Wang, Wensheng Zhang, and Hang Li. Listwise approach to learning to rank : theory and algorithm. In Proceedings of the 25th international conference on Machine learning, pages 1192–1199, 2008.

[ZBD+21] Gabriel Zaid, Lilian Bossuet, François Dassance, Amaury Habrard, and Alexandre Venelli. Ranking loss : Maximizing the success rate in deep learning side-channel analysis. IACR Transactions on Cryptographic Hardware and Embedded Systems, pages 25–55, 2021.

[ZBHV20a] Gabriel Zaid, Lilian Bossuet, Amaury Habrard, and Alexandre Venelli. Methodology for efficient cnn architectures in profiling attacks. IACR Transactions on Cryptographic Hardware and Embedded Systems, 2020(1) :1–36, 2020.

[ZBHV20b] Gabriel Zaid, Lilian Bossuet, Amaury Habrard, and Alexandre Venelli. Understanding methodology for efficient cnn architectures in profiling attacks. Cryptology ePrint Archive, Report 2020/757, 2020. https://eprint.iacr.org/2020/757.

[ZZN+20] Jiajia Zhang, Mengce Zheng, Jiehui Nan, Honggang Hu, and Nenghai Yu. A novel evaluation metric for deep learning-based side channel analysis and its extended application to imbalanced data. IACR Transactions on Cryptographic Hardware and Embedded Systems, 2020(3) :73–96, 2020.

[ZZT09] Li Zhuang, Feng Zhou, and J Doug Tygar. Keyboard acoustic emanations revisited. ACM Transactions on Information and System Security (TISSEC), 13(1) :1–26, 2009.

Table des figures

1.1 Phase de profilage des traces. 9
1.2 Phase de correspondance entre les traces et les profils. 9

2.1 Exemple de l'impact des opérations AddRoundKey et SubBytes sur la consommation de courant dans le modèle de l'identité sans ajout de bruit. 17
2.2 Étapes d'une attaque template et la correspondance d'une attaque basée sur le Deep Learning. 21
2.3 Représentation des biais. 23
2.4 Représentation du potentiel d'approximation de différentes familles de fonction. 25
2.5 Exemple d'une fonction de perte à minimiser. 28
2.6 Exemple de perceptron. 31
2.7 Exemple de MLP à trois couches. 33
2.8 Exemple d'une opération de convolution. 35
2.9 Fresque résumant les articles majeurs portant sur l'application des techniques de deep learning aux attaques par canaux auxiliaires. 42

3.1 Description du processus d'early stopping. 53
3.2 Évolution de l'accuracy et du rang de la bonne clé après l'utilisation de 5000 traces d'attaque évaluées sur l'ensemble d'entraînement et de validation au cours de l'entraînement de CNN_{best}. 55
3.3 Exemple d'évolution de $\Delta^1_{train,val}$ au cours de l'entraînement d'un réseau. La courbe représentant $\Delta^1_{train,val}$ est faite à l'aide d'une moyenne glissante de fenêtre de taille 10. 60
3.4 Description du processus d'early stopping utilisant la métrique $\Delta^d_{train,val}$. . . 61
3.5 Calcul de $\Delta^1_{train,val}$ pour des epochs consécutives lorsque le calcul est plus court que l'entraînement d'une epoch 61
3.6 Calcul de $\Delta^1_{train,val}$ pour des epochs consécutives lorsque le calcul est plus long que l'entraînement d'une epoch 62
3.7 Évolution de $\Delta^1_{train,val}$ et N^1_{train} durant l'entraînement du réseau CNN_{best} et comparaison avec les résultats présentés dans [BPS+19]. 65
3.8 Évolution de $\Delta^1_{train,val}$ et N^1_{train} durant l'entraînement du réseau CNN_{BV} 0CONV en utilisant la fonction de perte MSE et comparaison avec les résultats présentés dans [vdVP19]. 66
3.9 Évolution de $\Delta^1_{train,val}$ et N^1_{train} durant l'entraînement du réseau CNN_{BV} 0CONV en utilisant la fonction de perte CCE. 67

3.10 Évolution de $\Delta^1_{train,val}$ et N^1_{train} durant l'entraînement du réseau CNN_{BV} 1CONV en utilisant la fonction de perte CCE. 68

