Étude de l'utilisation de réseaux de neurones artificiels pour des calculs de haute performance dédiés à la modélisation du transport de sources énergétiques

Corisande Lamy

THÈSE PRÉSENTÉE
POUR OBTENIR LE GRADE DE
DOCTEUR
DE L'UNIVERSITÉ DE BORDEAUX

ÉCOLE DOCTORALE MATHÉMATIQUES ET INFORMATIQUE

MATHÉMATIQUES APPLIQUÉES ET CALCUL SCIENTIFIQUE

Par **Corisande LAMY**

Étude de l'utilisation de réseaux de neurones artificiels pour des calculs de haute performance dédiés à la modélisation du transport de sources énergétiques

Directeurs de thèse :
Bruno DUBROCA, Jean-Luc FEUGEAS, Philippe NICOLAÏ

Soutenue le 21 novembre 2022

Membres du jury :

Laurent BOUDIN	Maître de conférence	Sorbonne Université	Rapporteur
Frédérique CHARLES	Maîtresse de conférence	Sorbonne Université	Examinatrice
Bruno DUBROCA	Directeur de Recherche	Université de Bordeaux	Directeur
Jean-Luc FEUGEAS	Directeur de Recherche	Université de Bordeaux	Co-directeur
Gilles KLUTH	Ingénieur Chercheur	CEA/DAM	Examinateur
Philippe NICOLAÏ	Directeur de Recherche	Université de Bordeaux	Invité
Marina OLAZABAL	Directrice de Recherche	CEA/DAM	Examinatrice
Guillaume PERRIN	Directeur de Recherche	Université Gustave Eiffel	Rapporteur
Gaël POETTE	Ingénieur Chercheur	CEA/DAM	Invité
Olivier SAUT	Directeur de Recherche	CNRS	Président du jury

Étude de l'utilisation de réseaux de neurones pour des calculs de haute performance dédiés à la modélisation du transport des sources énergétiques

Résumé : Quels que soient leurs domaines d'application, les codes de calcul modernes sont soumis à des exigences de rapidité de calcul et d'optimisation d'occupation de la place mémoire, qui requièrent l'utilisation de méthodes numériques évoluées. Pour répondre à ces besoins de traitement de données massives et de calculs intensifs, les méthodes d'apprentissage profond se positionnent comme des réponses alternatives efficaces et extrêmement attractives.

Ce manuscrit de thèse présente une étude sur le couplage de réseaux de neurones artificiels à des codes de calcul haute performance dédiés à des simulations des phénomènes de physique complexes, plus particulièrement en lien avec des théories de transport. Le premier cadre d'application concerne la simulation d'expériences de fusion par confinement inertiel pour la production d'énergie, et en particulier, la modélisation du transport de chaleur électronique non local. Le second domaine d'application est le transport et dépôt d'énergie de particules en radiothérapie pour le traitement des cancers.

Les deux études du couplage de réseaux de neurones artificiels à des codes de calcul haute performance, développés par l'équipe de recherche IFCIA au laboratoire CELIA, sont présentées dans ce manuscrit, et montrent des résultats encourageants : pour des critères de précision largement suffisant pour ces applications, nous obtenons des gains de temps considérables. Cette étude préliminaire de faisabilité encourage l'équipe du CELIA à poursuivre ce travail dans les années à venir.

Mots-clés : réseaux de neurones artificiels, transport de sources énergétiques, modélisation, apprentissage profond, fusion par confinement inertiel, radiothérapie

---

Study of the use of neural networks for high-performance calculations dedicated to the modeling of the transport of energy sources

Abstract: Whatever their fields of application, modern computation codes are subject to requirements for speed of calculation and optimization of memory space occupation, which necessitate the use of advanced numerical methods. To meet these needs for massive data processing and intensive calculations, deep learning methods offer effective and extremely attractive alternative responses.

This thesis manuscript presents a study on the coupling of artificial neural networks to high performance computing codes dedicated to simulations of complex physical phenomena, more particularly in connection with transport theories. The first application framework concerns the simulation of inertial confinement fusion experiments for the production of energy, and in particular, the modeling of nonlocal electronic heat transport. The second field of application is the transport and deposition of energy from particles in radiotherapy for the treatment of cancers.

The two studies of the coupling of artificial neural networks to high performance computing codes developed by the IFCIA research team at the CELIA laboratory, are presented in this manuscript, and show encouraging results: for precision criteria largely sufficient for these applications, we obtain considerable time savings. This preliminary feasibility study encourages the CELIA team to continue this work in the years to come.

Keywords: artificial neural networks, transport of energy sources, mathematical modelling, deep learning, inertial confinement fusion, radiotherapy

---

Centre Lasers Intenses et Applications
UMR 5107 Université de Bordeaux-CNRS-CEA, 33405 Talence, France.

Table des matières

Introduction

Les enjeux des codes de calcul haute performance

De nos jours, la simulation de phénomènes physiques complexes est devenue un outil essentiel et complémentaire à la théorie, et à la préparation et l'interprétation d'expériences, qui servent à observer et analyser le réel. En effet, lorsque les équations mathématiques qui décrivent les phénomènes physiques sont trop complexes et que les outils théoriques ne sont pas suffisants pour calculer les solutions exactes, la simulation permet d'approcher ces solutions : ceci à été rendu possible grâce aux progrès considérables dans le domaine des méthodes numériques et des moyens de calcul. La simulation permet en outre de diminuer les coûts de production en se substituant à des expériences qui seraient trop coûteuses, voire impossibles (par exemple, en 1996, la France a signé le Traité d'Interdiction Complète des Essais nucléaires).

Ainsi, la simulation de phénomènes physiques implique le développement de codes de calcul où les solutions des équations théoriques sont approchées à l'aide d'outils de calcul scientifique élaborés. Les codes de calcul modernes doivent ainsi être précis, mais aussi performants, en termes de rapidité de calcul et d'occupation mémoire. Pour ce faire, ils bénéficient des nombreux outils numériques performants pour la résolution d'équations, mais aussi de moyens informatiques de parallélisation de codes comme MPI [1] ou OpenMP [2]. L'émergence de supercalculateurs pour le calcul intensif a également permis d'augmenter la complexité des systèmes résolus : par exemple, le supercalculateur EXA1, qui a été inauguré le 13 septembre 2022, possède une puissance de calcul de 23.2 pétaflops (soit 23.2 millions de milliards d'opérations par seconde), et cette capacité sera amenée par la suite à augmenter avec les besoins du Commissariat à l'Énergie Atomique et aux Énergies Alternatives (CEA).

Aujourd'hui, il est indéniable que les progrès scientifiques sont intimement liés aux enjeux sociétaux. Le CEA participe notamment à cet effort, en finançant la recherche en lien avec les sciences de l'atome, et plus particulièrement la question du nucléaire. Le travail présenté dans ce manuscrit s'intéresse à deux cadres d'application en particulier : la fusion par confinement inertiel et la radiothérapie.

La fusion par confinement inertiel pour la production d'énergie

À l'heure de la transition énergétique, le développement de nouvelles sources d'énergie renouvelables est capitale. L'énergie provenant de l'éolien, du solaire, ou encore de l'hydroélectrique, n'est malheureusement pas suffisante pour répondre aux besoins de la population mondiale, qui a presque quadruplé en l'espace d'un siècle. La principale source d'électricité en France provient des centrales nucléaires (en 2020, le nucléaire représente 66.5% de l'électricité consommée en France). Aujourd'hui, l'énergie produite par les centrales nucléaires provient de la fission d'atomes d'uranium. Cette réaction a l'avantage de ne pas produire de CO_2, mais a l'inconvénient de produire des déchets radioactifs. Ce sont

ces déchets qui posent un problème, d'une part du point de vue de leur stockage, et d'autre part des problèmes de sécurité des installations, comme en attestent les catastrophes de Tchernobyl en 1986 et Fukushima en 2011. Une autre approche semble cependant prometteuse et fait l'objet de recherches très actives : la fusion nucléaire contrôlée. Les réactions de fusion sont la source d'énergie des étoiles et leur exploitation sur terre requière le contrôle de conditions thermodynamiques extrêmes. Le développement de lasers de haute puissance rend possible la reproduction de ces phénomènes sur terre.

En effet, une réaction de fusion consiste en l'appariement de deux noyaux légers, produisant un noyau plus lourd. Cependant, il existe une répulsion coulombienne entre les noyaux, empêchant de fait leur rapprochement. Pour le provoquer, il faut vaincre cette répulsion coulombienne en communiquant beaucoup d'énergie aux noyaux. Aux densités d'énergie atteintes, le milieu réactionnel est à l'état plasma. La communauté scientifique s'est accordée sur le choix de deux noyaux légers, le deutérium (D) et le tritium (T), dont l'énergie de barrière de Coulomb est plus faible, facilitant la fusion des noyaux. Ces deux éléments sont également facilement produits : le deutérium se trouve en quantité presque illimitée dans l'eau de mer, et le tritium peut être obtenu par une réaction du lithium, présent dans la croûte terrestre. De la fusion de ces deux noyaux sera engendré un noyau plus lourd : l'hélium, aussi appelé particule α, ainsi qu'un neutron. C'est l'énergie libérée par la réaction de fusion ($\sim$17.6MeV) qui sera extraite pour être utilisée par la suite. L'expérience menée au National Ignition Facility du laboratoire national Lawrence Livermore en Californie le 8 août 2021 constitue une avancée majeure dans la recherche sur la fusion, puisque 1.3 mégajoules d'énergie thermonucléaire ont été récupérés grâce aux 1.9 mégajoules d'énergie produite par 192 lasers [3] : le rendement de cette expérience est donc de 0.7, soit le plus proche du seuil d'ignition, c'est-à-dire (c-à-d) le moment où l'on produit plus d'énergie que l'on en fournit au système, et qui est de 1.

Pour recréer les conditions nécessaires pour la fusion dans un réacteur nucléaire, plusieurs approches existent :

- la fusion par confinement magnétique (ou FCM) [4], où le plasma est confiné par des champs magnétiques dans une boîte "invisible" car il ne rentre pas en contact avec les parois du récipient ;
- la fusion par confinement inertiel (ou FCI) [5], où un combustible est confiné par sa propre inertie (Figure 1)
  - par attaque indirecte : les lasers irradient les parois d'un cylindre creux, appelé hohlraum, entourant la cible de deutérium-tritium, et l'énergie déposée sur la paroi du hohlraum est convertie en rayons X qui vont illuminer la cible et provoquer son implosion ;
  - ou bien par attaque directe : les lasers irradient directement la cible de deutérium-tritium.

Deux autres techniques ont également été développées pour la FCI. Elles visent à séparer la phase de compression de la phase d'allumage. Ce ralentissement de l'implosion va permettre d'atténuer les instabilités hydrodynamiques inhérentes à la FCI, et qui déforment la cible et peuvent alors empêcher l'allumage. Avec ces méthodes, l'intensité des lasers est de fait diminuée, ce qui entraîne une température et une pression relativement basses dans le point chaud de la cible. Une énergie additionnelle est donc nécessaire pour provoquer l'allumage [5] : soit par un faisceau de particules chargées produit par un laser à impulsion courte et intense (on parle de schéma d'allumage rapide), soit par une onde de choc sphérique provoquée par une hausse de l'intensité laser en fin d'impulsion (on parle de schéma d'allumage par choc).

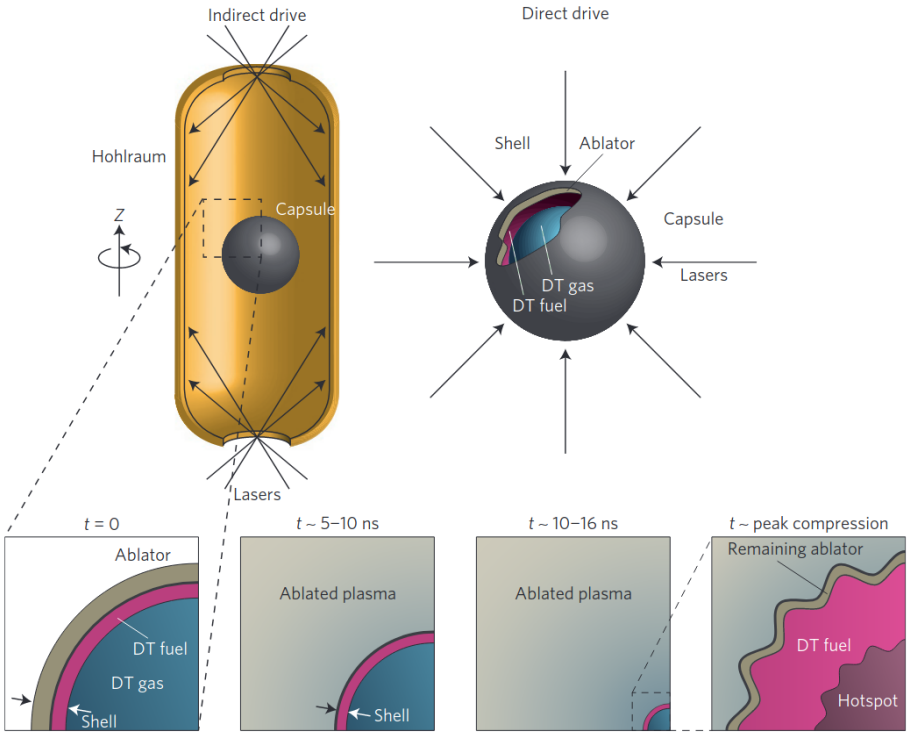


FIGURE 1 – Schémas des allumages en FCI par attaque indirecte (à gauche) et directe (à droite) et graphes de l'évolution hydrodynamique au sein de la cible (en bas). [5]

Une meilleure compréhension et modélisation des processus physiques impliqués dans la réaction de fusion est indispensable pour arriver à faire de cette nouvelle source d'énergie propre une réalité. Les simulations d'expériences de FCI impliquent non seulement d'étudier le transport cinétique collisionnel de plusieurs espèces de particules chargées, mais aussi d'autres phénomènes complexes tels que le transfert thermique, la production de neutrons, ou encore les interactions laser-plasma. Ce travail de thèse s'intéresse à un phénomène particulier qui a lieu lors des expériences de FCI : le transport de chaleur des électrons impliqué dans la conversion de l'énergie du laser en énergie cinétique lors de l'implosion d'une cible [6, 7].

Le transfert d'énergie par les électrons dans un plasma par électron est un processus clé en physique des hautes densités d'énergie (*High-Energy-Density* ou HED) [6–8]. Les plasmas créés dans les expériences de FCI et FCM sont loin de l'équilibre thermodynamique local [9, 10]. De ce fait, le transfert d'énergie par transport d'électrons ne peut pas être décrit par une loi de Fourier dépendant uniquement des conditions locales, car le libre parcours moyen des électrons peut être long par rapport à la longueur du gradient de température. Le transport est donc non local, ce qui signifie que le flux d'énergie à une position donnée peut dépendre de conditions thermodynamiques distantes. Ce problème est connu depuis les années 1980 [11], mais le calcul précis du flux non local à de grandes échelles spatiales et temporelles reste un défi. Plusieurs modèles macroscopiques approximatifs ont été développés sur la base des caractéristiques cinétiques du flux non local [12–14], et certains d'entre eux sont implémentés dans des codes hydrodynamiques, permettant ainsi une modélisation plus précise des expérimentations [15–17]. C'est un tel modèle, le modèle non local de Schurtz-Nicolaï-Busquet [13], qui est implémenté dans le code hydrodynamique CHIC [18] dans sa version magnétisée de Nicolaï-Feugeas-Schurtz [19], développé par l'équipe IFCIA du CELIA et considéré dans ce travail de thèse. Ce modèle a démontré

son efficacité depuis de nombreuse années dans de nombreux codes, en France (CHIC [18], FCI2 [20]) et à l'étranger (DUED [21] en Italie, PINOCO2D [22] au Japon, HYDRA [23] et DRACO [24] aux États Unis). L'objectif dans ce travail de thèse fut de remplacer le module de transport électronique non-local du code CHIC – grâce auquel nous avons pu générer dans un premier temps une base de données conséquente et maîtrisée - par un réseau de neurones artificiels.

La radiothérapie pour le traitement des cancers

La radiothérapie est un moyen de traitement des cancers utilisant des rayonnements ionisants (photons, électrons, etc.), afin de détruire les cellules cancéreuses, tout en préservant le mieux possible les tissus sains. Il s'agit d'une méthode très utilisée dans les traitements des cancers, puisqu'on estime que la moitié des patients atteints d'un cancer sont traités avec cette technique, souvent en complément d'autres traitements, comme l'hormonothérapie, la chimiothérapie, et enfin la chirurgie. Le terme de rayonnement désigne le faisceau de particules énergétiques qui va parcourir le corps du patient jusqu'à atteindre la zone à traiter. Lors de ce trajet, les particules du faisceau interagissent avec le milieu dans lequel elles se déplacent. Ces particules dites incidentes (ou particules injectées) peuvent entrer en collision avec les particules du milieu lors de ce trajet, et il y a alors un transfert d'énergie. Lors d'une collision, une partie de l'énergie des particules transportées peut ainsi être transférée vers les particules constituant le milieu : c'est ce dépôt d'énergie qui constitue ce que l'on appelle la dose, exprimée en $\mathrm{J.kg^{-1}}$ ou encore en gray ($1\mathrm{Gy} = 1\ \mathrm{J.kg^{-1}}$). Ce phénomène a des effets biologiques sur les milieux vivants, comme provoquer la dégradation voire la mort cellulaire [25–27]. Suivant le type de tumeur, différents traitements par rayons ionisants peuvent être employés, notamment la radiothérapie externe, où la source de rayons est à distance du patient.

L'administration d'une dose en radiothérapie est donc définie par trois facteurs : la dose par séance d, le nombre de séances n (soit $D = n \times d$) et la durée totale ou étalement du traitement (t). Du fait du fractionnement de la dose, les cellules irradiées vont se réparer entre deux séances. Fort heureusement, la capacité de régénération des cellules saines est plus élevée que celle des cellules cancéreuses : c'est sur cet effet différentiel entre tissus sains et tumoraux que repose l'efficacité de la radiothérapie [25, 26]. De plus, la destruction de la tumeur par rayons ionisants doit se faire de telle manière que les effets indésirables sur les tissus sains voisins soient limités. Pour ce faire, la balistique optimale de l'irradiation (i.e. le nombre de faisceaux, leurs placements, leur intensité, etc.) doit être déterminée par le Treatment Planning System ou TPS [26, 28] (Figure 7.4b).

Il existe aujourd'hui de nombreux outils de calcul de dose pour l'aide à la planification de traitement en radiothérapie, et deux grandes familles de modèles sont disponibles aujourd'hui dans les TPS. La première est la famille des méthodes Monte Carlo [29], qui résolvent les équations cinétiques de transport de Boltzmann de manière statistique. Les modèles probabilistes, tels que FLUKA [30] ou encore GEANT4 [31], sont capables de faire un calcul de référence proche de la solution exacte de la distribution de dose, même pour des géométries arbitrairement hétérogènes, mais leurs temps de calcul restent élevés. La deuxième famille est celle des modèles basés sur les noyaux (*kernel based*), qui offrent une alternative rapide et robuste aux algorithmes de Monte Carlo. Parmi cette famille, les modèles de *pencil beam* sont les plus répandus [32–34]. Mais, là encore, les modèles sont limités, cette fois-ci à cause des géométries auxquelles ils peuvent être appliqués. Un troisième moyen de calcul de dose a émergé ces dernières années : les modèles déterministes [35, 36], qui reposent sur l'introduction des moments (semblables à des moyennes)

successifs de la fonction de distribution, solution des équations de Boltzmann. Ils sont donc capables de modéliser la totalité des interactions physiques ayant lieu pendant le transport des particules dans le milieu, en un temps réduit par rapport aux méthodes de Monte Carlo [37]. C'est à un un tel modèle que l'on s'intéresse ici : le modèle aux moments M_1, développé par le groupe de recherche IFCIA du CELIA [38, 39] et implémenté dans le code KIDS. Dans le cadre de ce travail de thèse, l'objectif fut d'associer un réseau de neurones artificiels à ces méthodes déjà performantes pour en améliorer encore les performances, en termes de temps de calcul.

Brève histoire de l'Intelligence Artificielle et du *Deep Learning*

La conférence de Dartmouth (Dartmouth Summer Research Project on Artificial Intelligence) organisée durant l'été 1956 au Darmouth College par Marvin Minsky et John McCarthy, est considérée comme l'acte de fondateur de l'Intelligence Artificielle (IA). Cette discipline concerne les machines et ordinateurs capables de reproduire des actions jusque-là propres à l'humain. Il est important de distinguer le *Deep Learning* - qui correspond aux réseaux de neurones artificiels profonds - et le Machine Learning (ML) ou encore l'IA qui englobent des notions beaucoup plus vastes. La question même de ce qu'est une IA a fait naître deux idéologies : l'IA symbolique et l'IA connexionniste. La première a eu plus de succès et d'avancées dans les premières années de l'évolution de l'IA, tout d'abord avec le concept de machine universelle d'Alan Turing en 1937, puis le premier agent conversationnel Eliza développé par Joseph Weizenbaum en 1963, et enfin avec l'avènement des systèmes experts et la programmation logique en 1982, au terme du premier "hiver de l'IA" qui aura duré une dizaine d'années. Les co-fondateurs de l'IA font partie de ce mouvement de pensée pour lequel une machine intelligente se base sur les connaissances déjà connues et transmises par l'homme pour résoudre des problèmes.

À l'opposé, l'IA connexionniste se base sur des exemples de solutions de problèmes afin d'interpoler des solutions pour de nouveaux problèmes, par des procédés statistiques. Cette deuxième vision de l'IA s'inspire du fonctionnement biologique du corps humain et des neurones. Cette piste est présentée une première fois en 1943 par Warren McCulloch et Walter Pitts, dont les travaux portaient sur une description du fonctionnement des neurones du cerveau humain à l'aide de circuits électriques [40]. C'est en 1957 que le mouvement voit son premier succès, avec le perceptron de Frank Rosenblatt et le premier algorithme d'apprentissage [41]. Le mouvement est vivement critiqué et s'ensuivent alors plusieurs décennies durant lesquelles les financements de la recherche en IA sont gelés, et les limites des puissances de calcul et de mémoire des machines disponibles à l'époque limitent toute avancée. C'est finalement en 2003 que le mouvement est relancé, à l'initiative de trois chercheurs, Geoffrey Hinton, Joshua Bengio et Yann LeCun. Ils développent les réseaux de neurones dits profonds, et en 2012, c'est le réseau convolutif profond AlexNet [42] qui remporte le concours de reconnaissance d'images ImageNet [43]. Aujourd'hui, les deux approches sont encore très actives et continuent de s'opposer.

Même si les applications principales du *Deep Learning* concernent le traitement d'images (segmentation, classification), depuis plusieurs années, le *Deep Learning* est aussi de plus en plus présent dans des applications de résolution d'équations et de simulation, notamment avec l'émergence d'une nouvelle classe de réseaux appelés *Physics Informed Neural Networks* [44]. Si l'un des challenges du développement de code de calcul haute performance est la diminution de leur temps d'exécution, alors les réseaux de neurones artificiels pourraient s'avérer des outils de calcul puissants. En effet, une fois entraînés, les performances des réseaux de neurones artificiels s'expriment en secondes, voir en millisecondes,

en faisant ainsi des outils capables d'être utilisés en temps réel, comme par exemple pour la détection d'obstacle par des robots autonomes [45].

Problématique de thèse et plan détaillé du manuscrit

La problématique du travail présenté dans ce manuscrit est la suivante :

Un réseau de neurones artificiels peut-il se substituer à un modèle physique dans un code de calcul haute performance, et par quels moyens ?

L'objectif de ce travail est donc d'explorer les performances du remplacement, partiel ou total, d'un module de physique complexe présent dans des codes de calcul de production ou d'étude développés au CELIA (CHIC et KIDS) par un réseau de neurones artificiels.

Ce manuscrit de thèse est organisé en quatre parties. Tout d'abord, la première partie du manuscrit présente un état de l'art sur l'apprentissage profond : les notions de base des réseaux de neurones artificiels sont introduites au Chapitre 1, avec les principes de fonctionnement de deux types de réseaux (réseaux denses et réseaux convolutifs), et les techniques d'apprentissage ; puis l'émergence récente de ces outils dans les simulations numériques est discutée au Chapitre 2, avec d'abord une présentation des *Physics Informed Neural Networks* et des librairies informatiques de *Deep Learning*. Puis, la deuxième partie du manuscrit présente un état de l'art sur les deux théories de transport évoquées précédemment : le Chapitre 3 présente le code CHIC ainsi que la théorie du transport de chaleur non local pour la fusion par confinement inertiel, et le Chapitre 4 présente le code KIDS ainsi que la théorie du transport de particules pour le calcul de dose en radiothérapie. Ensuite, la troisième partie du manuscrit présente les premières analyses exploratoires sur les réseaux de neurones, effectuées au début de la thèse : le Chapitre 5 montre les résultats des premières utilisations de RNA, d'abord pour des problèmes de classification binaire, puis pour la résolution de l'équation de la chaleur 1D par des PINN, et enfin, pour la modélisation du flux de chaleur non local de Luciani-Mora-Virmont [12] ; le Chapitre 6 montre une réflexion sur le lien théorique entre deux fonctions clés pour l'apprentissage des réseaux. Enfin, la quatrième et dernière partie de ce manuscrit présente les principaux résultats sur le couplage de réseaux de neurones artificiels à des codes de calcul haute performance : le Chapitre 7 présente les résultats de l'apprentissage et du couplage de réseaux de neurones artificiels denses au code CHIC pour des simulations 1D d'une part, et 2D d'autre part, ainsi que les premiers résultats pour des réseaux convolutifs ; le Chapitre 8 présente les résultats de l'apprentissage et du couplage de réseaux de neurones artificiels denses au code KIDS, ainsi qu'une analyse sur le temps d'exécution du code développé durant la thèse. Ce manuscrit termine par une conclusion sur tous les résultats présentés, et expose les perspectives qui en découlent.

Première partie

Apprentissage profond

Chapitre 1

Théorie des réseaux de neurones artificiels

Ce chapitre introduit les notions de base sur les réseaux de neurones utilisées dans la suite de ce manuscrit.

1.1 Introduction : de l'Intelligence Artificielle à l'Apprentissage Profond

1.1.1 Les techniques d'apprentissage automatique

Le Machine Learning, ou apprentissage automatique, est une sous-branche de l'IA qui repose sur l'exploitation de données afin qu'un algorithme s'améliore avec l'expérience, c'est-à-dire qu'il apprenne. Ce domaine est très large et on y trouve de nombreux algorithmes, tels que les arbres de décision, qui produisent une prédiction tout en restant compréhensible pour l'utilisateur (en opposition aux réseaux de neurones, souvent qualifiés de boîtes noires), et qu'on retrouve dans de nombreux domaines comme la médecine ou l'exploitation de données. Un autre exemple est le clustering ou partitionnement de données : les données sont divisées en groupes selon leur ressemblance, comme pour la recommandation de films ou de musique.

Les réseaux de neurones artificiels font également partie des algorithmes de ML. Le terme de *Deep Learning* fait référence à la profondeur des réseaux de neurones et à leur capacité d'apprentissage automatique. Il existe en réalité de nombreux types de réseaux de neurones, en plus des réseaux profonds classiques, comme les réseaux convolutifs, spécialisés dans le traitement d'images basé sur l'application de filtres (par exemple la recherche de contours pour reconnaître un objet spécifique), ou encore les réseaux récurrents, spécialisés dans l'analyse de séries temporelles et ainsi très utilisés en analyse de texte et reconnaissance de parole.

L'apprentissage automatique s'appuie donc sur l'expérience et l'exploitation de données. Il existe de fait plusieurs méthodes d'apprentissage, mais les plus communément utilisées sont les suivantes :

. l'apprentissage supervisé : principalement utilisée pour les réseaux de neurones ; le réseau apprend à l'aide d'un jeu de données comprenant des entrées et des sorties, ou cibles (par exemple, dans la reconnaissance de chiffres, les entrées sont les images et les sorties sont les chiffres à identifier) ; le but de l'apprentissage est alors de réussir à faire une prédiction la plus proche possible de la cible ;

. l'apprentissage par renforcement : au lieu de données d'apprentissage, l'algorithme est directement en contact avec un environnement (par exemple un robot qui doit marcher sur un sol accidenté) et va apprendre de ses actions dans cet environnement et les mémoriser (le mouvement de jambe qui permet au robot d'avancer est perçu comme positif, celui qui le fera tomber comme négatif) ;
. l'apprentissage non supervisé : l'algorithme est uniquement fourni en données d'entrée et doit extraire de lui-même de l'information afin de partitionner les données et d'en extraire une structure sous-jacente ; le *clustering* est la méthode d'apprentissage non supervisé la plus répandue, et dans laquelle la machine trouve des points communs entre des données d'un groupe hétérogène et identifie d'elle-même des sous-classes de données homogènes ; il existe également des réseaux de neurones, appelés auto-encodeurs, qui utilisent cette méthode, et dont le but est le plus souvent de réduire la dimension des données ou de les débruiter.

C'est la méthode d'apprentissage supervisé qui a été utilisée durant le travail de thèse. Elle est décrite de manière plus détaillée dans la suite de ce chapitre.

1.1.2 Du neurone biologique au neurone artificiel

L'IA connexionniste s'est inspiré du neurone biologique, dont le fonctionnement peut se résumer en quatre éléments (figure 1.1) :

. les dendrites, qui reçoivent l'information d'autres neurones,
. le corps cellulaire et son noyau, qui sont responsables du traitement de l'information reçue grâce aux dendrites,
. l'axone, par lequel l'information du neurone est transportée pour ensuite être transmise par la suite à d'autres neurones,
. les terminaisons neuronales (ou synapses), qui permettent de connecter l'axone au dendrites d'autres neurones pour leur transmettre une information.

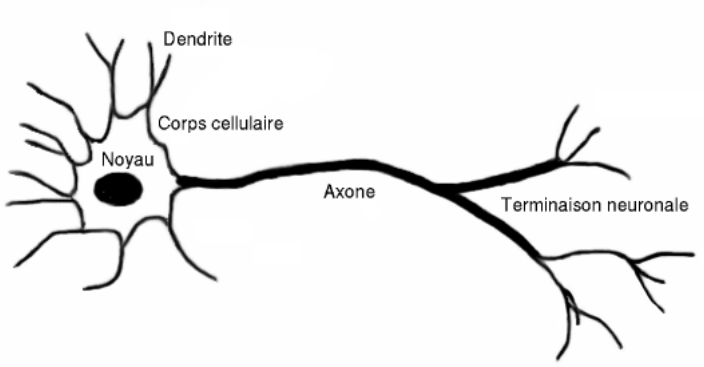


FIGURE 1.1 – Schéma d'un neurone biologique

De manière similaire, le perceptron (figure 1.2) est défini par :

$$\widehat{y} = H(z) = \begin{cases} 1 \text{ si } z \geq 0, \\ \\ 0 \text{ sinon,} \end{cases} \tag{1.1.1}$$

$$z = \sum_{i=1}^{n} w_i x_i + b, \tag{1.1.2}$$

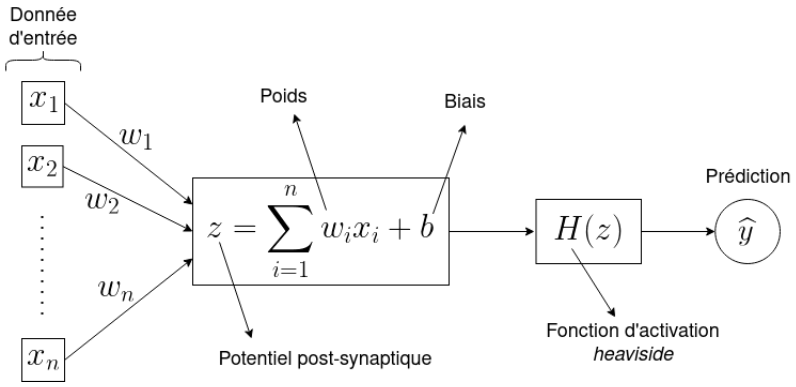


FIGURE 1.2 – Schéma d'un perceptron

où

- $(x_i)_{1 \leq i \leq n}$ sont les n dimensions de la donnée d'entrée, qu'on peut assimiler aux dendrites du neurone biologique ;
- $(w_i)_{1 \leq i \leq n}$ sont les poids du perceptron et b son biais, que l'on peut assimiler aux terminaisons neuronales ;
- z est le potentiel post-synaptique et H est la fonction d'activation *heaviside* (ou *threshold*), que l'on peut assimiler au corps cellulaire ;
- $\widehat{y}$ est la prédiction (ou sortie), que l'on peut assimiler à l'axone.

La sortie du perceptron rend compte du signe du potentiel post-synaptique : si ce dernier est négatif, alors la sortie vaut zéro, mais si elle est positive ou nulle, alors la sortie vaut 1. Le perceptron est de ce fait un classificateur binaire, du fait que sa sortie ne puisse prendre que deux valeurs distinctes.

L'apprentissage du perceptron, et de manière plus générale des réseaux de neurones artificiels, s'appuie sur l'ajustement des poids et du biais. S'agissant d'un apprentissage dit supervisé, si l'on note y la sortie cible, et $\widehat{y}$ la sortie prédite par le perceptron, la règle d'apprentissage du perceptron s'écrit alors :

$$w_i \longleftarrow w_i + \alpha(y - \widehat{y})x_i, \tag{1.1.3}$$

où α est un taux d'apprentissage, s'apparentant au pas de descente d'une descente de gradient classique. Nous verrons plus tard dans ce chapitre pourquoi cette analogie est faite ici.

Le perceptron est de fait un agent intelligent trop rudimentaire pour traiter des problèmes plus complexes : pour s'attaquer à ces problèmes, il faut organiser plusieurs neurones dans ce que l'on appelle un réseau.

1.2 Architectures de réseaux neuronaux

1.2.1 Les réseaux à propagation avant

Dès lors que l'on connecte plusieurs neurones entre eux, on parle de réseau de neurones. C'est la manière d'organiser ces neurones entre eux qui va déterminer l'architecture du réseau. Les réseaux dits denses, également appelés perceptrons multicouches (*Multi Layer Perceptrons* ou MLP) forment l'architecture la plus élémentaire. On les appelle aussi les réseaux de neurones à propagation avant (*Feedforward Neural Networks* ou FNN), à cause de la manière dont l'information se propage de la couche d'entrée à la couche de sortie.

Dans ces réseaux, le fonctionnement de chaque neurone est à l'image de celui du perceptron, à cela près qu'au lieu d'une fonction *heaviside*, on utilise une fonction continue dérivable, communément appelée fonction d'activation. On considère le neurone j de la couche $\mathcal{C}^l$: l'activation du neurone (i.e. sa sortie), notée a_j^l, est calculée de telle sorte que

$$a_j^l = g(z_j^l), \tag{1.2.1}$$

$$z_j^l = \sum_{i=1}^{n} w_{ij}^l a_i^{l-1} + b_j^l, \tag{1.2.2}$$

où
- z_j^l est le potentiel post-synaptique du neurone j de la couche $\mathcal{C}^l$,
- w_{ij}^l est le poids reliant le neurone i de la couche $\mathcal{C}^{l-1}$ au neurone j de la couche $\mathcal{C}^l$,
- b_j^l est le biais du neurone j de la couche $\mathcal{C}^l$,
- g est une fonction d'activation continue.

Le tableau 1.1 donne une liste non exhaustive de fonctions d'activations et leurs dérivées. On verra plus tard dans ce chapitre que certaines sont associées en dernière couche à une fonction perte selon le type de problème à résoudre.

Nom	Équation	Dérivée
Heaviside ou *threshold*	$h(z_j^l) = \begin{cases} 1 \text{ si } z_j^l \geq 0 \\ 0 \text{ si } z_j^l < 0 \end{cases}$	$h'(z_j^l) = \delta(z_j^l)$
Sigmoïd	$\sigma(z_j^l) = \frac{1}{1+e^{-z_j^l}}$	$\sigma'(z_j^l) = \sigma(z_j^l)(1-\sigma(z_j^l))$
Tangente hyperbolique	$\tanh(z_j^l) = \frac{e^{z_j^l}-e^{-z_j^l}}{e^{z_j^l}+e^{-z_j^l}}$	$\tanh'(z_j^l) = 1-\tanh(z_j^l)^2$
Softmax	$s(z_j^l) = \frac{e^{z_j^l}}{\sum_{k\in\mathcal{C}^l} e^{z_k^l}}$	$s'(z_j^l) = s(z_j^l)(1-s(z_j^l))$
Rectified Linear Unit (ReLU)	$ReLU(z_j^l) = \begin{cases} z_j^l \text{ si } z_j^l \geq 0 \\ 0 \text{ si } z_j^l < 0 \end{cases}$	$ReLU'(z_j^l) = \begin{cases} 1 \text{ si } z_j^l \geq 0 \\ 0 \text{ si } z_j^l < 0 \end{cases}$
Identité	$I(z_j^l) = z_j$	$I'(z_j^l) = 1$

Tableau 1.1 – Quelques fonctions d'activations et leurs dérivées

Dans un FNN, les neurones sont donc organisés par couches, et l'information est transportée de la première à la dernière couche. Comme illustré Figure 1.3, il existre trois types de couches :
- la couche d'entrée : c'est là qu'est insérée la donnée d'entrée, ce ne sont donc pas à proprement parler des neurones qui constituent cette couche, puisqu'il n'y a ni fonction d'activation ni pondération de l'information ;

- la couche de sortie : les sorties des neurones de cette couche correspondent à la prédiction du réseau, que l'on note $\widehat{y}$ (ou $(\widehat{y}_j)_j$ si la sortie fait plus d'1 dimension), et nous verrons également par la suite que la fonction d'activation de cette couche est choisie en fonction du type de sortie cible ;
- les couches cachées.

De manière générale, lorsque l'on dénombre le nombre de couches d'un réseau, on ne compte pas la couche d'entrée.

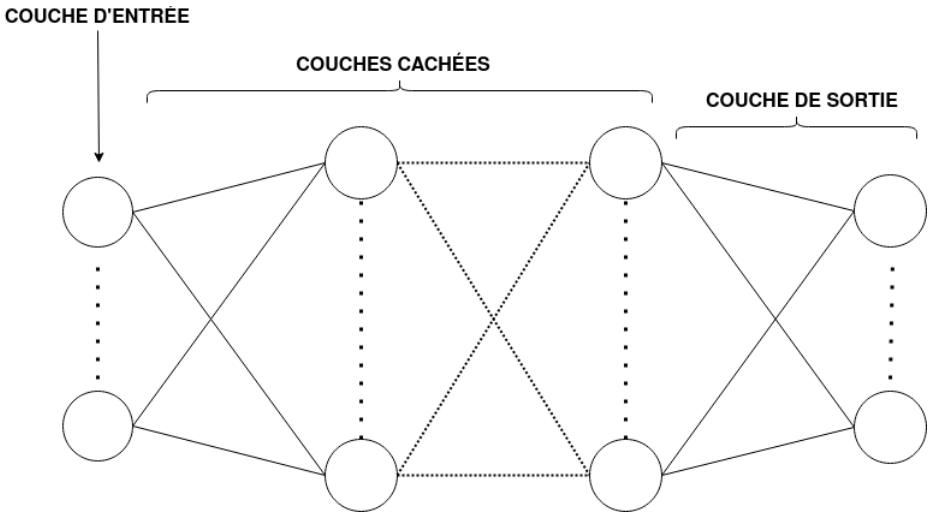


FIGURE 1.3 – Schéma d'un réseau à propagation avant.

De part son architecture, un FNN ne traite les données d'entrée et de sortie que sous la forme de vecteurs. Lorsque l'on traite une donnée en 2 dimensions, comme par exemple une image, on perd donc toute notion d'espace. L'architecture présentée dans la prochaine section est, elle, capable de conserver cette notion d'espace en 2 dimensions : ce sont les réseaux dits convolutifs.

1.2.2 Les réseaux convolutifs

Les réseaux convolutifs (Convolutional Neural Networks ou CNN) ont la particularité de conserver la forme des données en 2 dimensions de type image [46, 47]. Comme leur nom l'indique, ces réseaux sont basés sur l'utilisation de l'opération de convolution, au lieu de la multiplication matrice-vecteur qu'on trouve chez les FNN. Ils gardent les autres fonctionnalités des FNN, à savoir l'utilisation de fonctions d'activation et la même règle d'apprentissage, qui sera explicitée dans la prochaine partie de ce chapitre.

Dans l'exemple Figure 1.4, on a défini un noyau de convolution de hauteur $h_n = 3$ et de largeur $l_n = 3$ (à noter que ce noyau contient en fait des poids). Dans cet exemple de convolution, le noyau parcourt l'image avec un pas $p_l = 1$ dans le sens longitudinal et $p_h = 1$ dans le sens transversal. Pour chaque morceau d'image de la même taille, on effectue un produit élément à élément, appelé produit de Hadamard. Le résultat de tous ces produits forme une nouvelle image, appelée Z par analogie avec le potentiel post-synaptique d'un réseau dense. Pour toute convolution, on peut calculer les dimensions $l_{out} \times h_{out}$ de l'image en sortie, en fonction des dimensions de l'image d'entrée de taille $l_{in} \times h_{in}$, de celles du noyau de convolution de taille $l_n \times h_n$, et des pas (p_l, p_h) :

$$l_{out} = ceil\left(\frac{l_{in}-l_n+1}{p_l}\right) \quad \text{et} \quad h_{out} = ceil\left(\frac{h_{in}-h_n+1}{p_h}\right),$$

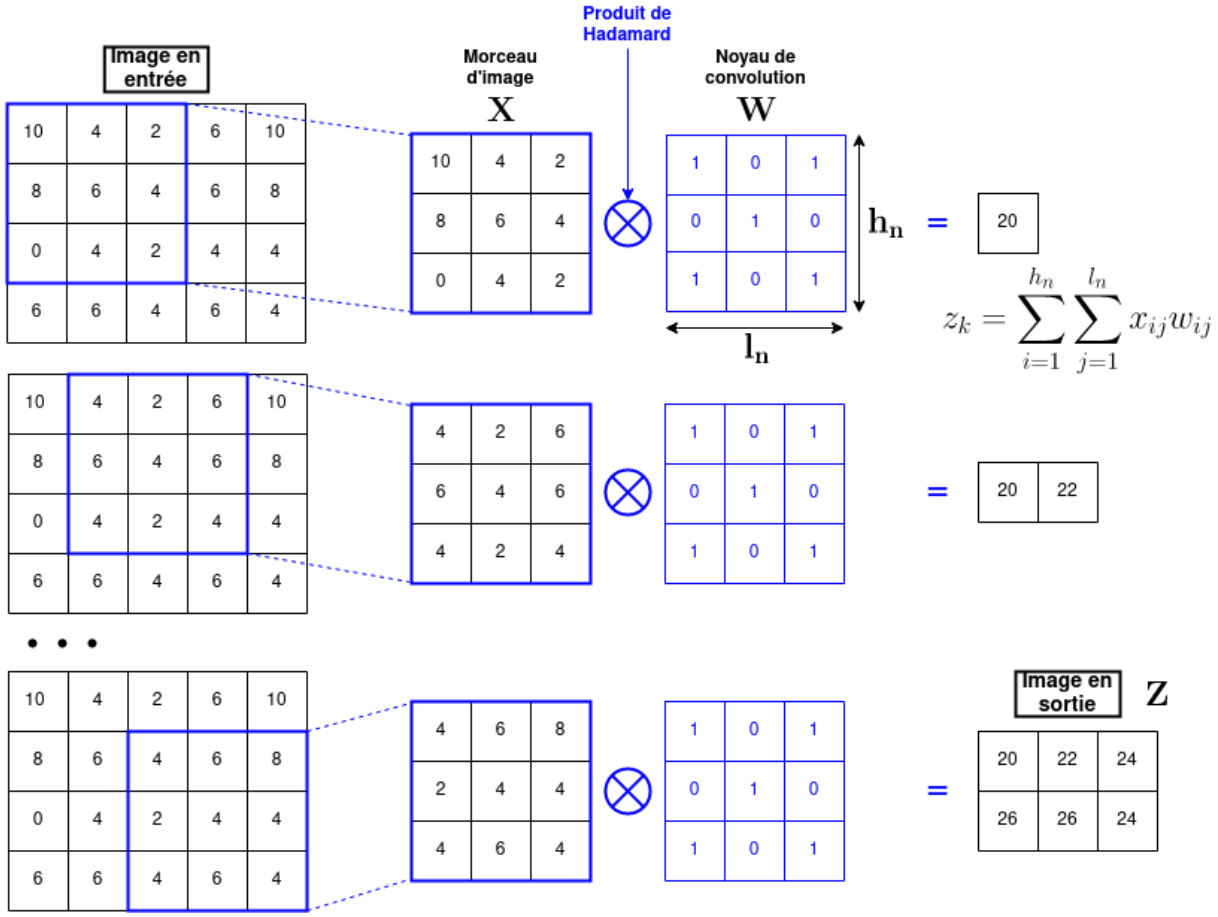


FIGURE 1.4 – Schéma de l'opération de convolution dans un CNN.