4.1 Effet de la batch normalization sur la distribution de l'entrée d'une couche. 76
4.2 Changement dans la direction du gradient selon la taille du pas utilisé précédemment. 78
4.3 Exemple des effects du dropout sur un réseau complètement connecté. . . . 83
4.4 Traces de consommation de courant pour une desynchronisation $n = 0$ (en bleu) et $n = 100$ (en orange). 84
4.5 Illustration du ré-ordonnancement de la clé. 85
4.6 Évolution de $\Delta^1_{train,val}$ pour CNN_{best} durant l'entraînement pour différents nombres d'exemples d'entraînement sur Desync0. 87
4.7 Évolution du rang de la bonne clé après 5000 traces d'attaque durant l'entraînement du réseau CNN_{best} pour différents nombres d'exemples d'entraînement et différents niveaux de désynchronisation. 87
4.8 Évolution de $\Delta^1_{train,val}$ pour CNN_{bn} durant l'entraînement pour différents nombres d'exemples d'entraînement et différents niveaux de désynchronisation. 89
4.9 Évolution du rang de la bonne clé après 5000 traces d'attaque durant l'entraînement du réseau CNN_{bn} pour différents nombres d'exemples d'entraînement et différents niveaux de désynchronisation. 89
4.10 Évolution de $\Delta^1_{train,val}$ pour $\text{CNN}_{\text{bnreg}}$ durant l'entraînement pour différents nombres d'exemples d'entraînement et différents niveaux de désynchronisation. 91
4.11 Évolution du rang de la bonne clé après 5000 traces d'attaque durant l'entraînement du réseau $\text{CNN}_{\text{bnreg}}$ pour différents nombres d'exemples d'entraînement et différents niveaux de désynchronisation. 92

5.1 Schéma illustrant la méthode d'attaque classique des attaques par canaux auxiliaires et les changements de la nouvelle méthode. 98
5.2 Évolution du rang moyen de la bonne clé du même ensemble de prédictions avec ou sans ordonnancement des prédictions. 100
5.3 Évolution du rang moyen de la bonne clé pour deux ordonnancements différents lorsque l'attaque est réussie. 101
5.4 Exemple de l'utilisation de distingueurs (maximum, minimum, médiane) pour ordonner les prédictions de 4 classes possibles (1,2,3 et 4). 104
5.5 Évolution du rang moyen de la bonne clé pour des attaques utilisant un ordonnancement aléatoire et un ordonnancement basé sur un distingueur (minimum et médiane) des prédictions du réseau CNN_{best} sur la base de données ASCAD clé fixe et l'ensemble Desync0. 105
5.6 Évolution du rang moyen de la bonne clé pour des attaques utilisant un ordonnancement aléatoire et un ordonnancement basé sur un distingueur (minimum et médiane) des prédictions du réseau CNN_{best} sur l'ensemble ASCAD clé fixe Desync50. 106

5.7 Évolution du rang moyen de la bonne clé pour des attaques utilisant un ordonnancement aléatoire et un et un ordonnancement inversé basé sur un distingueur (minimum et médiane) des prédictions du réseau CNN_{best} sur l'ensemble ASCAD clé fixe Desync50. 107
5.8 Évolution du rang moyen de la bonne clé pour des attaques utilisant 5000 traces et un ordonnancement aléatoire et un ordonnancement inversé basé sur un distingueur (minimum et médiane) des prédictions du réseau CNN_{best} sur l'ensemble ASCAD clé fixe Desync50. 108
5.9 Évolution du rang moyen de la bonne clé pour des attaques utilisant un ordonnancement aléatoire et un ordonnancement inversé basé sur un distingueur (minimum et médiane) des prédictions du réseau CNN_{best} sur l'ensemble ASCAD clé fixe Desync100. 108
5.10 Évolution du rang moyen de la bonne clé pour des attaques utilisant 5000 traces et un ordonnancement aléatoire ou un ordonnancement inversé basé sur un distingueur (minimum et médiane) des prédictions du réseau CNN_{best} sur l'ensemble ASCAD clé fixe Desync100. 109
5.11 Évolution de la moyenne des prédictions ordonnées par ordre croissant de valeur pour les différentes niveaux de désynchronisation. 110
5.12 Valeurs minimales des prédictions pour Desync50 et Desync100 classées en fonction de la désynchronisation des traces. 111
5.13 Évolution du rang moyen de la bonne clé pour des attaques utilisant un ordonnancement aléatoire et un ordonnancement basé sur un distingueur (minimum et médiane) des prédictions du réseau CNN_{best} sur l'ensemble ASCAD clé variable Desync0. 111
5.14 Évolution du rang moyen de la bonne clé pour des attaques utilisant différents nombres de traces et un ordonnancement aléatoire et un ordonnancement inversé basé sur un distingueur (minimum et médiane) des prédictions du réseau CNN_{best} sur l'ensemble ASCAD clé variable Desync50. . . . 112
5.15 Évolution du rang moyen de la bonne clé pour des attaques utilisant différents nombres de traces et un ordonnancement aléatoire et un ordonnancement inversé basé sur un distingueur (minimum et médiane) des prédictions du réseau CNN_{best} sur l'ensemble ASCAD clé variable Desync100. . . 113
5.16 Évolution de la moyenne des prédictions ordonnées par ordre croissant de valeur pour différentes niveaux de désynchronisation. 115
5.17 Valeurs minimales des prédictions pour Desync50 et Desync100 classées en fonction de la désynchronisation des traces. 116
5.18 Valeurs médianes des prédictions pour Desync50 et Desync100 classées en fonction de la désynchronisation des traces. 116
5.19 Évolution du rang moyen de la bonne clé pour des attaques utilisant un ordonnancement basé sur le minimum des prédictions d'un template réalisé avec 10 000 traces. 118