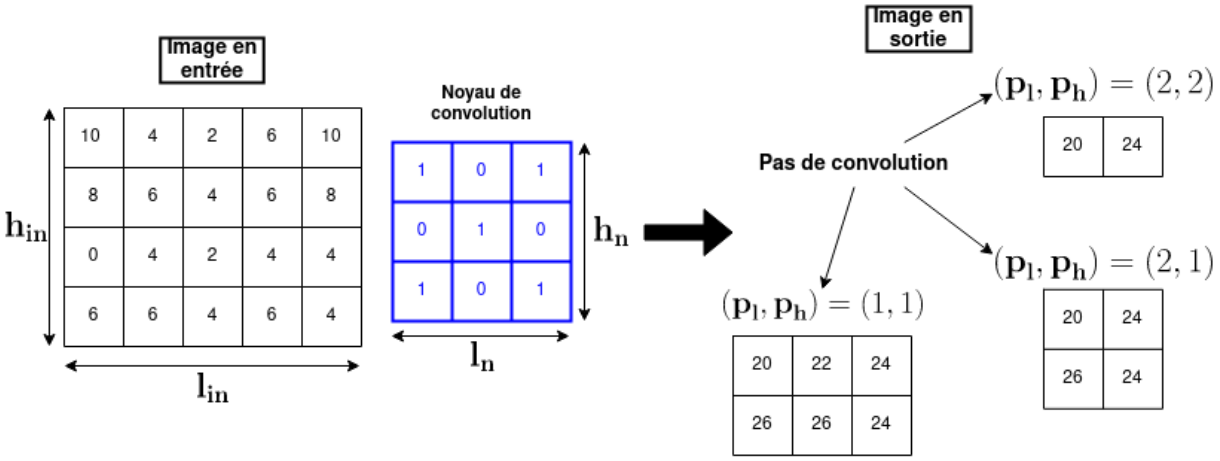


FIGURE 1.5 – Exemples de résultats d'une convolution sur une image de taille 5×4 avec un noyau de convolution de taille 3×3, pour différents pas (p_l, p_h).

où *ceil* désigne l'arrondi au supérieur. La Figure 1.5 montre trois exemples de résultats pour une image de taille 5×4 et un noyau de convolution de taille 3×3, avec les pas (1,1), (2,2) et (2,1). Avec la définition algébrique précédente, on trouve bien que les tailles d'image en sortie sont respectivement 3×2, 2×1 et 2×2.

On appelle cette convolution la convolution sans *padding* ou avec un *valid padding* : le noyau est déplacé sur l'image d'entrée de telle sorte que seules les valeurs de l'image sont prises en comptes, et qu'au besoin, des valeurs de l'image peuvent même être négligées.

C'est le cas dans l'exemple Figure 1.5 avec un pas (2,2) : du fait que $p_h = 2$, la dernière ligne de l'image a été négligée. Cet effet peut être indésirable dans le cas où l'information au bord de l'image serait importante et ne doit pas être omise.

Pour y remédier, on peut mettre en place un *zero padding* autour de l'image afin de n'omettre aucune valeur. Comme son nom l'indique, ce *padding* consiste à ajouter des zéros autour de l'image (des mailles "fantômes" en sorte). Ici encore, on peut calculer quelles seront les dimensions de l'image en sortie, ainsi que le nombre de lignes h_{pad} et de colonnes l_{pad} qu'il faudra ajouter à l'image en entrée :

$$
\begin{aligned}
l_{out} &= ceil\left(\frac{l_{in}}{p_l}\right) \text{ et } h_{out} = ceil\left(\frac{h_{in}}{p_h}\right), \\
l_{pad} &= max\left((l_{out}-1)p_l + l_n - l_{in}, 0\right), \\
h_{pad} &= max\left((h_{out}-1)p_h + h_n - h_{in}, 0\right).
\end{aligned}
$$

Si le nombre de ligne (colonne respect.) à ajouter est pair, alors elles sont distribuées de manière égale de part et d'autre de l'image. Si ce nombre est impair, alors la ligne (colonne respect.) supplémentaire sera ajoutée en bas (à droite respect.) de l'image. Autrement dit :

$$
\begin{aligned}
l_{pad,g} &= floor(l_{pad}/2) \text{ et } l_{pad,d} = l_{pad} - l_{pad,g}, \\
h_{pad,h} &= floor(h_{pad}/2) \text{ et } h_{pad,b} = h_{pad} - l_{pad,h},
\end{aligned}
$$

où $floor$ désigne l'arrondi à l'inférieur, $l_{pad,g}$ et $l_{pad,d}$ désignent les colonnes de *padding* à gauche et à droite respectivement, et $h_{pad,h}$ et $h_{pad,b}$ désignent les lignes de *padding* en haut et en bas respectivement. Un exemple de *zero padding* est montré Figure 1.6. Avec les pas (1,1) et (2,1), on trouve en effet que $l_{pad} = 2$ et $h_{pad} = 2$ (cadrant vert), et avec le pas (2,2), on trouve que $l_{pad} = 2$ et $h_{pad} = 1$.

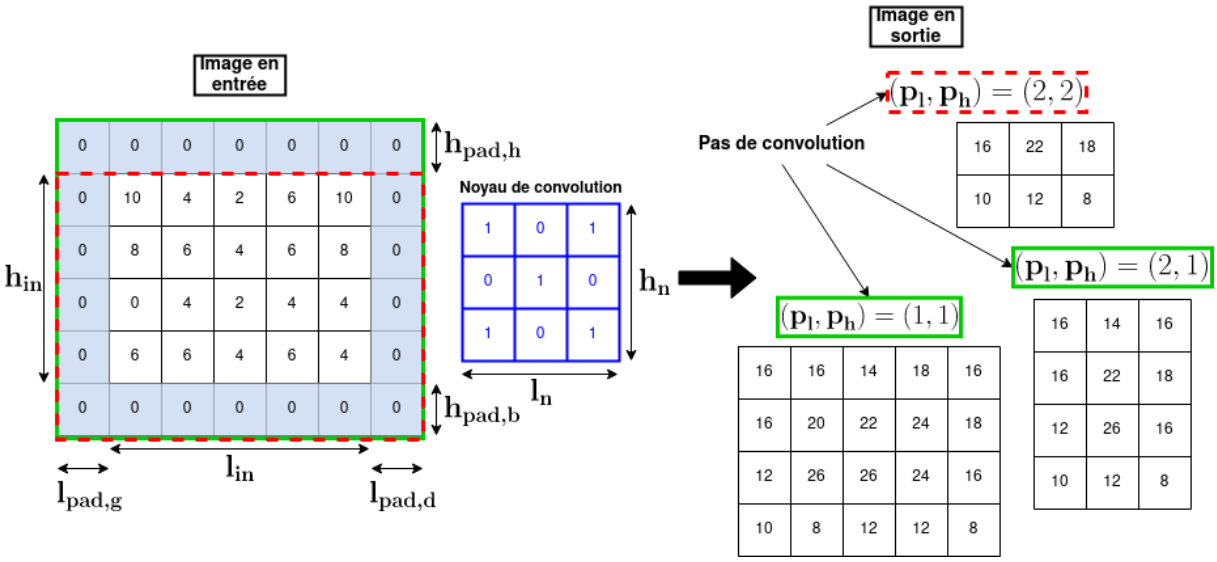


FIGURE 1.6 – Exemples de résultats d'une convolution sur une image de taille 5×4 avec un noyau de convolution de taille 3×3 et un *zero padding*, pour différents pas (p_l, p_h).

Pour simplifier les exemples présentés ici, les noyaux de convolutions sont définis comme des matrices à 2 dimensions, alors qu'en réalité ils en ont 3 : cette troisième dimension correspond au nombre de canaux d'entrée (ou *input channels)*. Les exemples montrés ici n'admettent qu'une seule image en entrée. On peut imaginer qu'au lieu d'une

image en noir et blanc, l'image d'entrée soit en couleur, c'est-à-dire qu'il y aurait 3 cannaux d'entrée correspondant aux couleurs de base (rouge-bleu-vert). De plus, de manière générale, une couche de convolution dans un CNN est constituée de plusieurs noyaux, qui produisent chacun une image (qui passe également par la fonction d'activation), et ce sont ces images qui deviennent à leur tour l'entrée de la couche de convolution suivante.

1.3 Apprentissage des réseaux de neurones

1.3.1 Minimisation de la fonction perte et descente de gradient

L'apprentissage des réseaux de neurones est basé sur la minimisation d'une fonction dite fonction perte (ou *loss*), qui mesure une certaine distance, ou erreur, entre la cible et la prédiction, notées respectivement y^d et $\widehat{y}^d$ pour une donnée d quelconque. En effet, le but de l'apprentissage est de réduire l'écart (l'erreur) entre les prédictions et les cibles, c'est-à-dire de trouver un minimum, au moins local, de la fonction perte. Le tableau 1.2 présente une liste non exhaustive de fonctions perte et leurs dérivées.

Nom	Équation	Dérivée
Entropie binaire	$loss(y, \widehat{y}) = -y \log(\widehat{y}) - (1-y)\log(1-\widehat{y})$	$\frac{\partial loss}{\partial \widehat{y}} = \frac{-(y-\widehat{y})}{\widehat{y}(1-\widehat{y})}$
Entropie catégorique	$loss(y, \widehat{y}) = \sum_j -y_j \log(\widehat{y}_j)$	$\frac{\partial loss}{\partial \widehat{y}_k} = \frac{-y_k}{\widehat{y}_k}$
Erreur quadratique	$loss(y, \widehat{y}) = \frac{1}{2}\sum_j (\widehat{y}_j - y_j)^2$	$\frac{\partial loss}{\partial \widehat{y}_k} = \widehat{y}_k - y_k$

Tableau 1.2 – Quelques fonctions pertes (ou *loss*) et leurs dérivées

La méthode sur laquelle sont basés tous les algorithmes d'apprentissage modernes (aussi appelés optimiseurs) n'est autre que la descente de gradient couramment utilisée en calcul numérique. L'algorithme du gradient s'appuie sur le fait qu'un minimum local de la fonction objectif J est atteint lorsque sa dérivée est nulle. L'algorithme itératif part d'un point θ aléatoire, et à chaque itération, le paramètre θ est mis à jour en prenant comme direction de descente l'opposé de la dérivée de J, multiplié par un pas de descente α bien choisi, jusqu'à ce qu'un certain critère d'arrêt soit atteint. On peut ainsi écrire la formule générale :

$$\theta \longleftarrow \theta - \alpha \frac{\partial J(\theta)}{\partial \theta}. \tag{1.3.1}$$

Le but de l'apprentissage d'un réseau est en réalité de minimiser la perte totale $\mathcal{L}_T$, qui est une moyenne des *loss* sur l'ensemble des données d'apprentissage $\mathcal{D}_a$, c'est-à-dire :

$$\mathcal{L}_T = \frac{1}{card(\mathcal{D}_a)} \sum_{d=1}^{card(\mathcal{D}_a)} loss(y^d, \widehat{y}^d). \tag{1.3.2}$$

Dans le cas de l'apprentissage d'un réseau, on a donc besoin de mettre à jour chaque paramètre θ (les poids et biais) à l'aide de la dérivée de la perte totale. Autrement dit, si l'on écrit la descente de gradient définie par l'équation (1.3.1) pour la perte totale $\mathcal{L}_T$:

$$\theta \leftarrow \theta - \alpha \frac{1}{card(\mathcal{D}_a)} \sum_{d=1}^{card(\mathcal{D}_a)} \frac{\partial}{\partial \theta} loss(y^d, \widehat{y}^d). \tag{1.3.3}$$

Le principal avantage de cet algorithme est que le faible nombre de mises à jour des paramètres engendre un gradient plus stable, ce qui permet une convergence plus stable elle aussi, pour certains problèmes. Cet algorithme se prête également à une implémentation parallélisée. Cependant, cette formulation implique que l'on doit faire un calcul de dérivée pour chaque donnée d'apprentissage et chaque paramètre du réseau. S'agissant de *Deep Learning*, où les réseaux possèdent de nombreux paramètres et les bases de données sont très grandes de part la taille et le nombre de données, le coût de calcul devient trop important et l'apprentissage sera alors très lent. La stabilité du gradient peut également provoquer une convergence prématurée vers un minimum local qui serait mauvais par rapport à d'autres. De plus, l'implémentation de cet algorithme implique que toute la base de données doit être accessible en mémoire.

Le gradient stochastique (*Stochastic Gradient Descent* ou SGD) est une variation de la descente de gradient où seule la dérivée d'une donnée aléatoire est utilisée pour la mise à jour des paramètres. Si une itération correspond à mise à jour des paramètres, on appelle *epoch* l'ensemble des itérations qui auront permis de parcourir toutes les données d'apprentissage 1 fois. C'est-à-dire qu'avec la descente de gradient classique, il n'y a qu'1 itération par *epoch* car on calcule la moyenne des dérivées sur toutes les données, alors qu'avec le gradient stochastique il y a autant d'itérations au cours d'1 *epoch* qu'il y a de données d'apprentissage. Plus précisément, cet algorithme s'écrit :

$$\theta \leftarrow \theta - \alpha \frac{\partial}{\partial \theta} loss(y^d, \widehat{y}^d), \tag{1.3.4}$$

où la donnée (X^d, y^d) a été choisie aléatoirement parmi les données d'apprentissage. Le gradient stochastique permet ainsi un gain en termes de temps de calcul et d'espace mémoire à chaque itération. Néanmoins, la mise à jour fréquente des paramètres introduit des oscillations du gradient qui peuvent empêcher d'atteindre un minimum local qui serait meilleur que la plupart, voire le minimum global. De plus, avec cette implémentation, on perd toute possibilité de paralléliser les calculs, donc on perd également en efficacité.

C'est un compromis entre la descente de gradient classique et le gradient stochastique qui est privilégiée pour l'apprentissage des RNA : le gradient par *minibatch*. Au lieu de faire une moyenne des gradients pour la totalité des données d'apprentissage, on va diviser la base de données en paquets appelés *batches*. Par analogie, le gradient stochastique revient à définir des *batches* de taille 1. L'algorithme du gradient par *minibatch* s'écrit alors :

$$\theta \leftarrow \theta - \alpha \frac{1}{n} \sum_{d=1}^{n} \frac{\partial}{\partial \theta} loss(y^d, \widehat{y}^d), \tag{1.3.5}$$

où $1 < n < card(\mathcal{D}_{batch})$ désigne la taille d'un *batch*. Il y aura alors autant d'itérations dans 1 epoch qu'il y a de *batches*. On retrouve la possibilité de paralléliser les calculs de dérivées du gradient classique sans l'inconvénient d'avoir besoin de toutes les données en mémoire immédiate. On garde également des oscillations du gradient comme observé chez le gradient stochastique, seulement de manière moins intense.

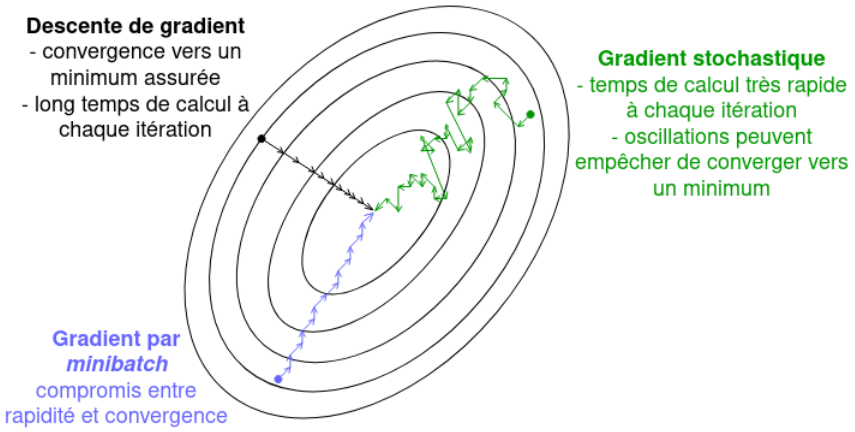


FIGURE 1.7 – Schéma résumant les comportements de la descente de gradient (noir), du gradient stochastique (vert), et du gradient par *minibatch* (bleu).

Le schéma présenté Figure 1.7 résume les comportements des trois algorithmes qui viennent d'être présentés : la descente de gradient est la technique présentant le moins d'oscillations, tandis que le gradient stochastique est le plus bruité ; finalement, le gradient par *minibatch* est le compromis entre les deux, présentant quelques oscillations.

Dans certains cas, ces algorithmes ne permettent pas d'atteindre un minimum optimum. En effet, ils ont tendance à rester coincés sur des points selles [48, 49] où le gradient est donc quasi nul, entraînant ainsi la mise à jour des paramètres de stagner dans cette zone de gradient quasi nul durant plusieurs itérations, jusqu'à un arrêt prématuré de l'entraînement. Une technique pour éviter cela est de garder en mémoire les évènements des itérations précédentes, comme une accélération, afin de s'éloigner des points selles. C'est pour cela que l'on introduit le gradient au moment [50], où un nouveau terme, noté $v^{(t)}$ à l'itération courante, accumule les gradients, tout en donnant plus d'importance au gradient courant grâce à une pondération, autrement dit :

$$v^{(t)} = \beta v^{(t-1)} + \alpha \nabla_\theta \mathcal{L}_T^{(t)}(\{X, Y\}_b), \tag{1.3.6}$$

$$\theta \leftarrow \theta - v^{(t)}, \tag{1.3.7}$$

où $v^{(t)}$ est le moment à l'itération courante, $\nabla_\theta \mathcal{L}_T^{(t)}(\{X, Y\}_b)$ désigne gradient à l'itération courante calculé sur les données d'un *batch*, α est le taux d'apprentissage, et β est un nombre réel entre 0 et 1 de pondération des contributions des gradients antérieurs. En développant l'écriture du moment $v^{(t)}$ sur toutes les itérations, et si α est supposé constant, on trouve alors :

$$v^{(t)} = \alpha \sum_{p=1}^{t} \beta^{t-p} \nabla_\theta \mathcal{L}_T^{(p)}(\{X, Y\}_b). \tag{1.3.8}$$

En conséquence, si $\beta \to 0$ alors la contribution des gradients antérieurs décroit trop rapidement. Une valeur par défaut couramment utilisée est $\beta = 0.9$ [49, 50].

Des optimiseurs encore plus performants ont été développées ces dernières années et sont aujourd'hui utilisés pour l'apprentissage des réseaux profonds [49] :

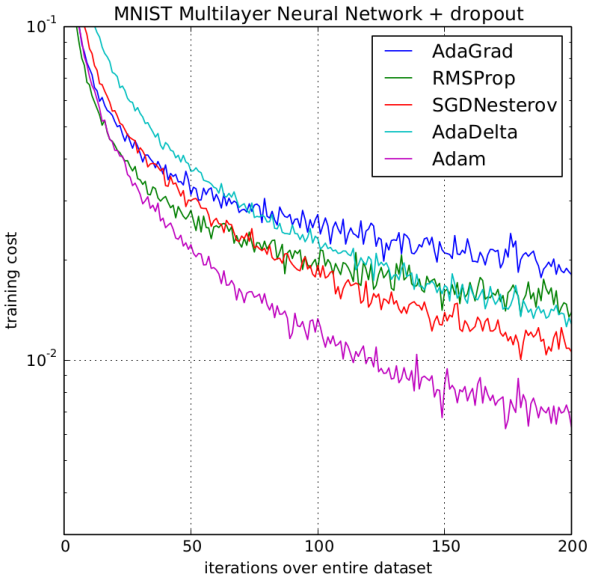


FIGURE 1.8 – Comparaison des performances de plusieurs optimiseurs sur la base de données MNIST [51].

. SGDNesterov [52] : au lieu de calculer le moment pour l'itération courante et d'utiliser ensuite les moments antérieurs accumulés, SGDNesterov utilise d'abord les moments accumulés pour se déplacer sur la fonction à minimiser, calcule le moment à l'itération courante en ce nouveau point, puis corrige la mise à jour des paramètres ;
. Adagrad [53] : ici, le taux d'apprentissage est adapté aux paramètres mis à jour, augmentant le taux lorsqu'un paramètre est associé à des caractéristiques peu fréquentes dans les données, ou au contraire en le diminuant s'ils sont associés à des caractéristiques fréquentes ;
. Adadelta [54] : une variante de Adagrad où la variation du taux d'apprentissage n'est plus monotone et l'accumulation des moments est restreinte à quelques itérations ;
. RMSprop : en tout point identique à Adadelta mais développé indépendamment au même moment et non publié ;
. Adam [51] : un algorithme usant à la fois d'un taux d'apprentissage adaptif type Adagrad, et de deux moments.

Une comparaison de leurs performances sur la base MNIST est présentée figure 1.8 à titre d'exemple.

Le point commun de tous ces algorithmes est l'utilisation des dérivées de la fonction perte par rapport à chaque paramètre du réseau. Il est évident que s'agissant de réseaux profonds, comptant des milliers, voire des millions, de paramètres, le calcul exact de toutes ces dérivées serait trop coûteux. L'apprentissage des réseaux repose en fait sur une optimisation de ce calcul de dérivées, appelée rétropropagation.

1.3.2 Mise à jour des paramètres et rétropropagation des gradients

Bien que la rétropropagation soit utilisée depuis longtemps, ce n'est qu'à la fin des années 1980 qu'une justification théorique est donnée, d'abord en 1986 par Rumerhalt *et*

al [55], puis en 1988 par Yann LeCun [56]. Le terme de rétropropagation est choisi en opposition avec la propagation avant, où l'information en entrée est transportée jusqu'à la dernière couche : on va partir de la dernière couche pour remonter jusqu'à l'entrée afin de calculer toutes les dérivées successives.

Prenons la descente de gradient stochastique définie équation (1.3.4). Afin de simplifier les notations, on omet celle indiquant la donnée et on définit plutôt la prédiction comme étant une activation, comme présenté équation (1.2.1). On ajoute également une information sur la couche où se trouve le paramètre que l'on veut mettre à jour, soit :

$$w_{ij}^{l+1} \leftarrow w_{ij}^{l+1} - \alpha \frac{\partial}{\partial w_{ij}^{l+1}} loss(y, a_j^{L+1}), \tag{1.3.9}$$

où w_{ij}^{l+1} est le poids qui connecte le neurone i de la couche $\mathcal{C}^l$ au neurone j de la couche $\mathcal{C}^{l+1}$, et L est le nombre de couches cachées du réseau (donc $a^{L+1} = \widehat{y}$). La démonstration est faite pour les poids uniquement, la méthode étant la même pour les biais.

En utilisant les équations (1.2.1) et (1.2.2) et la règle de dérivation en chaîne, la dérivée de la fonction perte en fonction du poids w_{ij}^l peut de fait être définie par :

$$\frac{\partial}{\partial w_{ij}^l} loss(y, a^{L+1}) = \frac{\partial loss(y, a^{L+1})}{\partial a_j^l} \frac{\partial a_j^l}{\partial z_j^l} \frac{\partial z_j^l}{\partial w_{ij}^l}. \tag{1.3.10}$$

La dérivée de la *loss* par rapport au poids w_{ij}^l est donc le produit de trois dérivées :

- $\frac{\partial loss(y, a^{L+1})}{\partial a_j^l}$, soit la dérivée de la *loss* par rapport à l'activation du neurone j sur la couche $\mathcal{C}^l$,
- $\frac{\partial a_j^l}{\partial z_j^l}$, soit la dérivée de l'activation du neurone j sur la couche $\mathcal{C}^l$ par rapport au potentiel post-synaptique de ce même neurone,
- $\frac{\partial z_j^l}{\partial w_{ij}^l}$, soit la dérivée du potentiel post-synaptique du neurone j de la couche $\mathcal{C}^l$ par rapport au poids w_{ij}^l.

La troisième dérivée se calcule aisément pour tout poids du réseau : $\frac{\partial z_j^l}{\partial w_{ij}^l} = a_i^{l-1}$.

En revanche, pour les couches cachées, on peut exprimer la dérivée de la *loss* par rapport à l'état du neurone j de la couche $\mathcal{C}^l$ à partir de la dérivée à la couche $\mathcal{C}^{l+1}$:

$$\begin{aligned}
\frac{\partial loss(y, a^{L+1})}{\partial a_j^l} &= \sum_{k \in \mathcal{C}^{l+1}} \frac{\partial loss(y, a^{L+1})}{\partial a_k^{l+1}} \frac{\partial a_k^{l+1}}{\partial a_j^l}, \\
&= \sum_{k \in \mathcal{C}^{l+1}} \frac{\partial loss(y, a^{L+1})}{\partial a_k^{l+1}} \frac{\partial a_k^{l+1}}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{\partial a_j^l}, \\
&= \sum_{k \in \mathcal{C}^{l+1}} \frac{\partial loss(y, a^{L+1})}{\partial a_k^{l+1}} \frac{\partial a_k^{l+1}}{\partial z_k^{l+1}} w_{jk}^{l+1}.
\end{aligned}$$

Ainsi, la dernière couche $L+1$ permet de calculer la dérivée à la couche L, qui permet de calculer la dérivée à la couche $L-1$, et ainsi de suite juqu'à l'entrée.

Posons maintenant la variable intermédiaire :

$$\Delta_j^l = -\frac{\partial loss(y, a^{L+1})}{\partial a_j^l} \frac{\partial a_j^l}{\partial z_j^l}.$$

Cette dérivée se calcule aisément pour $l = L + 1$ (la couche de sortie). Mais pour les couches cachées, on va utiliser le résultat précédent et faire le développement :

$$\begin{aligned}\Delta_j^l &= -\frac{\partial loss(y, a^{L+1})}{\partial a_j^l}\frac{\partial a_j^l}{\partial z_j^l}, \\ &= \frac{\partial a_j^l}{\partial z_j^l}\sum_{k\in\mathcal{C}^{l+1}} -\frac{\partial loss(y, a^{L+1})}{\partial z_k^{l+1}}\frac{\partial z_k^{l+1}}{\partial a_j^l}, \\ &= \frac{\partial a_j^l}{\partial z_j^l}\sum_{k\in\mathcal{C}^{l+1}} \Delta_k^{l+1} w_{jk}^{l+1}.\end{aligned}$$

Au final, on a donc définit $(\Delta_j^l)_{j\in\mathcal{C}^l}^{1\leq l\leq L+1}$ tel que :

. $\Delta^{L+1} = -\dfrac{\partial loss(y, a^{L+1})}{\partial a^{L+1}}\dfrac{\partial a^{L+1}}{\partial z^{l+1}}$ pour tout neurone de la couche de sortie $\mathcal{C}^{L+1}$,

. $\Delta_j^l = \dfrac{\partial a_j^l}{\partial z_j^l}\displaystyle\sum_{k\in\mathcal{C}^{l+1}} \Delta_k^{l+1} w_{jk}^{l+1}$ pour le neurone j de la couche cachée $\mathcal{C}^l$ $(1 \leq l \leq L)$.

et la descente de gradient stochastique (1.3.9) s'écrit maintenant :

$$w_{ij}^l \leftarrow w_{ij}^l + \alpha a_i^{l-1}\Delta_j^l, \tag{1.3.11}$$
$$b_j^l \leftarrow b_j^l + \alpha\Delta_j^l. \tag{1.3.12}$$

1.3.3 Performance de l'entraînement

Fort d'une règle d'apprentissage et d'un algorithme de rétropropagation pour la mise à jour des paramètres, il reste une question à élucider : comment s'assurer du bon déroulement d'un entraînement et de la qualité du réseau en résultant ?

L'évolution de la *loss* durant l'apprentissage est un premier indicateur de la performance du réseau. En effet, une *loss* qui décroît est le signe que la descente de gradient continue de progresser, et si elle termine sur un plateau alors un minimum est atteint. En plus de la *loss*, on va également utiliser des fonctions appelées *metrics* pour mesurer la performance du réseau, comme par exemple le taux de réussite lors d'une classification. Mais mesurer la performance uniquement sur des données que le réseau connait n'est pas suffisant. En effet, le but de l'entraînement est d'obtenir un réseau capable d'interpoler des solutions à partir d'un jeu de données restreint, mais néanmoins représentatif du problème à résoudre : on parle de la capacité de généralisation du réseau. Il est donc capital de tester la performance du réseau sur des données jamais utilisées pour la mise à jour des paramètres. Pour ce faire, il est d'usage de séparer la base de données en deux, voire trois, sous-jeux de données :

. l'ensemble d'apprentissage ou *train set* : ce sont les données utilisées durant la descente de gradient pour mettre à jour les paramètres du réseau, et tout au long de l'apprentissage elles sont également utilisées pour faire des mesures de performance ;
. l'ensemble de validation ou *validation set* : en plus des mesures sur l'ensemble d'apprentissage, on fait des mesure sur cet ensemble, qui contient des données que le réseau n'a jamais vues (i.e. qui ne servent pas à la mise à jour des paramètres), ce qui donnera une idée de la capacité de généralisation du réseau ;
. l'ensemble test ou *test set* : ce troisième ensemble est utilisé pour comparer la performance de plusieurs réseaux entraînés afin de choisir lequel garder.

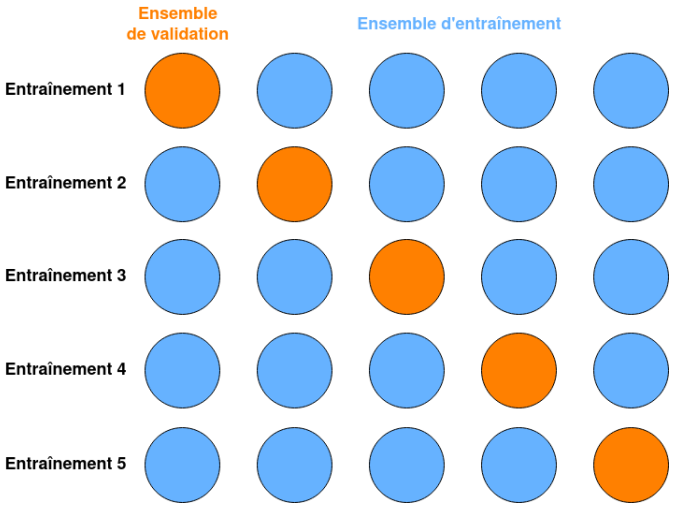


FIGURE 1.9 – Schéma du déroulement des entraînements avec la méthode de validation croisée ou *k-folds*.

Il n'existe pas de base théorique sur le choix des proportions des différents ensembles. De manière générale, l'ensemble d'apprentissage doit présenter plus de données que les deux autres ensembles, le plus souvent environ 70% du jeu de données de départ. Les proportions pour les deux autres ensembles peuvent être égales, par exemple 15%, sinon la proportion de l'ensemble de validation sera un peu plus élevée que celle de l'ensemble test, par exemple 20%-10%.

Cependant, on peut se trouver dans le cas d'un jeu de données trop petit (moins de 10000 données). Il faut alors soit procéder à une augmentation de données, c'est-à-dire créer des variations des données afin d'augmenter artificiellement la base, soit il est recommandé d'utiliser une validation croisée, ou méthode des *k-folds* [57, 58]. Dans ce dernier cas, on ne définit pas d'ensemble test, mais le jeu de données est divisé en k ensembles de proportions égales. Le réseau est alors entraîné à k reprises : à chaque entraînement, un ensemble différent est définit comme ensemble de validation, et les $k-1$ autres ensembles forment l'ensemble d'apprentissage, comme illustré figure 1.9 où une base de données est divisée en 5 *folds* produisant ainsi 5 entraînements. Au final, on aura une performance moyenne sur k ensembles de validation différents.

L'observation simultanée des performances sur l'ensemble d'entraînement et l'ensemble de validation tout au long de l'entraînement permet de détecter des comportements non désirés : le sur-apprentissage ou *overfitting*, et le sous-apprentissage ou *underfitting* [57,58], comme illustré figure 1.10. Le sur-apprentissage signifie que le réseau a appris les données d'entraînement au lieu de généraliser. Il est caractérisé par un décollement de la *loss* en validation, qui accroît tandis que celle en entraînement continue de décroître. Il existe des méthodes pour pallier ce problème :

- en premier lieu, diminuer la complexité du réseau, c'est-à-dire le nombre de neurones cachés, peut suffire ;
- on peut également arrêter l'entraînement prématurément pour éviter de rentrer dans la phase de sur-apprentissage, on parle alors de *early stopping* ;
- il se peut également que la cause soit un manque de données par rapport à la complexité du problème, et il faut donc soit faire de l'augmentation de données, soit simplifier le problème et les données ;
- certains neurones peuvent également s'être trop spécialisés dans un certain type de données, délaissant les autres, et dans ce cas là on peut forcer les neurones à

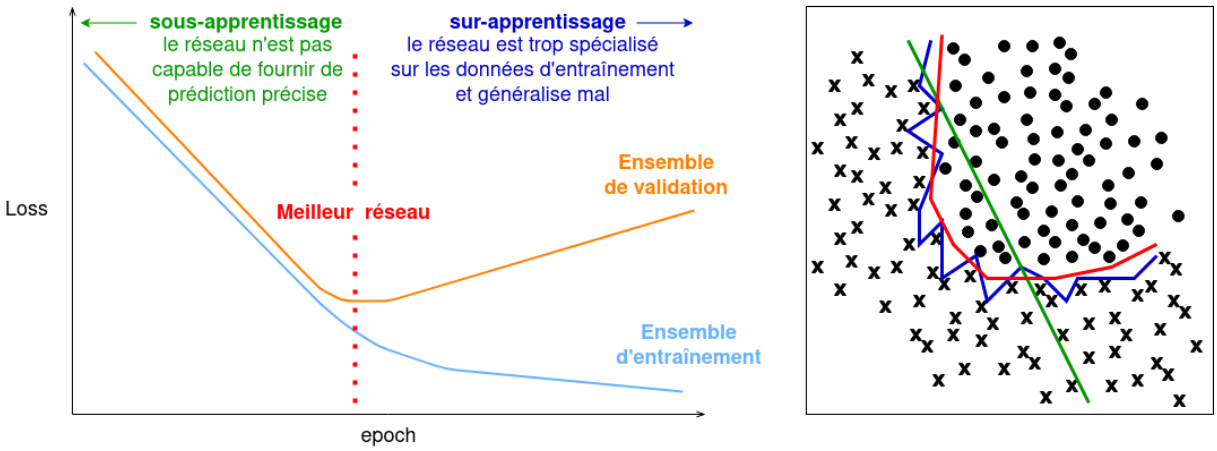


FIGURE 1.10 – Principe de sous- et sur-apprentissage (gauche), et exemple sur une classification binaire (droite).

"s'éteindre" de manière aléatoire durant l'entraînement pour moins les soliciter et éviter qu'ils se spécialisent, c'est ce que l'on appelle le *dropout* ;

. enfin, une dernière technique utile est la régularisation de la *loss*, c'est-à-dire qu'en ajoutant un terme à la *loss*, généralement la norme L^1 ou L^2 des paramètres, on limite les valeurs des paramètres pour éviter qu'ils soient trop petits ou trop grands.

À l'inverse, le sous-apprentissage signifie que le réseau n'arrive pas à faire de prédiction précise. Il est caractérisé par des valeurs de *loss* et de *metrics* trop élevés à la fin de l'apprentissage. Pour y remédier, on peut soit augmenter la complexité du réseau, soit augmenter la complexité de la base de données.

1.3.4 *Hyperparameter tuning* ou réglage des hyperparamètres

Les hyperparamètres sont tous les paramètres qui sont choisis par l'utilisateur avant l'apprentissage, et qui définissent l'architecture du réseau ainsi que le déroulement de l'apprentissage, contrairement aux paramètres que sont les poids et les biais, qui eux sont ajustés durant l'apprentissage. Les hyperparamètres rencontrés jusque là sont :

. le nombre de couches cachées,
. le nombre de neurones de chaque couche pour les FNN,
. les dimensions des noyaux de convolution et leur nombre à chaque couche pour les CNN,
. les fonctions d'activation,
. la fonction perte,
. le taux d'apprentissage et ses paramètres s'il décroit durant l'apprentissage (on parle de *learning rate decay* [59]),
. l'optimiseur et ses paramètres,
. la taille de *batch*,
. le nombre d'*epochs*,
. le *dropout*,
. la régularisation de la *loss*,

auxquels viennent s'ajouter encore d'autres hyperparamètres, dont on ne fera pas la liste ici. Néanmoins il est important de souligner la difficulté que soulève la définition de certains de ces paramètres. Un cas particulier est le choix de la fonction perte, qui est lié au problème à résoudre et impacte de plus le choix de la fonction d'activation en sortie.

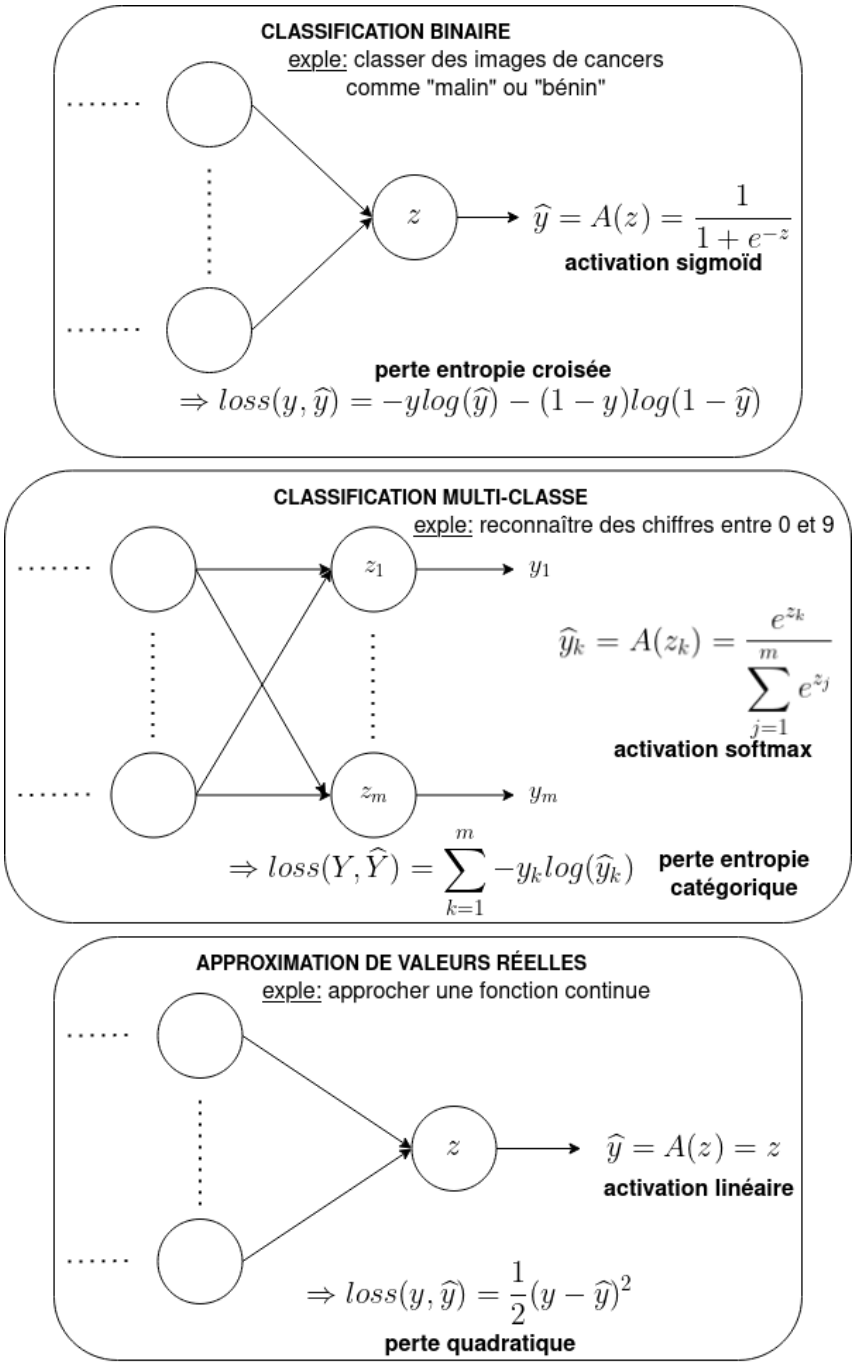


FIGURE 1.11 – Choix de la fonction perte et de la fonction d'activation de la couche de sortie suivant le type de problème à résoudre.

La Figure 1.11 présente chaque fonction perte présentée dans le Tableau 1.2, le type de problème pour lequel elle est recommandée, et à quelle fonction d'activation du Tableau 1.1 on l'associe. Ces choix sont motivés par les valeurs que prennent les fonctions d'activation. En effet, la fonction sigmoïd retourne une valeur réelle comprise entre 0 et 1, qui est interprétée comme une probabilité par la fonction perte entropie croisée. Une fois associées, ces deux fonctions permettent de résoudre des problèmes de classification binaire : la prédiction du réseau correspond à la probablité que la donnée d'entrée appartienne à une des deux classes de données, et la perte est d'autant plus faible que cette prédiction est proche de la valeur cible associée. De manière similaire, la fonction d'acti-

vation *softmax* est utilisée pour les problèmes de classification où il y a plus de 2 classes de données : chaque neurone en sortie représente alors la probabilité d'appartenir à une de ces classes. Finalement, on peut voir l'entropie croisée comme un cas particulier de l'entropie catégorique. Enfin, les problèmes de régression linéaire, où l'on veut approcher des valeurs réelles, sont résolus en associant l'activation linéaire (dont le résultat est une valeur réelle non bornée) à l'erreur quadratique. Il n'existe pas de règle quant au choix de la fonction d'activation sur les couches cachées, bien que les activations les plus souvent utilisées sont les fonctions ReLU, sigmoïd et tangente hyperbolique [57].

Un premier moyen de régler le reste des hyperparamètres est en faisant une recherche manuelle du jeu d'hyperparamètres produisant la meilleure généralisation. Pour cela, il faut comprendre l'influence de chaque hyperparamètre sur la performance du réseau [57, 58, 60]. Avec cette technique, il est recommandé de régler chaque paramètre l'un après l'autre, en commençant par ceux reconnus par la communauté comme ayant la plus grande influence sur l'apprentissage. Il est par exemple recommandé de commencer par régler le taux d'apprentissage, tandis que la taille de batch peut être réglée en dernier [60]. On appelle cette technique la descente coordonnée, ou *coordinate descent*, car on règle tour à tour chaque hyperparamètre jusqu'à atteindre la meilleure combinaison rencontrée durant la recherche manuelle. Cependant, toutes les combinaisons d'hyperparamètres n'auront pas été explorées, et on peut alors avoir manqué la meilleure combinaison possible.

Si l'on veut faire une recherche exhaustive de toutes les combinaisons d'hyperparamètres possibles, alors on peut utiliser la recherche par quadrillage ou *grid search*. Une grille de valeurs régulière est d'abord déterminée pour chaque hyperpramètre et on peut ensuite lancer en parallèle des entraînements sur chaque combinaison. En pratique, vu le coût numérique qu'une recherche exhaustive implique, on va plutôt pratiquer plusieurs recherches par quadrillage à la suite, en s'attachant à régler seulement quelques hyperparamètres à la fois, les autres hyperparamètres restant fixes. Par exemple, dans le cas d'un taux d'apprentissage variable, il existe plusieurs méthodes pour faire décroître sa valeur durant l'apprentissage, chacune avec un certain nombre de paramètres : une première recherche peut être menée sur la méthode de décroissance du taux d'apprentissage (on parle de *scheduler*) en fixant les valeurs initiale et finale du taux, puis une fois le meilleur *scheduler* identifié, la recherche suivante porterait sur trouver le meilleur jeu de valeurs initiale et finale [60]. Une variante de cette recherche par quadrillage est la recherche aléatoire, c'est-à-dire qu'au lieu de définir une grille de valeurs uniforme pour chaque hyperparamètre, on va plutôt tirer des valeurs aléatoirement dans un domaine défini [61].

Les réseaux de neurones trouvent leur utilité dans de nombreux domaines, mais celui qui nous intéresse ici est celui de la simulation numérique : le prochain chapitre dresse un état des lieux quant à l'utilisation des RNA dans la modélisation de phénomènes physiques.

Chapitre 2

L'émergence du *Deep Learning* dans les simulations numériques

Depuis leur découverte, les réseaux de neurones artificiels ont été utilisés avec succès et ont prouvé leur efficacité dans de nombreux domaines, comme le traitement du langage naturel [62], la détection de fraude [63], l'évaluation du risque de crédit [64], l'industrie automobile [65], ou encore l'imagerie médicale [66], pour n'en nommer que quelques-uns. Ces dernières années, les RNA ont notamment fait une percée fulgurante dans la modélisation de phénomènes physiques [67–76].

Pour la simulation de phénomènes physiques, les RNA ne sont généralement pas utilisés tels quels, mais plutôt couplés à des modèles de calcul classiques qui résolvent des équations aux dérivées partielles à l'aide de schémas numériques. Les RNA peuvent alors venir remplacer ou consolider une partie de ces codes, afin d'en améliorer les performances. Dans ce contexte, les RNA ont prouvé qu'ils pouvaient réduire les temps de calcul tout en préservant une bonne précision lorsqu'ils sont correctement entraînés, et pouvaient même permettre des calculs en temps réel [77–79]. Des travaux récents ont même permis l'émergence d'un nouveau type de RNA : les *Physics Informed Neural Networks* ou PINN [44, 80, 81], où les équations sont introduites dans la définition de la fonction perte.

Dans un premier temps, cette nouvelle classe des réseaux de neurones est décrite dans ses versions les plus simples : l'optimisation sans contraintes et l'optimisation avec contrainte. Des versions plus avancées, usant notamment d'outils numériques classiques comme les méthodes Runge-Kutta, sont également discutées. Dans la seconde et dernière partie de ce chapitre, la question de la disponibilité des librairies de *Deep Learning* pour une implémentation dans les grands codes de calcul est abordée.

2.1 Physics Informed Neural Networks

Ces réseaux exploitent deux forces des RNA. Premièrement, leurs capacités d'approximateurs universels, c'est-à-dire que pour n'importe quel problème $y = f(x)$, il existe un réseau pouvant approcher la fonction f. Les premières démonstrations de cette propriété portaient sur des réseaux avec une unique couche cachée avec un nombre arbitrairement grand de neurones [82–84], puis plus tard elles ont été étendues aux MLP [85]. Deuxièmement, les premiers travaux de Lagaris *et al* [86] ont montré qu'un réseau de neurones était dérivable. La dérivation de la sortie d'un réseau en fonction de la donnée d'entrée est aisément réalisable grâce à un outil comme la différentiation automatique [87], déjà couramment employé dans les librairies de *Deep Learning*.

Ces deux caractéristiques, couplées à des librairies de *Deep Learning* optimisées, permettent d'implémenter la principale nouveauté des PINN : remplacer les *loss* classiques par des équations aux dérivées partielles décrivant des lois physiques. Considérons les équations différentielles partielles avec des conditions aux limites, écrit sous la forme générale :

$$\begin{cases} \mathcal{G}(x, u(x), \nabla u(x), ...) = 0 \; \forall x \in \Omega, \\ \mathcal{F}(u(x)) = f(x) \; \forall x \in \partial\Omega. \end{cases} \tag{2.1.1}$$

On sait que la dérivée d'un réseau de neurone par rapport à n'importe laquelle de ses entrées est un autre réseau [86]. On peut donc dériver la sortie du réseau $\mathcal{N}_\theta(x)$ par rapport aux entrées (temps, espace, énergie, etc) et écrire une fonction perte qui utilise ces dérivées, et ainsi utiliser les techniques de minimisation de fonction perte usuelles. Deux approches existent parmi cette famille de réseaux de neurones.

La première est l'optimisation sans contrainte [86] : on suppose que la prédiction du réseau ne vérifie pas les conditions aux limites, et on va alors construire une solution $\widehat{u}(x)$ qui va, elle, les vérifier. On est donc ici face à un apprentissage non supervisé, puisqu'on n'utilisera pas de cible pour l'évaluation de la fonction perte.

La reconstruction de la solution est définie par :

$$\widehat{u}(x) = A(x) + B(x)\mathcal{N}_\theta(x) \; \forall x \in \Omega, \tag{2.1.2}$$

de manière à ce que $\mathcal{F}(A(x)) = f(x)$ et $B(x) = 0$ lorsque $x \in \partial\Omega$. La perte totale est alors définie par :

$$\mathcal{L}_T = \sum_{x_i \in \Omega} |\mathcal{G}(x_i, \widehat{u}(x_i), \nabla\widehat{u}(x_i), ...)|^2. \tag{2.1.3}$$

La deuxième approche est l'optimisation avec contrainte [44], où l'on suppose au contraire que la prédiction du réseau vérifie les conditions aux limites. Ainsi, en plus de contenir les équations, la fonction perte impose également à la sortie du réseau de respecter les conditions aux limites dès lors qu'une donnée se trouve sur une des limites du domaine (condition initiale ou condition aux bords). Pour ce faire, on forme un ensemble de points de collocation pour les conditions aux limites. Sur ces points de collocation, on aura finalement ce qui s'apparente à un apprentissage semi-supervisé : une partie de l'apprentissage va dépendre de l'équation à résoudre (donc apprentissage non supervisé faisant appel directement aux équations), tandis que l'autre partie va dépendre des conditions limites établies (donc apprentissage supervisé). Sur le reste du domaine, on utilise uniquement un apprentissage non supervisé, où on a recours aux équations.

On pose maintenant $\widehat{u}(x) = \mathcal{N}_\theta(x)$, et la perte totale est alors définie par :

$$\mathcal{L}_T = \sum_{x_i \in \Omega} |\mathcal{G}(x_i, \widehat{u}(x_i), \nabla\widehat{u}(x_i), ...)|^2 + \sum_{x_j \in \partial\Omega} |\mathcal{G}(x_j, \widehat{u}(x_j), \nabla\widehat{u}(x_j), ...) - f(x_j)|^2. \tag{2.1.4}$$

Ces deux méthodes semblent apporter une solution à la contrainte que pose l'établissement d'une base de données. En effet, dans le cadre d'une application à la simulation de phénomènes physiques, les données d'apprentissage pourraient provenir soit de codes de calcul numériques, soit d'expériences. Ces deux types de données peuvent cependant être difficiles à générer en quantité suffisante pour que le réseau apprenne, du fait du temps de calcul trop coûteux pour une modélisation classique, ou de la reproductibilité peu aisée de certaines expériences, etc. Les données peuvent également ne pas être assez bonnes ou

exactes, à cause de l'ordre du schéma numérique et l'accumulation d'erreurs d'arrondis pour les codes de calcul, ou encore des erreurs de mesure durant une expérience.

Dans leur article, Raissi et al. [80] proposent d'ailleurs une version des PINN qui usent d'un schéma numérique de Runge-Kutta afin de s'affranchir des points de collocation. Depuis, plusieurs études ont même prouvé que les PINN pouvaient être utilisés sans avoir besoin d'aucune donnée d'apprentissage [88, 89]. D'autres ont également prouvé que les PINN pouvaient utilement être associés aux méthodes numériques classiques. C'est le cas notament dans l'article de Markidis [90], où un PINN est associé à un solveur de Gauss-Seidel pour résoudre l'équation de Poisson 2D avec une précision et une rapidité de calcul similaire à celle d'une librairie optimisée comme PETSc ; ou encore l'article de Jin et al. [91], où un PINN est enrichi avec des propriétés des schémas numériques préservant l'asymptotique pour la résolution d'équations de transport linéaires. Enfin, si jusqu'alors les PINN étaient exclusivement des réseaux à propagation avant, des architectures mêlant PINN et réseaux convolutifs ont également vu le jour [92–94].

Néanmoins, comme l'a mentionné Markidis dans son article [90], l'utilisation des PINN dans les grands codes de calcul est limitée par la disponibilité des librairies de *Deep Learning*.

2.2 Les librairies de *Deep Learning*

La grande majorité des librairies de *Deep Learning* sont implémentées avec une interface Python. Pour preuve, dans son rapport sur l'état de l'apprentissage automatique et de la science des données [95], la plateforme web Kaggle (propriété de Google où sont organisées des compétitions en science des données) établit une liste des librairies les plus utilisées par la communauté de science des données (Figure 2.1), et parmis elles, on retrouve trois librairies de *Deep Learning* : TensorFlow [96], Keras [97] et PyTorch [98]. Plusieurs articles établissant des listes des librairies de *Deep Learning* viennent également confirmer ces résultats [99–101]. Il existe également des librairies disponibles dans d'autres langages, comme le Deep Learning Toolbox de MATLAB, ou encore le Cognitive Toolkit (CNTK) développé par Microsoft (implémenté en C++).

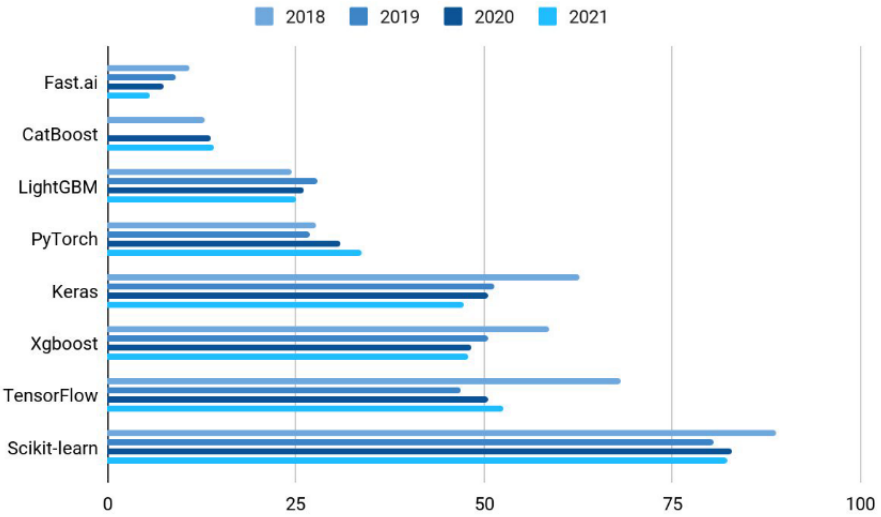


FIGURE 2.1 – Librairies de *machine learning* les plus utilisées d'après le *State of Machine Learning and Data Science 2021* de la plateforme Kaggle [95]

Le langage Fortran est le premier langage de haut niveau, initialement développé par l'ingénieur John Backus entre 1954 et 1957 pour l'IBM 704 (le premier ordinateur commercialisé par IBM). Depuis, le langage a évolué, avec plusieurs mises à jour majeures comme Fortran 77, Fortran 90 (qui lui a permis de devenir une norme ISO), et Fortran 2008. On parle d'ailleurs aujourd'hui de Fortran moderne en opposition aux versions antérieures. Bien que peu répandu dans l'industrie, Fortran est très utilisé pour la résolution de systèmes à grande échelle et de nombreux grands codes de calculs, comme Flash [102], Fluka [30], EPOCH [103], NEMO [104], ou encore HYCOM [105], sont implémentés en Fortran. Le langage dispose également de librairies de calcul scientifique hautement performantes, telles que LAPACK [106] et PETSc [107], et d'outils pour le calcul haute performance parallélisé, avec notamment OpenMP [2] et MPI [1].

L'ajout d'outils de *Deep Learning* pourrait se révéler bénéfique pour les grands codes de calcul et la simulation numérique en général. Mais l'arrivée tardive des RNA dans les simulations numériques tient certainement au fait qu'il y a peu d'échanges entre les communautés de la science des données et du calcul scientifique. Markidis a également remarqué ce manque d'outils de *Deep Learning* en Fortran [90]. Il a d'ailleurs obtenu ses résultats exclusivement avec des librairies Python : la librairie DeepXDE [108], qui utilise Tensorflow en back-end et est dédiée à l'implémentation de PINN, ainsi que Cython [109] et petsc4py [110] pour la partie solveur numérique classique.

Fort heureusement, ce nouvel engouement pour le *Deep Learning* dans le calcul scientifique a inspiré le développement de deux nouvelles librairies implémentées en Fortran pour répondre à la demande croissante : la librairie neural-fortran [111], une librairie entièrement parallélisée, et *Fortran Keras Bridge* ou FKB [112], une interface entre un environnement Python, où l'on peut utiliser la librairie Keras, et un environnement Fortran. La première librairie s'attache donc à traduire en langage Fortran les outils déjà existants. Dans son article, Curcic compare son outil (non optimisé) aux librairies Keras et Tensorflow pour la base de données MNIST sur 1 processeur de calcul et obtient une performance similaire en termes de temps de calcul (environ 14s pour neural-fortran contre 12.4 pour Keras+Tensorflow), mais aussi une meilleure performance en termes d'espace mémoire (220MB pour neural-fortran contre 359MB pour Keras-Tensorflow), prouvant le potentiel de cette nouvelle librairie [111]. La librairie FKB, quant à elle, propose un pont entre le langage Fortran et le langage Python afin de bénéficier de la librairie déjà existante et fortement optimisée Keras. De plus, FKB a été développée avec une sous-couche usant de la librairie neural-fortran afin de convertir des réseaux Keras en Fortran. Ces deux librairies commencent à être utilisées depuis peu dans des simulations à grande échelle [113–116]), et elles ont même inspiré l'émergence de deux autres libraires, SmartSim [117] et Fortnet [118].

Dans cette première partie de la thèse, on a introduit la théorie générale derrière les RNA, et discuté de leur émergence dans les simulations numériques. L'objet de cette thèse est d'utiliser des RNA pour des simulations de transport de sources énergétiques : la deuxième partie introduit deux théories du transport, avec en premier lieu la théorie du transport de chaleur non local dans les plasmas.

Deuxième partie

Théories de transport pour des simulations de physique complexe

Chapitre 3

Transport de chaleur pour la fusion par confinement inertiel

Dans l'introduction de ce manuscrit, l'importance de la description du transfert de chaleur des électrons lors des expériences de FCI a été adressée. Ce chapitre vient élaborer ce point, en présentant d'une part le code de simulation d'expériences de fusion par confinement inertiel développé au sein du CELIA dans la première section, puis, en introduisant dans la deuxième section des modèles de transport de flux de chaleur, qui seront par ailleurs étudiés dans la suite de cette thèse.

3.1 CHIC : un code hydrodynamique dédié à la simulation d'expériences en FCI

Il existe trois approches en physique des plasmas :

- La théorie particulaire : on décrit le mouvement individuel des particules, des ions et des électrons, qui sont sujets à l'influence de champs magnétiques ;
- La théorie cinétique : au lieu de décrire les particules de manière individuelle, les équations cinétiques le font de manière statistique, en modélisant la fonction de distribution des particules, qui représente la probabilité de présence des particules dans l'espace des phases ;
- La théorie hydrodynamique : considérée comme la plus simple manière de décrire un plasma, elle convient pour un grand nombre de phénomènes dans les interactions laser-plasma, mais ne convient pas dans le cas hors équilibre local ; en particulier, cette théorie calcule des grandeurs moyennes comme la densité et la température, en supposant que la fonction de distribution a la forme d'une Maxwellienne.

Les codes cinétiques (OSHUN [119], SPARK [120]) sont les outils de référence pour les simulations de physique à haute densité d'énergie (*high energy-density* ou HED). Néanmoins, ces codes ne peuvent pas être utilisés pour reproduire des expériences de fusion sur des temps longs (nanoseconde) et de grande dimensions (millimètre) : dans ce cas, on préfère les codes hydrodynamiques, comme le code CHIC [18].

Le code CHIC est un code bidimensionnel ALE (Arbitrary Lagrangian Eulerian) d'hydrodynamique radiatique permettant notamment de faire des simulations d'expériences de FCI. Le coeur du code est la résolution d'un modèle hydrodynamique de plasma bi-température, dont le système d'équations est défini comme :

$$\begin{cases} \dfrac{\partial}{\partial t}\rho + \overrightarrow{\nabla} \cdot \rho\overrightarrow{u} = 0, \\ \dfrac{\partial}{\partial t}\rho\overrightarrow{u} + \overrightarrow{\nabla}(p_e + p_i) = 0, \\ \dfrac{\partial}{\partial t}(\dfrac{3}{2}p_e) + \overrightarrow{\nabla} \cdot \overrightarrow{q_e} = \Omega_{ei}(T_i - T_e) + W_l + W_r + W_f, \\ \dfrac{\partial}{\partial t}(\dfrac{3}{2}p_i) - \overrightarrow{\nabla} \cdot k_i\overrightarrow{\nabla}T_i = \Omega_{ie}(T_e - T_i), \end{cases} \tag{3.1.1}$$

où ρ est la densité du plasma et $\overrightarrow{u}$ sa vitesse, p_α est la pression des particules α et T_α leur température, $\Omega_{\alpha\beta}$ est un terme d'échange entre les particules α et les particules β, $\overrightarrow{q_e}$ est le flux de chaleur des électrons, et W_l, W_r, W_f sont des termes sources décrivant respectivement le laser, le transport radiatif et la fusion nucléaire. Le système (3.1.1) ne possédant que 4 équations pour 5 inconnues, il doit être fermé par un modèle pour calculer le flux de chaleur des électrons $\overrightarrow{q_e}$. À ces équations se rajoutent également les équations d'état (*Equations Of State* ou EOS).

Dans le code CHIC, deux modèles de conduction thermique sont implémentés dans le module de conduction thermique, auquel viennent s'ajouter également d'autres modules décrivant des phénomènes physiques complexes. La structure du code CHIC est schématisée Figure 3.1.

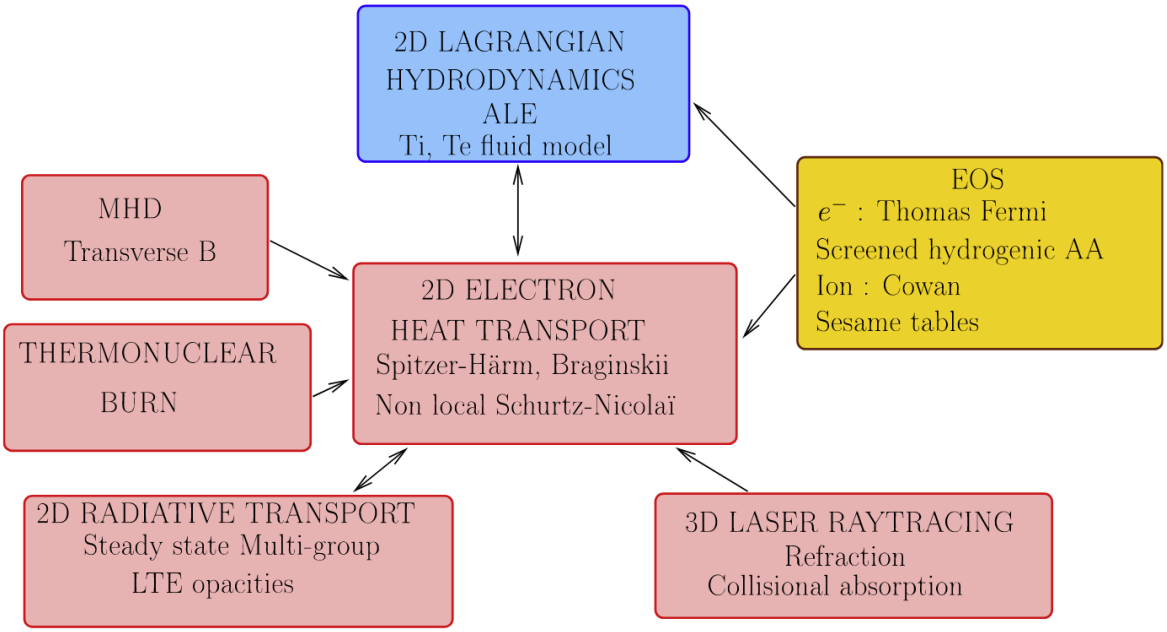


FIGURE 3.1 – Structure du code hydrodynamique CHIC. [18]

Trois modèles de conduction thermique dans les plasmas produits par laser sont définis dans la section suivante : le modèle local de Spitzer-Härm, et les modèles non locaux de Luciani-Mora-Virmont et de Schurtz-Nicolaï-Busquet. Le premier et le dernier sont notamment implémentés dans le module de conduction thermique du code CHIC.

3.2 Conduction thermique dans les plasmas produits par laser

3.2.1 Le modèle local de Spitzer-Härm

Le modèle de Spitzer-Härm [121] défini le flux des électrons comme étant :

$$q_{SH} = -\kappa_{SH}(T_e)\nabla T_e, \tag{3.2.1}$$

où T_e est la température du milieu considéré, et $\kappa_{SH}(T_e)$ est le coefficient de conductivité thermique qui dépend de la température ($\propto T_e^{5/2}$).

Lorsque les gradients de température sont raides, l'hypothèse d'équilibre local sur laquelle repose le flux de Spitzer-Härm n'est plus valable. Puisque dans ce cas, la théorie de Spitzer-Härm surestime le flux de chaleur, et il est d'usage de limiter la valeur du flux local par un coefficient ajustable dans les codes de simulation. Cependant, cette limitation ad hoc ne permet pas de retrouver toutes les propriétés du flux de chaleur et l'utilisation de modèles non locaux devient nécessaire [9, 10].

3.2.2 Modèles non locaux

En présence de forts gradients, l'hypothèse d'équilibre thermodynamique local, sur laquelle repose la théorie de Spitzer-Härm, n'est plus valide : le ratio entre le libre parcours moyen des particules (la distance parcourue par une particule entre 2 collisions, notée λ_e) et la longueur de gradient de température ($L_T = \frac{T}{\nabla T}$) est élevé. En conséquence, l'influence des zones de forts gradients de température dépasse le voisinage local de la zone considérée, et peut modifier le flux en un point distant du domaine de quatre manières, comme illustré Figure 3.2 :

1. par une rotation : dans les zones où il suit le gradient de température, le flux local peut être dévié par la contribution non locale due à la présence d'un fort gradient dans la zone d'étude ;
2. par un préchauffage : dans les zones froides, les électrons dont le libre parcours moyen est grand comparé aux longueurs de gradient peuvent contribuer au chauffage du plasma ;
3. par une limitation : les flux non locaux sont naturellement limités à une fraction du flux libre (flux théorique maximum) ;
4. par un flux antinaturel : dans le cas d'un faible gradient de température proche d'une zone de très fort gradient, le flux local peut être réduit par une contribution non locale dominante dans la direction opposée.

L'idée sous-jacente aux modèles non locaux a été proposée par Luciani et Mora [123] : une formulation basée sur une convolution sur l'espace physique du flux de Spitzer-Härm par un propagateur G, dont la portée est fonction de la distance que peuvent parcourir les électrons, c'est-à-dire

$$q_e(x) = \int q_{SH}(x')G(x, x')dx'. \tag{3.2.2}$$

De nombreux noyaux de convolution ont été proposés [12–14, 16, 17]. On s'intéresse à deux d'entre eux en particulier : le modèle de Luciani-Mora-Virmont et le modèle de Schurtz-Nicolaï-Busquet.

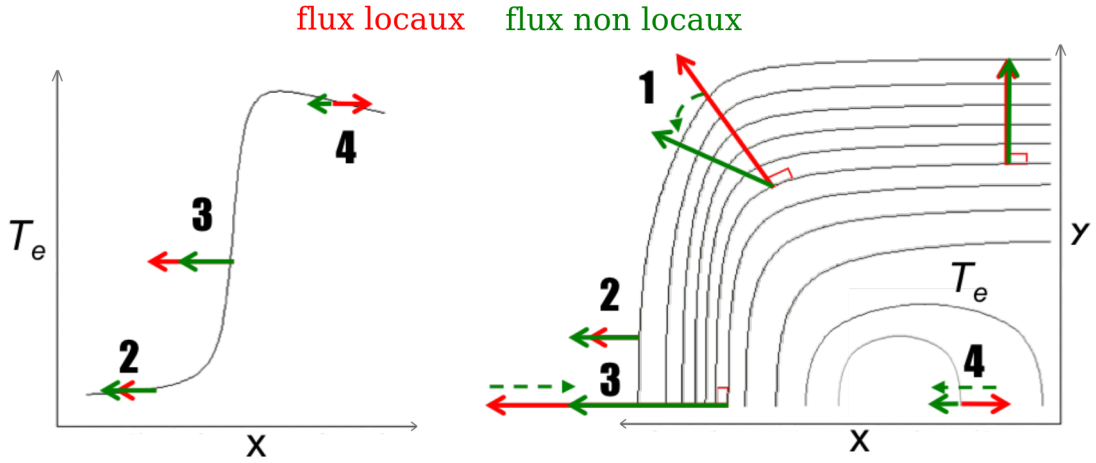


FIGURE 3.2 – Principaux effets non locaux rencontrés dans une simulation de FCI : la rotation (1), le préchauffage (2), la limitation (3) et le flux antinaturel (4). [122]

3.2.2.1 Le modèle de Luciani-Mora-Virmont

Le modèle de Luciani-Mora-Virmont [12] défini le flux des électrons comme étant :

$$q_{LMV} = \int \omega(x, x') q_{SH}(x') \frac{1}{2\lambda(x')} dx', \tag{3.2.3}$$

où le noyau de convolution $\omega(x, x')$ est définit par

$$\omega(x, x') = \exp\left(\frac{1}{n_e(x')\lambda(x')} \left| \int_x^{\prime x} n_e(x'') dx'' \right| \right), \tag{3.2.4}$$

avec $\lambda(x')$ le parcours effectif et n_e la densité électronique. Ce modèle n'est toutefois valable qu'en 1D. C'est pour cela qu'un modèle similaire à celui-ci et valable pour les cas 2D et 3D a été développé par Schurtz-Nicolaï-Busquet [13].

3.2.2.2 Le modèle de Schurtz-Nicolaï-Busquet

Dans leur article, Schurtz, Nicolaï et Busquet proposent une extension multi-dimensionnelle du flux non local LMV [13]. Par analogie, le modèle de Schurtz-Nicolaï-Busquet définit le flux de chaleur en multidimension par :

$$\mathbf{q_{SNB}}(\mathbf{r}) = \int_{4\pi} \int_0^{\infty} \mathbf{\Omega} \otimes \mathbf{\Omega} W_0(\mathbf{r}, \mathbf{r} + s\mathbf{\Omega}) \mathbf{q_{SH}}(\mathbf{r} + s\mathbf{\Omega}) \frac{ds}{\lambda_e(\mathbf{r} + s\mathbf{\Omega})} d\mathbf{\Omega}, \tag{3.2.5}$$

où $W_0(\mathbf{r}, \mathbf{r}')$ est un noyau de convolution

Le flux non local SNB (3.2.5) est en fait le moment d'ordre 1 de la solution de l'équation de transport :

$$\mathbf{\Omega} \cdot \nabla Q(\mathbf{\Omega}, \mathbf{r}) = \frac{1}{\lambda_e(\mathbf{r})} \left(\frac{3}{4\pi} \mathbf{\Omega} \cdot \mathbf{q_{SH}}(\mathbf{r}) - Q(\mathbf{\Omega}, \mathbf{r}) \right). \tag{3.2.6}$$

Les électrons n'ayant pas tous la même vitesse, l'espace des énergies est découpé en N_g groupes, et on pose :

$$\mathbf{U}_g = \frac{1}{24} \int_{E_{g-1}/kT}^{E_g/kT} \beta^4 e^{-\beta} \mathbf{q}_{SH} d\beta \quad \text{et} \quad \lambda_g = 2 \left(\frac{E_{g-1/2}}{T_e} \right)^2 \lambda_e.$$

L'équation (7.2.1) s'écrit alors pour chaque groupe en énergie g :

$$\mathbf{\Omega} \cdot \nabla Q_g(\mathbf{\Omega}, \mathbf{r}) = \frac{1}{\lambda_g(\mathbf{r})} \left(\frac{3}{4\pi} \mathbf{\Omega} \cdot \mathbf{U_g}(\mathbf{r}) - Q_g(\mathbf{\Omega}, \mathbf{r}) \right), \tag{3.2.7}$$

et il en découle que

$$\mathbf{q_{SNB}}(\mathbf{r}) = \sum_{g=1}^{N_g} \int_{S_2} Q_g(\mathbf{\Omega}, \mathbf{r}) \mathbf{\Omega} d^2\mathbf{\Omega}.$$

On définit maintenant le moment d'ordre 0 de Q_g, soit

$$H_g(\mathbf{r}) = \int_{S_2} Q_g(\mathbf{\Omega}, \mathbf{r}) d^2\mathbf{\Omega}.$$

L'approximation P1 de Q_g s'écrit alors :

$$Q_g(\Omega, \mathbf{r}) = \tfrac{1}{4\pi} H_g(\mathbf{r}) + \tfrac{3}{4\pi} \mathbf{\Omega} \cdot \mathbf{q_g}(\mathbf{r}).$$

On remplace cette expression dans l'équation (3.2.7), et en calculant les moments d'ordre 0 et d'ordre 1 on trouve finalement :

$$\begin{cases} H_g(\mathbf{r}) = -\lambda_g(\mathbf{r}) \nabla \cdot \mathbf{q}_g(\mathbf{r}), \\ \dfrac{\lambda_g(\mathbf{r})}{3} \nabla H_g(\mathbf{r}) = \mathbf{U}_g(\mathbf{r}) - \mathbf{q}_g(\mathbf{r}). \end{cases} \tag{3.2.8}$$

Le moment d'ordre 0 de Q_g est alors solution de l'équation :

$$\left(\frac{1}{\lambda_g(\mathbf{r})} - \nabla \frac{\lambda_g(\mathbf{r})}{3} \nabla \right) H_g(\mathbf{r}) = -\nabla \cdot \mathbf{U}_g(\mathbf{r}), \tag{3.2.9}$$

et le flux de chaleur non local SNB s'écrit finalement :

$$\mathbf{q}_{SNB}(\mathbf{r}) = \mathbf{q}_{SH}(\mathbf{r}) - \sum_{g=1}^{N_g} \frac{\lambda_g(\mathbf{r})}{3} \nabla H_g(\mathbf{r}). \tag{3.2.10}$$

Les questions de modélisation du transport de sources énergétiques apparaissent dans d'autres domaines que la FCI : le prochain chapitre introduit la théorie du transport de particules pour la radiothérapie.

Chapitre 4

Transport de particules pour le calcul de dose en radiothérapie

Ce chapitre décrit le code KIDS, un code aux moments développé au CELIA. D'abord, la première partie introduit les interactions des particules chargées qui sont modélisées. Ensuite, la seconde partie du chapitre présente la dérivation du modèle aux moments.

4.1 Description des interactions des particules chargées

La figure 4.1 présente les principales interactions des particules chargées du rayon ionisant avec celles des atomes constituant le corps du patient.

Elles se distinguent en trois types d'interactions :

- La diffusion élastique, où un électron ou un positron est dévié sans qu'il n'y ait perte d'énergie : la seule interaction dans cette catégorie est l'effet Mott [124] ;
- La diffusion inélastique radiative, où un électron ou un positron perd de l'énergie sous forme de photon : la seule interaction dans cette catégorie est l'effet Bremsstrahlung [125] ;
- La diffusion inélastique collisionnelle, où la particule injectée vient exciter un électron lié à un atome ou une molécule constituant le milieu, et entraîne sa séparation de l'atome :
  - pour les électrons, on parle d'effet Möller [126] ;
  - pour les positrons, d'effet Bhabha [127] ;
  - et pour les photons, ce sont l'effet photo-électrique [128] et l'effet Compton [129].

À noter que dans toutes ces interactions, il y a conservation de la quantité d'énergie. On remarque également figure 4.1 que l'on a mis un indice sur la particule résultante (par exemple γ') lorsque la particule incidente perd de l'énergie mais n'est pas absorbée par l'atome. Par la suite, on emploiera le terme *scattering* au lieu du terme de diffusion.

On doit se placer sous plusieurs hypothèses afin de formuler le modèle aux moments dans la partie suivante. Tout d'abord, les collisions entre 2 particules transportées sont négligées, car elles arrivent très rarement, et leur effet sur le milieu est par conséquent également négligé. On considère également que les particules injectées dans le domaine sont beaucoup moins nombreuses que celles le constituant, mais aussi que les particules du domaine (c'est-à-dire celles constituant le corps du patient) sont fixes : le milieu est donc assimilé à une distribution de particules initiale donnée. Enfin, le flux de particules injectées est considéré comme stationnaire, car le temps requis pour que ce flux devienne stationnaire est négligeable comparé au temps d'irradiation.

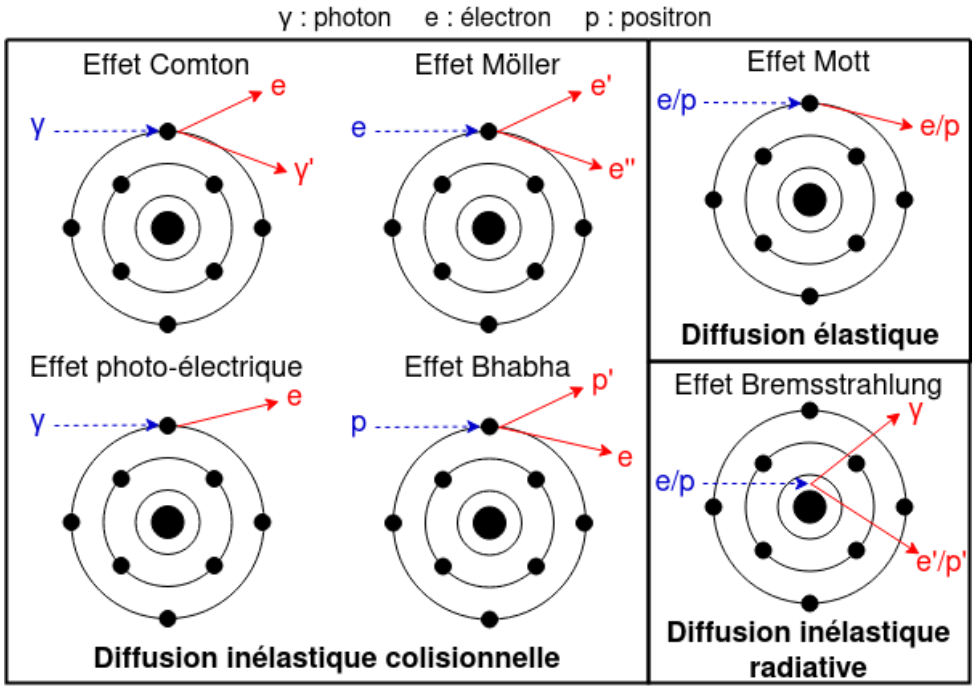


FIGURE 4.1 – Récapitulatif des principales interactions de particules chargées.

4.2 Dérivation du modèle aux moments M_1

Les hypothèses précédentes, couplées aux équations de transport de particules de Boltzmann-Vlasov, conduisent à l'équation stationnaire linéarisée suivante :

$$\begin{aligned}\boldsymbol{\Omega}.\nabla_{\boldsymbol{x}}\Psi(E,\boldsymbol{\Omega}) = &\int_{\mathbb{R}^+}\int_{S_2}\sigma_s(E',E,\boldsymbol{\Omega}'.\boldsymbol{\Omega})\Psi(E',\boldsymbol{\Omega}')\mathrm{d}\boldsymbol{\Omega}'\mathrm{d}E' \\ &-\int_{\mathbb{R}^+}\int_{S_2}\sigma_s(E,E',\boldsymbol{\Omega}.\boldsymbol{\Omega}')\mathrm{d}\boldsymbol{\Omega}'\mathrm{d}E'\Psi(E,\boldsymbol{\Omega}),\end{aligned} \tag{4.2.1}$$

où S_2 est la sphère unité sur $\mathbb{R}^3$, $\boldsymbol{\Omega}$ est la direction normalisée de propagation des particules dans l'espace des phases, $\Psi(E,\boldsymbol{\Omega})$ est la densité de probabilité, et $\sigma_s(E',E,\boldsymbol{\Omega}'.\boldsymbol{\Omega})$ est la section efficace différentielle, c'est-à-dire la probabilité qu'une particule à l'état $(E',\boldsymbol{\Omega}')$ se retrouve à l'état $(E,\boldsymbol{\Omega})$ après une collision . La dépendance en espace est omise ici afin d'alléger l'écriture.

On introduit également la section efficace totale σ_T telle que :

$$\sigma_T(E) = \int_{\mathbb{R}^+}\int_{S_2}\sigma_s(E,E',\boldsymbol{\Omega}.\boldsymbol{\Omega}')\mathrm{d}\boldsymbol{\Omega}'\mathrm{d}E'.$$

On obtient finalement un système de 3 équations, correspondant chacune à un type de particules (photons γ, électrons e et positrons p), et tenant compte des interactions provoquant la perte ou le gain de particules de ce type. Ainsi, l'équation pour les particules de type $j \in \{\gamma, e, p\}$ s'écrit de manière générique :

$$\boldsymbol{\Omega}.\nabla_{\boldsymbol{x}}\Psi^j(E) + \sigma_T^j(E)\Psi^j(E) = \sum_i \int_{\mathbb{R}^+}\int_{S_2}\sigma_s^{ij}(E',E,\boldsymbol{\Omega}'.\boldsymbol{\Omega})\Psi^i(E')\mathrm{d}\boldsymbol{\Omega}'\mathrm{d}E', \tag{4.2.2}$$

où les i, j sont les particules transportées, et σ_s^{ij} est la section efficace de génération de particules j par les particules i. Ces équations ont une dépendance en espace (la position

définie par les coordonnées (x, y, z) et la direction de vol définie par les angles (θ, ϕ)), et en énergie (E), soit 6 variables, ce qui serait trop coûteux à résoudre numériquement. C'est pour cela que l'on introduit la méthode aux moments, qui repose sur l'intégration de la densité de probabilité.

En plus de réduire le nombre de variables, cette méthode permet de maintenir une bonne précision de calcul. Elle est souvent utilisée pour la résolution d'équations de transport dans des domaines comme l'astrophysique, la physique des plasmas, ou encore la dynamique des gaz raréfiés. Le modèle M_1 présenté ici est dérivé des moments angulaires du système (4.2.2) et d'une relation de fermeture entropique [130]. L'application première de ce modèle était le transfert radiatif [130], et ce n'est que plus tard tard qu'il fut étendu à la radiothérapie [131].

Pour trouver ces moments, il suffit d'intégrer la densité de probabilité sur une sphère unité de l'espace des phases. On peut alors prendre plusieurs moments successifs, et les trois premiers moments sont ainsi définis comme :

$$\begin{aligned}
\Psi_0(\mathbf{r}, E) &= \int_{S_2} \Psi(\mathbf{r}, E, \mathbf{\Omega}) \mathrm{d}\mathbf{\Omega}, \\
\mathbf{\Psi_1}(\mathbf{r}, E) &= \int_{S_2} \mathbf{\Omega} \Psi(\mathbf{r}, E, \mathbf{\Omega}) \mathrm{d}\mathbf{\Omega}, \\
\overline{\mathbf{\Psi}}_{\mathbf{2}}(\mathbf{r}, E) &= \int_{S_2} \mathbf{\Omega} \otimes \mathbf{\Omega} \Psi(\mathbf{r}, E, \mathbf{\Omega}) \mathrm{d}\mathbf{\Omega}
\end{aligned} \tag{4.2.3}$$

où le moment d'ordre zéro, Ψ_0, est un scalaire, le moment d'ordre un, $\mathbf{\Psi_1}$, est un vecteur, et le moment d'ordre deux, $\overline{\mathbf{\Psi}}_{\mathbf{2}}$, est un tenseur. Après intégration sur l'espace des énergies, Ψ_0 et $\mathbf{\Psi_1}$ permettront de reconstruire respectivement l'énergie et le flux de particules.

Le modèle aux moments M_1 s'écrit finalement :

$$\begin{cases}
\boldsymbol{\nabla_r} \cdot \mathbf{\Psi_1}^i(E) + \sigma_T^i(E) \Psi_0^i(E) = \displaystyle\sum_{k=[i,j]} \int_{\mathbb{R}^+} \sigma_0^{ki}(E', E, \mathbf{\Omega'}.\mathbf{\Omega}) \Psi_0^k(E') \mathrm{dE}', \\
\boldsymbol{\nabla_r} \cdot \overline{\mathbf{\Psi}}_{\mathbf{2}}^{\,i}(E) + \sigma_T^i(E) \mathbf{\Psi_1}^i(E) = \displaystyle\sum_{k=[i,j]} \int_{\mathbb{R}^+} \sigma_1^{ki}(E', E, \mathbf{\Omega'}.\mathbf{\Omega}) \mathbf{\Psi_1}^k(E') \mathrm{dE}'.
\end{cases} \tag{4.2.4}$$

Ce système a 2 équations pour 3 inconnues (Ψ_0, $\mathbf{\Psi_1}$ et $\overline{\mathbf{\Psi}}_{\mathbf{2}}$). Il faut donc une fermeture afin d'exprimer $\overline{\mathbf{\Psi}}_{\mathbf{2}}$ en fonction des deux autres moments :

$$\overline{\mathbf{\Psi}}_{\mathbf{2}}(E) = \left[\frac{1 - \chi(||\boldsymbol{\alpha}||)}{2} \overline{\boldsymbol{Id}} + \frac{3\chi(||\boldsymbol{\alpha}||) - 1}{2} \boldsymbol{n} \otimes \boldsymbol{n} \right] \Psi_0, \tag{4.2.5}$$

où $\overline{\boldsymbol{Id}}$ est la matrice identité, et

$$\begin{cases}
\boldsymbol{n} = \dfrac{1}{||\mathbf{\Psi_1}||} \mathbf{\Psi_1}, \\
\boldsymbol{\alpha} = \dfrac{1}{\Psi_0} \mathbf{\Psi_1}, \\
\chi(||\boldsymbol{\alpha}||) = \dfrac{1 + ||\boldsymbol{\alpha}||^2 + ||\boldsymbol{\alpha}||^4}{3}.
\end{cases}$$

Cette fermeture résulte de la maximisation de l'entropie du système considéré, sous la contrainte de la résolution des deux premiers moments. Ce problème d'optimisation avec contraintes conduit à une solution pour la densité de probabilité strictement positive, qui s'écrit sous la forme d'une exponentielle d'un polynôme du premier degré. Ainsi, on a toujours $\Psi_0 \geq 0$, et par Cauchy-Schwarz, on a de plus $||\alpha|| \leq 1$. Ces deux conditions définissent l'ensemble des états admissibles.

Au final, ce système doit être résolu pour chaque type de particules, sur un maillage en espace de 1 à 3 dimensions, et sur un maillage en énergie à 1 dimension. Ce système possède également une propriété d'addition des effets des groupes en énergie. En effet, chaque groupe en énergie peut être calculé indépendament, et leurs effets sont additionnés pour calculer la dose totale à l'aide du premier moment grâce à la relation [131] :

$$D(\mathbf{r}) = \frac{T}{\rho(\mathbf{r})} \int_0^\infty S_P(\mathbf{r}, E)\Psi_0(\mathbf{r}, E)\mathrm{dE} \tag{4.2.6}$$

avec T est le temps d'irradiation du patient, $\rho(\mathbf{r})$ la densité des tissus irradiés, et $S_P(\mathbf{r}, E)$ le pouvoir d'arrêt.

Deux théories du transport de sources énergétiques ont été présentées dans cette deuxième partie du manuscrit. Avant d'appliquer sur chacune d'entre elles les connaissances sur les RNA abordées dans la première partie de cette thèse, on a d'abord étudié des cas d'école dont les résultats sont présentés au chapitre suivant.

Troisième partie

Premières analyses exploratoires des réseaux de neurones artificiels

Chapitre 5

Premières utilisations des RNA : de la classification binaire à la régression linéaire

Suite au Chapitre 1 présentant les notions fondamentales sur les réseaux de neurones profonds, ce chapitre sert d'étude préliminaire quant à leur utilisation pour résoudre différents types de problèmes, allant de la classification binaire à la régression linéaire. La première partie de ce chapitre présente une étude sur plusieurs problèmes de classification binaire. Tous les entraînements ont été réalisés grâce à un code Python implémenté sans librairie de *deep learning*, le but étant dans un premier temps de ne pas utiliser les réseaux comme boîte noire, et de compléter les connaissances théoriques avec une connaissance pratique des algorithmes d'apprentissage de base. La deuxième partie explore l'utilisation des PINN. Le cas choisi est l'équation de la chaleur, et on compare les performances des techniques exposées ultérieurement. Enfin, la troisième partie présente une première étude sur l'apprentissage de réseaux de neurones pour la modélisation du flux de chaleur non local. On ne s'intéresse qu'à des cas à 1 dimension, et de ce fait, le modèle choisi ici est le modèle de Luciani-Mora-Viremont.

5.1 RNA et problèmes de classification

On peut diviser les problèmes de classification en trois familles :
- la classification binaire, s'il s'agit de distinguer entre 2 classes de données ;
- la classification multi-classe, si le nombre de classes est supérieur à 2 ;
- la classification multi-label, si la donnée peut appartenir à plusieurs classes.

Il est d'usage d'apprendre au réseau à prédire la probabilité qu'une donnée appartienne à une certaine classe, plutôt que de lui faire prédire directement la classe de la donnée (ce qui s'apparenterait plutôt à une régression linéaire). Pour ce faire, dans le cas des classifications multi-classe et multi-label, la couche de sortie est constituée d'autant de neurones que de classes, et la sortie de chaque neurone correspond à la probabilité que la donnée d'entrée appartienne à une certaine classe. Par exemple, pour une classification d'images de chiffres manuscrits, on aura autant de neurones en sortie que de chiffres, soit 10 au total, et la sortie du neurone $n \in \{0, ..., 9\}$ correspond à la probabilité que l'image contienne le chiffre n. Dans le cas de la classification binaire, un seul neurone en sortie suffit : une probabilité ≥ 0.5 signifie que le réseau reconnaît la classe A, et à l'inverse, une probabilité < 0.5 signifie que le réseau reconnaît la classe B.

Dans cette partie on s'intéresse uniquement à des problèmes de classification binaire. On considère des données d'entrée correspondant aux coordonnées de points dans l'espace 2D, et des sorties cibles correspondant à la couleur des points. Dans un premier temps, on montrera les limites du perceptron de Rosenblatt, qui ne peut que résoudre des problèmes avec des frontières de décision dites linéaires. C'est cette limitation qui pousse à augmenter le nombre de neurones et former un réseau, permettant ainsi de résoudre des problèmes avec des frontières de décision dites non linéaires.