5.20 Évolution du rang moyen de la bonne clé pour des attaques utilisant un ordonnancement normal et un ordonnancement inversé basés sur le minimum des prédictions d'un template réalisé avec 20 000 traces. 118
5.21 Évolution du rang moyen de la bonne clé pour des attaques utilisant un ordonnancement normal et un ordonnancement inversé basés sur le minimum des prédictions d'un template réalisé avec 50 000 traces. 119
5.22 Schéma d'un réseau siamois. Les deux réseaux partagent les mêmes poids et reçoivent chacun une entrée différente. Les scores qu'ils déterminent sont ensuite comparés pour être utilisés dans la fonction de perte. 123
5.23 Schéma illustrant l'évolution du rang de différentes hypothèses de clés après l'ajout d'une nouvelle prédiction. La bonne clé est représentée en rouge et les hypothèses mieux placées sont représentées en bleu. 126
5.24 Exemple d'évolution du rang de la bonne clé en utilisant un ordre déterminé par le rang du bon label dans chaque prédiction. La valeur du rang du bon label est également affichée pour chaque trace. 127
5.25 Schéma illustrant la nouvelle méthode d'attaque des attaques par canaux auxiliaires utilisant un réseau siamois pour déterminer un ordonnancement des prédictions. 132
5.26 Évolution du rang moyen de la bonne clé pour des attaques utilisant un ordre normal déterminé par le minimum et un ordre déterminé par le score sur l'ensemble de données ASCAD clé fixe Desync0. 134
5.27 Évolution du rang moyen de la bonne clé pour des attaques utilisant un ordre inversé déterminé par le minimum et un ordre déterminé par le score sur l'ensemble de données ASCAD clé fixe Desync50. 134
5.28 Évolution du rang moyen de la bonne clé pour des attaques utilisant un ordre inversé déterminé par le minimum et un ordre déterminé par le score sur l'ensemble de données ASCAD clé fixe Desync100. 135
5.29 Évolution du rang moyen de la bonne clé pour des attaques utilisant un ordre normal déterminé par le minimum et un ordre déterminé par le score sur l'ensemble de données ASCAD clé variable Desync0. 136
5.30 Évolution du rang moyen de la bonne clé pour des attaques utilisant un ordre inversé déterminé par le minimum et un ordre déterminé par le score sur l'ensemble de données ASCAD clé variable Desync50. 137
5.31 Évolution du rang moyen de la bonne clé pour des attaques utilisant un ordre inversé déterminé par le minimum et un ordre déterminé par le score sur l'ensemble de données ASCAD clé variable Desync100. 138

Liste des tableaux

3.1 Résumé des résultats obtenus avec en **gras** les choix fait grâce à $\Delta^1_{train,val}$ . . 69

4.1 Résumé des différentes méthodes utilisées et explorées pour l'amélioration des performances des réseaux de neurones dans le cadre d'attaques par canaux auxiliaires. 74
4.2 Tableau récapitulatif des tests fait sur le dropout et la régularisation L2. . . 90
4.3 Résumé des résultats de CNN_{best}, CNN_{bn} et $\text{CNN}_{\text{bnreg}}$ en fonction des valeurs de $\Delta^1_{train,val}$, N^1_{val}, N^*_a et le nombre d'epochs d'entraînement pour l'entraînement utilisant 190 000 traces d'entraînement. 93
4.4 Résumé des résultats d'autres travaux sur les bases de données ASCAD en fonction du nombre de traces nécessaires pour réussir les attaques. 94

A.1 Hyperparamètres du réseau CNN_{best} [BPS+19]. 147
A.2 Hyperparamètres du réseau CNN_{BV} [vdVP19] 148
A.3 Hyperparamètres du réseau CNN_{bn}. 149
A.4 Hyperparamètres du réseau $\text{CNN}_{\text{bnreg}}$. 150
A.5 Hyperparamètres du réseau MLP_{score}. 151