5.1.1 Classification binaire avec frontière de décision linéaire

Par définition, le perceptron recherche une frontière linéaire, qu'on appelle également frontière de décision, afin de séparer les données en deux classes. En effet, on rappelle que la prédiction du perceptron $\widehat{y}$ est définie par :

$$\widehat{y} = H(z) = \begin{cases} 1 \text{ si } z \geq 0, \\ \\ 0 \text{ sinon,} \end{cases} \quad \text{et} \quad z = \sum_{i=1}^{n} w_i x_i + b,$$

et que la règle d'apprentissage du perceptron est :

$$\begin{aligned} w_i &\longleftarrow w_i + \alpha(y - \widehat{y})x_i, \\ b &\longleftarrow b + \alpha(y - \widehat{y}). \end{aligned} \tag{5.1.1}$$

Dans le cas de données d'entrée à 2 dimensions, notées (x_1, x_2), l'apprentissage du perceptron (Algorithme 1) équivaut en réalité à la recherche d'une droite d'équation :

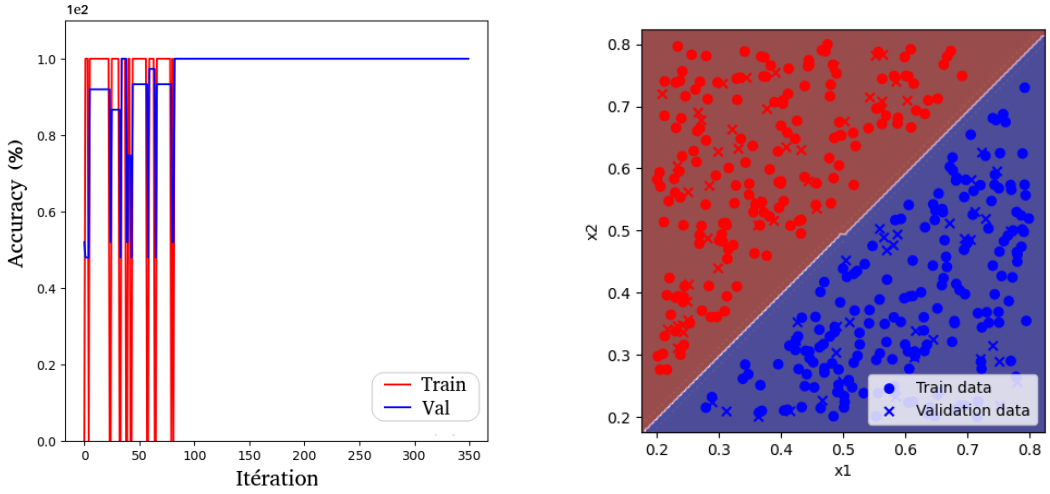
$$\sum_{i=1}^{2} w_i x_i + b = 0 \Longleftrightarrow x_2 = -\frac{w_1}{w_2} x_1 - \frac{b}{w_2}.$$

En d'autres termes, les poids définissent la pente de la frontière, et le biais définit l'ordonnée à l'origine. On comprend alors l'utilité du biais : si on l'omet, on aurait toujours une frontière passant par l'origine du repère.

La Figure 5.1 présente les résultats obtenus avec un perceptron de Rosenblatt sur un jeu de données séparables linéairement. Ce jeu de données est constitué de 500 points

Algorithme 1 Apprentissage du perceptron de Rosenblatt

Définir: α, N_{epochs}
Initialiser: W, b
  Tant que $epoch \leq N_{epochs}$ **faire**
    Pour $k \leq card(\mathcal{D}_{train})$ **faire**
      $y \leftarrow H(W^T X_k + b)$
      Si $y \neq \widehat{y}$ **alors**
        $W \leftarrow W + \alpha(y_k - \widehat{y_k})X_k$
        $b \leftarrow b + \alpha(y_k - \widehat{y_k})$
      Fin Si
    Fin Pour
  Fin Tant que



(a) Taux de réussite

(b) Carte de prédiction

FIGURE 5.1 – Apprentissage d'un perceptron sur des données séparables linéairement.

rouges ou bleus, avec une répartition de 70% des données pour l'apprentissage (soit 350 données) et 15% pour la validation (soit 75 données) et 15% pour le test. Le taux d'apprentissage ici est $\alpha = 0.01$. La courbe du taux de réussite Figure 5.1a montre que l'algorithme converge dès la 83^e itération, c'est-à-dire qu'en ayant vu seulement 83 données sur les 350 disponibles, le perceptron a pu déterminer une frontière de décision séparant les données correctement. On constate également de fortes oscillations sur les premières itérations, dû au fait qu'une seule donnée est évaluée à chaque itération. En effet, comme le neurone voit les données une à une, le taux de réussite est exactement 1 ou zéro, et la mise à jour des paramètres avec une donnée peut avoir rendu la frontière de décision incorrecte pour la donnée suivante.

On peut voir la répartition des données sur la Figure 5.1b, où les ronds sont les données de l'ensemble d'apprentissage et les croix celles de l'ensemble de validation. On peut également y voir la frontière de décision qui a été identifiée par le perceptron à la fin de l'apprentissage en arrière-plan : celle-ci sépare effectivement les données.

Prenons maintenant un neurone formel avec une fonction d'activation sigmoïd, notée σ, et l'entropie croisée comme fonction perte. On a vu au Chapitre 1 que la règle d'apprentissage définie comme une descente de gradient s'écrit :

$$\theta \leftarrow \theta - \alpha \frac{\partial}{\partial \theta} loss(y, \widehat{y}).$$

La dérivée de l'entropie croisée se calcule à l'aide de la règle des dérivées en chaîne :

$$\begin{aligned}
\frac{\partial}{\partial \theta} loss(y, \widehat{y}) =& \frac{\partial loss(y, \widehat{y})}{\partial \widehat{y}} \times \frac{\partial \sigma(z)}{\partial z} \times \frac{\partial z}{\partial \theta}, \\
=& \frac{-(y - \widehat{y})}{\widehat{y}(1 - \widehat{y})} \times \widehat{y}(1 - \widehat{y}) \times \frac{\partial z}{\partial \theta}, \\
=& -(y - \widehat{y}) \frac{\partial z}{\partial \theta},
\end{aligned}$$

soit finalement

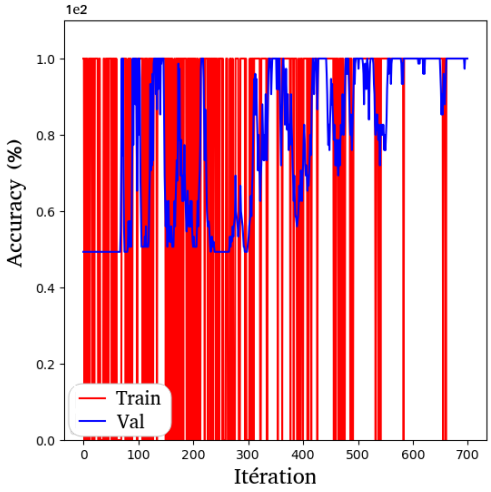
$$\begin{aligned}
w_i &\leftarrow w_i + \alpha(y - \widehat{y}) x_i, \\
b &\leftarrow b + \alpha(y - \widehat{y}).
\end{aligned} \tag{5.1.2}$$

Algorithme 2 Apprentissage d'un neurone simple avec activation sigmoïd et *loss* entropie croisée (*batchsize*=1)

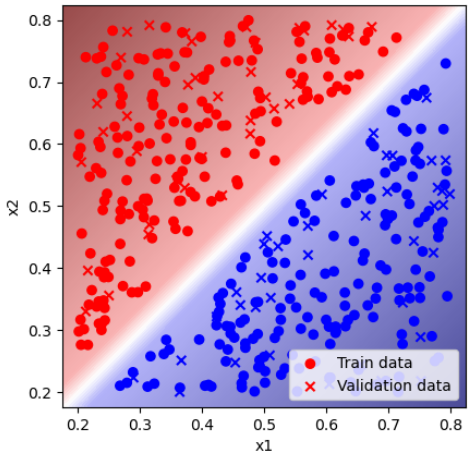
Définir: α, N_{epochs}
Initialiser: W, b
  Tant que $epoch \leq N_{epochs}$ **faire**
    Pour $k \leq card(\mathcal{D}_{train})$ **faire**
      $\widehat{y}_k \leftarrow \sigma(W^T X_k + b)$
      $W \leftarrow W + \alpha(y_k - \widehat{y}_k)X_k$
      $b \leftarrow b + \alpha(y_k - \widehat{y}_k)$
    Fin Pour
  Fin Tant que

On retombe finalement sur la même règle d'apprentissage que le perceptron (Algorithme 2).

La Figure 5.2 présente l'apprentissage d'un neurone avec activation sigmoïd et *loss* entropie croisée sur le même jeu de données que précédemment, ainsi qu'avec le même taux d'apprentissage $\alpha = 0.01$. La Figure 5.2a montre que le taux de réussite oscille bien plus que chez le perceptron. La Figure 5.2b montre la répartition des données dans les ensembles d'entraînement (ronds) et de validation (croix), ainsi que la frontière de décision identifiée par le neurone à la fin de l'apprentissage en arrière-plan. On note ici que les couleurs en arrière-plan sont moins fortes que chez le perceptron : cela vient du fait que le perceptron prédit des valeurs entières qui sont 0 ou 1, alors que le neurone avec sigmoïd prédit des valeurs réelles, comprises entre 0 et 1, laissant ainsi apparaître plus de nuance quant à la certitude des prédictions (on peut dire que lorsque la prédiction est très proche de 0 ou de 1, le neurone est "sûr" de sa réponse).



(a) Taux de réussite



(b) Carte de prédiction

FIGURE 5.2 – Apprentissage d'un neurone avec activation sigmoïd et *loss* entropie croisée sur des données séparables linéairement (*batchsize*=1).

Algorithme 3 Apprentissage d'un neurone simple avec activation sigmoïd, *loss* entropie croisée et *batchsize* quelconque

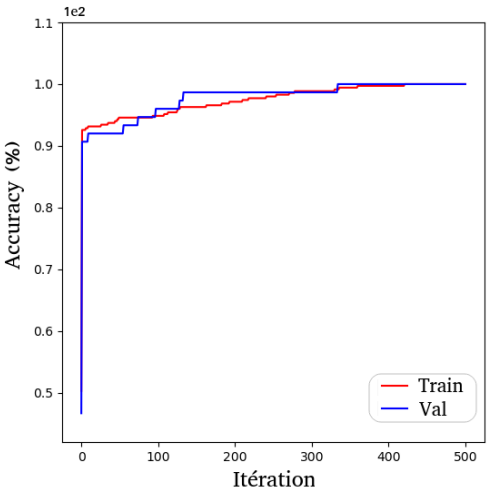
Définir: $\alpha, N_{epochs}, nbatchs, batchsize$
Initialiser: W, b
  Tant que $epoch \leq N_{epochs}$ **faire**
    Pour $b \leq nbatchs$ **faire**
      $S_W = 0, S_b = 0$
      Pour $k \leq batchsize$ **faire**
        $\widehat{y}_k \leftarrow \sigma(W^T X_k + b)$
        $S_W \leftarrow S_W + (y_k - \widehat{y}_k) X_k$
        $S_b \leftarrow S_b + (y_k - \widehat{y}_k)$
      Fin Pour
      $W \leftarrow W + \alpha(S_W / batchsize)$
      $b \leftarrow b + \alpha(S_b / batchsize)$
    Fin Pour
  Fin Tant que

On prend maintenant la règle d'apprentissage pour un *batch* de taille quelconque :

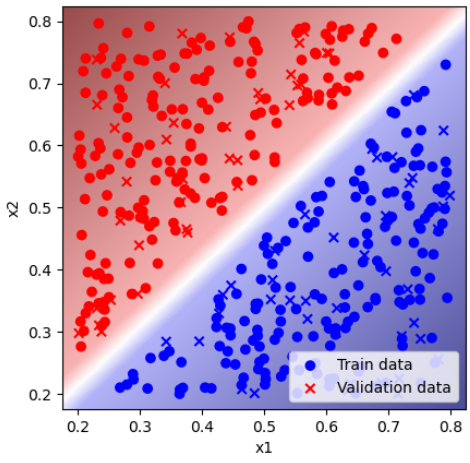
$$\theta \leftarrow \theta - \alpha \frac{1}{batchsize} \sum_{d=1}^{batchsize} \frac{\partial}{\partial \theta} loss(y^d, \widehat{y}^d),$$

où *batchsize* désigne la taille d'un *batch* (Algorithme 3).

On a vu au Chapitre 1 que l'augmentation du batchsize atténue les oscillations : c'est bien ce que l'on observe Figures 5.3 et 5.4. En comparant l'évolution du taux de réussite avec *batchsize=ntrain* (Figure 5.3a), *batchsize=ntrain/2* (Figure 5.4a) et *batchsize=ntrain/10* (Figure 5.4c), on constate en effet que plus la taille de batch est petite, plus on observe des oscillations.

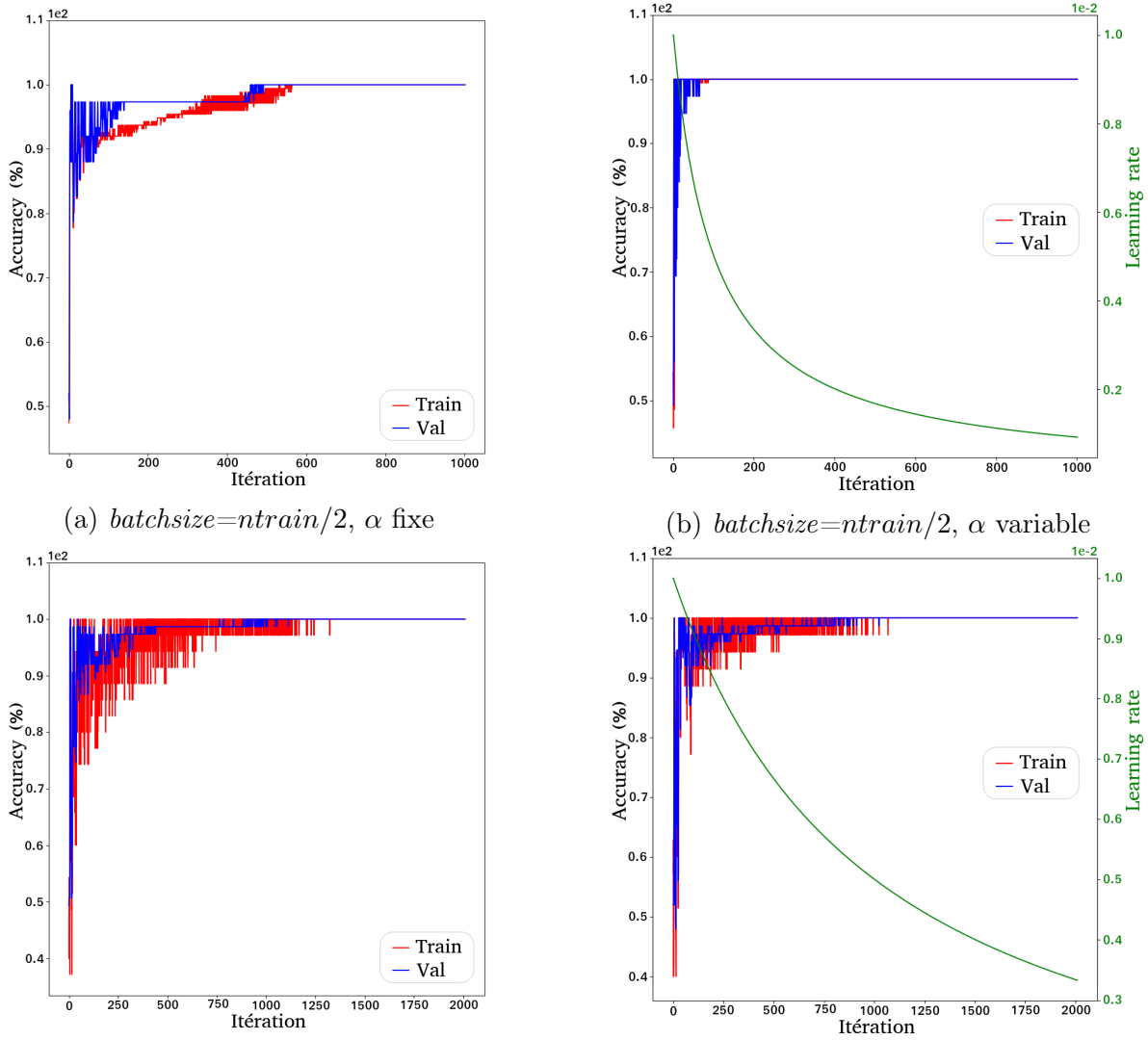


(a) Taux de réussite



(b) Carte de prédiction

FIGURE 5.3 – Apprentissage d'un neurone avec activation sigmoïd et *loss* entropie croisée sur des données séparables linéairement (*batchsize=ntrain*).



(a) *batchsize*=*ntrain*/2, α fixe

(b) *batchsize*=*ntrain*/2, α variable

(c) *batchsize*=*ntrain*/10, α fixe

(d) *batchsize*=*ntrain*/10, α variable

FIGURE 5.4 – Apprentissage d'un neurone simple : influence du *batchsize* et du taux d'apprentissage variable sur les oscillations du taux de réussite.

Historiquement, deux visions s'opposent : d'un côté le *batch training*, et de l'autre l'*online training* [132]. Le *batch training* consiste à mettre à jour les paramètres à l'aide de l'accumulation des gradients évalués avec chaque donnée d'entraînement (et non pas sur un *minibatch*, comme son nom semble l'insinuer) ; à l'inverse, l'*online* training consiste à mettre à jour les paramètres avec le gradient évalué sur une donnée à la fois [57]. Pendant longtemps, les adeptes du *batch training* ont assuré que cette technique calcule la véritable direction du gradient pour les mises à jour des paramètres, et qu'en conséquence, elle serait plus correcte, en plus d'être plus rapide. À l'opposé, les adeptes de l'*online training* ont prouvé que la mise à jour des paramètres à chaque donnée laisse la possibilité à l'algorithme de descente de se corriger avec la donnée suivante si la mise à jour l'a mené dans une direction qui finalement s'éloigne d'un minimum local [132]. Dans leur article, Wilson et Martinez ont également montré que l'argument de rapidité du *batch training* tenait au fait que les études jusqu'alors avaient été menées pour de faibles taux d'apprentissage, et qu'avec des taux plus élevés, l'*online training* convergeait plus rapidement vers un taux de réussite élevé que le *batch training* avec un taux d'apprentissage faible [132].

Un moyen souvent utilisé pour maîtriser les oscillations dû à l'utilisation de *batchs* est de faire varier le taux d'apprentissage durant l'entraînement. En plus de cela, le taux variable peut améliorer la performance finale d'un réseau tout en diminuant le temps d'apprentissage (i.e. le nombre d'itérations). Il existe plusieurs méthodes pour faire décroitre la valeur du taux d'apprentissage, qu'on appelle plus communément *schedulers* [96]. Parmi les *schedulers* les plus communément utilisés, on trouve :

- le *time decay*, où $\alpha_t = \dfrac{\alpha_0}{1 + decay \cdot t}$,
- le *step decay*, où $\alpha_t = \alpha_0 \cdot decay^{t/drop}$,
- et l'*exponential decay*, où $\alpha_t = \alpha_0 \cdot exp(-decay \cdot t)$,

avec α_t le taux d'apprentissage à l'itération t, *decay* une valeur réelle entre 0 et 1 définissant le taux de décroissance, et *drop* un entier définissant le nombre d'itérations entre chaque changement de valeur du taux dans le cas du *step decay*.

On regarde maintenant l'influence d'un taux d'apprentissage variable avec *time decay* sur le neurone avec activation sigmoïd et *loss* entropie croisée, lorsque *batchsize*=$ntrain/10$ (Figure 5.4d) et *batchsize*=$ntrain/2$ (Figure 5.4b) : en adaptant la valeur du *decay* à chaque *batchsize* (respectivement $decay = 10^{-3}$ et $decay = 10^{-2}$), on constate finalement qu'avec un taux d'apprentissage décroissant, les courbes d'apprentissage sont plus lisses et qu'on atteint plus rapidement 100% de réussite sur les ensembles d'entraînement et de validation.

Si un perceptron ou un neurone simple est garanti de converger pour des données séparables linéairement, ce n'est pas le cas pour des données non séparables linéairement. Dans ce cas, l'algorithme pourra au mieux converger vers une solution avec l'erreur la plus petite possible. Afin de traiter des frontières de décision non linéaire, il faut associer plusieurs neurones ensemble et former un réseau.

5.1.2 Classification binaire avec frontière de décision non linéaire

Dans un premier temps, on s'intéresse à la classification de données distribuées en damier à 4 zones. En principe, un réseau avec une couche cachée de 4 neurones, comme décrit Figure 5.5, est suffisant afin de résoudre ce problème (1 neurone caché pour chaque zone du damier). S'agissant d'un réseau avec une couche cachée, on a maintenant besoin d'expliciter l'algorithme d'apprentissage en deux phases : d'abord la passe avant, puis la rétropropagation des gradients. Dans un premier temps, on définit les matrices et vecteurs d'intérêt mentionnés Figure 5.5 :

- les poids et biais :
  - $\mathbf{w} = \begin{pmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{24} \end{pmatrix}$ où w_{ij} est le poids reliant l'entrée i au neurone j de la couche cachée,
  - $\mathbf{b} = \begin{pmatrix} b_1 & b_2 & b_3 & b_4 \end{pmatrix}$ où b_j est le biais du neurone j de la couche cachée,
  - $\mathbf{v} = \begin{pmatrix} v_1 & v_2 & v_3 & v_4 \end{pmatrix}$ où v_j est le poids reliant le neurone j de la couche cachée au neurone de sortie,
  - $\tilde{b}$ le biais du neurone de sortie,
- les signaux post-synaptiques et activations :
  - $\mathbf{z} = \left(\mathbf{z}_k\right)_{1 \leq k \leq batchsize}$ où $\mathbf{z}_k = \begin{pmatrix} z_{k,1} & z_{k,2} & z_{k,3} & z_{k,4} \end{pmatrix}$ avec $z_{k,j}$ le signal post-synaptique du neurone j de la couche cachée pour la donnée k,

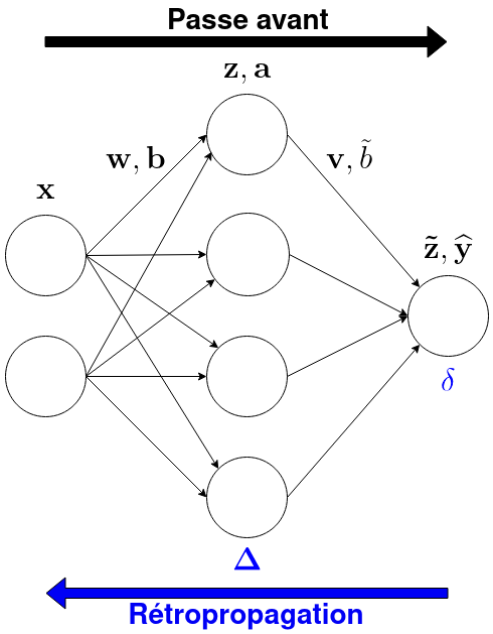


FIGURE 5.5 – Schéma d'un réseau à 1 couche cachée de 4 neurones et 1 neurone en sortie

- $\mathbf{a} = \left(\mathbf{a}_k\right)_{1\leq k\leq batchsize}$ où $\mathbf{a}_k = \begin{pmatrix} a_{k,1} & a_{k,2} & a_{k,3} & a_{k,4} \end{pmatrix}$ avec $a_{k,j}$ l'activation du neurone j de la couche cachée pour la donnée k,
- $\tilde{\mathbf{z}} = \left(\tilde{z}_k\right)_{1\leq k\leq batchsize}$ où $\tilde{z}_k$ est le signal post-synaptique du neurone de sortie pour la donnée k,
- $\widehat{\mathbf{y}} = \left(\widehat{y}_k\right)_{1\leq k\leq batchsize}$ où $\widehat{y}_k$ est l'activation du neurone de sortie pour la donnée k,

- les gradients :
  - $\mathbf{\Delta} = \left(\mathbf{\Delta}_k\right)_{1\leq k\leq batchsize}$ où $\mathbf{\Delta}_k \in \mathbb{R}^4$ est le gradient à la couche cachée pour la donnée k,
  - $\delta = \left(\delta_k\right)_{1\leq k\leq batchsize}$ où $\delta_k \in \mathbb{R}$ est le gradient à la couche de sortie pour la donnée k,
- $\mathbf{x} = \left(\mathbf{x}_k\right)_{1\leq k\leq batchsize}$ où $\mathbf{x} = \begin{pmatrix} x_1 & x_2 \end{pmatrix}$ est la donnée d'entrée k.

La Figure 5.5 montre les deux étapes de calcul qui sont maintenant nécessaires pour la mise à jour des paramètres : la passe avant où l'information d'entrée est propagée vers la couche de sortie, puis la rétropropagation des gradients. Nous avions précédemment explicité la règle de mise à jour des paramètres pour un réseau à L couches cachées au Chapitre 1 :

$$\begin{aligned} w_{ij}^l &\leftarrow w_{ij}^l + \alpha a_i^{l-1} \Delta_j^l, \\ b_j^l &\leftarrow b_j^l + \alpha \Delta_j^l, \end{aligned}$$

où $(\Delta_j^l)_{j\in\mathcal{C}^l}^{1\leq l\leq L+1}$ est la variable rétropropagée de la couche de sortie vers la première couche cachée, et définie par :

- $\Delta^{L+1} = -\dfrac{\partial loss(y, a^{L+1})}{\partial a^{L+1}} \dfrac{\partial a^{L+1}}{\partial z^{l+1}}$ pour tout neurone de la couche de sortie $\mathcal{C}^{L+1}$,
- $\Delta_j^l = \dfrac{\partial a_j^l}{\partial z_j^l} \displaystyle\sum_{k\in\mathcal{C}^{l+1}} \Delta_k^{l+1} w_{jk}^{l+1}$ pour le neurone j de la couche cachée $\mathcal{C}^l$ $(1 \leq l \leq L)$.

Algorithme 4 Apprentissage d'un réseau à 1 couche cachée pour une classification binaire

Définir: $\alpha, N_{epochs}, nbatchs, batchsize$
Initialiser: $\mathbf{w}, \mathbf{b}, \mathbf{v}, \tilde{b}$
  Tant que $epoch \leq N_{epochs}$ **faire**
    Pour $b \leq nbatchs$ **faire**
      Pour $k \leq batchsize$ **faire**
        $\mathbf{z}_k \leftarrow \mathbf{w}^T \mathbf{x}_k + \mathbf{b}$
        $\mathbf{a}_k \leftarrow f(\mathbf{z}_k)$
        $\tilde{z}_k \leftarrow \mathbf{v}^T \mathbf{a}_k + \tilde{b}$
        $\widehat{y}_k \leftarrow \sigma(\tilde{z}_k)$
        $\delta_k \leftarrow -\dfrac{\partial loss(y_k, \widehat{y}_k)}{\partial \widehat{y}_k} \dfrac{\partial \sigma(\tilde{z}_k)}{\partial \tilde{z}_k}$
        $\mathbf{\Delta_k} \leftarrow (\overrightarrow{\nabla}_{\mathbf{z}_k} \cdot \mathbf{a}_k) \cdot (\delta_k \mathbf{v})$
      Fin Pour
      $\mathbf{w} \leftarrow \mathbf{w} + \alpha \overline{\mathbf{\Delta} \cdot \mathbf{x}}$
      $\mathbf{b} \leftarrow \mathbf{b} + \alpha \overline{\mathbf{\Delta}}$
      $\mathbf{v} \leftarrow \mathbf{v} + \alpha \overline{\delta \cdot \mathbf{a}}$
      $\tilde{b} \leftarrow \tilde{b} + \alpha \overline{\delta}$
    Fin Pour
  Fin Tant que

On peut maintenant écrire cette règle pour le réseau à 1 couche cachée considéré ici :

$$\begin{aligned} &\delta \leftarrow -\frac{\partial loss(y, \widehat{y})}{\partial \widehat{y}} \frac{\partial \sigma(\tilde{z})}{\partial \tilde{z}}, \\ &v_i \leftarrow v_i + \alpha a_i \delta, \\ &\tilde{b} \leftarrow \tilde{b} + \alpha \delta, \\ &\Delta_j \leftarrow \frac{\partial a_j}{\partial z_j} \times \delta \times v_j, \\ &w_{ij} \leftarrow w_{ij} + \alpha x_i \Delta_j, \\ &b_j \leftarrow b_j + \alpha \Delta_j, \end{aligned} \tag{5.1.3}$$

d'où l'Algorithme 4, où $(\mathbf{\Delta}_k \cdot \mathbf{x}_k)_j = (\mathbf{\Delta}_k)_j \times (\mathbf{x}_k)_j$ désigne le produit élément à élément de deux matrices de même taille, et $\overline{\mathbf{\Delta}} = \dfrac{1}{batchsize} \sum_{k}^{batchsize} \mathbf{\Delta}_k$.

Les premiers tests sur cette base de données ont été peu concluant : en effet, les seules frontières de décision trouvées par les réseaux étaient en fait des frontières linéaires. Ces résultats prouvent principalement deux choses : d'une part qu'initialiser les paramètres à zéro n'est pas une bonne stratégie, et d'autre part que la simple descente de gradient n'est pas suffisante.

Il est en fait d'usage d'initialiser les poids avec des valeurs aléatoires (mais proches de zéro). En effet, si des neurones d'une couche sont liés à la même donnée d'entrée avec les mêmes valeurs de paramètres et ont une fonction d'activation identique, alors l'algorithme de descente les mettra à jour avec exactement les mêmes valeurs [57] et ne pourra donc pas converger. Ainsi, il est impératif d'initialiser les poids avec des valeurs aléatoires pour casser la symétrie (les biais eux peuvent être initialisés à zéro). Les premières techniques d'initialisation des poids était basées sur des règles heuristiques simples, à savoir des

valeurs aléatoires comprises entre 0 et 1 ou bien entre -1 et 1, mais des techniques plus pointues ont vu le jour depuis. Parmi les plus connues, on trouve :

- l'initialisation Xavier [133], où les valeurs sont distribuées selon une distribution uniforme entre les valeurs $-1/\sqrt{n}$ et $1/\sqrt{n}$ avec n le nombre de dimensions de la couche précédente,
- l'initialisation Xavier normalisée, où les valeurs sont distribuées selon une distribution uniforme entre les valeurs $-\frac{\sqrt{6}}{\sqrt{n+m}}$ et $\frac{\sqrt{6}}{\sqrt{n+m}}$ avec n le nombre de dimensions de la couche précédente et m le nombre de dimensions de la couche qu'on initialise,
- l'initialisation He [134], où les valeurs sont distribuées selon une gaussienne avec une moyenne de 0 et un écart-type de $\sqrt{2/n}$ avec n le nombre de dimensions de la couche précédente.

Les performances de chacune de ces initialisations, ainsi que l'initialisation à zéro, sont comparées Figure 5.6. Ces résultats sont obtenus pour l'apprentissage du même réseau que précédemment, à savoir un réseau de 1 couche cachée à 4 neurones avec activation sigmoïd. Comme déjà mentionné plus haut, il a également fallut rajouter un moment dans l'algorithme de descente. On rappelle la définition de la règle de mise à jour vue au Chapitre 1 :

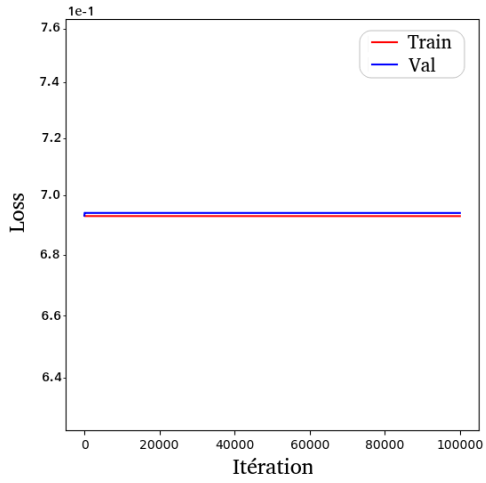
$$\begin{aligned} m^{(t)} &= \beta m^{(t-1)} + \alpha \nabla_\theta \mathcal{L}_T^{(t)}(\{X, Y\}_b), \\ \theta &\leftarrow \theta - m^{(t)}, \end{aligned}$$

où $v^{(t)}$ est le moment à l'itération courante, $\nabla_\theta \mathcal{L}_T^{(t)}(\{X, Y\}_b)$ désigne gradient à l'itération courante calculé sur les donnée d'un *batch*, α est le taux d'apprentissage, et β est un nombre réel entre 0 et 1 de pondération des contributions des gradients antérieurs (plus communément appelé moment). En développant cette définition pour notre réseau de 1 couche cachée à 4 neurones, on trouve finalement :

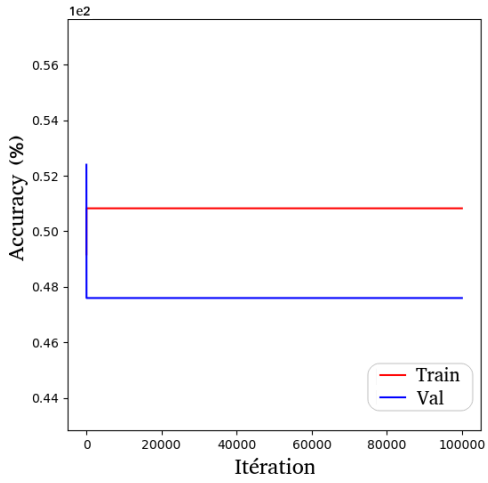
$$\begin{aligned} \delta &\leftarrow -\frac{\partial loss(y, \widehat{y})}{\partial \widehat{y}} \frac{\partial \sigma(\tilde{z})}{\partial \tilde{z}}, \\ m_{v_i} &\leftarrow \beta m_{v_i} + \alpha a_i \delta \\ v_i &\leftarrow v_i + m_{v_i}, \\ m_{\tilde{b}} &\leftarrow \beta m_{\tilde{b}} + \alpha \delta \\ \tilde{b} &\leftarrow \tilde{b} + m_{\tilde{b}}, \\ \Delta_j &\leftarrow \frac{\partial a_j}{\partial z_j} \times \delta \times v_j, \\ m_{w_{ij}} &\leftarrow \beta m_{w_{ij}} + \alpha x_i \Delta_j \\ w_{ij} &\leftarrow w_{ij} + m_{w_{ij}}, \\ m_b &\leftarrow \beta m_b + \alpha \Delta_j, \\ b_j &\leftarrow b_j + m_b, \end{aligned} \tag{5.1.4}$$

d'où l'Algorithme 5.

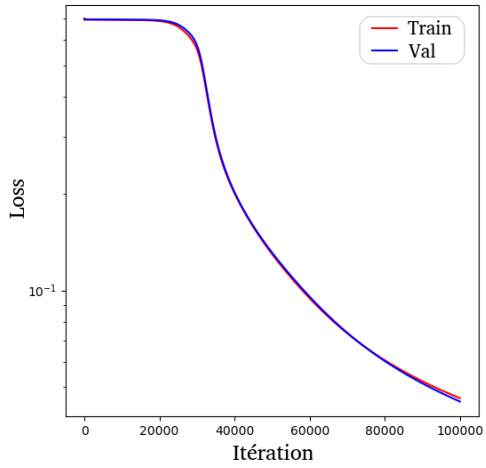
On entraîne la même architecture, avec $\alpha = 0.01$, $batchsize = ntrain/10$, et $\beta = 0.01$, et en faisant varier l'initialisation des poids. Tout d'abord, sur les Figures 5.6a et 5.6b, on observe en effet que l'algorithme est comme coincé lorsque les poids sont initialisés à zéro, et ce même avec l'ajout d'un moment : la *loss* reste constante pendant tout l'entraînement et le taux de réussite n'évolue pas non plus. En revanche, on constate que l'algorithme arrive à converger avec les initialisations aléatoires Xavier (Figures 5.6c et 5.6d) et Xavier normalisée (Figures 5.6e et 5.6f). On peut voir sur les évolutions des *loss* (Figures 5.6c et



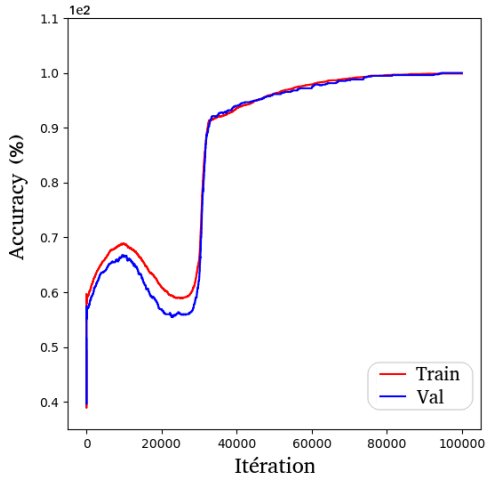
(a) Initialisation zéro : *loss*



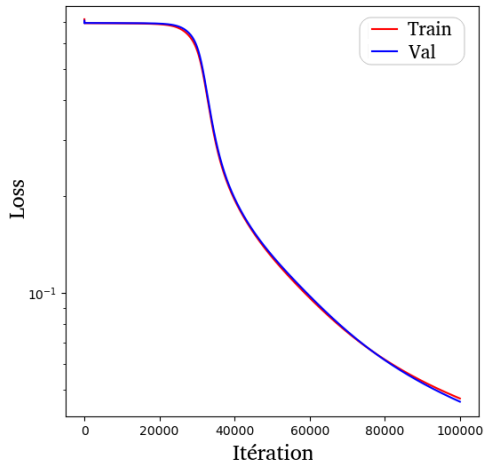
(b) Initialisation zéro : taux de réussite



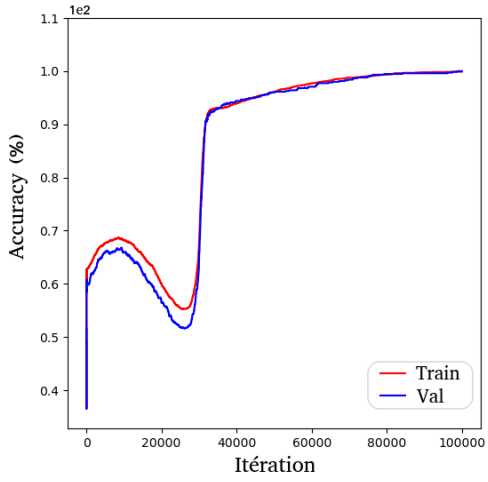
(c) Initialisation Xavier : *loss*



(d) Initialisation Xavier : taux de réussite



(e) Initialisation Xavier normalisée : *loss*



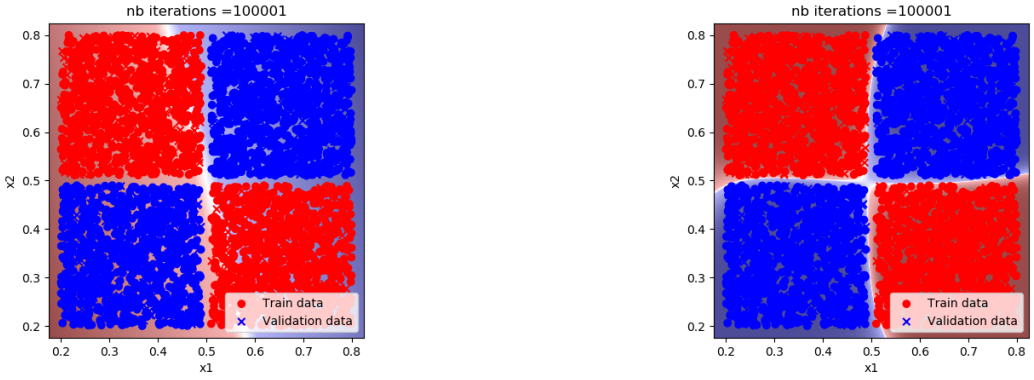
(f) Initialisation Xavier normalisée : taux de réussite

FIGURE 5.6 – Apprentissage d'un réseau à 1 couche cachée de 4 neurones : influence de l'initialisation des poids

Algorithme 5 Apprentissage d'un réseau à 1 couche cachée pour une classification binaire (avec moment)

Définir: $\alpha, \beta, N_{epochs}, nbatchs, batchsize$
Initialiser: $\mathbf{w}, \mathbf{b}, \mathbf{v}, \tilde{b}$
  $\mathbf{m_v} \leftarrow 0$, $m_{\tilde{b}} \leftarrow 0$, $\mathbf{m_w} \leftarrow 0$, $\mathbf{m_b} \leftarrow 0$
  Tant que $epoch \leq N_{epochs}$ **faire**
    Pour $b \leq nbatchs$ **faire**
      Pour $k \leq batchsize$ **faire**
        $\mathbf{z}_k \leftarrow \mathbf{w}^T \mathbf{x}_k + \mathbf{b}$
        $\mathbf{a}_k \leftarrow f(\mathbf{z}_k)$
        $\tilde{z}_k \leftarrow \mathbf{v}^T \mathbf{a}_k + \tilde{b}$
        $\widehat{y}_k \leftarrow \sigma(\tilde{z}_k)$
        $\delta_k \leftarrow -\frac{\partial loss(y_k, \widehat{y}_k)}{\partial \widehat{y}_k} \frac{\partial \sigma(\tilde{z}_k)}{\partial \tilde{z}_k}$
        $\mathbf{\Delta_k} \leftarrow (\overrightarrow{\nabla}_{\mathbf{z}_k} \cdot \mathbf{a}_k) \cdot (\delta_k \mathbf{v})$
      Fin Pour
      $\mathbf{m_w} \leftarrow \beta \mathbf{m_w} + \alpha \overline{\mathbf{\Delta} \cdot \mathbf{x}_k}$
      $\mathbf{m_b} \leftarrow \beta \mathbf{m_b} + \alpha \overline{\mathbf{\Delta}}$
      $\mathbf{m_v} \leftarrow \beta \mathbf{m_v} + \alpha \overline{\delta \cdot \mathbf{a}}$
      $m_{\tilde{b}} \leftarrow \beta m_{\tilde{b}} + \alpha \overline{\delta}$
      $\mathbf{w} \leftarrow \mathbf{w} + \mathbf{m_w}$
      $\mathbf{b} \leftarrow \mathbf{b} + \mathbf{m_b}$
      $\mathbf{v} \leftarrow \mathbf{v} + \mathbf{m_v}$
      $\tilde{b} \leftarrow \tilde{b} + m_{\tilde{b}}$
    Fin Pour
  Fin Tant que

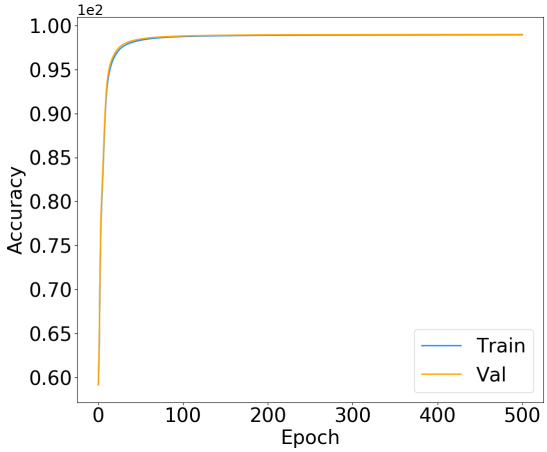
5.6e) qu'après plus de 30 000 itérations à stagner autour d'une valeur, l'algorithme fini enfin par diminuer fortement : l'algorithme a en fait réussi à sortir d'un minimum local grâce à l'ajout du moment. Parallèlement, on voit qu'au même moment, le taux de réussite (Figures 5.6d et 5.6f) fait un bon, passant d'environ 60% à plus de 90%, confirmant ainsi que le réseau, en s'éloignant du premier minimum local, a réussi à en trouver un meilleur.



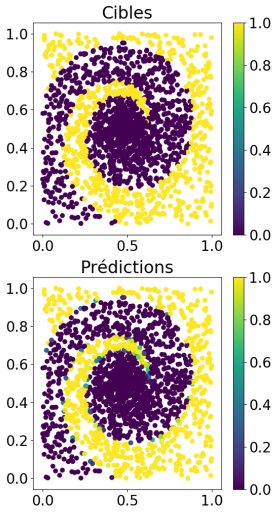
(a) Initialisation zéro

(b) Initialisation Xavier

FIGURE 5.7 – Cartes de prédiction de l'apprentissage d'un réseau à 1 couche cachée de 4 neurones : influence de l'initialisation des poids



(a) Taux de réussite



(b) Prédiction sur l'ensemble test

FIGURE 5.8 – Entraînement d'un réseau à 2 couches cachées de 10 neurones avec sigmoïd, sur des données organisées en spirale

La Figure 5.7 présente les cartes de prédiction à la fin de l'entraînement. Comme déjà mentionné, avec l'initialisation zéro, le réseau ne reconnait qu'une frontière linéaire (Figure 5.7a), alors que les quatres zones du damier sont bien reconnues avec une initialisation aléatoire, et en l'occurence ici il s'agit de l'initialisation Xavier (Figure 5.7b).

Un optimiseur plus avancé peut également permettre de résoudre des problèmes avec des frontières encore plus complexes. Par exemple, on a entraîné un réseau à 2 couches cachées de 10 neurones avec sigmoïd, sur des données organisées en spirale, à l'aide de l'optimiseur Adam. Les résultats, obtenus grâce à la librairie Tensorflow, sont montrés Figure 5.8 : le taux de réussite augmente durant l'entraînement jusqu'à atteindre 99% (Figure 5.8a), performance qui est confirmée sur l'ensemble test (Figure 5.8b).

5.2 Résolution de l'équation de la chaleur par des PINN

On s'intéresse maintenant aux PINN introduits au Chapitre 2, pour résoudre l'équation de transport de chaleur 1D. D'abord, le système et sa solution analytique sont présentés. Ensuite, deux *loss* sont dérivées. Enfin, les résultats d'une étude sur l'influence du nombre de neurones cachés et leur répartition sont présentés.

5.2.1 Équation et solution analytique

On considère l'équation de la chaleur pour $x \in [0, 1]$ avec conditions de Dirichlet :

$$\begin{cases} \dfrac{\partial u}{\partial t} - \alpha \dfrac{\partial^2 u}{\partial x^2} = 0, \\ u(0, x) = sin(2\pi x), \\ u(t, 0) = u(t, 1) = 0, \end{cases} \tag{5.2.1}$$

dont on peut calculer la solution exacte en employant la méthode de séparation des variables, c'est-à-dire que l'on cherche une solution de la forme $u(t,x) = f(t)g(x)$. En remplaçant cette expression dans l'équation, on trouve finalement :

$$f'(t)g(x) = \alpha f(t)g''(x) \Longleftrightarrow \frac{1}{\alpha}\frac{f'(t)}{f(t)} = \frac{g''(x)}{g(x)}.$$

Chaque membre de cette équation doit être constant, soit :

$$\begin{cases} \dfrac{f'(t)}{f(t)} = c_0\alpha, \\ \dfrac{g''(x)}{g(x)} = c_0, \end{cases} \Longleftrightarrow \begin{cases} f'(t) - c_0\alpha f(t) = 0, \\ \\ g''(x) - c_0 g(x) = 0. \end{cases}$$

En intégrant la première équation de ce système, on obtient $f(t) = c_1 exp(c_0\alpha t)$. Comme $\alpha > 0$, c'est le coefficient c_0 qui définit l'évolution de la température. Or, si $c_0 > 0$, alors la température croît exponentionnellement, ce qui n'est pas admissible. De plus, si $c_0 = 0$, alors la température resterait constante, donc il faut que $c_0 < 0$.

On pose maintenant $c_0 = -\omega^2$ avec ω un réel arbitraire, et on reprend l'équation sur g. Il en découle l'équation du second ordre homogène :

$$g''(x) + \omega^2 g(x) = 0.$$

On cherche alors une solution de la forme $g(x) = exp(mx)$, avec comme équation caractéristique $m^2 + \omega^2 = 0$, soit $m = \pm i\omega$, donc $g(x) = c_2 cos(\omega x) + c_3 sin(\omega x)$. La solution $u(t,x)$ s'écrit alors : $u(t,x) = exp(c_0\alpha t)[Acos(\omega x) + Bsin(\omega x)]$.

Or $u(t,0) = 0$, donc $A = 0$. De plus, $u(t,1) = 0$ donc $sin(\omega) = 0 \Longleftrightarrow \omega = k\pi$. Au final, $u(t,x) = Bexp(-\alpha(k\pi)^2 t)sin(k\pi x)$. En réalité, pour chaque valeur de k, on a une solution élémentaire $u_k(t,x) = b_k exp(-\alpha(k\pi)^2 t)sin(k\pi x)$.

D'après le principe de superposition, toute combinaison linéaire de solutions de cette forme est aussi solution de l'équation du système (5.2.1). Finalement, les solutions générales s'écrivent :

$$u(t,x) = \sum_{k=1}^{\infty} u_k(t,x).$$

Pour satisfaire la condition initiale, il faut alors que $\displaystyle\sum_{k=1}^{\infty} u_k(t,x) = u_0(x)$.

On multiplie par $sin(n\pi x)$ et on intègre sur $[0,1]$, et comme

$$\int_0^1 sin(k\pi x)sin(n\pi x)\,dx = \frac{1}{2}\delta_{n,k},$$

on trouve finalement,

$$b_k = 2\int_0^1 u_0(s)sin(k\pi s)\,ds.$$

Pour la condition initiale du système (5.2.1), on trouve alors :

$$b_k = 2\int_0^1 sin(2\pi s)sin(k\pi s)\,ds = \delta_{k,2} = 1$$

La solution exacte de ce système est donc :

$$u(t,x) = exp(-\alpha 4\pi^2 t)sin(2\pi x). \tag{5.2.2}$$

Pour simplifier le problème, on choisit $\alpha = \dfrac{1}{4\pi^2}$.

5.2.2 Mise en place des PINN

L'architecture d'un PINN est la même qu'un réseau dense à propagation avant (Figure 5.9), à cela près que la *loss* est définie à l'aide des équations que l'on veut résoudre. On sait que l'on va devoir dériver la prédiction du réseau par rapport aux variables t et x pour définir les équations dans la loss : il faut donc que ces deux variables soient en entrée du réseau, comme illustré Figure 5.9. Définissons maintenant la *loss* pour chacune des méthodes mentionnées au Chapitre 2, afin de résoudre le système (5.2.1). On verra ensuite ce que le calcul de ces dérivées partielles implique pour l'apprentissage.

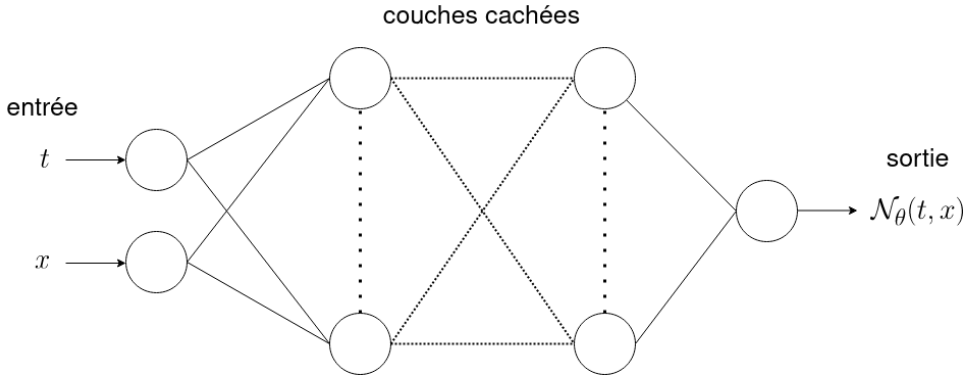


FIGURE 5.9 – Schéma d'un PINN avec un nombre de couches cachées et de neurones cachés quelconque, et prenant le point (t, x) comme entrée

5.2.2.1 Optimisation sans contrainte

Avec cette méthode, on définit une solution $\widehat{u}(t,x)$ construite à partir de la prédiction du réseau $\mathcal{N}_\theta(t,x)$ [86] :

$$\widehat{u}(t,x) = sin(2\pi x) + tsin(2\pi x)\mathcal{N}_\theta(t,x). \tag{5.2.3}$$

Cette fonction vérifie bien les conditions aux limites du système 5.2.1, c'est-à-dire $\widehat{u}(0,x) = sin(2\pi x)$ et $\widehat{u}(t,0) = \widehat{u}(t,1) = 0$.

Pour l'apprentissage du réseau, on définit maintenant la fonction perte :

$$loss(\mathcal{N}_\theta(t,x)) = \frac{1}{card(\Omega)} \sum_{(t,x)\in\Omega} \left| \frac{\partial \widehat{u}(t,x)}{\partial t} - \alpha \frac{\partial^2 \widehat{u}(t,x)}{\partial x^2} \right|^2, \tag{5.2.4}$$

où Ω est un ensemble de points d'apprentissage $(x,t) \in [0,1]^2$ et $|\Omega|$ désigne le nombre de points définis dans le domaine Ω. On remarque également qu'avec cette fonction coût, on n'a pas besoin de données cibles, et que seuls les couples d'entrées (t,x) forment la base de données d'apprentissage.

5.2.2.2 Optimisation avec contrainte

Avec cette approche, on pose :

$$\widehat{u}(x) = \mathcal{N}_\theta(t,x), \tag{5.2.5}$$

où $\mathcal{N}_\theta(t,x)$ est la prédiction du réseau [80, 81].

En prenant en compte les conditions aux limites du système (5.2.1), la fonction perte est alors définie par :

$$loss(\mathcal{N}_\theta(t,x)) = \frac{1}{card(\Omega)} \sum_{(t,x)\in\Omega} |\frac{\partial \widehat{u}(t,x)}{\partial t} - \alpha \frac{\partial^2 \widehat{u}(t,x)}{\partial x^2}|^2 \tag{5.2.6}$$
$$+ \frac{1}{card(\partial\Omega_t)} \sum_{(t,x)\in\partial\Omega_t} |\widehat{u}(t,x)) - sin(2\pi x)|^2 + \frac{1}{card(\partial\Omega_x)} \sum_{(t,x)\in\partial\Omega_x} |\widehat{u}(t,x))|^2,$$

où $\partial\Omega_t = \{0\} \times [0,1]$ et $\partial\Omega_x = [0,1] \times \{0,1\}$ sont respectivement les domaines des conditions initiales et des conditions aux bords. On remarque qu'avec cette fonction coût, on a besoin de deux types de données cibles : les points tels que $(t,x) \in \partial\Omega_t$, et ceux tels que $(t,x) \in \partial\Omega_x$.

5.2.2.3 Dérivation d'un réseau de neurone

Le calcul de ces deux *loss* implique non seulement la sortie du réseau de neurone, mais également les dérivées partielles par rapport aux deux dimensions d'entrée, à savoir t et x : dans leur article, Lagaris, Likas et Fotiadis ont démontré qu'un réseau était en effet dérivable par rapport à ses entrées [86].

Prenons un réseau avec une couche cachée de I neurones avec activation sigmoïd et 1 seul neurone en sortie avec activation linéaire (cette activation est nécessaire pour ne pas borner la valeur en sortie). Ce réseau prend en entrée une donnée $x = (x_1, \cdots, x_N)$. Notons w_{in} le poids reliant la dimension n de l'entrée au neurone i de la couche cachée, et v_i le neurone reliant le neurone i de la couche cachée au neurone de sortie. La sortie du neurone, notée $\mathcal{N}$ vaut alors :

$$\begin{cases} \mathcal{N} = \sum_{i=1}^{I} v_i \sigma_i, \\ \\ \sigma_i = \sigma(z_i) = \sigma(\sum_{n=1}^{N} w_{in} x_n + b_i). \end{cases}$$

Maintenant, calculons la dérivée d'ordre k par rapport à x_n de $\mathcal{N}$. Comme la fonction sigmoïd est de classe $\mathcal{C}^\infty$, on trouve que :

$$\begin{aligned} \frac{\partial^k \mathcal{N}}{\partial x_n^k} &= \sum_{i=1}^{I} v_i \left(\frac{\partial z_i}{\partial x_n}\right)^k \sigma_i^{(k)}, \\ &= \sum_{i=1}^{I} v_i w_{in}^k \sigma_i^{(k)} \ \forall k \in \mathbb{N}^* \backslash \{0\}, \ \forall n = 1, \cdots, N. \end{aligned}$$

De cette expression on peut maintenant en déduire la différentielle du réseau $d\mathcal{N}$. En effet :

$$\begin{aligned} \frac{\partial^{\lambda_n} \mathcal{N}}{\partial x_n^{\lambda_n}} = \sum_{i=1}^{I} v_i w_{in}^{\lambda_n} \sigma_i^{(\lambda_n)} &\iff \frac{\partial^{\lambda_{n-1}}}{\partial x_{n-1}^{\lambda_{n-1}}} \left(\frac{\partial^{\lambda_n} \mathcal{N}}{\partial x_n^{\lambda_n}}\right) = \sum_{i=1}^{I} v_i w_{in}^{\lambda_n} w_{i,n-1}^{\lambda_{n-1}} \sigma_i^{(\lambda_n + \lambda_{n-1})}, \\ &\iff \frac{\partial^{\lambda_1}}{\partial x_1^{\lambda_1}} \cdots \frac{\partial^{\lambda_n}}{\partial x_n^{\lambda_n}} \mathcal{N} = \sum_{i=1}^{I} v_i \left(\prod_{n=1}^{N} w_{in}^{\lambda_n}\right) \sigma_i^{(\sum_{n=1}^{N} \lambda_n)}, \end{aligned}$$

```
import tensorflow as tf

# Definition de l'equation de la chaleur :
# on derive la sortie du reseau par rapport aux entrees (t,x)
def heat_equation_1d(t, x, network):
    alpha = 1.0/(4.0*np.pi*np.pi)
    with tf.GradientTape() as tape1:
        tape1.watch(x)
        with tf.GradientTape() as tape2:
            tape2.watch([t,x])
            u = network(tf.concat([2*t-1,2*x-1], 1))
        u_t, u_x = tape2.gradient(u, [t, x])
    u_xx = tape1.gradient(u_x, x)
    return u_t - alpha*u_xx
```

donc

$$d\mathcal{N} = \sum_{i=1}^{I} v_i P_i \sigma_i^{(\Lambda)} dx, \tag{5.2.7}$$

où $P_i = \prod_{n=1}^{N} w_{in}^{\lambda_n}$ et $\Lambda = \sum_{n=1}^{N} \lambda_n$. En d'autres termes, la dérivée d'un réseau à 1 couche cachée n'est autre qu'un autre réseau à 1 couche cachée : les poids w_{ij} et biais b_i sont les mêmes pour les deux réseaux, mais les poids de la couche de sortie du réseau dérivé sont remplacés par $v_i P_i$ et l'activation des neurones de sa couche cachée est également remplacée par la dérivée d'ordre Λ de la sigmoïd. Comme la dérivée du réseau par rapport à ses entrées est un autre réseau, on peut facilement calculer la dérivée du deuxième réseau par rapport aux paramètres du premier réseau et utiliser un algorithme de descente de gradient pour régler les poids.

La dérivation se fait aisément avec la librairie Tensorflow [96], comme le montre le code ci-dessus. Dans cet exemple, l'outil **GradientTape**, basé sur la méthode de dérivation automatique [87], se charge de dériver les lignes de calcul qui vont suivre par rapport aux variables demandées grâce à la commande **tape.watch**. On remarque qu'inclure une boucle de dérivation dans une autre permet de calculer une dérivée seconde.

5.2.3 Résultats

On prend le réseau décrit Figure 5.9, et pour différentes combinaisons du nombre de couches et du nombre de neurones par couche, on évalue la performance des *loss* (5.2.4) et (5.2.6). Pour tous ces entraînements, on utilise l'optimiseur Adam sur un *batch* de taille *ntrain* avec un taux d'apprentissage de 0.001 pendant 1000 *epochs*. Les paramètres sont initialisés à l'aide de la méthode Xavier mentionnée précédemment, et on choisit l'activation tangente hyperbolique sur toutes les couches cachées.

Pour ce qui est de la base de données, $N_\Omega = 10\ 000$ points $(t, x) \in \Omega = [0, 1]^2$ ont été générés, ainsi que $N_{\partial\Omega} = 200$ points de collocation sur les frontières du domaine pour les conditions limites, soit 10 200 points en tout. On a fait le choix ici de ne pas définir d'ensemble test mais de plutôt utiliser la méthode de validation croisée introduite au Chapitre 1. On rappelle qu'avec cette méthode, les données sont séparées en k *folds* (i.e. en sous-ensembles) et pour chaque architecture on lance k apprentissages avec un *fold* différent utilisé comme ensemble de validation (le reste des *folds* constitue l'ensemble

Couches \ Neurones	4	8	16	32	64
2	1.27e-04	2.21e-05	1.27e-05	7.23e-06	8.59e-06
4	3.13e-05	6.01e-06	1.71e-06	2.12e-06	2.41e-06
6	1.54e-05	5.80e-06	1.25e-06	6.38e-07	1.86e-06
8	1.28e-05	3.44e-06	2.49e-07	5.37e-07	3.76e-07
10	5.45e-06	2.02e-06	2.78e-07	5.69e-07	4.93e-07

(a) Optimisation sans contrainte

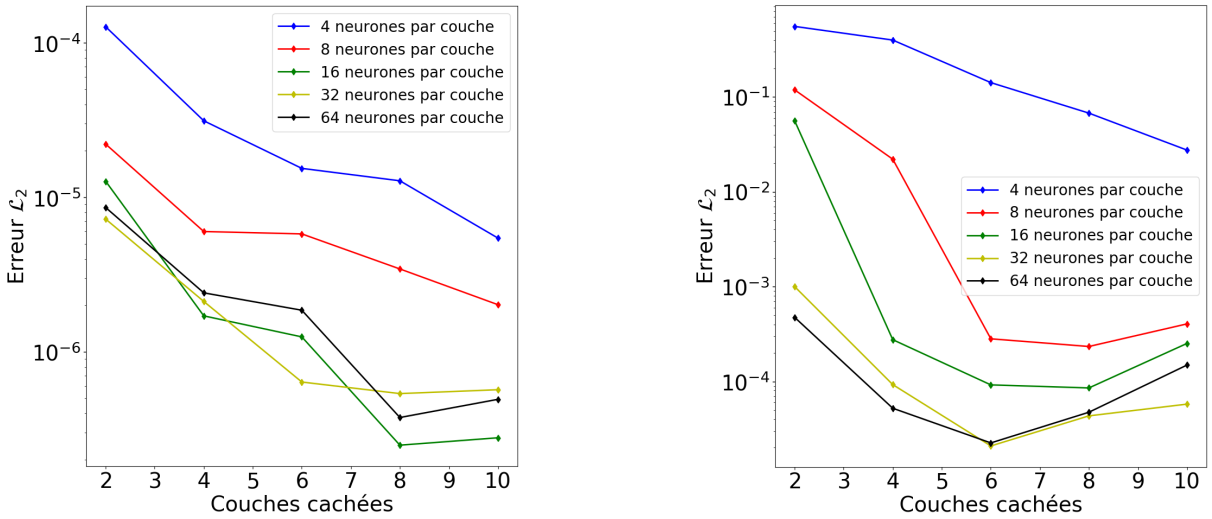
Couches \ Neurones	4	8	16	32	64
2	5.56e-01	1.19e-01	5.60e-02	1.00e-03	4.74e-04
4	3.99e-01	2.21e-02	2.76e-04	9.27e-05	5.22e-05
6	1.42e-01	2.82e-04	9.21e-05	2.09e-05	2.25e-05
8	6.77e-02	2.34e-04	8.55e-05	4.34e-05	4.74e-05
10	2.76e-02	4.05e-04	2.52e-04	5.77e-05	1.49e-04

(b) Optimisation avec contrainte

Tableau 5.1 – Erreur L^2 sur l'ensemble de validation à la fin de l'entraînement

d'entraînement). Les points ont donc été répartis en 5 *folds*, en prenant soin de répartir les 200 points supplémentaires sur les frontières de manière équitable dans tous les *folds*. Le tableau 5.1 et la Figure 5.10 présentent une synthèse de ces résultats, avec l'erreur L^2 moyenne sur l'ensemble de validation à la fin de l'entraînement, pour un nombre variable de couches cachées et de neurones par couche.

Tout d'abord, on constate qu'à première vue, les résultats sont meilleurs avec l'optimisation sans contrainte. En effet, la plus faible erreur L^2 atteinte avec cette méthode est de 2.49e-07, alors qu'elle est de 2.09e-05 pour l'optimisation avec contrainte. Il faut noter cependant qu'une partie des hyperparamètres a été fixée : ce choix permet de mieux voir l'influence seule du nombre de neurones cachés et leur répartition dans les couches cachées, mais il faudrait en réalité régler les hyperparamètres pour chaque architecture.



(a) Optimisation sans contrainte

(b) Optimisation avec contrainte

FIGURE 5.10 – Erreur L^2 en fonction du nombre de couches cachées pour différentes valeurs de neurones par couche

Analysons d'abord le Tableau 5.1a, dont les résultats sont également mis en valeur sur la Figure 5.10a. La première observation que l'on peut faire est que l'augmentation du nombre de neurones cachés améliore les performances, jusqu'à un certain seuil. En effet, les courbes de la Figure 5.10a ont toutes une tendance à décroitre lorsque l'on augmente le nombre de couches cachées. Cependant on remarque deux choses : d'abord, avec 10 couches à 16, 32 ou 64 neurones, la performance s'est légèrement détériorée (elle reste tout de même meilleure que 10 couches à 4 ou 8 neurones) ; ensuite, que globalement, mettre 64 neurones par couche cachée ne produit pas les meilleurs résultats. En somme, ces résultats nous montrent qu'augmenter le nombre de neurones cachés, et en conséquence le nombre de paramètres, a des limites. On parle également d'expressibilité du réseau : plus un réseau a de paramètres plus il est capable de gérer des données complexes. Mais si les données ne sont pas aussi complexes que l'on croit, alors augmenter l'expressibilité du réseau peut au contraire provoquer du surapprentissage. Dans notre cas, les données sont clairement simples, puisque la meilleure performance est obtenue avec un réseau de 8 couches cachées à 16 neurones (soit 1 969 paramètres).

La même analyse est valable pour les résultats de l'optimisation avec contrainte (Tableau 5.1b et Figure 5.10b), où le réseau de 6 couches cachées à 32 neurones (soit 5 409 paramètres) est celui qui a la meilleure performance.

5.3 Réseaux denses pour le calcul du flux non local de Luciani-Mora-Virmont

Avant de coupler un réseau de neurone à un code hydrodynamique pour le calcul du flux non local, on a fait une étude de faisabilité avec un modèle plus simple que le modèle Schurtz-Nicolaï-Busquet : le modèle de Luciani-Mora-Virmont, présenté brièvement au Chapitre 3. Cette étude s'est déroulée en deux temps : d'abord la génération d'une base de données, puis l'entraînement de réseaux. Le but étant uniquement d'évaluer si un réseau pouvait calculer un flux non local, il n'y a pas eu de couplage avec un code hydrodynamique. La démarche et les résultats obtenus sont présentés ci-dessous.

5.3.1 Génération de la base de données

On veut générer une base de données où la cible est le flux non local de Luciani-Mora-Virmont [12] en tout point d'un espace 1d discrétisé. On choisit d'entraîner des réseaux denses, ce qui implique que le nombre de points du maillage devra rester fixe pour toutes les données générées.

Prenons un domaine $[a, b]$ et définissons $dx = (b-a)/n$ le pas de discrétisation de cet espace 1d, de telle sorte que $x_i = a + (i-1) * dx$ pour $1 \leq i \leq n+1$. On notera par la suite f_i la fonction f approchée au point x_i.

D'après la définition du flux non local de Luciani-Mora-Virmont donnée par l'équation (3.2.3) au Chapitre 3, pour chaque point de maillage, il nous faut calculer le flux local de Spitzer-Härm [121], ainsi qu'un noyau de convolution.

On calcule en premier lieu le flux de Spitzer-Härm, définit par l'équation (3.2.1). Pour ce faire, on approche le gradient de la température avec une différence finie, et pour $1 \leq i \leq n$, on écrit alors :

$$q_{SH,i} = -\kappa_{SH,i} \frac{T_{i+1} - T_i}{dx}, \tag{5.3.1}$$

avec

$$\kappa_{SH,i} = 8\left(\frac{2}{\pi}\right)^{3/2} \frac{1}{\sqrt{n_e} Z e^4 ln\Lambda_{ei}} \frac{Z}{Z + 0.2lnZ + 3.44} \left(\frac{T_i + T_{i+1}}{2}\right)^{5/2}, \tag{5.3.2}$$

où n_e est la densité, Z le nombre ionique, e la charge électronique, et $ln\Lambda_{ei}$ le logarithme coulombien électron-ion.

On calcule maintenant le noyau de convolution $\omega(x, x')$ définit par l'équation (3.2.4). On note $\omega_{ij} = \omega(x_i, x_j)$. En supposant que la densité est uniforme, on trouve finalement :

$$\omega_{ij} = \frac{1}{2\lambda_j} exp(-\left|\frac{x_i - x_j}{\lambda_j}\right|), \tag{5.3.3}$$

avec

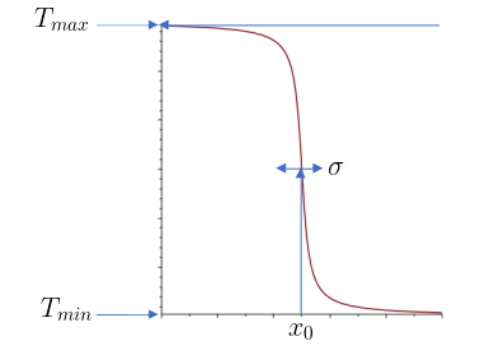
$$\lambda_j = 32\lambda_{0,j} = \frac{(kT_j)^2}{4\pi n_e Z e^4 ln\Lambda_{ei}}, \tag{5.3.4}$$

où λ est le parcours effectif, λ_0 le libre parcours moyen, et k la constante de Boltzmann.

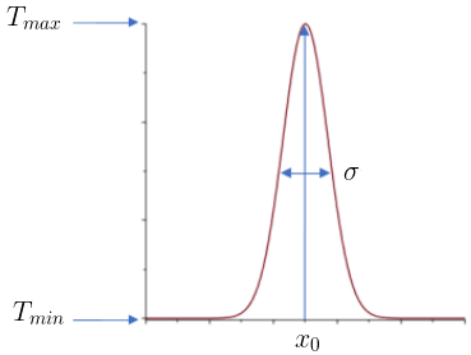
On peut maintenant évaluer le flux non local de Luciani-Mora-Virmont en utilisant une somme de Riemann pour approcher l'intégrale. Pour $1 \leq i \leq n$, on obtient ainsi :

$$q_{LMV,i} = \sum_{j=1}^{n} \omega_{ij}\, q_{SH,j}\, dx. \tag{5.3.5}$$

Pour générer les profils de flux non locaux, on génère d'abord des profils de température (Figure 5.11) : des profils en forme de marche (Figure 5.11a) et des profils en forme de gaussienne (Figure 5.11b). Pour ces deux types, on fait varier σ, x_0, T_{min} et T_{max} de telle manière que lorsque l'on calcule le ratio $max(\frac{q_{LMV}}{q_{SH}})$, les données soient quasi uniformément réparties entre 0 (i.e. des profils faiblement locaux) et 1 (i.e. des profils fortement locaux), comme illustré Figure 5.12. En tout, 20 988 profils ont été générés, dont 10 505 avec des profils de température type marche et 10 483 avec des profils de température type gaussienne. En somme, un profil complet de température et le flux non local correspondant constituent un couple entrée-sortie.

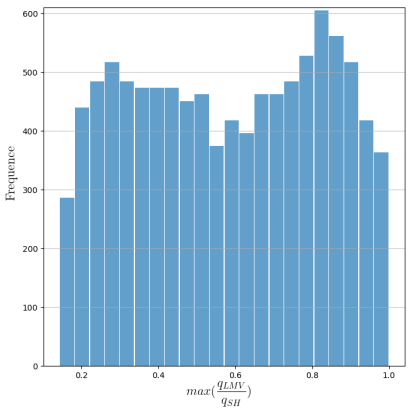


(a) Profils type marche

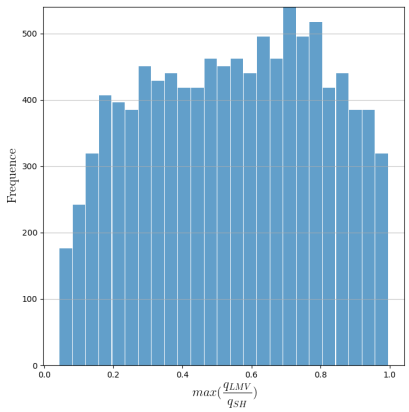


(b) Profils type gaussienne

FIGURE 5.11 – Types de profils de température utilisés pour générer les flux non locaux de Luciani-Mora-Virmont



(a) Profils type marche



(b) Profils type gaussienne

FIGURE 5.12 – Histogrammes de répartition des données générées.

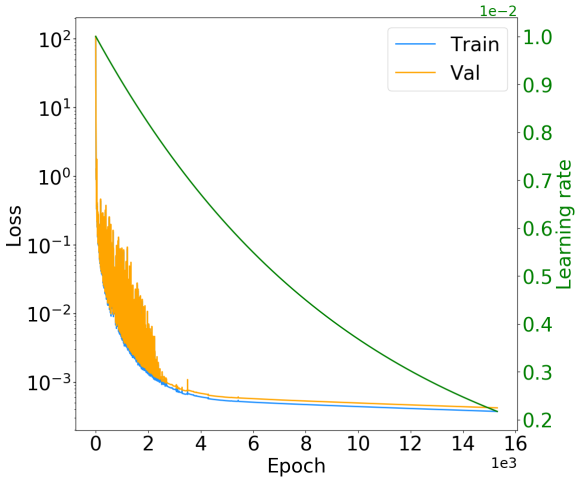
5.3.2 Entraînements de réseaux de neurones artificiels

Afin d'évaluer la performance des réseaux, on utilise maintenant un nouveau marqueur de performance : pour chaque donnée, on calcule l'erreur relative entre prédiction et cible en chaque point, et si l'erreur maximum est inférieure à $p\%$, alors on considère que le réseau a correctement prédit la sortie. Cette fonction permet d'une part de définir un critère de précision sur la sortie du réseau, et d'autre part de se rendre compte de la part de prédictions qui respectent ce critère.

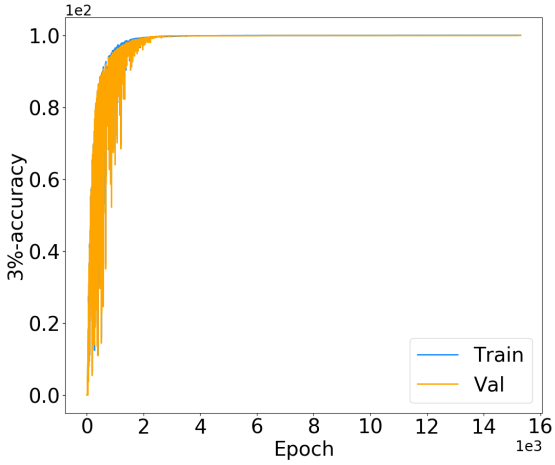
Dans un premier temps, des réseaux ont été entrainés sur chaque sous-base de données séparément, puis enfin sur la base de données complète. À chaque fois, de nombreux entraînements ont été menés, en faisant varier les hyperparamètres : seuls les meilleurs sont présentés ici.

5.3.2.1 Sur des données de type marche

Pour cette première partie de la base de données, les meilleurs résultats ont été obtenus avec un réseau de 2 couches cachées à 300 neurones avec activation sigmoïd. On a utilisé une *loss* des moindres carrés et l'optimiseur Adam avec un taux d'apprentissage décrois-

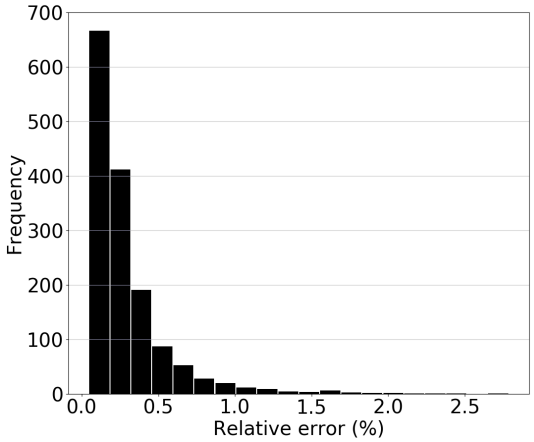


(a) Évolution de la *loss*

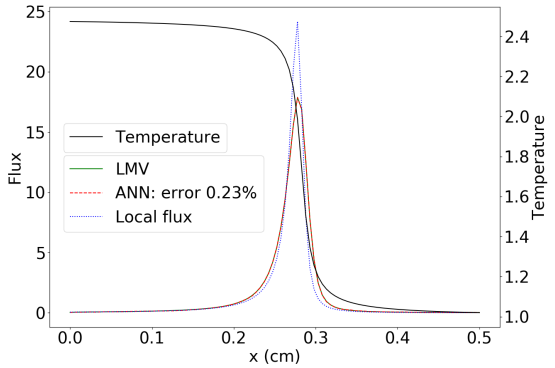


(b) Évolution du taux de réussite à 3%

FIGURE 5.13 – Résultats de l'entraînement



(a) Histogramme des erreurs relatives



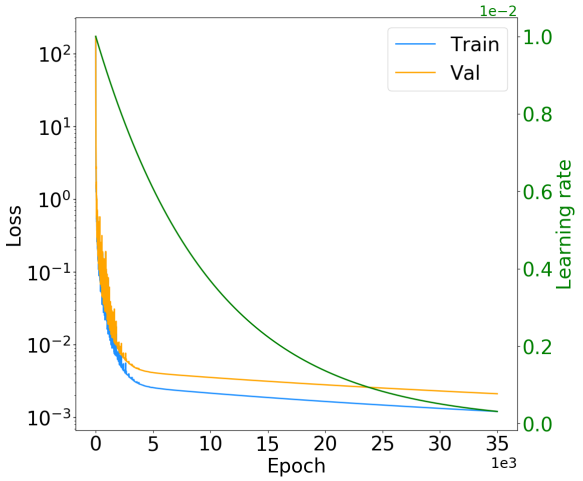
(b) Exemple de profil

FIGURE 5.14 – Performances sur l'ensemble test

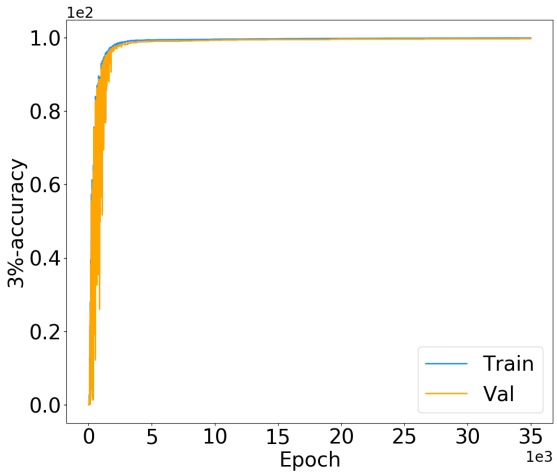
sant (*time decay*) sur un *batch* de taille 500. La Figure 5.13 présente les performances de l'apprentissage : les courbes ont les mêmes tendances sur les ensembles d'entraînement et de validation, où la *loss* diminue jusqu'à atteindre un plateau (Figure 5.13a), et le taux de réussite à 3% augmente jusqu'à atteindre 100% (Figure 5.13b). Les performances sont similaires sur l'ensemble test (Figure 5.14) : la Figure 5.14a montre la répartition des données test selon l'erreur commise par le RNA et on constate qu'aucune donnée ne présente une erreur supérieure à 3%, l'erreur maximum étant de 2.79% ; cette tendance est de plus confirmée par l'exemple de profil présenté Figure 5.14b où l'une erreur relative est de 0.23%.

5.3.2.2 Sur des données de type gaussien

Pour la deuxième partie de la base de données, les meilleurs résultats ont été obtenus avec un réseau de 3 couches cachées à 200 neurones avec activation ReLU. On a utilisé une *loss* des moindres carrés et l'optimiseur Adam avec un taux d'apprentissage décroissant (*time decay*) sur un *batch* de taille 500. Les résultats de l'entraînement sont présentés Figure 5.15 : la *loss* diminue similairement sur les ensembles d'entraînement et de valida-

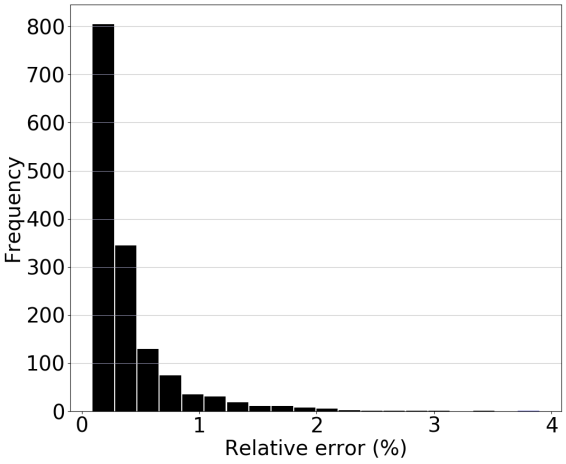


(a) Évolution de la *loss*

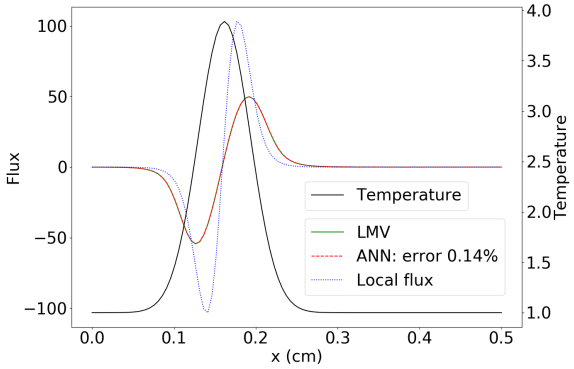


(b) Évolution du taux de réussite à 3%

FIGURE 5.15 – Résultats de l'entraînement



(a) Histogramme des erreurs relatives



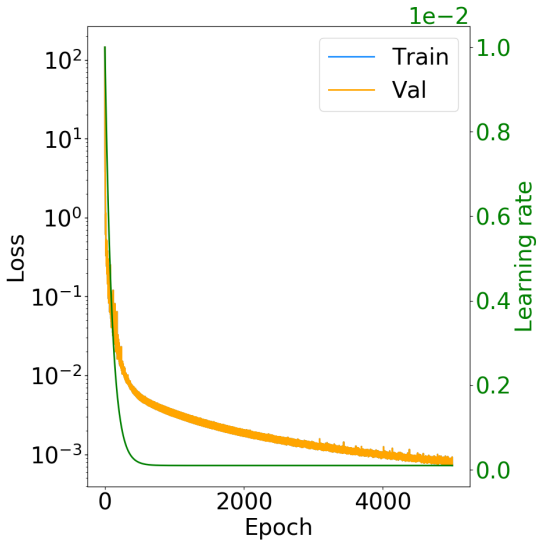
(b) Exemple de profil

FIGURE 5.16 – Performances sur l'ensemble test

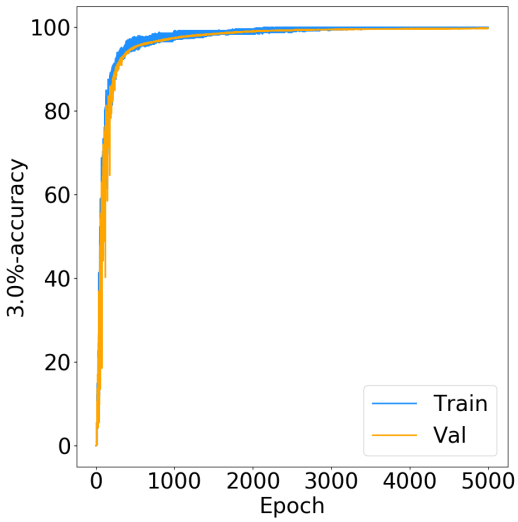
tion (Figure 5.15a), et le taux de réussite à 3% augmente jusqu'à atteindre 100% (Figure 5.15b). Les performances sont encore une fois similaires sur l'ensemble test (Figure 5.16) : Figure 5.16a montre que le réseau est à 99.8% de réussite à 3% avec une erreur maximum de 3.9% ; un exemple de prédiction est présenté Figure 5.16b.

5.3.2.3 Sur des données de type marche et gaussien

Enfin, tous les profils sont rassemblés puis répartis dans les ensembles d'entraînement (14 000 données), de validation (4 000 données) et de test (2 988 données), en prenant soin que chaque ensemble présente la même proportion de données de type marche et de type gaussien. Le meilleur réseau trouvé est un réseau de 2 couches cachées à 300 neurones avec activation sigmoïd. Pour son entraînement, on a utilisé une *loss* des moindres carrés et l'optimiseur Adam avec un taux d'apprentissage variable (*time decay*) et un *batch* de taille 500. L'évolution de la *loss* et du taux de réussite à 3% sont présentés Figure 5.17 : la *loss* diminue avec la même tendance sur les ensembles d'entraînement et de validation (Figure 5.17a), tandis que le taux de réussite augmente jusqu'à atteindre 100% sur les deux

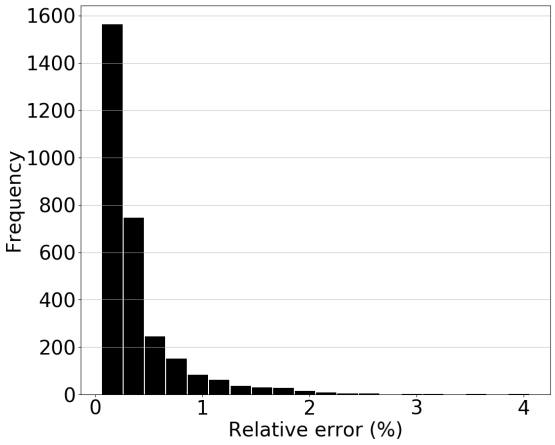


(a) Évolution de la *loss*

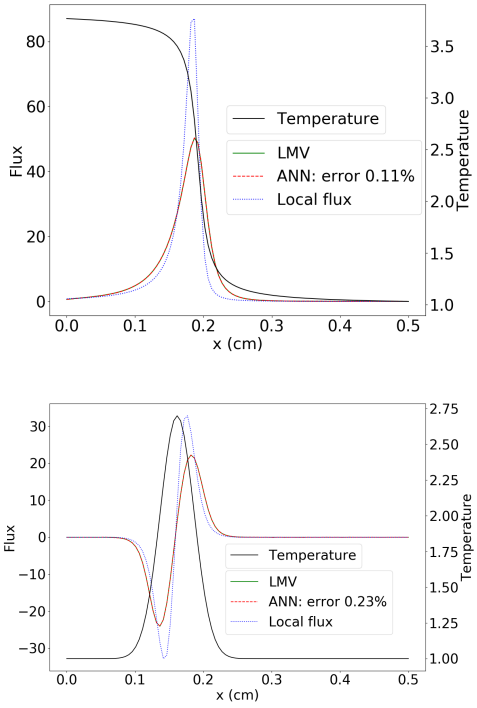


(b) Évolution du taux de réussite à 3%

FIGURE 5.17 – Résultats de l'entraînement



(a) Histogramme des erreurs relatives



(b) Exemples de profils

FIGURE 5.18 – Performances sur l'ensemble test

ensembles également (Figure 5.17b). Les performances sur l'ensemble test sont présentées Figure 5.18 : l'histogramme des erreurs relatives montre que le réseau atteint 99.93% de réussite à 3%, avec une erreur maximum de 3.53% (Figure 5.18a) ; cette performance est de plus confirmée lorsque l'on compare la prédiction du réseau à la cible (Figure 5.18b).

5.4 Conclusion et perspectives

Dans ce chapitre, nous avons exploré l'utilisation de RNA pour la résolution de différents types de problèmes : la classification, la résolution d'équations différentielles, et la régression linéaire.

Nous avons d'abord abordé le problème de la classification, en se limitant à la classification binaire avec une base de données où les coordonnées de points dans l'espace 2D forment les données en entrée, et leur couleur (bleu ou rouge) constitue la cible en sortie. On définit ainsi un problème où l'on peut explicitement séparer les données par ce que l'on appelle une frontière de décision. Nous avons pu voir que si cette frontière est linéaire, alors un simple neurone suffit à résoudre le problème. Par contre, dès lors que cette frontière est non linéaire, il faut utiliser un réseau avec au moins 1 couche cachée. Par la même occasion, nous avons pu attester des performances de l'algorithme de descente de gradient, qui nécessite l'ajout d'un moment lorsqu'on considère des frontières de décision non linéaires.

Ensuite, nous avons étudié un type de réseaux spécifique à la résolution d'équations différentielles : les *Physics Informed Neural Networks* ou PINN. L'apprentissage de ces réseaux ne se fait pas avec les fonctions perte classique : ce sont plutôt les équations à résoudre qui définissent la *loss*. En effet, cela est possible car la prédiction d'un réseau est dérivable par rapport à son entrée. Nous avons vu deux manières de définir cette *loss* différentielle : soit en contraignant la prédiction pour qu'elle vérifie les conditions limites (on parle alors d'optimisation sous contraintes), soit en construisant une solution qui vérifie les conditions aux limites à partir de la prédiction du réseau (on parle alors d'optimisation sans contrainte). On s'est intéressé au cas de l'équation de la chaleur 1D, et on a étudié le comportement de deux types d'optimisation. Les résultats ont montré que les deux méthodes étaient capables de résoudre l'équation, avec une erreur L^2 sur l'ensemble test de l'ordre de 10^{-6} sans contrainte, et 10^{-4} avec contrainte.

Enfin, nous avons réalisé une étude qui a pour vocation de servir de preuve de principe pour le travail présenté dans la quatrième partie de cette thèse : nous avons entraîné des réseaux denses pour qu'ils modélisent le flux non local de Luciani-Mora-Virmont. Les résultats ont montré qu'un réseau dense était capable d'approché un profil de flux non local avec une erreur inférieur à 3% en tout point du maillage. Ces résultats nous confirment ainsi qu'un réseau est capable de modéliser ce phénomène physique complexe, laissant ainsi la possibilité de considérer un autre modèle de flux non local, mais aussi de coupler un RNA à un code hydrodynamique, comme on le verra par la suite au Chapitre 7.

Ces premières utilisations des RNA ont cependant soulevé la question qu'est le lien théorique entre fonction d'activation et fonction perte. En effet, durant toutes les études menées ici, les choix de fonction d'activation et de fonction perte étaient basés sur des lois empiriques, validées par la communauté du *deep learning*. Une réflexion sur le lien théorique entre la fonction perte entropie croisée et la fonction d'activation sigmoïd est présentée au chapitre suivant.

Chapitre 6

Recherche d'un lien entre fonction d'activation et fonction perte

L'entraînement d'un réseau de neurones artificiels repose sur le choix d'hyperparamètres pertinents. Ces derniers sont nombreux : répartition des neurones dans les différentes couches, choix d'une ou plusieurs fonctions d'activation à travers les couches, initialisation des poids, valeur du taux d'apprentissage, nombre d'itérations, fonction perte, optimiseur, etc. Un inconvénient non négligeable des réseaux de neurones est le fait qu'il n'existe pas de théorie établie sur le choix de ces hyperparamètres. Néanmoins, il existe des règles empiriques.

Le choix des fonctions d'activation et de perte repose sur de telles lois : en pratique, on associe une fonction perte à une activation sur la couche de sortie en fonction du type de problème considéré et du type de sortie que l'on souhaite obtenir. Par exemple, si la sortie est une valeur réelle, comme la valeur d'une fonction, il faudra utiliser une fonction d'activation non bornée telle que l'activation linéaire, et la fonction de coût appropriée pour ce type de sortie est la fonction des moindres carrés. Ou encore, si la sortie est une probabilité, il est recommandé d'utiliser une activation sigmoïde couplée à l'entropie croisée comme fonction perte.

Le travail présenté ici vise à établir un lien théorique entre la fonction d'activation sigmoïde et la fonction de coût d'entropie croisée, et repose sur la formulation de l'apprentissage d'un neurone simple en problème d'optimisation. La première partie de ce chapitre présente des définitions et théorèmes d'optimisation, tirés d'un cours de Laurent Guillopé [135]. La deuxième partie présente les résultats, appuyés par les tests numériques explorés dans la troisième et dernière partie.

6.1 Optimisation sous contrainte

6.1.1 Définitions générales

Un problème d'optimisation consiste en la recherche des valeurs de minimum (ou de maximum) d'une fonction J appelée fonction d'objectif ou fonction de coût, soit

$$\min_{x \in U} J(x). \tag{6.1.1}$$

À noter que la recherche du maximum de J revient à la recherche du minimum de la fonction oposée $-J$.

Définition 6.1.1 *Soit U un ouvert de $\mathbb{R}^n$ et $J : U \in \mathbb{R}^n \longrightarrow \mathbb{R}$ une fonction d'objectif.*
(i) Le point x_0 est un minimum strict de J si $J(x_0) < J(x)$ pour $x \in U \backslash x_0$.
(ii) Le point x_0 est un minimum global de J si $J(x_0) \leq J(x)$ pour $x \in U$.
(iii) Le point x_0 est un minimum local (resp. local strict) de J s'il existe un voisinage V de x_0 tel que $J(x_0) \leq J(x)$ pour $x \in V$ (resp. $J(x_0) < J(x)$ pour $x \in V \backslash x_0$).

Définition 6.1.2 *Soient U un ouvert de $\mathbb{R}^n$, $J : U \in \mathbb{R}^n \longrightarrow \mathbb{R}$ une fonction d'objectif. Soient $f_i : U \longrightarrow \mathbb{R}$ pour $0 \leq i \leq m$ et $g_j : U \longrightarrow \mathbb{R}$ pour $0 \leq j \leq p$. Le problème d'optimisation avec les contraintes d'égalité $f_i(x) = 0$ pour $0 \leq i \leq m$ et les contraintes d'inégalité $g_j(x) \geq 0$ pour $0 \leq j \leq p$ est la recherche de minimum*

$$min\ \{J(x) : x \in U, f_i(x) = 0\ \forall 0 \leq i \leq m, g_j(x) \geq 0\ \forall 0 \leq j \leq p\}. \tag{6.1.2}$$

Définition 6.1.3 *Soit U un ouvert de $\mathbb{R}^n$ et $J : U \in \mathbb{R}^n \longrightarrow \mathbb{R}$. La différentielle de J au point $x \in U$ est l'application linéaire*

$$f'(h) = f'(h_1, \cdots, h_n) = \partial_{x_1} f(x) h_1 + \cdots + \partial_{x_n} f(x) h_n = \langle \nabla f(x), h \rangle, \tag{6.1.3}$$

et telle que

$$f(x + h) = f(x) + f'(h) + \|h\| \epsilon(h). \tag{6.1.4}$$

6.1.2 Optimisation avec contraintes d'égalité

Soit U un ouvert de $\mathbb{R}^n$. On s'intéresse ici aux problèmes d'optimisations

$$\mathcal{P}_{U,f} : \min_{\substack{x \in U \subset \mathbb{R}^n \\ f_i(x) = 0\ i=1,\ldots,m}} J(x). \tag{6.1.5}$$

Définition 6.1.4 *Soient U un ouvert de $\mathbb{R}^n$, $J : U \subset \mathbb{R}^n \longrightarrow \mathbb{R}$ une fonction d'objectif et $f : U \longrightarrow \mathbb{R}^m$ des fonctions de classe $\mathcal{C}^1$, déterminant le problème d'optimisation avec contrainte d'égalité $\mathcal{P}_{U,f}$. Si $m = 1$, le lagrangien $\mathcal{L}_{J,f}$ associé est la fonction définie sur $U \times \mathbb{R}$ par*

$$(x, \lambda) \in U \times \mathbb{R} \longmapsto \mathcal{L}_{J,f}(x, \lambda) = J(x) - \lambda g(x).$$

De manière plus générale, si $m > 1$, le lagrangien est définit par

$$(x, \Lambda = (\lambda_1, ..., \lambda_m) \in U \times \mathbb{R}^m \longmapsto \mathcal{L}_{J,f}(x, \Lambda) = J(x) - \sum_{k=1}^{m} \lambda_k g_k(x) = J(x) - \langle \Lambda, g(x) \rangle.$$

Les coefficients $\lambda, \Lambda, \lambda_1, ..., \lambda_m$ sont appelés multiplicateurs de Lagrange.

Théorème 6.1.5 *Soient U un ouvert de $\mathbb{R}^n$, $J : U \in \mathbb{R}^n \longrightarrow \mathbb{R}$ une fonction d'objectif et $f : U \longrightarrow \mathbb{R}^m$ des fonctions de classe $\mathcal{C}^1$, $\mathcal{P}_{J,f}$ le problème défini par $\min_{f(x)=0} J(x)$ et son lagrangien $\mathcal{L}_{J,f} : (x, \lambda) \in U \times \mathbb{R}^m \longmapsto J(x) - \langle \Lambda, g(x) \rangle$. Soit x_* un minimum du problème $\mathcal{P}_{J,f}$.*
Si la différentielle $g'(x_*)$ est surjective, alors il existe un $\Lambda_* \in \mathbb{R}^m$ tel que (x_*, Λ_*) soit un point critique du lagrangien $\mathcal{L}_{J,f}$, i.e.

$$\partial_{x_j} J(x_*) = \langle \Lambda_*, \partial_{x_j} g(x_*) \rangle,\ j = 1, ..., n,\ g(x_*) = 0,$$

soit

$$\nabla J(x_*) = \langle \Lambda_*, \nabla g(x_*) \rangle,\ g(x_*) = 0. \tag{6.1.6}$$

Le multiplicateur Λ_* est unique.

6.1.3 Optimisation avec contraintes d'inégalité

Soit U un ouvert de $\mathbb{R}^n$. On s'intéresse ici aux problèmes d'optimisations

$$\mathcal{P}_{U,g} : \min_{\substack{x\in U\subset\mathbb{R}^n \\ g_i(x)\geq 0 \ \ i=0,\dots,p}} J(x). \tag{6.1.7}$$

Définition 6.1.6 *Soient U un ouvert de $\mathbb{R}^n$ et les fonctions $g_j : U \longrightarrow \mathbb{R}$ avec $0 \leq j \leq p$.*

(i) La contrainte $g_j(x) \geq 0$ est dite saturée en $x_* \in U$, et le point x_* actif relativement à la contrainte g_j , si $g_j(x_*) = 0$. On note S_{x_*} l'ensemble des indices $j = 1, \cdots, p$ tels que la contrainte $g_j(x) \geq 0$ soit saturée en x_* . Ainsi, si $j \notin S_{x_*}$, alors $g_j(x_*) > 0$.

(ii) Les contraintes $g_j(x) \geq 0, j = 1, \cdots, p$ sont dites régulières en x_*, et le x_* un point régulier, si la matrice jacobienne de $g_{S_{x_*}} = (g_j)_{j\in S_{x_*}}$ est de rang $\#S_{x_*}$ i. e. si les gradients $\nabla g_j(x_*), j \in S_{x_*}$ sont linéairement indépendants

Théorème 6.1.7 (Karush-Kuhn-Tucker) *Soit une partie C de U un ouvert de $\mathbb{R}^n$ et les fonctions J, $(g_j)_{j=1,\dots,p}$ définies sur C et différentiables, et définissant le problème d'optimisation avec contraintes d'inégalité*

$$\mathcal{P}_{C,g} : \min_{\substack{x\in C\subset\mathbb{R}^n \\ g_i(x)\geq 0 \ \ i=0,\dots,p}} J(x). \tag{6.1.8}$$

Soit x_* un point régulier pour les contraintes $g_j(x) \geq 0, j \in S_{x_*}$ saturées en x_*. Si x_* est un minimum du problème avec contraintes d'inégalité $\mathcal{P}_{C,g}$, alors il existe un unique $\Lambda_* = (\lambda_{*1}, \cdots, \lambda_{*p})$ tel que

(i) le lagrangien $\mathcal{L}(x, \lambda_1, \cdots, \lambda_p) = J(x) - \sum_{j=1}^{p} \lambda_j g_j(x)$ est critique en les variables x au point $(x_*, \Lambda_* = (\lambda_{*j})_{j=1,\dots,p})$, soit

$$\nabla_x \mathcal{L}(x_*, \Lambda_*) = \nabla_x J(x_*) - \sum_{j=1}^{p} \lambda_{j*} \nabla_x g_j(x_*) = 0; \tag{6.1.9}$$

(ii) $\lambda_{*j} = 0$ si x_* n'est pas actif pour la contrainte g_j ;

(iii) $\lambda_{*j} \geq 0$ si x_* est actif pour la contrainte g_j.

6.1.4 Optimisation avec contraintes d'égalité et d'inégalité

Théorème 6.1.8 *Soit J une fonction objectif définie sur U un ouvert de $\mathbb{R}^n$ et différentiable, avec les contraintes en égalités $f_i = 0$, $i = 1, \cdots, m$ et en inégalités $g_j \geq 0$, $j = 1, \cdots, p$ et $x_k \geq 0$, $k = 1, \cdots, p$. Soit x_* un minimum du problème*

$$\mathcal{P}_{C,f,g} : \min_{\substack{x\in C\subset\mathbb{R}^n \\ f_i(x)=0 \ \ i=1,\dots,m \\ g_i(x)\geq 0 \ \ i=0,\dots,p}} J(x). \tag{6.1.10}$$

Définissions S_{x_*} l'ensemble des indices de contraintes $g_j \geq 0$ saturées en x_* et $\mathcal{L}_{x_*}$ le lagrangien défini par

$$\mathcal{L}_{x_*}(x, \Lambda, M) = J(x) - \sum_{i=1}^{m} \lambda_i f_i(x) - \sum_{j\in S_{x_*}} \mu_j g_j(x). \tag{6.1.11}$$

On suppose le minimum x_* régulier pour les contraintes, i. e. la différentielle de $x \mapsto ((f_i)_i, (g_j)_{j\in S_{x_*}})$ est surjective en x_*. Il existe alors des multiplicateurs $(\Lambda_* = (\lambda_{*i})_{i=1,\cdots,m})$ et $M_* = (\mu_{*j})_{j\in S_{x_*}})$ tels que

$$\partial_x \mathcal{L}_{x_*}(x_*, \Lambda_*, M_*) = 0,\ \mu_j \geq 0,\ j \in S_{x_*}. \tag{6.1.12}$$

6.2 Formulation de l'apprentissage d'un neurone simple en problème d'optimisation sous contrainte

On considère un neurone formel à n entrées, notées $(x_i)_{1\leq i\leq n}$, dont la tâche est une classification binaire. Nous avons vu que l'apprentissage de ce neurone revient à minimiser la fonction perte, qui mesure l'écart entre la prédiction $\widehat{y}$ et la cible y, sur un ensemble de données d'apprentissage à l'aide d'une descente de gradient stochastique : la minimisation se fait sur l'ensemble des paramètres (i.e. poids et biais). Ici, on s'intéresse plutôt à une minimisation dépendant de la prédiction et de la cible. Pour cela, on écrit cette minimisation sous forme de problème d'optimisation avec contraintes d'égalités et d'inégalités, en s'appuyant sur le théorème 6.1.8.

En effet, on peut écrire les contraintes suivantes :

- on sait que pour une classification binaire, la cible vaut toujours 0 ou 1 : on peut donc définir la contrainte d'égalité $y(1-y) = 0$;
- durant l'apprentissage, on va également imposer que la prédiction soit comprise entre 0 et 1, donc on peut définir les contraintes d'ingégalité $\widehat{y} \geq 0$ et $1 - \widehat{y} \geq 0$.

Le problème de minimisation avec contraintes d'égalité et d'inégalité de l'apprentissage d'un neurone simple pour une classification binaire peut alors s'écrire :

$$\mathcal{P}_{C,g,h} : \quad \min_{\substack{y,\widehat{y} \\ g:y(1-y)=0 \\ h_1:\widehat{y}\geq 0 \\ h_2:1-\widehat{y}\geq 0}} \quad loss(y, \widehat{y}), \tag{6.2.1}$$

où $loss(y, \widehat{y}) = -ylog(\widehat{y}) - (1-y)log(1-\widehat{y})$ est la fonction perte de *cross-entropy*. La *loss* et les trois contraintes sont de classe $\mathcal{C}^1$ pour $y \in \{0, 1\}$ et $\widehat{y} \in]0, 1[$. On note $(y_*, \widehat{y}_*)$ un minimum du problème (6.2.1).

Le lagrangien du problème (6.2.1) est défini par :

$$\mathcal{L}(y, \widehat{y}, \lambda, \mu) = loss(y, \widehat{y}) - \lambda y(1-y) - \mu\widehat{y}(1-\widehat{y}), \tag{6.2.2}$$

et son gradient est :

$$\nabla_{y,\widehat{y}} \mathcal{L}(y, \widehat{y}, \lambda, \mu) = \Big(log(\frac{1-y}{y}) - \lambda(1-2y), \frac{\widehat{y}-y}{\widehat{y}(1-\widehat{y})} - \mu(1-2\widehat{y})\Big). \tag{6.2.3}$$

On considère dans un premier temps que les contraintes h_1 et h_2 ne sont pas saturées, c-à-d que $\mu_1 = \mu_2 = 0$. Le minimum $(y_*, \widehat{y}_*)$ est bien un point régulier car $S_{y_*,\widehat{y}*} = \emptyset$ et la différentielle de $x \longmapsto \Big(g(y_*, \widehat{y}_*)\Big) \neq 0$. On a alors le système :

$$\begin{cases} \dfrac{\partial}{\partial y} loss(y_*, \widehat{y}_*) = \lambda_* \dfrac{\partial}{\partial y} g(y_*, \widehat{y}_*), \\ \dfrac{\partial}{\partial \widehat{y}} loss(y_*, \widehat{y}_*) = \lambda_* \dfrac{\partial}{\partial \widehat{y}} g(y_*, \widehat{y}_*), \\ g(y_*, \widehat{y}_*) = 0, \end{cases} \tag{6.2.4}$$

$$\iff \begin{cases} \lambda_* = log(\frac{1-\widehat{y}_*}{\widehat{y}_*}), \\ y = x, \\ x = 0, \end{cases} \quad \text{ou} \quad \begin{cases} \lambda_* = -log(\frac{1-\widehat{y}_*}{\widehat{y}_*}), \\ y = x, \\ x = 1, \end{cases} \tag{6.2.5}$$

donc on trouve deux minimums, $(0,0)$ et $(1,1)$, dont le multiplicateur de lagrange est $\lambda_* = +\inf$.

Lorsque seule la contrainte h_1 est saturée, c-à-d $\mu_1 \geq 0$ et $\mu_2 = 0$, le minimum $(y_*, \widehat{y}_*)$ est un point régulier. En effet, $S_{y_*,\widehat{y}*} = 1$ et la différentielle de $x \longmapsto \Big(g(y_*, \widehat{y}_*), h_1(y_*, \widehat{y}_*)\Big)$ est surjective car

$$\begin{pmatrix} \partial_y g(y_*, \widehat{y}_*) & \partial_{\widehat{y}} h_1(y_*, \widehat{y}_*) \\ \partial_y g(y_*, \widehat{y}_*) & \partial_{\widehat{y}} h_1(y_*, \widehat{y}_*) \end{pmatrix} = \begin{pmatrix} 1-2x & 0 \\ 0 & 1 \end{pmatrix}$$

forme une famille de vecteurs libres. On a alors le système :

$$\begin{cases} \frac{\partial}{\partial y} loss(y_*, \widehat{y}_*) = \lambda_* \frac{\partial}{\partial y} g(y_*, \widehat{y}_*) + \mu_{1_*} \frac{\partial}{\partial y} h_1(y_*, \widehat{y}_*), \\ \frac{\partial}{\partial \widehat{y}} loss(y_*, \widehat{y}_*) = \lambda_* \frac{\partial}{\partial \widehat{y}} g(y_*, \widehat{y}_*) + \mu_{1_*} \frac{\partial}{\partial \widehat{y}} h_1(y_*, \widehat{y}_*), \\ g(y_*, \widehat{y}_*) = 0, \\ h_1(y_*, \widehat{y}_*) = 0, \end{cases} \tag{6.2.6}$$

$$\iff \begin{cases} \lambda_* = log(\frac{1-\widehat{y}_*}{\widehat{y}_*}), \\ \mu_{1_*} = \frac{y}{y(1-y)}, \\ x = 0, \\ y = 0, \end{cases} \quad \text{ou} \quad \begin{cases} \lambda_* = -log(\frac{1-\widehat{y}_*}{\widehat{y}_*}), \\ \mu_{1_*} = \frac{y-1}{y(1-y)}, \\ x = 1, \\ y = 0. \end{cases} \tag{6.2.7}$$

Le point $(1,0)$ est incompatible avec un minimum car dans ce cas, $\mu_{1_*} = -\inf$. Le point $(0,0)$ est encore une fois identifié comme un minimum, avec $\mu_{1_*} = 1$ et $\lambda_* = +\inf$.

De la même manière, lorsque seule la contrainte h_2 est saturée, c-à-d $\mu_1 = 0$ et $\mu_2 \geq 0$, le minimum $(y_*, \widehat{y}_*)$ est un point régulier. En effet, $S_{y_*,\widehat{y}*} = 2$ et la différentielle de $x \longmapsto \Big(g(y_*, \widehat{y}_*), h_2(y_*, \widehat{y}_*)\Big)$ est surjective car

$$\begin{pmatrix} \partial_y g(y_*, \widehat{y}_*) & \partial_{\widehat{y}} h_2(y_*, \widehat{y}_*) \\ \partial_y g(y_*, \widehat{y}_*) & \partial_{\widehat{y}} h_2(y_*, \widehat{y}_*) \end{pmatrix} = \begin{pmatrix} 1-2x & 0 \\ 0 & -1 \end{pmatrix}$$

forme une famille de vecteurs libres. On a alors le système :

$$\begin{cases} \frac{\partial}{\partial y} loss(y_*, \widehat{y}_*) = \lambda_* \frac{\partial}{\partial y} g(y_*, \widehat{y}_*) + \mu_{2_*} \frac{\partial}{\partial y} h_2(y_*, \widehat{y}_*), \\ \frac{\partial}{\partial \widehat{y}} loss(y_*, \widehat{y}_*) = \lambda_* \frac{\partial}{\partial \widehat{y}} g(y_*, \widehat{y}_*) + \mu_{2_*} \frac{\partial}{\partial \widehat{y}} h_2(y_*, \widehat{y}_*), \\ g(y_*, \widehat{y}_*) = 0, \\ h_2(y_*, \widehat{y}_*) = 0, \end{cases} \tag{6.2.8}$$

$$\iff \begin{cases} \lambda_* = log(\frac{1-\widehat{y}_*}{\widehat{y}_*}), \\ \mu_{2_*} = \frac{-y}{y(1-y)}, \\ x = 0, \\ y = 1, \end{cases} \quad \text{ou} \quad \begin{cases} \lambda_* = -log(\frac{1-\widehat{y}_*}{\widehat{y}_*}), \\ \mu_{2_*} = \frac{-(y-1)}{y(1-y)}, \\ x = 1, \\ y = 1. \end{cases} \tag{6.2.9}$$

Cette fois, c'est le point $(0, 1)$ qui est incompatible avec un minimum car $\mu_{2_*} = -\inf$. Quant au point $(1, 1)$, il est encore une fois identifié comme un minimum, avec $\mu_{2_*} = 1$ et $\lambda_* = +\inf$.

Lorsque les deux contraintes h_1 et h_2 sont saturées, c-à-d $\mu_1, \mu_2 \geq 0$, le point $(y_*, \widehat{y}_*)$ n'est pas un point régulier car les gradients des contraintes ne forment pas une famille de vecteurs libres. Il n'y a donc pas de système à résoudre dans ce cas.

Au final, on a trouvé deux minimums, les points $(0, 0)$ et $(1, 1)$. Ceci est confirmé lorsque l'on regarde les lignes de niveaux de la fonction $loss(y_*, \widehat{y}_*)$ Figure 6.1. Néanmoins, ces deux points ne sont pas des points critiques (i.e. $\nabla loss \neq 0$). Le seul point critique de la fonction entropie croisée est le point $(\frac{1}{2}, \frac{1}{2})$, qui est aussi un point selle.

On a également trouvé à chaque fois que

$$\lambda_* = \pm \log(\frac{1 - \widehat{y}_*}{\widehat{y}_*}), \tag{6.2.10}$$

que l'on peut également écrire comme

$$\widehat{y}_* = \frac{1}{1 + exp(\pm\lambda_*)},$$

et on trouve finalement que pour atteindre le minimum du problème d'optimisation considéré, la prédiction soit être le résultat d'une sigmoïd sur le multiplicateur de lagrange λ_*. À cause de la nature de la fonction sigmoïd, on remarque qu'en réalité, on n'aura jamais exactement $\widehat{y}_* = y$, mais plutôt $\widehat{y}_* \to y$. À noter également que jusqu'alors, on n'avait fait aucune supposition sur la nature de la fonction d'activation du neurone.

En régression logistique, la fonction d'activation recommandée est toujours une sigmoïd. Celle-ci prend en variable la somme pondérée des entrées x_i et de leurs poids w_i. On vient de prouver que ce choix est judicieux puisqu'il assure d'atteindre le minimum théorique de la fonction d'entropie croisée.

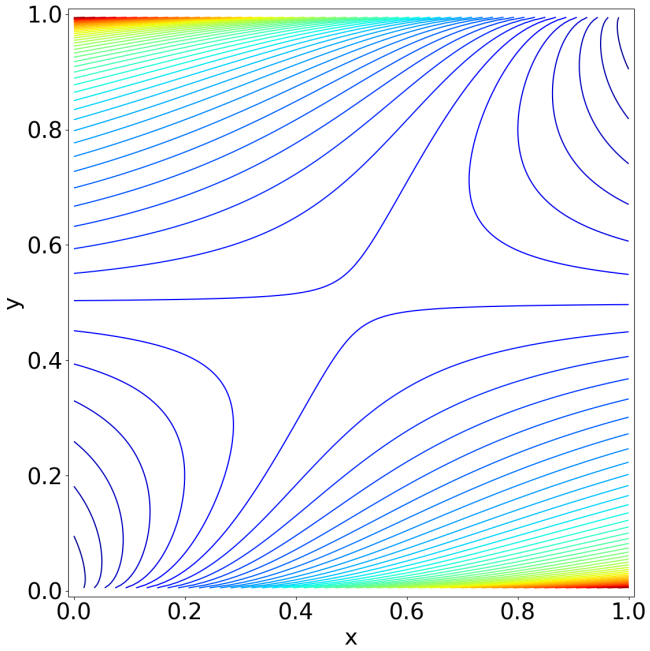


FIGURE 6.1 – Lignes de niveaux de $xlog(y) - (1-x)log(1-y)$

6.3 Test numérique

Maintenant, on souhaite voir si ce résultat théorique est valide numériquement. Pour cela, on reprend les données binaires utilisées au chapitre précédent et présentés Figure 6.2, et on définit un neurone simple avec 2 poids, 1 biais, et une activation sigmoïd (Figure 6.3).

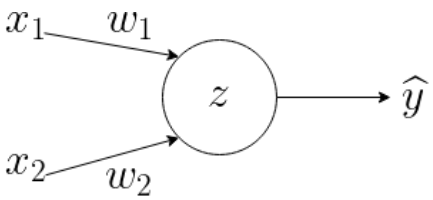


FIGURE 6.2 – Neurone simple

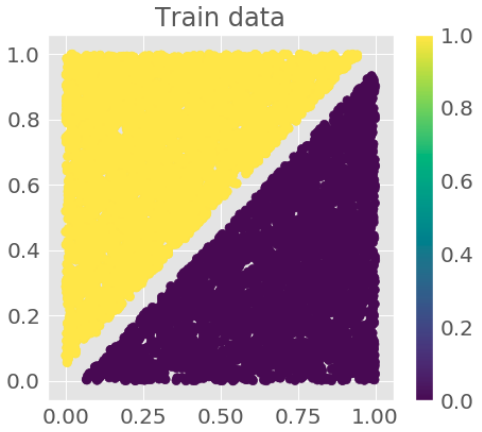


FIGURE 6.3 – Données d'entraînement

On compare les performances d'entraînement avec plusieurs fonctions perte :
- l'entropie croisée : $loss(y, \widehat{y}) = ylog(\widehat{y}) - (1-y)log(1-\widehat{y})$;
- les moindres carrés : $loss(y, \widehat{y}) = \frac{1}{2}(\widehat{y} - y)^2$;
- les moindres carrés logarithmiques (*mean squared logarithmic error* ou MSLE) : $loss(y, \widehat{y}) = log(y+1) - log(\widehat{y}+1)$.

Durant les entraînement, nous avons enregistré le taux de réussite à 1% d'erreur relative maximum, présentés Figure 6.4. Sur ce graphe, on constate que l'entropie croisée est la *loss* ayant la meilleure performance lorsqu'elle est associée à l'activation sigmoïd, confirmant ainsi le résultat précédent.

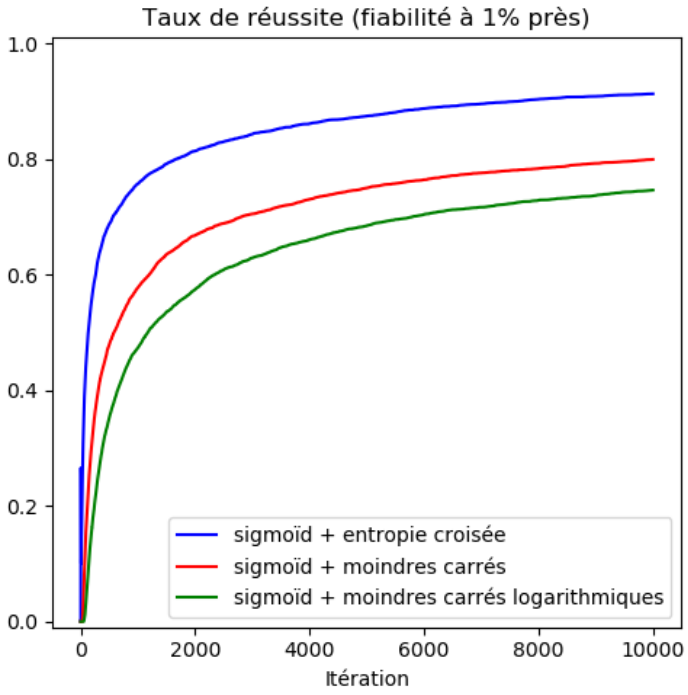


FIGURE 6.4 – Comparaison du taux de réussite à 1% pour différents choix de fonction perte

6.4 Conclusion et perspectives

En posant l'apprentissage d'un neurone simple, où l'entropie croisée est la fonction perte, comme un problème de minimisation avec contraintes d'égalité et d'inégalité, nous avons finalement trouvé que pour tout minimum du problème, la prédiction du neurone était en fait le résultat d'une fonction sigmoïd. Une rapide étude numérique a confirmé qu'en pratique, c'était bien l'association entropie croisée-sigmoïd qui donnait les meilleurs résultats d'entraînement.

Cependant, les réseaux largement utilisés aujourd'hui sont des réseaux profonds. Il serait donc intéressant d'étudier le problème de minimisation dans le cas de réseaux avec 1 couche cachée, et d'étendre la démonstration aux réseaux plus profonds. D'autres fonctions pertes, comme l'erreur quadratique, pourraient également être explorée.

Dans la prochaine partie de ce manuscrit, nous présentons deux études sur l'utilisation de RNA pour la modélisation du transport de sources énergétiques : d'abord dans le cadre de la FCI, avec la modélisation du flux de chaleur non local par des RNA couplés à un code hydrodynamique ; puis dans le cadre de la radiothérapie, avec le calcul de dose par des RNA couplés à un code aux moments.

Quatrième partie

Applications

Chapitre 7

Couplage de réseaux de neurones à un code hydrodynamique pour la FCI

Au Chapitre 5, nous avions introduit une première utilisation des RNA pour la modélisation du flux non local 1D avec le modèle de Luciani-Mora-Virmont. Pour cela, nous avions généré une base de données où les profils en température constituaient les entrées, et les flux non locaux les sorties. Les résultats positifs de cette étude préliminaire servent de preuve de concept à l'étude menée ici, à savoir : l'apprentissage d'un RNA pour qu'il modélise un flux non local, ainsi que son couplage à un code hydrodynamique.

La première partie de ce chapitre présente la stratégie de couplage ainsi que la méthode de génération des données d'apprentissage. Ces données sont séparées en deux types : les profils 1D, et les profils 2D. La deuxième partie de ce chapitre présente les résultats des apprentissages et du couplage pour des données 1D. Il en est de même pour les données 2D dans la troisième partie. La quatrième partie présente des résultats préliminaires sur la base de données 1D avec des réseaux convolutifs. Enfin, ce chapitre se termine par une conclusion où l'on aborde également les perspectives.

Une partie des résultats présentés ici a fait l'objet d'un article publié cette année [136].

7.1 Stratégie de couplage et de génération de données

Avant toute chose, il convient de définir une stratégie de couplage de RNA au code hydrodynamique CHIC [18]. D'après la discussion du Chapitre 2, plusieurs stratégies sont envisageables :

- selon le type de réseau : à propagation avant ou convolutif ;
- selon le type d'entraînement : supervisé ou PINN ;
- selon la librairie de *deep learning* : en Python (Tensorflow ou Keras) ou en Fortran (neural-fortran ou FKB).

Au vu des équations du modèle physique que l'on souhaite remplacer (à savoir le modèle SNB), l'utilisation d'un PINN serait trop hasardeuse : on se tourne donc vers l'apprentissage supervisé, nécessitant la génération d'une base de données. Vient ensuite le choix de la librairie utilisée pour l'apprentissage des réseaux. Les investigations menées jusqu'ici nous ont conduits d'une part à savoir coder les algorithmes de base du *deep learning* (propagation avant, rétropropagation, descente de gradient, etc.), et d'autre part à savoir utiliser la librairie de *deep learning* très utilisée qu'est Tensorflow. Il est donc plus naturel de continuer à utiliser cette librairie que d'apprendre à en maîtriser une nouvelle comme neural-fortran ou FKB, d'autant plus que Tensorflow offre une gamme d'outils bien

FIGURE 7.1 – Schéma d'une ablation laser : la cible en plastique (en gris) est frappée par le faisceau laser à droite.

plus complète. Nos connaissances théoriques nous ont permis de développer avec aisance un module RNA contenant deux routines : l'une pour la lecture des poids et des biais à partir d'un fichier en début de la simulation, l'autre reproduisant la passe-avant. Les entraînements de réseaux sont ainsi donc effectués en dehors du code hydrodynamique, et les poids et biais sont enregistrés dans un fichier. Dès qu'un RNA suffisamment précis est identifié, il est couplé au code CHIC.

Les données d'apprentissage sont générées à l'aide du code CHIC [18], et plus particulièrement à l'aide du module de transport non local. Le modèle de transport d'électrons non local implémenté actuellement dans ce module est celui développé par Schurtz, Nicolaï et Busquet [13] : il calcule le flux de chaleur des électrons dans une approximation multigroupe, peut fonctionner dans plusieurs dimensions spatiales et tient compte des champs magnétiques externes ou auto-générés [19, 137]. Ce modèle est cependant chronophage et ralentit considérablement les simulations hydrodynamiques : l'objectif est de remplacer une partie du module de transport d'électrons SNB par un RNA.

Dorénavant, le flux non local sera calculé par un module RNA de la telle manière : à chaque pas de temps, CHIC calcule le profil hydrodynamique complet instantané (i.e. température T_e, densité n_e, etc.) et le transmet au RNA, qui calcule le flux non local (ou plutôt une portion du flux, qu'on explicitera dans la section suivante) et le renvoie au code hydro pour qu'il calcule le profil hydrodynamique complet au pas de temps suivant. Finalement, on verra par la suite que pour les données que nous avons générées, seul le profil de température suffit au réseau pour faire des prédictions suffisamment précises.

Des bases de données unidimensionnelles et bidimensionnelles ont été construites en lançant des simulations hydrodynamiques d'ablation laser où une cible en plastique est éclairée par un laser depuis le côté droit, comme illustré Figure 7.1. Pour chacune d'entre elle, nous avons entraîné des réseaux denses qui ont par la suite été couplés au code CHIC.

7.2 Résultats pour des cas 1D

7.2.1 Première base de données

7.2.1.1 Description de la base

Dans un premier temps, une base de données, que l'on nomme CHIC1D-prélim, a été générée en exécutant des simulations pour des intensités laser comprises entre 10^{13} W/cm^2 et 10^{15} W/cm^2 à une longueur d'onde de 0.35 μm, et un maillage lagrangien de 240 points. Le maillage a été défini de manière à ce que la convergence du schéma numérique soit atteinte avec un nombre minimal de points de maillage pour toutes les configurations considérées. Le maillage étant lagrangien, il paraissait judicieux en premier lieu de donner les coordonnées des points du maillage en plus de la température en entrée, comme illustré Figure 7.2. On verra par la suite que cela n'est finalement pas nécessaire.

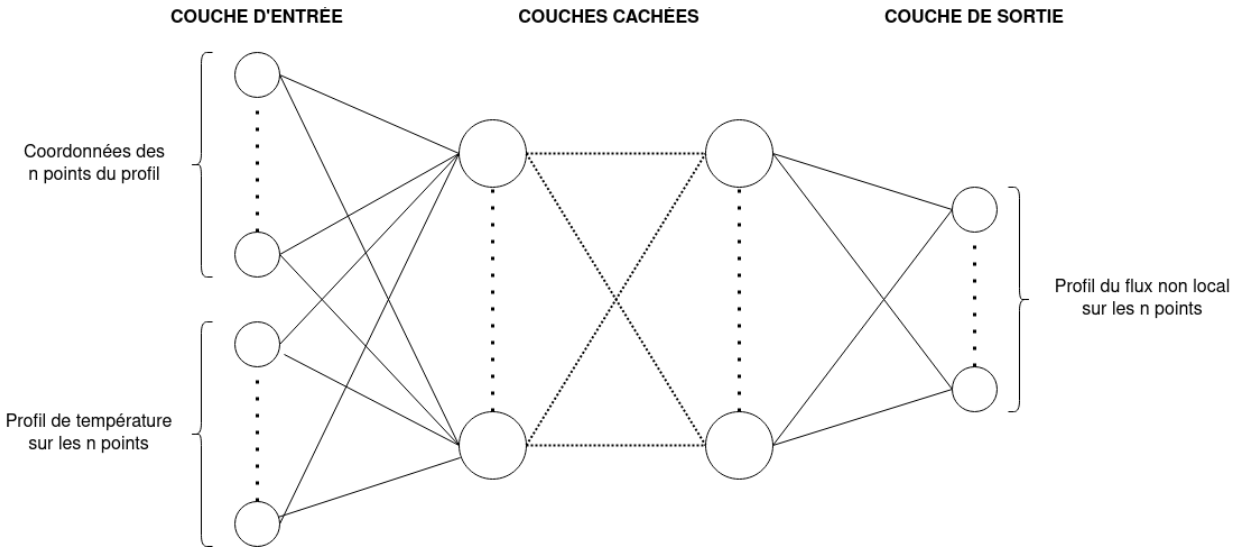


FIGURE 7.2 – Schéma de réseau dense pour la base de données CHIC1D-prélim

Pour chaque intensité laser, les coordonnées des points, ainsi que les valeurs de la température et du flux non local, ont été enregistrés toutes les 50 ps pour un temps de simulation total de 1 ns. Au total, 11 700 couples entrée-cible ont été collectés pour cette première base de données. La Figure 7.3 montre un exemple de profil de température et de flux non local, ainsi que la densité et le flux local associés.

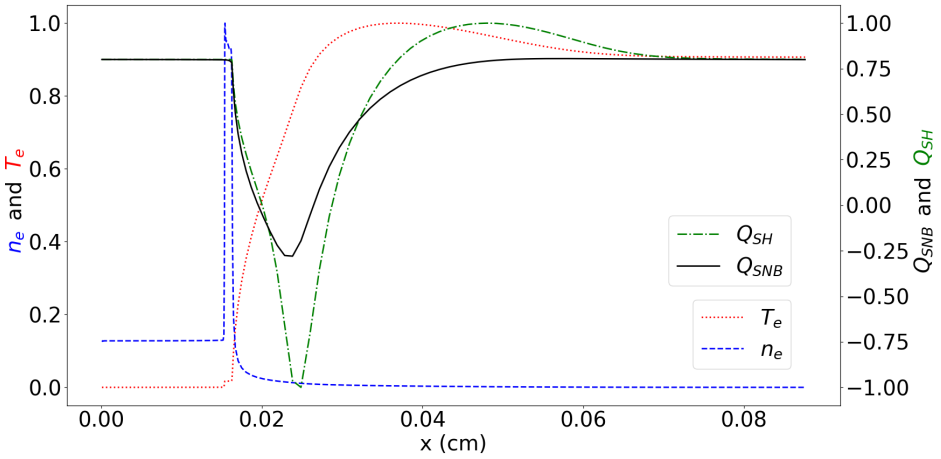
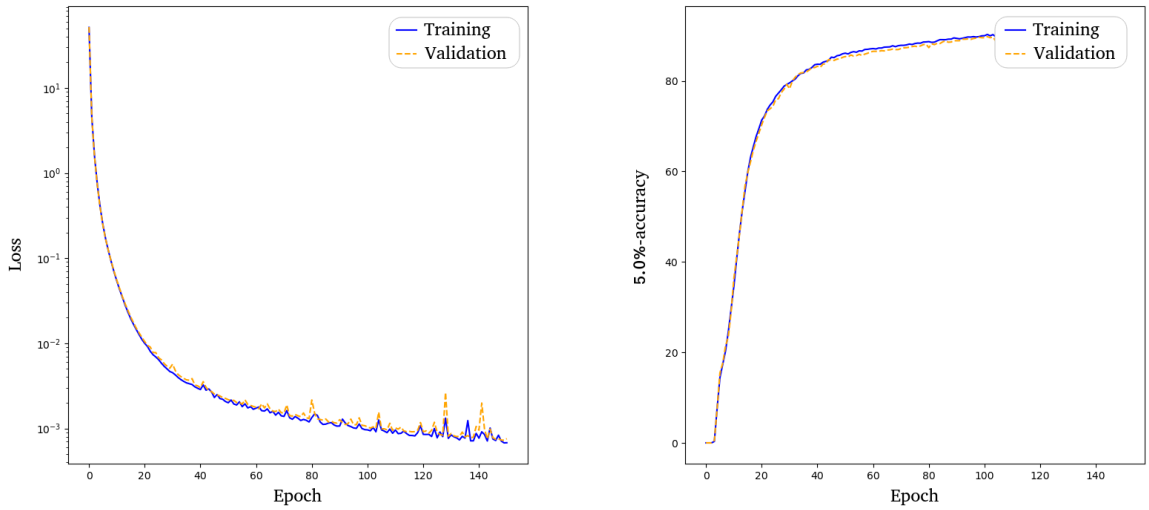


FIGURE 7.3 – Exemple de profil normalisé de la base CHIC1D-prélim, avec le flux non local Q_{SNB} (courbe noire solide), le flux local Q_{SH} (courbe verte pointillée avec tirets), la température T_e (courbe rouge pointillée) et la densité n_e (courbe bleue avec tirets).

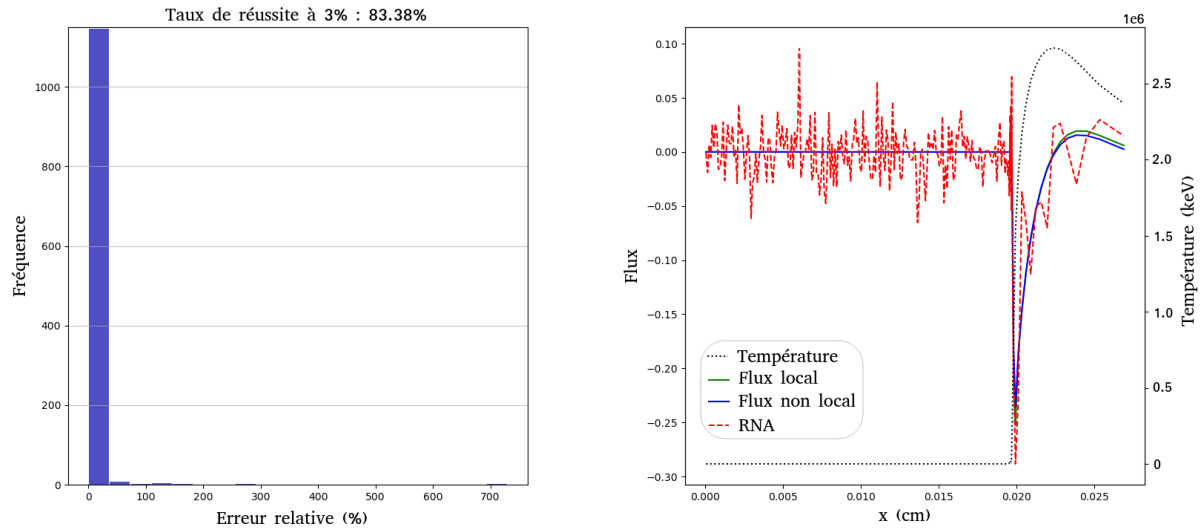
7.2.1.2 Entraînements de réseaux denses

Sur cette première base on s'est rendu compte que de manière générale les réseaux avaient du mal à converger vers le bon profil sur les intensités laser faibles (i.e. lorsqu'il y a peu d'effets non locaux, et que le flux non local est quasi zéro constant), mais aussi sur les premiers pas de temps (i.e. lorsque l'onde est au bord du mileu considéré, avec un gradient très fort).



(a) Évolution de la *loss* (b) Évolution du taux de réussite à 5%

FIGURE 7.4 – Résultats de l'entraînement sur la base CHIC1D-prélim complète

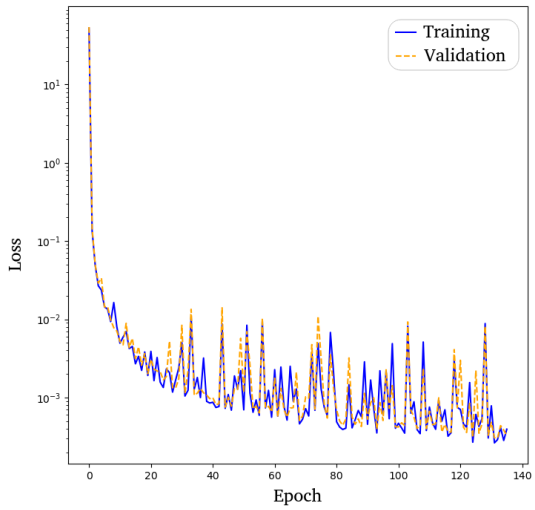


(a) Histogramme des erreurs relatives (b) Exemple de profil

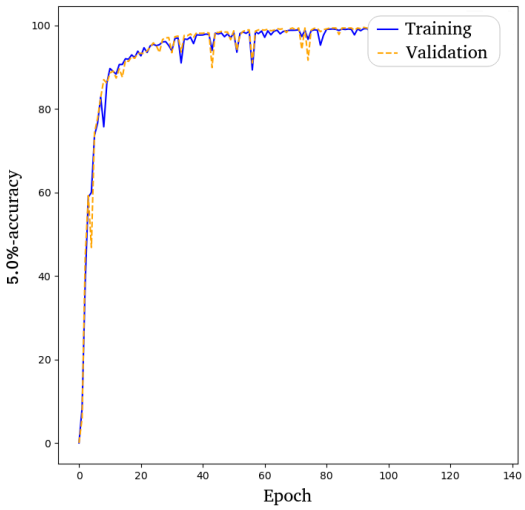
FIGURE 7.5 – Performances sur l'ensemble test de la base CHIC1D-prélim complète

Les Figures 7.4 et 7.5 montrent les résultats d'un entraînement sur la base CHIC1D-prélim. Les Figures 7.4a et 7.4b montrent respectivement l'évolution de la *loss* et du taux de réussite à 5% : on observe des tendances similaires sur les ensembles d'entraînement et de validation. La Figure 7.5a présente l'histogramme des erreurs relatives où seulement 83.38% des données tests respectent le critère de 3% d'erreur relative maximum et l'erreur maximum commise est supérieure à 700%. Un exemple de profil avec des erreurs élevées est montré Figure 7.5b.

Finalement, des tests ont montré qu'en omettant les premières intensités et les premiers pas de temps, on améliore la performance des réseaux. De plus, cela implique que

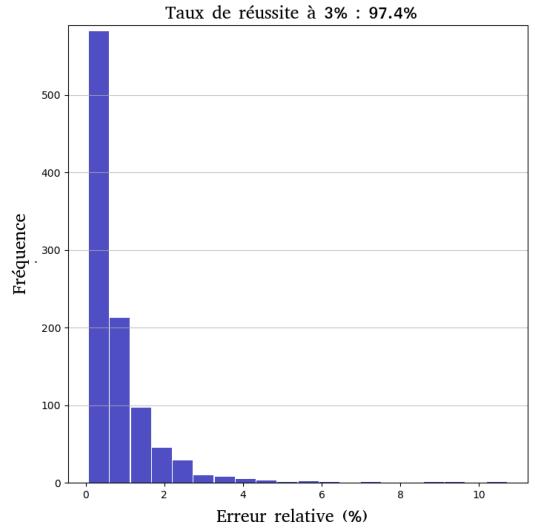


(a) Évolution de la *loss*

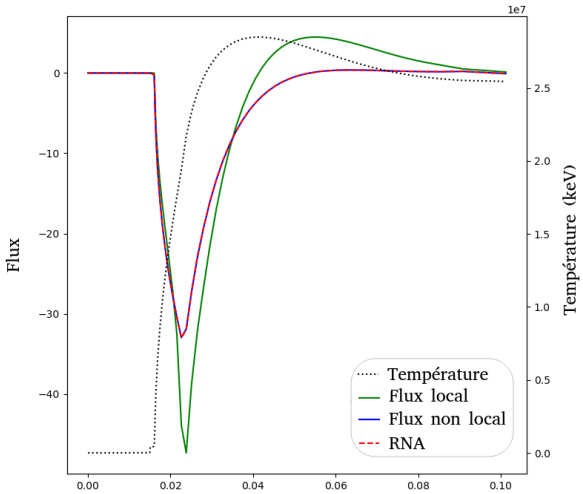


(b) Évolution du taux de réussite à 5%

FIGURE 7.6 – Résultats de l'entraînement sur la base CHIC1D-prélim partielle



(a) Histogramme des erreurs relatives



(b) Exemple de profil

FIGURE 7.7 – Performances sur l'ensemble test de la base CHIC1D-prélim partielle

sur les faibles intensités laser et pour les premiers pas de temps, le module SNB devra être appelé à la place du module RNA durant les simulations du code CHIC. Des résultats d'entraînements sont présentés dans l'annexe A : on y trouve notamment les résultats d'un entraînement sur la base CHIC1D-prélim sans les 5 premières intensités laser, ainsi que les résultats d'un entraînement sur la base CHIC1D-prélim sans les 2 premiers pas de temps. Ces résultats montrent une légère amélioration des performances des réseaux sur l'ensemble test, avec respectivement 94,16% de réussite à 3%, et 92,35% de réussite à 3%, contre 83,38% seulement lorsque l'on utilise la base de données complète.

Les résultats d'un entraînement sur la base CHIC1D-prélim sans les 25 premières intensités laser ni les 2 premiers pas de temps sont montrés Figures 7.6 et 7.7. D'abord, l'évolution de la *loss* et du taux de réussite à 5%, respectivement Figure 7.6a et 7.6b, montrent des tendances similaires sur les ensembles d'entraînement et de validation, malgré les oscillations. Ensuite, l'histogramme des erreurs relatives sur l'ensemble test (Figure 7.7a) montre qu'on arrive enfin à des erreurs plus raisonnables : en effet, 97.4% des erreurs relatives sont inférieures à 3%, avec une erreur maximum aux alentours de 10%. La Figure 7.7b montre un exemple de profil : la prédiction du RNA est en effet très proche du flux non local cible.

Une question se pose également : est-il vraiment nécessaire de donner les coordonnées des points du maillage lagrangien, en plus de la température ? En effet, ce choix était surtout motivé par l'aspect physique du phénomène non local et le fait qu'il s'agisse d'un maillage lagrangien, qui se modifie selon la forme de la solution durant la simulation. Cependant, les RNA comme ceux que l'on utilise ne se soucie pas de la physique réelle, mais cherchent uniquement une corrélation entre un groupe d'entrées et un groupe de sorties. S'il s'agissait de PINN, le calcul d'un gradient impliquerait que les coordonnées soient indiquées en entrée : or, il s'agit de réseaux denses classiques. Après plusieurs tests, nous avons finalement conclu que comme les performances sont similaires avec et sans les coordonnées, celles-ci ne seraient dorénavant plus données en entrée du réseau.

Les Figures 7.8 et 7.9 montrent les résultats d'un entraînement sur la base CHIC1D-prélim partielle, et où seul le profil de température est donné en entrée. Encore une fois, les évolutions de la *loss* (Figure 7.8a) et du taux de réussite à 1% (Figure 7.8b) sont similaires sur les ensembles d'entraînement et de validation, et on atteint même presque 100% de réussite pour un critère à 1%. La performance est également excellente sur l'ensemble test, comme le montre l'histogramme de la Figure 7.9a où 99.2% des prédictions sur l'ensemble test ont une erreurrelative inférieure à 1%. Enfin, un exemple de prédiction est présenté Figure 7.9b, où la prédiction du RNA est très proche de la cible.

Enfin, un dernier point doit être abordé ici : la question du couplage au code hydrodynamique et à l'influence qu'auront les oscillations de la prédiction que nous avons pu observer jusqu'à présent, sur le reste de la simulation. En effet, des oscillations peuvent introduire des instabilités dans le schéma numérique, et engendrer des erreurs de calcul dans le reste du code. Il faut donc envisager une manière de les limiter à l'avenir. Pour cela, nous avons pensé à deux stratégies :

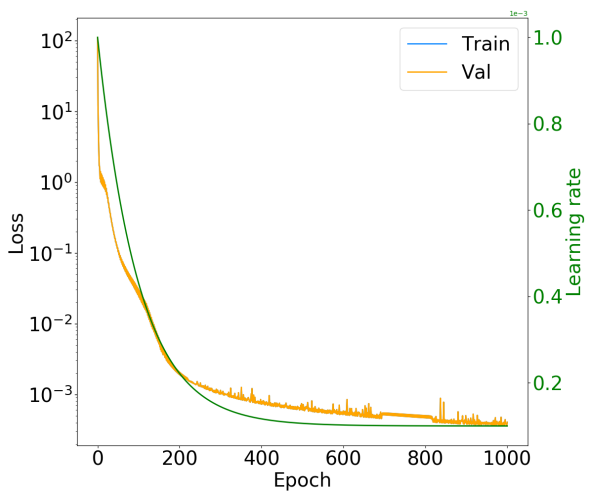
- traiter les oscillations en dehors de l'entraînement, avec une étape de lissage de la prédiction (qu'on présentera plus tard dans ce chapitre) ;
- traiter les oscillations pendant l'entraînement en remplaçant la cible en sorti par la divergence de la somme des contributions des groupes en énergie.

En effet, on rappelle que le flux de chaleur des électrons intervient dans une des équations de conservation de l'énergie du modèle hydrodynamique (3.1.1) défini au Chapitre 3. Plus généralement, cette équation de transport est du type :

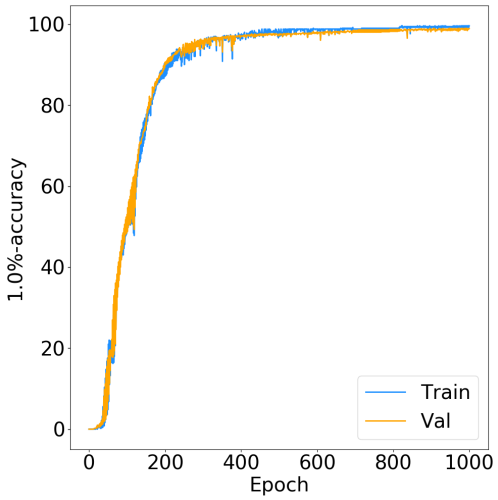
$$\partial_t(\rho\, C_{ve} T_e) + \nabla \cdot \mathbf{Q}_e = W, \tag{7.2.1}$$

où ρ est la densité du plasma, C_{ve} est la capacité thermique spécifique des électrons, T_e est la température et W est un terme source décrivant le dépôt d'énergie du laser et l'échange d'énergie avec les ions. Le flux de chaleur $\mathbf{Q}_e$ n'est autre que le flux de chaleur non local du modèle SNB défini par l'équation :

$$q_{SNB} = q_{SH} - \sum_{g=1}^{N_g} \frac{\lambda_g}{3} \nabla H_g, \tag{7.2.2}$$

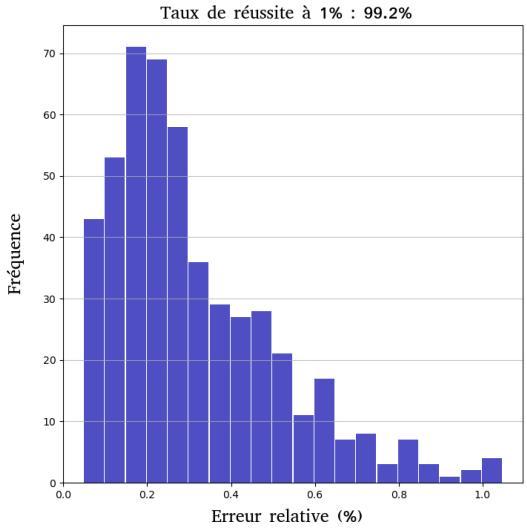


(a) Évolution de la *loss*

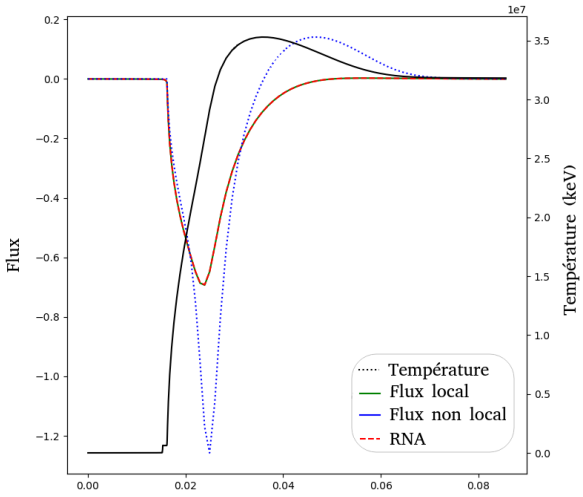


(b) Évolution du taux de réussite à 1%

FIGURE 7.8 – Résultats de l'entraînement sur la base CHIC1D-prélim partielle avec uniquement la température en entrée



(a) Histogramme des erreurs relatives



(b) Exemple de profil

FIGURE 7.9 – Performances sur l'ensemble test de la base CHIC1D-prélim partielle avec uniquement la température en entrée

où les expressions de λ_g et H_g sont exprimées plus en détail dans le Chapitre 3. Si on substitue cette expression dans l'équation (7.2.1), on trouve alors

$$\partial_t(\rho\, C_{ve} T_e) + \nabla \cdot \mathbf{Q}_{SH} = \nabla \cdot \left(\sum_{g=1}^{N_g} \frac{\lambda_g}{3} \nabla H_g \right) + W. \tag{7.2.3}$$

Le calcul du flux de chaleur local q_{SH} est simple, et il en est donc de même pour sa divergence. Quant au calcul de la somme des contributions de tous les groupes en énergie, il implique la résolution d'une équation sur H_g définie au Chapitre 3. Au final, nous avons donc choisi de remplacer la cible en sortie du réseau par la divergence de la correction

non locale, i.e. $S_{\text{nl}} = \nabla\left(\sum_{g=1}^{N_g} \frac{\lambda_g}{3} \nabla H_g\right)$ que l'on appelle dorénavant source non locale, ce qui permet de s'affranchir du calcul de la divergence de la prédiction du réseau.

Fort de toutes les observations faites ici, une nouvelle base de données est ainsi générée.

7.2.2 Base de données 1D finale

7.2.2.1 Description de la base

La base de données CHIC1D est formée en exécutant des simulations pour des intensités laser comprises entre 10^{13} W/cm^2 et 10^{15} W/cm^2 à une longueur d'onde de 0,35 μm, et un maillage lagrangien de 240 points. La plage d'intensité laser a été définie ainsi parce que les effets non locaux commencent à apparaître au-dessus d'une intensité laser d'environ 10^{14} W/cm^2, et à 10^{15} W/cm^2 ils sont suffisamment forts pour modifier les propriétés hydrodynamiques du milieu. Comme les cas 1D sont moins coûteux à générer, il était plus intéressant de tester l'apprentissage de RNA sur des profils avec des effets non locaux faibles à forts. Pour des intensités laser supérieures à 10^{16} W/cm^2, le code CHIC ne décrit pas correctement la physique de l'interaction des plasmas laser.

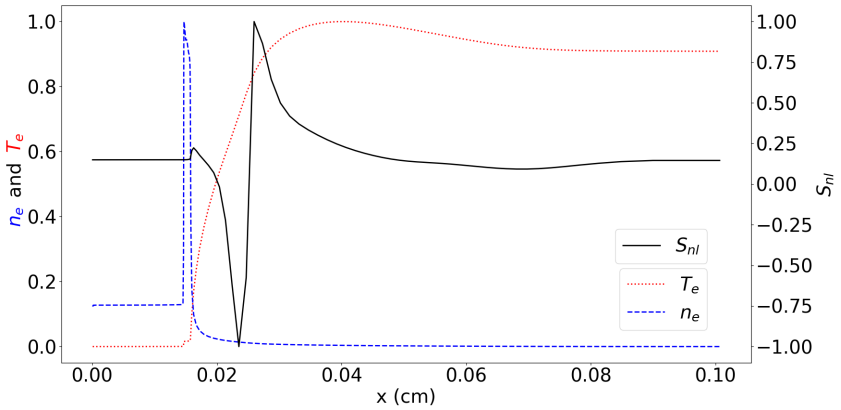


FIGURE 7.10 – Exemple de profil normalisé de la base CHIC1D : la source non locale S_{nl} (courbe noire solide), ainsi que la température T_e (courbe rouge pointillée) et la densité n_e (courbe bleue avec des tirets) correspondants.

Pour chaque intensité laser, les valeurs de la température et de la source non locale ont été enregistrées à tous les points de maillage toutes les 5 ps, à partir du point de la simulation où les effets non locaux sont suffisamment significatifs, généralement autour de 300 ps, et pour un temps total de simulation de 1 ns. Au total, 5738 cas 1D ont été collectés, dont 4500 formant l'ensemble d'apprentissage, 738 l'ensemble de validation et les 500 restants l'ensemble de test. Un exemple de profils 1D de la température et de la source de chaleur associée, ainsi que de la densité, est illustré à la Fig. 7.10. Le faisceau laser vient de la droite, où la température est élevée et la densité est faible, et le flux de chaleur est induit dans une zone de gradients abrupts, où la pression d'ablation entraîne un fort choc, se manifestant par un pic sur le profil de densité à gauche.

7.2.2.2 Entraînements de réseaux denses

Pour cette étude, nous avons choisi de fixer un nombre de neurones cachés, et de faire varier leur répartition sur plus ou moins de couches cachées. Le choix du nombre de neurones cachés ne pouvant être déterminé que de manière empirique, nous avons choisi

	nb de neurones cachés					
nb de couches cachées	240		480		960	
1	60,22 %	66,40 %	61,78 %	60,57 %	64,00 %	63,14 %
2	90,22 %	82,79 %	90,00 %	89,43 %	61,33 %	60,43 %
3	98,44 %	95,12 %	99,11 %	98,64 %	99,11 %	98,64 %
4	99,33 %	98,51 %	99,78 %	99,19 %	99,78 %	99,46 %
5	99,33 %	97,15 %	100,00 %	98,92 %	100,00 %	99,59 %
6	98,67 %	97,97 %	100,00 %	99,05 %	100,00 %	99,05 %
8	5,11 %	4,20 %	100,00 %	99,86 %	100,00 %	99,05 %
10	4,67 %	5,42 %	100,00 %	98,78 %	100,00 %	99,59 %

Tableau 7.1 – Taux de réussite à 3% sur les ensembles d'entraînement (valeurs bleues) et de validation (valeurs orange) - α_0=0.01 et *batchsize*=450

des valeurs multiples du nombre de maille des profils considérés, à savoir des multiples de 240. Trois valeurs de neurones cachés ont été choisies : 240, 480 et 960. Pour ces trois valeurs, on a entraîné des réseaux où les neurones ont été répartis équitablement sur 1, 2, 3, 4, 5, 6, 8 ou 10 couches cachées. Par exemple, un réseau à 960 neurones cachés répartis équitablement sur 6 couches signifie que chaque couche cachée possède 160 neurones. Nous avons également fait varier la taille de *batch* (90 ou 450). Enfin, nous avons fait varier la valeur du taux d'apprentissage initial (0.01 ou 0.005), auquel on ajoute un *time decay*. Seule une partie des résultats d'entraînements de réseaux est présentée ici, le reste étant disponible dans l'annexe B. Plus spécifiquement, on choisit de montrer ici les résultats des entraînements de réseaux pour un *batch* de taille 450 et α_0=0.01.

Le tableau 7.1 présente le taux de réussite à 3% à l'issu de l'entraînement sur les ensembles d'entraînement (valeurs bleues) et de validation (valeurs orange). La première observation faite ici est que de manière générale, les réseaux avec 240 neurones cachés ont des performances moins bonnes que ceux avec 480 ou 960 neurones cachés. De plus, on remarque qu'à partir d'une répartition sur 8 couches cachées, on a une dégradation nette des performances. On remarque aussi que les performances avec 480 ou 960 neurones cachés sont très similaires, et laissent entendre qu'un pallier est déjà atteint avec 480 neurones cachés. Par ailleurs, même si le résultat final est toujours bon pour les réseaux à 960 neurones cachés, l'évolution de la *loss* présente toujours du surappentissage.

Le tableau 7.2 présente les performances sur l'ensemble test à la fin de l'entraînement, à savoir le taux de réussite à 3% (valeurs bleues) ainsi que le taux de réussite pour un gamma index inférieur à 1.

	nb de neurones cachés					
nb de couches cachées	240		480		960	
1	65,40 %	49,20 %	62,00 %	44,80 %	61,60 %	39,00 %
2	84,00 %	67,20 %	88,80 %	75,60 %	59,80 %	33,00 %
3	96,40 %	87,80 %	99,00 %	95,20 %	98,20 %	93,80 %
4	98,00 %	90,80 %	99,80 %	98,80 %	99,00 %	98,40 %
5	98,80 %	90,00 %	99,60 %	99,20 %	99,40 %	99,40 %
6	99,20 %	94,20 %	98,60 %	98,20 %	98,80 %	98,40 %
8	5,00 %	2,40 %	99,60 %	98,80 %	100,00 %	99,20 %
10	5,40 %	1,60 %	99,60 %	98,80 %	99,40 %	98,80 %

Tableau 7.2 – Taux de réussite à 3% d'erreur (valeurs bleues) et taux de réussite pour un gamma index < 1 (valeurs orange) sur l'ensemble test - α_0=0.01 et *batchsize*=450

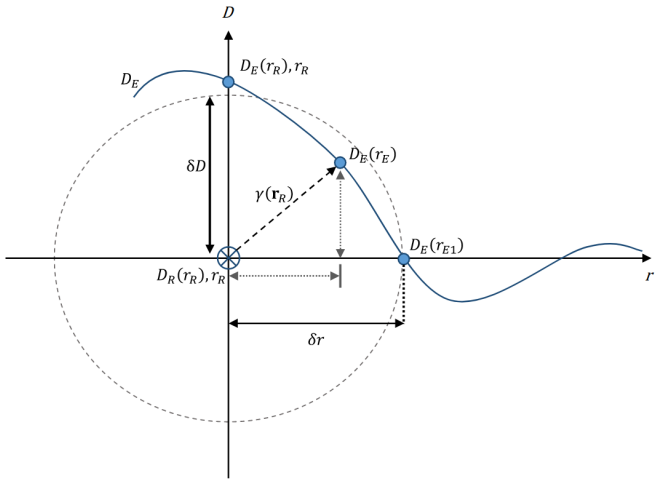
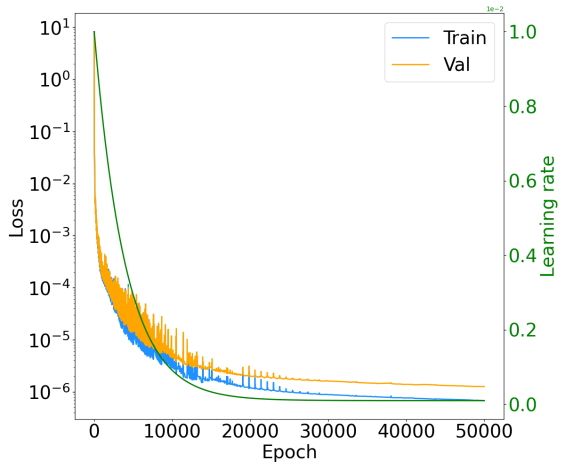


FIGURE 7.11 – Principe de calcul du gamma index [138]

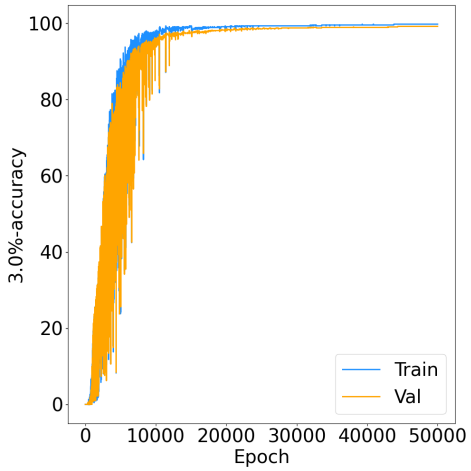
Le gamma index est couramment utilisé en radiothérapie [138, 139] : il évalue la distance entre une fonction prédite f_E et une fonction de référence f_R telle que

$$\gamma(x_R) = \min_{x_E} \Gamma(x_R, x_E),$$

$$\Gamma(x_R, x_E) = \sqrt{\frac{(x_E - x_R)^2}{\delta x^2} + \frac{[f_E(x_E) - f_R(x_R)]^2}{\delta f^2}},$$

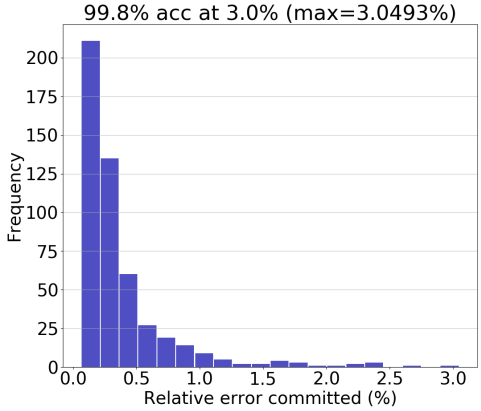


(a) Évolution de la *loss*

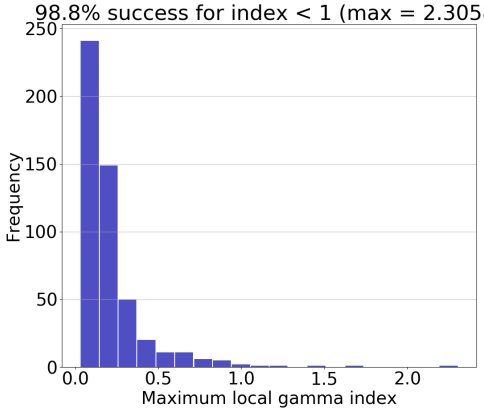


(b) Évolution du taux de réussite à 3%

FIGURE 7.12 – Résultats d'entraînement sur la base CHIC1D : réseau de 480 neurones cachés répartis équitablement sur 4 couches cachées avec activation sigmoïd, *batch* de taille 450 et α_0=0.01



(a) Histogramme des erreurs relatives

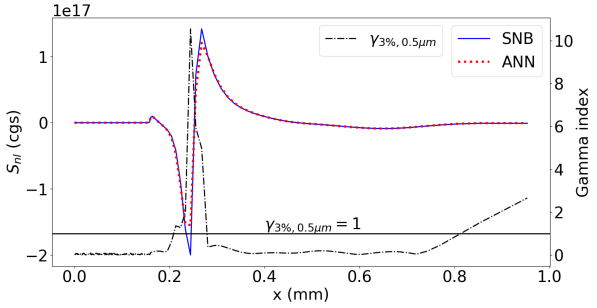


(b) Histogramme des gamma index

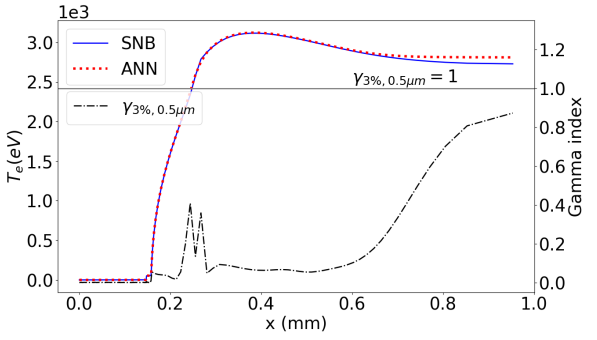
FIGURE 7.13 – Performances sur l'ensemble test de la base CHIC1D : réseau de 480 neurones cachés répartis équitablement sur 4 couches cachées avec activation sigmoïd, *batch* de taille 450 et α_0=0.01

où δx est un critère sur la distance et δf un critère sur la valeur, comme illustré Figure 7.11. Pour les sources non locales S_{nl} utilisées dans cette étude, nous avons défini $\delta x = 0{,}5\ \mu$m et $\delta f = 0,03 \max_x f_E(x)$: un gamma index inférieur à 1 signifie qu'en tout point du maillage, la variable f_E considérée a une erreur inférieure à 3% de la valeur maximale du profil cible sur une distance de 0,5 μm.

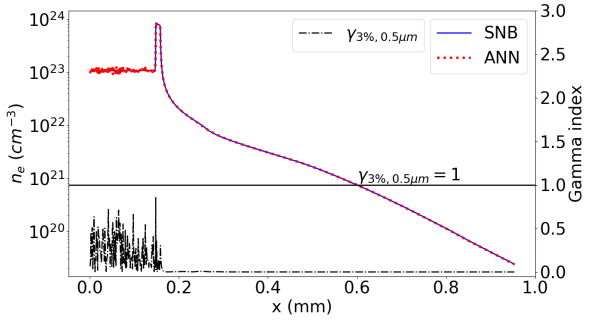
Parmi tous les réseaux testés durant cette étude (i.e. toutes valeurs de α_0 et taille de *batch* comprises), un d'entre eux présente les meilleurs résultats globaux : le réseau de 480 neurones cachés sur 4 couches cachées. Les graphes de performance d'entraînement de ce réseau sont présentées Figures 7.12 et 7.13 : les tendances similaires sur l'évolution de la *loss* et du taux de réussite à 3% sur les ensembles d'entraînement et de validation (Figure 7.12), le taux de réussite à 3% sur l'ensemble test qui reflète les performances d'entraînement (Figure 7.13a), et le taux de réussite pour un gamma index inférieur à 1



(a) Source non locale

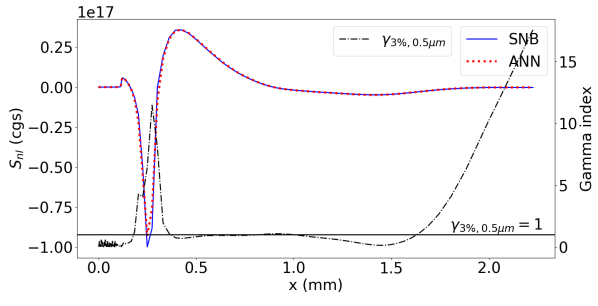


(b) Température

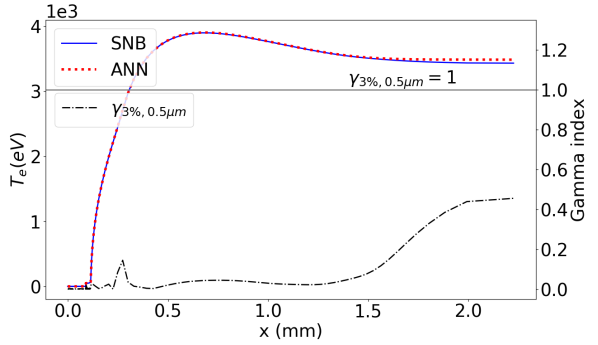


(c) Densité

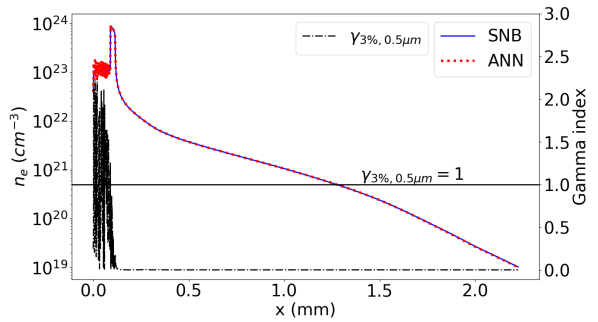
FIGURE 7.14 – Profils hydrodynamiques sans lissage à $t = 500$ps



(a) Source non locale



(b) Température



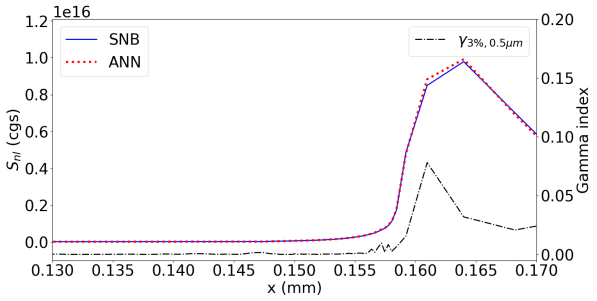
(c) Densité

FIGURE 7.15 – Profils hydrodynamiques sans lissage à $t = 1000$ps

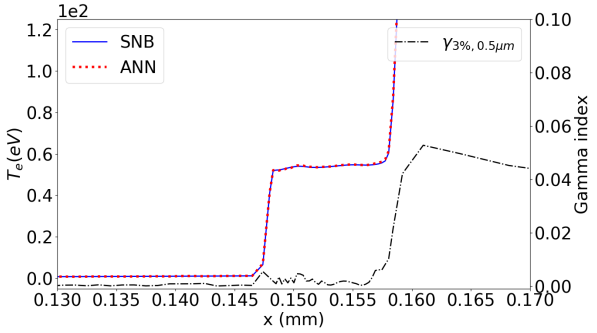
de 98.8% (Figure 7.13b). C'est ce réseau que nous avons couplé au code CHIC et dont on présente les résultats dans la prochaine section.

7.2.3 Couplage au code CHIC

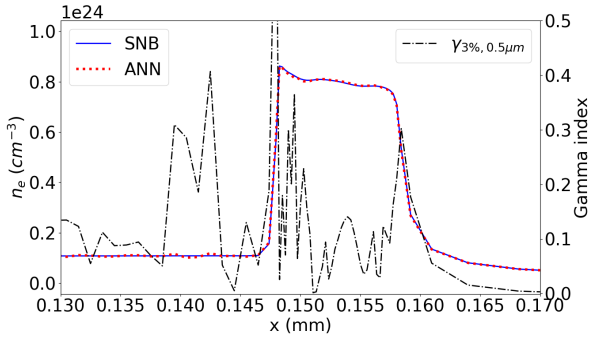
On couple maintenant le réseau retenu au code CHIC. Pour ce faire, on rappelle qu'un module RNA a été implémenté en Fortran afin de reproduire la passe avant d'un réseau dense. Les paramètres ont été enregistrés à l'issu de l'entraînement et sont importés en tout début de simulation. On teste d'abord le RNA sur une intensité laser qui faisait partie des intensités présentes dans la base de données, et on diminue le pas de temps de 5ps à 1ps, le RNA fera ainsi des prédictions à des pas de temps qu'il n'a jamais vus. Le maillage reste fixé à 240 mailles, la cible est encore en plastique, et la simulation dure 1ns. Pour comparer les résultats de la simulation RNA, nous avons également calculé la source issue du module SNB, auquel on fournit le profil en température calculé grâce à la source issue du module RNA. Les profils en température et en densité sont eux comparés aux



(a) Source non locale

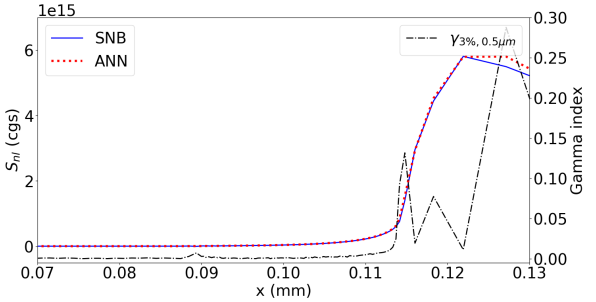


(b) Température

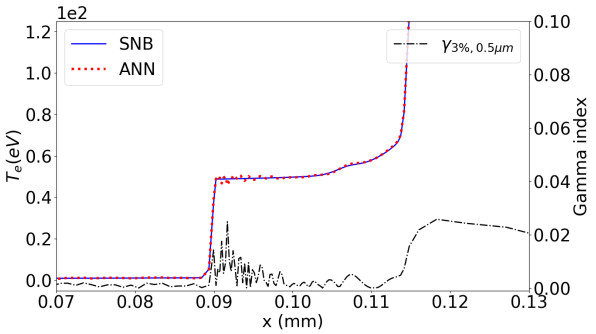


(c) Densité

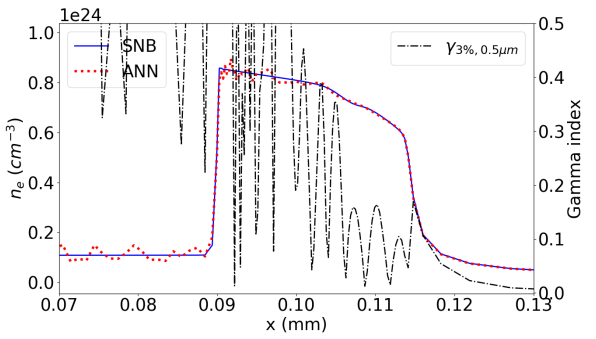
FIGURE 7.16 – Zoom sur les profils hydrodynamiques sans lissage à $t = 500$ps



(a) Source non locale



(b) Température

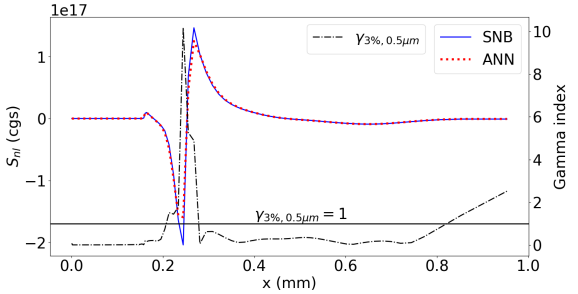


(c) Densité

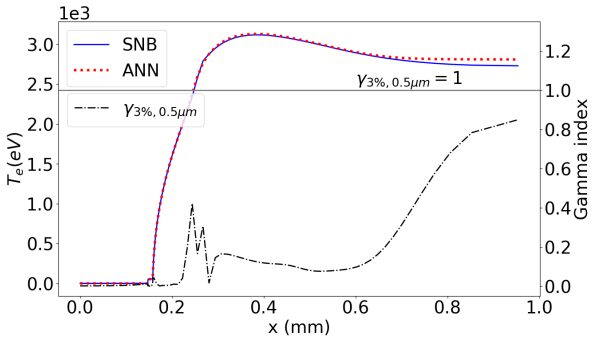
FIGURE 7.17 – Zoom sur les profils hydrodynamiques sans lissage à $t = 1000$ps

profils d'une simulation classique, c-à-d sans aucun RNA, et pour les mêmes paramètres physiques initiaux (intensité laser, maillage, etc.).

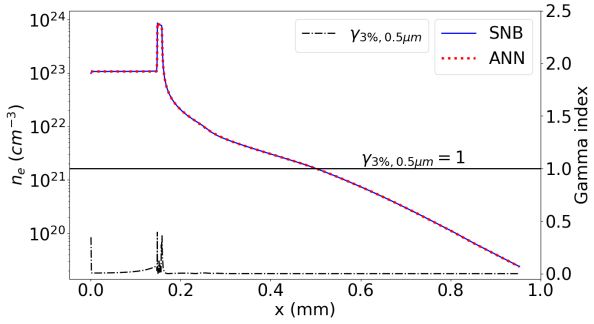
Les Figures 7.14 et 7.15 montrent respectivement les profils hydrodynamiques (i.e. source non locale, température et densité) à $t = 500$ps et $t = 1000$ps. Pour chaque profil, on trace également la valeur du gamma index en chaque point. Les Figures 7.14a et 7.15a montrent que la prédiction du RNA est majoritairement proche de la source calculée par le module SNB pour un même profil en température ($\gamma_{3\%,0.5\mu m} < 1$), sauf sur les forts gradients ainsi que sur le bord droit. Les profils en température du RNA (Figures 7.14b et 7.15b) sont également proches des profils en température de la simulation classique, puisque $\gamma_{3\%,0.5\mu m} < 1$ sur tout le domaine et pendant toute la simulation. On remarque toute fois un léger décollement entre les profils en température RNA et SNB, ainsi qu'une augmentation du gamma index, sur le bord droit du domaine, comme on l'avait déjà observé pour la source non locale. Quant aux profils en densité (Figures 7.14c et 7.15c), on n'observe aucun décollement entre les profils RNA et SNB sur le bord droit du domaine, cependant, on constate que des oscillations se sont accumulées sur le bord gauche du



(a) Source non locale

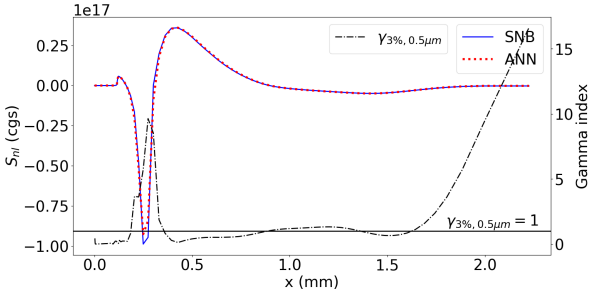


(b) Température

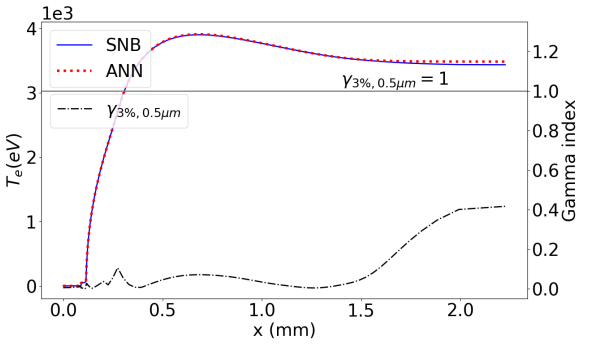


(c) Densité

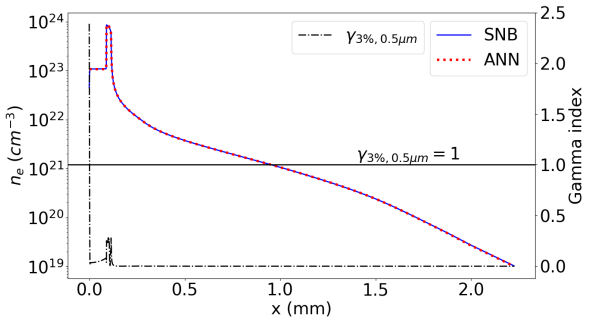
FIGURE 7.18 – Profils hydrodynamiques avec lissage à $t = 500$ps



(a) Source non locale



(b) Température



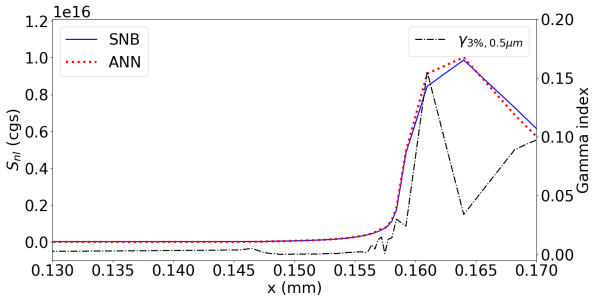
(c) Densité

FIGURE 7.19 – Profils hydrodynamiques avec lissage à $t = 1000$ps

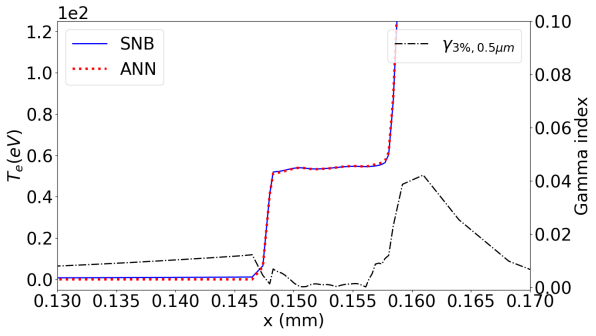
domaine : sur les profils de source non locale et de température, ces oscillations ne sont pas visibles à l'oeil nu, mais elles sont visibles sur les profils de gamma index.

Les Figures 7.16 et 7.17 montrent des zooms sur le bord gauche des profils hydrodynamiques, respectivement à $t = 500$ps et $t = 1000$ps. Les oscillations sur le profil de source non locale sont très faibles (Figures 7.16a et 7.17a) mais elles perturbent de manière préjudiciable la solution globale de la simulation hydrodynamique.

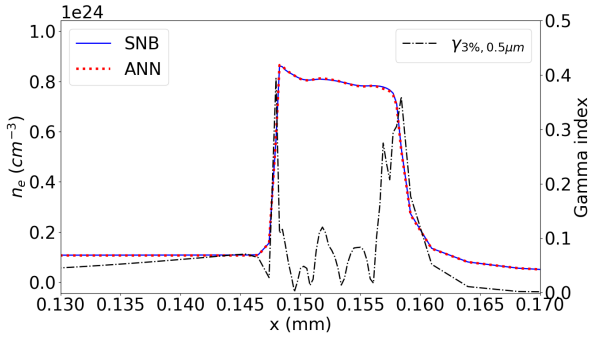
En effet, les oscillations ne sont pas observées aux régions du profil de source non locale où les gradients sont raides, mais bien là où la source est censée rester constante. Ces oscillations parasites se produisent notamment là où le plasma est dense, et où on observe une forte hausse de la densité. La prédiction du RNA est utilisée pour la résolution du système d'équations (3.1.1), et permet ainsi de calculer la température ainsi que la densité (la densité n_e est proportionnelle à la densité ρ qui apparaît dans les équations du système (3.1.1)). Or, ces deux grandeurs sont sensibles aux petites variations, en particulier la densité : ainsi, même une faible perturbation numérique sur la source non locale peut conduire à une augmentation exponentielle de la pression, qui provoquerait à son tour des oscillations de la densité qui ne sont pas tolérables (Figures 7.16c et 7.17c).



(a) Source non locale

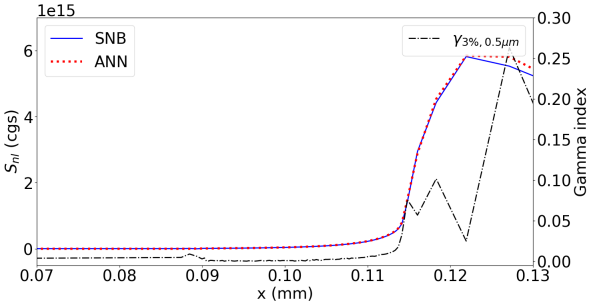


(b) Température

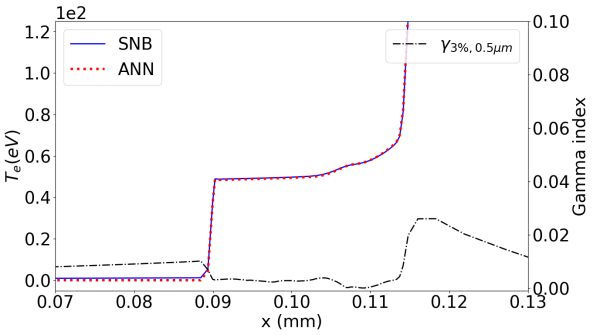


(c) Densité

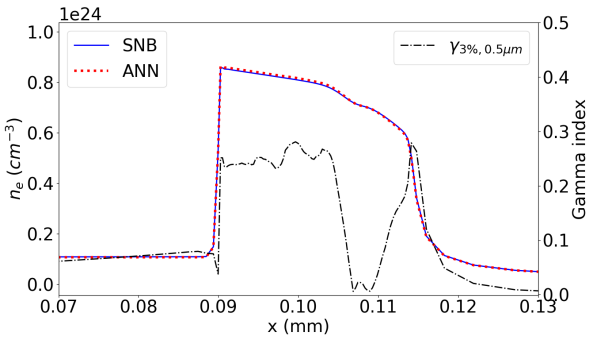
FIGURE 7.20 – Zoom sur les profils hydrodynamiques avec lissage à $t = 500$ps



(a) Source non locale



(b) Température

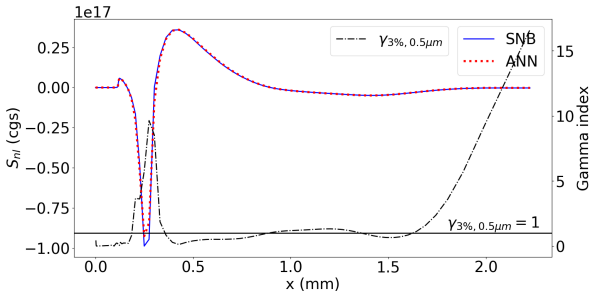


(c) Densité

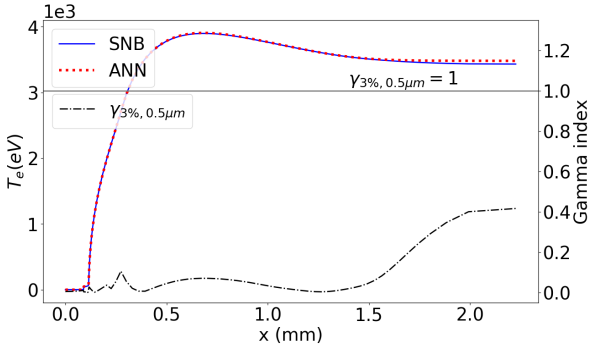
FIGURE 7.21 – Zoom sur les profils hydrodynamiques avec lissage à $t = 1000$ps

Sur la Figure 7.16, on observe qu'une légère perturbation de la source non locale a engendré de faibles oscillations sur le profil en température, qui ensuite ont engendré des oscillations plus fortes sur le profil en densité. Les oscillations sont plus perceptibles sur le gamma index, en particulier pour le profil en densité. On constate également que les oscillations s'accumulent et augmentent tout au long de la simulation. Sur la Figure 7.17, les oscillations à $t = 1000$ps (c-à-d à la fin de la simulation) sont nettement plus importantes sur les trois profils. Le gamma index local a également augmenté, et n'est même plus inférieur à 1 pour le profil en densité.

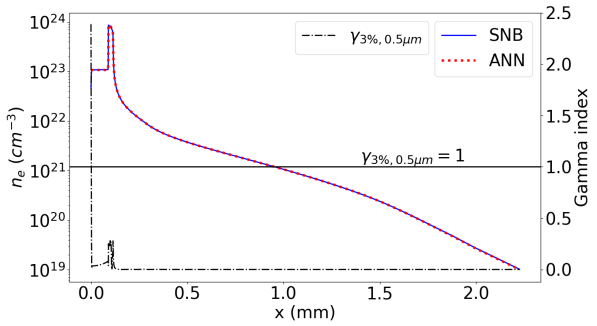
Une solution courante dans l'approche numérique classique est d'affiner le maillage sur la région d'intérêt, en particulier, dans les régions avec des gradients forts. Cependant, ce problème apparaît sur une partie du profil où la source non locale est constante en zéro, et il ne semble pas y avoir d'intérêt à augmenter la taille du maillage dans cette zone. C'est pourquoi la stratégie retenue ici est de plutôt lisser la prédiction du réseau. Un comportement oscillatoire similaire a été observé par Bois et al. [71], et ils ont aussi employé une stratégie de lissage pour remédier à ce problème.



(a) Source non locale

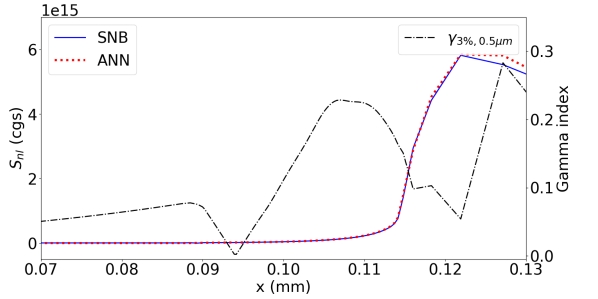


(b) Température

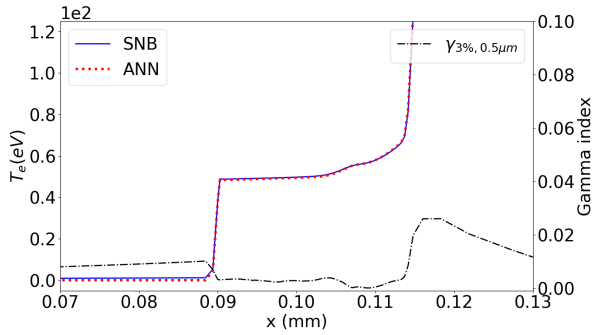


(c) Densité

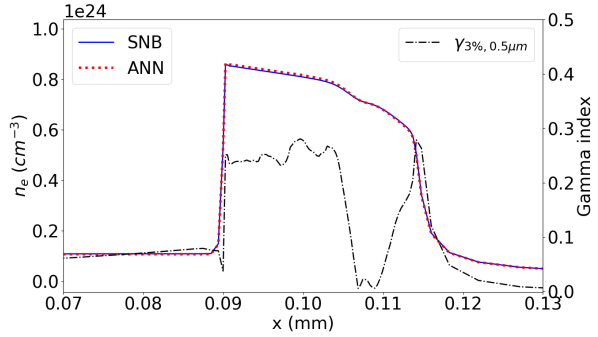
FIGURE 7.22 – Profils hydrodynamiques avec lissage à $t = 1000$ps (intensité laser inconnue du RNA)



(a) Source non locale



(b) Température



(c) Densité

FIGURE 7.23 – Zoom sur les profils hydrodynamiques avec lissage à $t = 1000$ps (intensité laser inconnue du RNA)

Étant donné que les oscillations sont concentrées sur la section du maillage où la température est basse et que des effets non locaux n'y sont pas attendus, le lissage va en fin de compte consister en fixer la valeur de la sortie prédite à zéro sur cette partie du maillage. De fait, nous avons intitulé ce lissage *zero smoothing* ou lissage zéro [136]. Les Figures 7.18 à 7.21 montrent les effets de cette stratégie de lissage sur la simulation RNA.

Sur ces Figures, on constate que le lissage a eu un fort impact sur la simulation, notamment sur la densité. En effet, bien que les oscillations soient toujours présentes sur le profil en température, elles sont plus faibles : le maximum du gamma index reste quasiment identique, mais les oscillations sur le pied d'onde (i.e. la première hausse en température) ont nettement diminué. De la même manière, les oscillations sur le profil en densité sont drastiquement réduites, et le gamma index local est également inférieur à 1 tout au long de la simulation en tout point du maillage.

Enfin, nous avons également voulu voir comment réagirait le RNA fasse à une intensité laser qui ne faisait pas partie des données d'entraînement mais qui fait partie de la plage

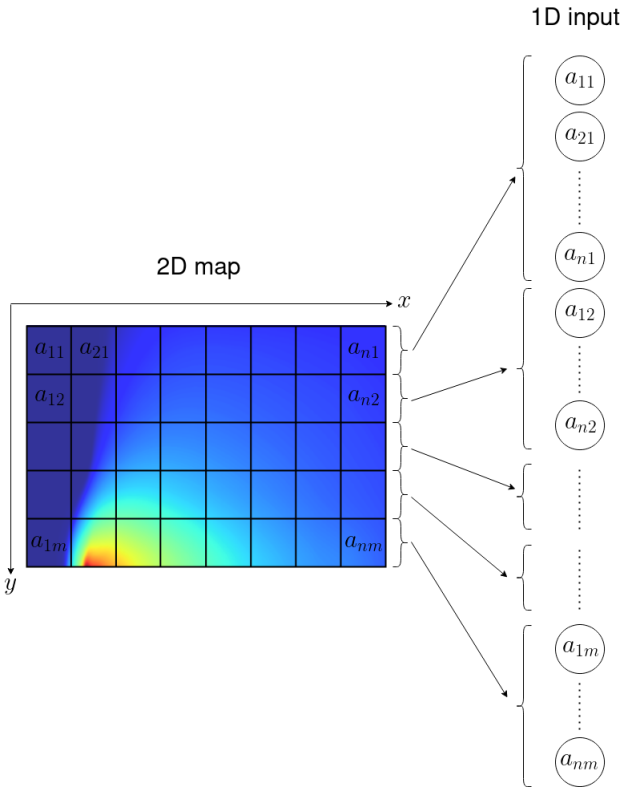


FIGURE 7.24 – Schéma de la transformation d'un profil 2D en un tableau 1D.

d'intensité laser balayée par les données d'entraînement. Les résultats à $t = 1000$ ps sont montrés Figures 7.22 et 7.23 (annexe B pour les résultats à $t = 500$ ps). La stratégie de lissage a également été utilisée pour cette simulation.

On constate Figure 7.22 que le RNA a encore une fois du mal à approcher les forts gradients (7.22a) mais que les profils en température (7.22b) et en densité (7.22c) sont très proches des profils issus d'une simulation où seul le module SNB était utilisé. Les zooms sur la portion gauche du domaine où se trouvent le pied de température et la hausse de la densité confirment qu'avec la stratégie de lissage, le RNA est même capable d'approcher une simulation pour une intensité laser inconnue (Figure 7.23). Ces résultats viennent confirmer le fait qu'un RNA est un très bon outil d'interpolation.

Une dernière analyse intéressante à faire ici est les conséquences de l'utilisation du RNA sur le temps de calcul. En moyenne, le temps de calcul de la source non locale est réduit d'un facteur 233 sur 1 pas de temps. Plus spécifiquement, le temps de calcul est en moyenne 1,6 ms pour la simulation avec RNA, sachant que 1,1% de ce temps est consacré au lissage de la prédiction, contre 375 ms pour la simulation classique. Ce gain est non négligeable et à ce stade, on s'attend à un gain au moins similaire en 2D.

7.3 Résultats pour des cas 2D

7.3.1 Description de la base

La durée d'une simulation étant nettement plus longue qu'en 1D, la base de données CHIC2D a été construite en lançant une seule simulation avec une intensité laser de 10^{15} W/cm^2. À la différence de la base CHIC1D dont les profils étaient calculés sur un maillage lagrangien, les profils de la base CHIC2D sont eux calculés sur une grille

eulérienne équidistante, avec 50 points le long de la direction de propagation du laser (axe x) et 20 points dans la direction transversale (axe y). La température et la source non locale ont été sauvegardées toutes les 1 ps à partir de $t = 100$ ps, jusqu'à la fin de la simulation ($t = 1000$ ps). Au total, 900 profils 2D ont été générés, dont 600 pour l'ensemble d'apprentissage, 150 pour l'ensemble de validation et 150 pour l'ensemble de test. Comme les réseaux utilisés sont des réseaux à propagation avant, chaque profil 2D a été transformé en un tableau unidimensionnel de longueur $N_i = 50 \times 20$ (Figure 7.24).

7.3.2 Entraînements de réseaux denses

Pour la base de données CHIC2D, nous avons fixé le nombre de couches cachées au nombre de 4 car les études menées jusqu'à maintenant ont prouvé qu'il n'était pas nécessaire d'en mettre plus pour atteindre une bonne performance, compte tenu des données considérées. Nous avons fait varier le nombre de neurones par couche cachée, ainsi que la valeur du taux d'apprentissage initial, et nous avons fixé la taille de *batch* à 60. Les résultats sont présentés dans les Tableaux 7.3 et 7.4.

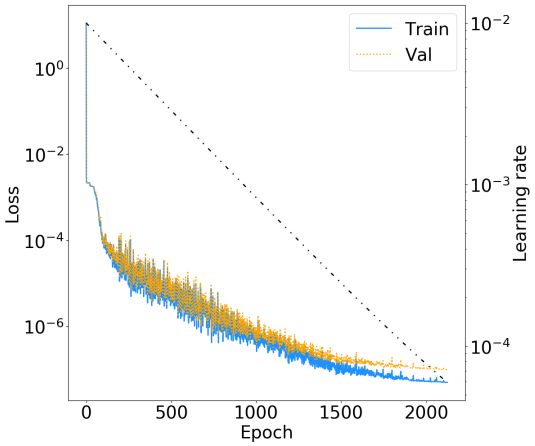
Le Tableau 7.3 présente le taux de réussite à 3% à l'issu des entraînements sur les ensemble d'entraînement et de validation. On constate que les performances sont excellentes

	4CC150N		4CC200N		4CC250N	
α_0=0,01	100,00 %	100,00 %	100,00 %	100,00 %	99,17 %	99,33 %
α_0=0,005	100,00 %	100,00 %	100,00 %	100,00 %	100,00 %	100,00 %
α_0=0,003	99,17 %	94,00 %	99,83 %	100,00 %	99,67 %	98,00 %
α_0=0,001	19,50 %	14,67 %	53,33 %	46,67 %	100,00 %	97,97 %

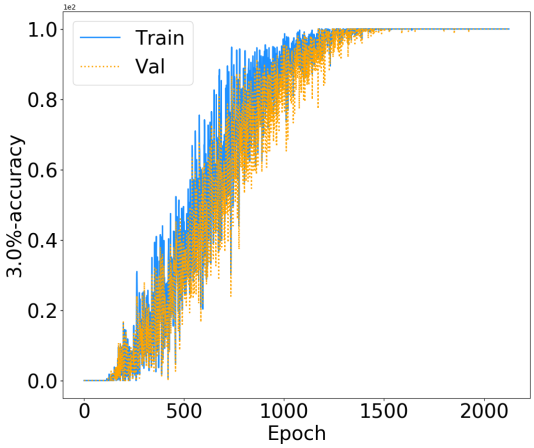
Tableau 7.3 – Taux de réussite à 3% sur les ensembles d'entraînement (valeurs bleues) et de validation (valeurs orange)

	4CC150N		4CC200N		4CC250N	
α_0=0,01	100,00 %	99,33 %	100,00 %	99,33 %	98,00 %	97,33 %
α_0=0,005	100,00 %	99,33 %	100,00 %	100,00 %	100,00 %	100,00 %
α_0=0,003	96,67 %	96,00 %	100,00 %	99,33 %	98,00 %	93,00 %
α_0=0,001	14,00 %	15,33 %	43,33 %	44,00 %	99,33 %	98,67 %

Tableau 7.4 – Taux de réussite à 3% d'erreur (valeurs bleues) et taux de réussite pour un gamma index < 1 (valeurs orange) sur l'ensemble test

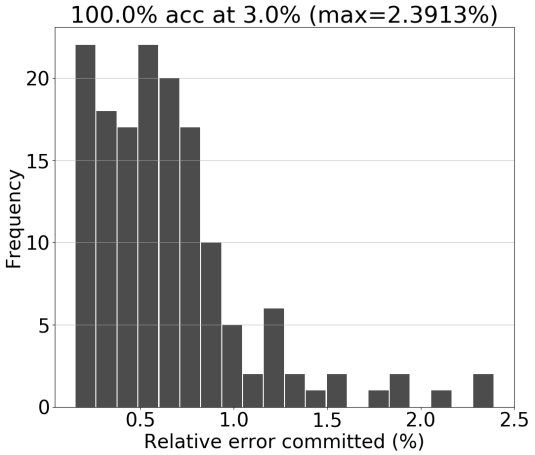


(a) Évolution de la *loss*

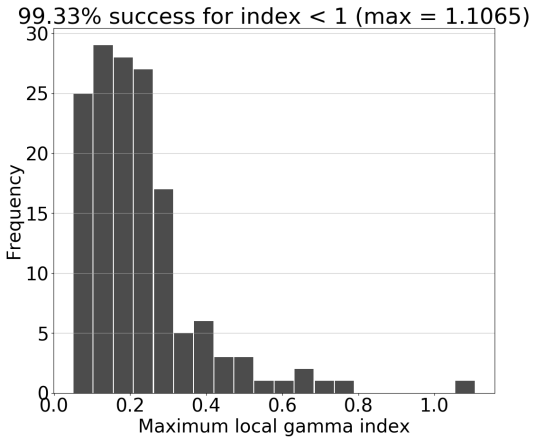


(b) Évolution du taux de réussite à 3%

FIGURE 7.25 – Résultats d'entraînement sur la base CHIC2D : réseau de 4 couches cachées à 150 neurones avec activation sigmoïd, *batch* de taille 60 et α_0=0.01



(a) Histogramme des erreurs relatives

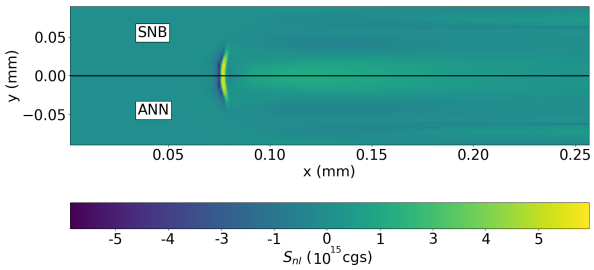


(b) Histogramme des gamma index

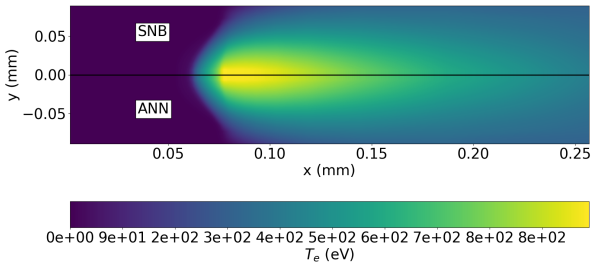
FIGURE 7.26 – Performances sur l'ensemble test de la base CHIC2D : réseau de 4 couches cachées à 150 neurones avec activation sigmoïd, *batch* de taille 60 et α_0=0.01

avec seulement 150 neurones par couche cachée, et qu'une faible performance est souvent associée à un trop faible taux d'apprentissage initial. Les mêmes remarques sont valables pour les performances sur l'ensemble test, présentées Figure 7.4.

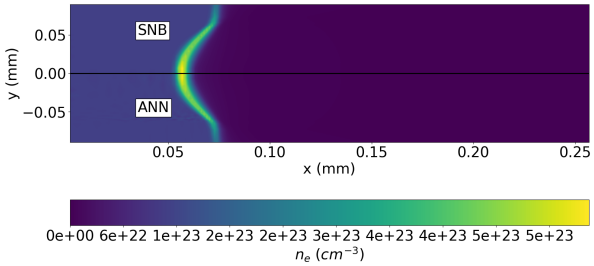
Finalement, aux vues des résultats des tableaux ainsi que des observations des graphes d'évolution de la *loss*, le meilleur réseau est celui de 4 couches cachées de 150 neurones avec un taux d'apprentissage initial de 0.01. Les résultats de son entraînement sont présentés Figures 7.25 et 7.26. On y observe en effet que les tendances des évolutions de la *loss* et du taux de réussite à 3% (respectivement Figures 7.25a et 7.25b) sont similaires sur les ensembles d'entraînement et de validation, et que le taux de réussite atteint 100% sur les deux ensembles à la fin de l'entraînement. Les performances sont également excellentes sur l'ensemble test, avec 100% de réussite à 3% (Figure 7.26a), ainsi que 99.33% de réussite pour un gamma index < 1 (Figure 7.26b). C'est donc ce réseau que l'on couple au code CHIC dans la prochaine section.



(a) Source non locale

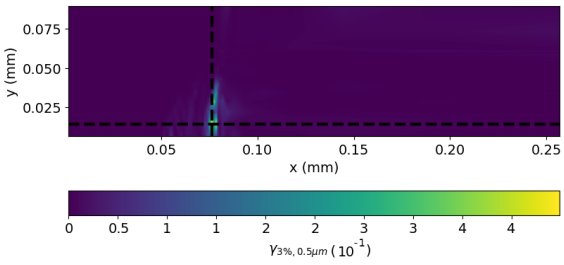


(b) Température

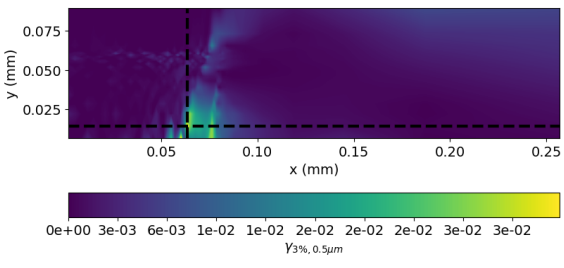


(c) Densité

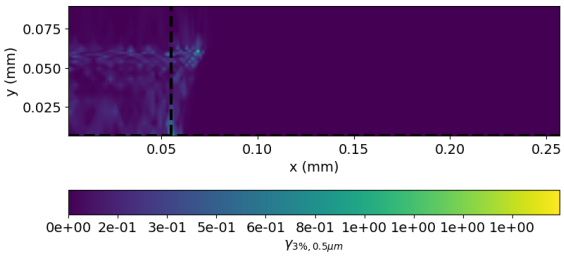
FIGURE 7.27 – Profils hydrodynamiques sans lissage à $t = 500$ps



(a) Source non locale



(b) Température



(c) Densité

FIGURE 7.28 – Gamma index des profils hydrodynamiques sans lissage à $t = 500$ps

7.3.3 Couplage au code CHIC

Pour le couplage, le même module RNA est utilisé. On ne montre ici que les profils hydrodynamiques à $t = 500$ps sur les Figures 7.27 à 7.30. Les profils à $t = 1000$ ps sont disponibles dans l'annexe C.

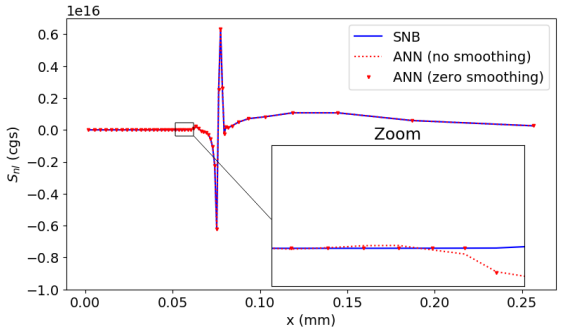
La Figure 7.27 montre une comparaison entre les profils provenant d'une simulation classique (partie supérieure des profils, nommée SNB) et la simulation avec le module RNA (partie inférieure des profils, nommée ANN). Les profils montrés ici sont ceux sans lissage : les profils avec lissage sont présentés dans l'annexe C et sont quasi identiques à ceux sans lissage, sur lesquels on ne distingue pas les oscillations à l'oeil nu. Même sans lissage, les profils RNA sont donc quasi identiques aux profils SNB.

La Figure 7.28 présente le gamma index local pour chaque profil hydrodynamique. En 2D, le calcule du gamma index implique l'ajout d'un critère en espace sur une sphère, noté δr, et le gamma index 2D s'écrit alors :

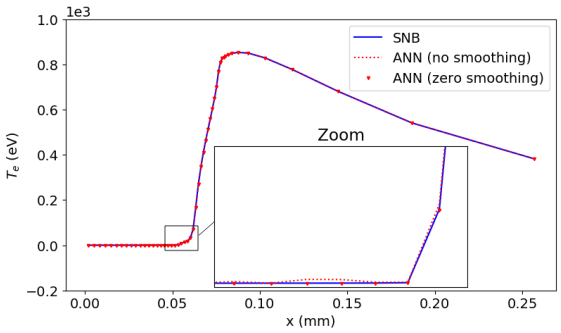
$$\gamma(r_R) = \min_{r_E} \ \Gamma(r_R, r_E), \ \ r_R = \sqrt{x_R^2 + y_R^2}$$

$$\Gamma(r_R, r_E) = \sqrt{\frac{(r_E - r_R)^2}{\delta r^2} + \frac{[f_E(r_E) - f_R(r_R)]^2}{\delta f^2}}.$$

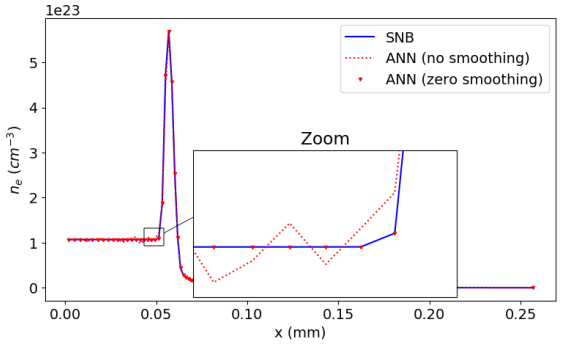
Pour cette étude, nous avons définit $\delta r = 0.5\mu$m et $\delta f = 0.03 \max_{x,y} f_E(x, y)$.



(a) Source non locale

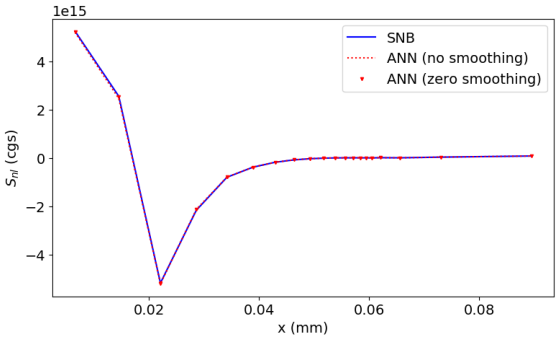


(b) Température

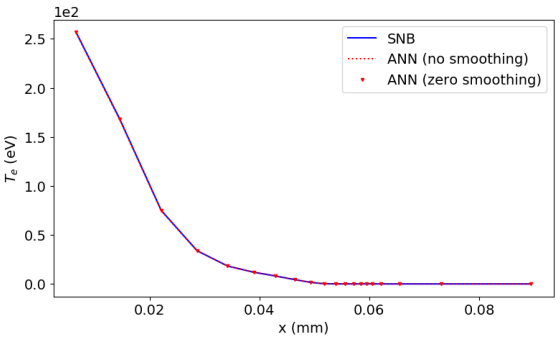


(c) Densité

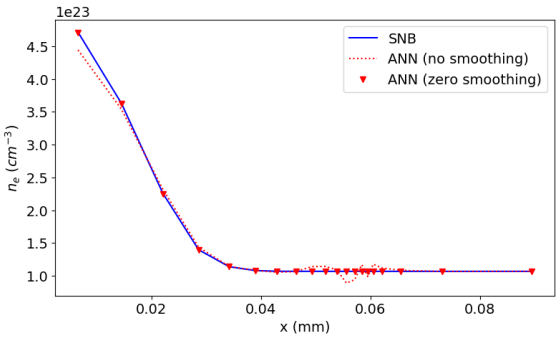
FIGURE 7.29 – Coupe longitudinale des profils hydrodynamiques à $t = 500$ps



(a) Source non locale



(b) Température



(c) Densité

FIGURE 7.30 – Coupe transversale des profils hydrodynamiques à $t = 500$ps

Bien que le critère choisi soit respecté pour tous les profils, les cartes 2D des gamma index semblent présenter des oscillations. Deux lignes pointillées sont tracées sur chaque carte, une longitudinale et l'autre transversale : elles se rencontrent là où le gamma index est le plus élevé. À chacune d'elle est associée une coupe 1D où l'on compare le profil de la simulation classique aux profils de la simulation RNA avec et sans lissage zéro : les coupes longitudinales sont présentées Figure 7.29 et les coupes transversales Figure 7.30.

Tout d'abord, on constate qu'encore une fois les oscillations sur la prédiction du réseau sont très faibles mais bien présentes, et par cascade elles se sont décuplées sur le profil en température, puis le profil en densité. Les coupes à $t = 1000$ ps présentées dans l'annexe C montrent également que les oscillations s'accumulent et augmentent durant la simulation.

Quant au lissage, nous avons utilisé la même stratégie qu'en 1D, c-à-d que la prédiction du réseau est fixée à zéro avant la hausse en température (portion gauche des profils). Les coupes 1D montrent que ce lissage a encore une fois réussi à atténuer les oscillations sur le profil en densité.

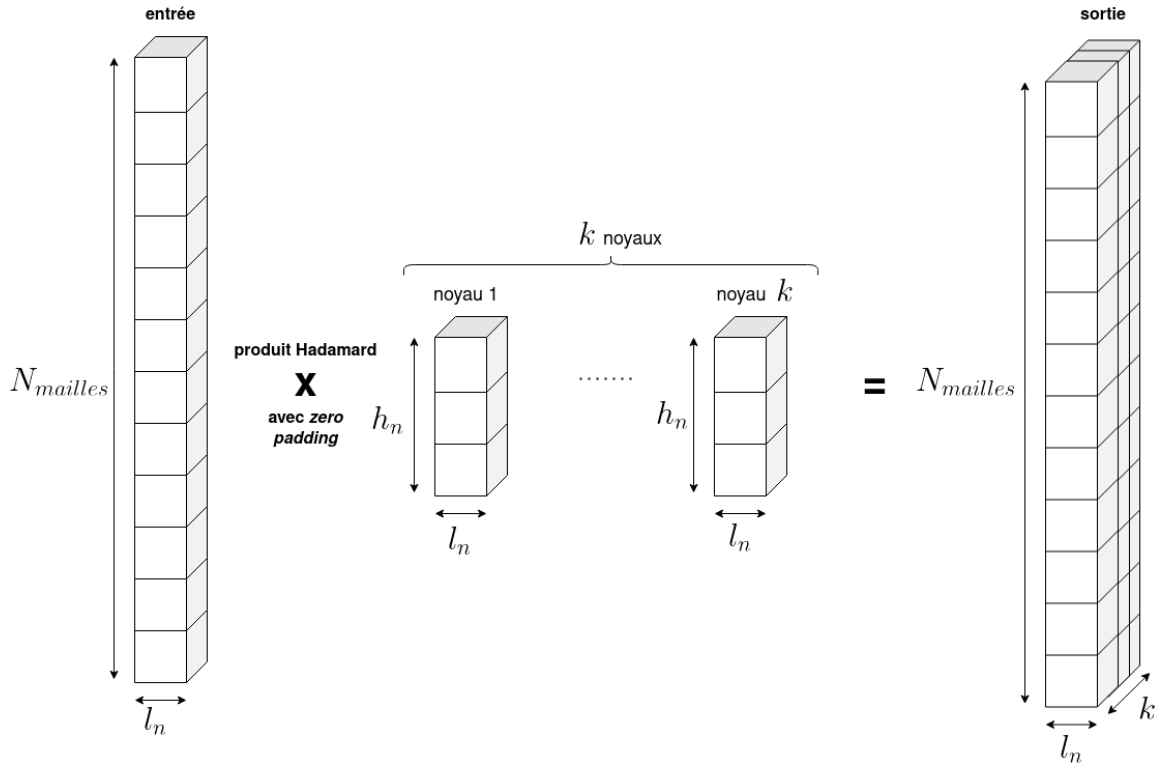


FIGURE 7.31 – Schéma d'une convolution sur une donnée 1D

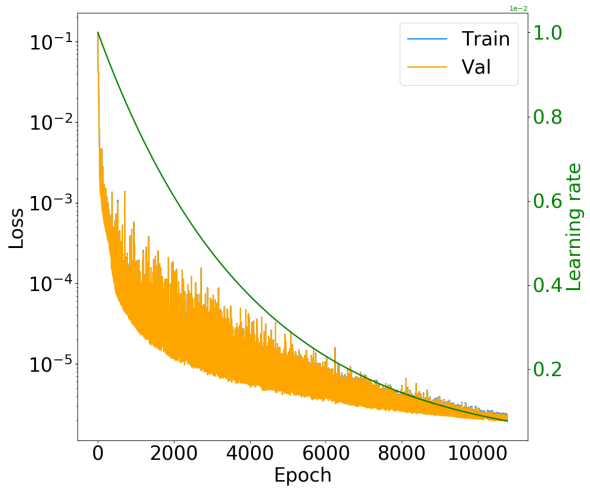
Enfin, le temps de calcul de la source non locale a été réduit d'un facteur 433 en moyenne à chaque pas de temps, passant de 282 ms avec le module SNB à 0,65 ms avec le module RNA. Il convient de noter qu'ici, seul 0,5 % du temps de calcul du module RNA est consacré au lissage de la prédiction.

7.4 Résultats préliminaires avec des réseaux convolutifs sur la base 1D

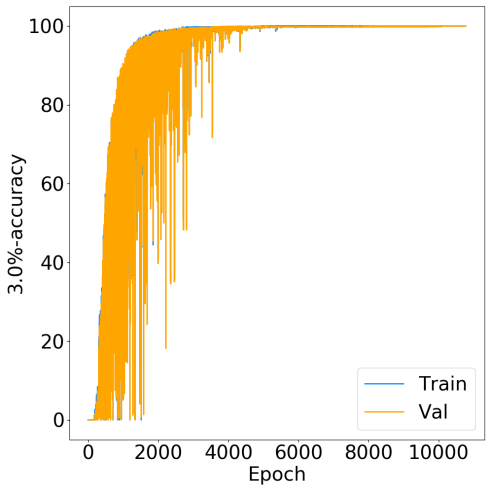
Au lieu de réseaux denses, on utilise maintenant des réseaux convolutifs, introduits au Chapitre 1. Ceux-ci sont largement utilisés en reconnaissance d'image, et ont l'avantage de garder la notion d'espace 2D des données. Ils pourraient également être entraînés sur des données avec des maillages différents, dans la mesure où les phénomènes physiques ont toujours lieu aux mêmes échelles (millimètre). On les teste ici sur la base CHIC1D.

Nous avons donc développé un module de réseaux convolutifs où se trouvent une fonction de convolution ainsi qu'une fonction de *zero padding*. La Figure 7.31 montre un schéma d'une convolution sur une donnée 1D telle que définie dans le module RNA convolutif.

Une fonctionnalité des réseaux convolutifs n'a pas été introduite au Chapitre 1 : le *pooling*. En effet, une couche convolutive est en réalité organisée en trois étapes : une première étape où une convolution est appliquée à l'entrée de la couche, une deuxième étape où la sortie de la convolution passe par une fonction d'activation, et enfin, une troisième étape de *pooling* où l'image est rétrécie en prenant une certaine moyenne de portions de l'image dont la taille est définie par l'utilisateur. Il existe de nombreuses fonctions de *pooling*, comme par exemple le *maxpooling*, où l'image en sortie ne contient que les maximums de portions de l'image avant *pooling*. S'agissant ici d'un problème

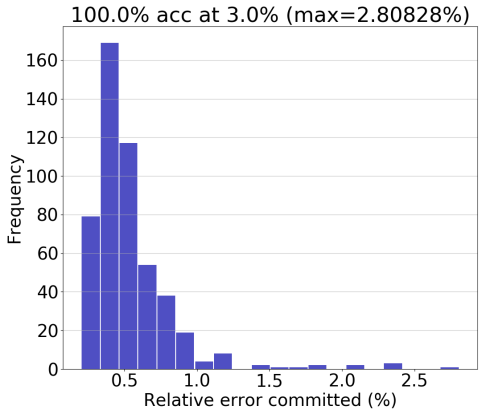


(a) Évolution de la *loss*

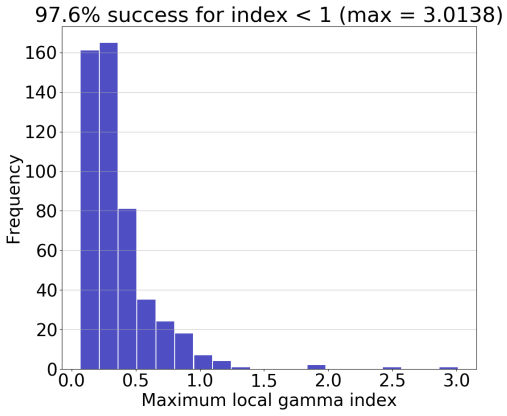


(b) Évolution du taux de réussite à 3%

FIGURE 7.32 – Résultats d'entraînement sur la base CHIC1D : réseau convolutif de 6 couches cachées avec k=16 noyaux de convolution de dimensions h_n=5 et l_n=1, et activation sigmoïd, *batch* de taille 450 et α_0=0.01



(a) Histogramme des erreurs relatives



(b) Histogramme des gamma index

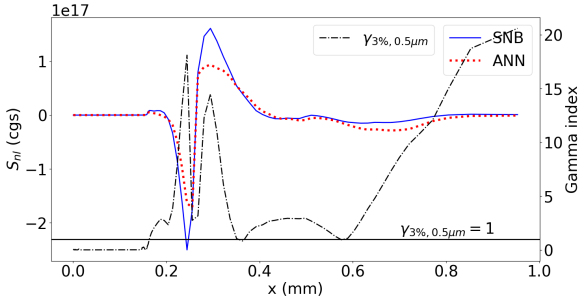
FIGURE 7.33 – Performances sur l'ensemble test de la base CHIC1D : réseau convolutif de 6 couches cachées avec k=16 noyaux de convolution de dimensions h_n=5 et l_n=1, et activation sigmoïd, *batch* de taille 450 et α_0=0.01

de régression où la prédiction du réseau doit avoir la même dimension que la donnée d'entrée, nous avons choisi de ne pas utiliser de *pooling* dans les réseaux que nous avons testés. Pour conserver la taille de l'entrée, on doit également définir le pas de translation transversale des noyaux de convolution comme étant $p_h = 1$ (il n'y a pas de translation longitudinale puisque les données sont 1D). D'après les définitions vues au Chapitre 1, on calcule également le *padding* nécessaire pour conserver la dimension des données, soit

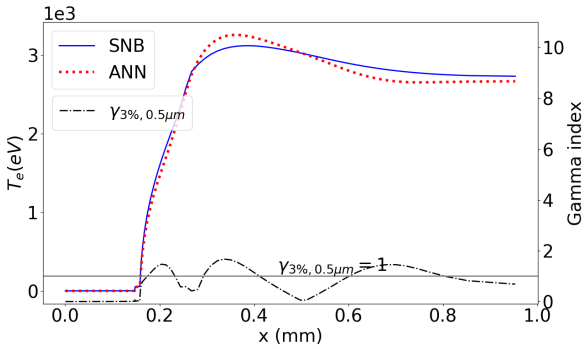
$$h_{pad} = max((N_{mailles} - 1)p_h + h_n - N_{mailles}, 0).$$

Les résultats d'entraînement du meilleur réseau convolutif trouvé sont présentés Figures 7.32 et 7.33. Ce réseau convolutif possède 6 couches cachées de k=16 noyaux de convolution de dimensions h_n=5 et l_n=1, et une fonction d'activation sigmoïd.

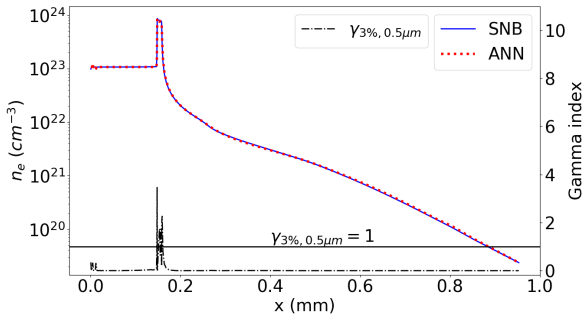
Comme chez les réseaux denses vus précédemment, les évolutions de la *loss* et du taux



(a) Source non locale

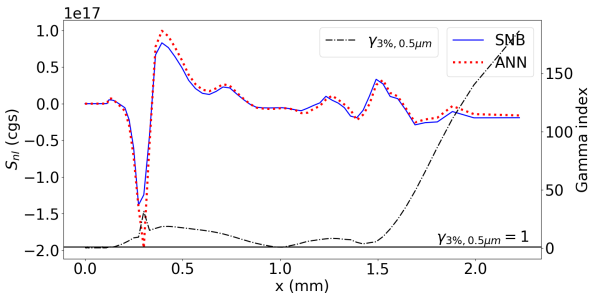


(b) Température

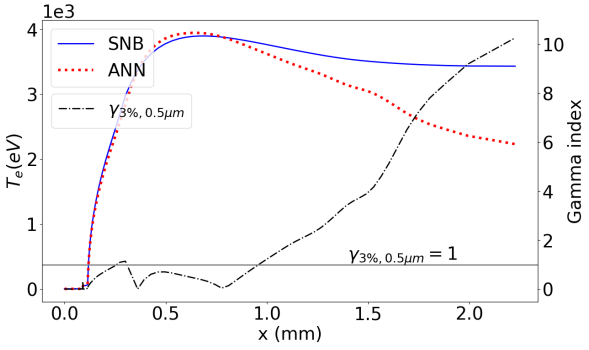


(c) Densité

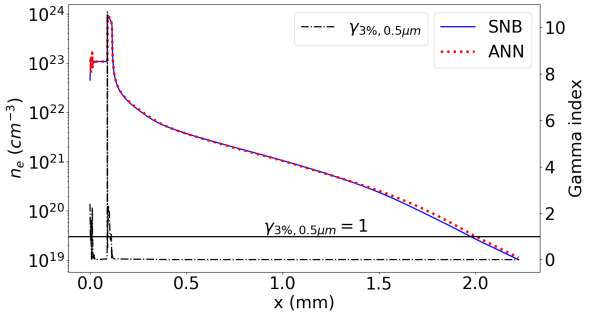
FIGURE 7.34 – Profils hydrodynamiques sans lissage à $t = 500$ps (RNA convolutif)



(a) Source non locale



(b) Température

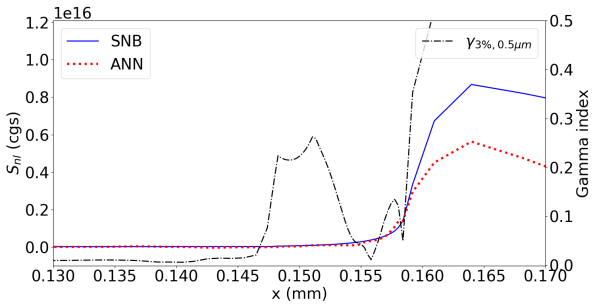


(c) Densité

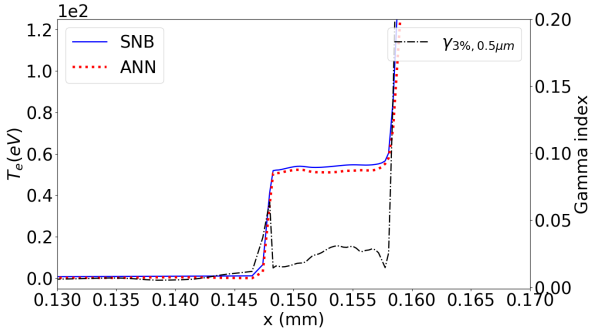
FIGURE 7.35 – Profils hydrodynamiques sans lissage à $t = 1000$ps (RNA convolutif)

de réussite à 3% (respectivement Figures 7.32a et 7.32b) ont des tendances similaires sur les ensembles d'entraînement et de validation, et le taux de réussite termine même sur un plateau à 100%. Les performances sur l'ensemble test sont également excellentes : le taux de réussite à 3% est de 100% avec une erreur relative maximum de 2.8% (Figure 7.33a) ; le taux de réussite pour un gamma index inférieur à 1, lui, est de 97.6%. En somme, ces résultats sont mêmes meilleurs que ceux du meilleur réseau dense sur la base CHIC1D, et on s'attend donc à une meilleure performance une fois couplé au code CHIC.

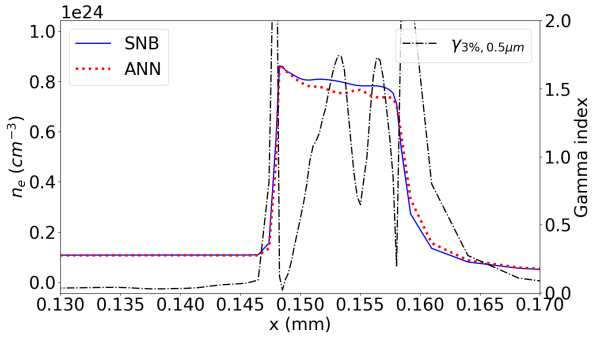
Les résultats d'une simulation RNA convolutif sans lissage sont montrés Figures 7.34 à 7.37. On constate que les résultats du réseau convolutif sont très différents de ceux du réseau dense. Tout d'abord, le profil de source non local (Figures 7.34a et 7.35a) est très différent de ceux normalement observés : les pics des gradients ne sont plus aussi prononcés, et sur la portion droite du domaine, la source n'est plus quasi constante comme nous avons pu le voir jusqu'à présent (Figures 7.14a, 7.18a et 7.22a). Cette pertubation engendre un fort décollement du profil en température (Figures 7.34b et 7.35b) sur cette même portion du domaine. Quant à la densité (Figures 7.34c et 7.35c), on observe égale-



(a) Source non locale

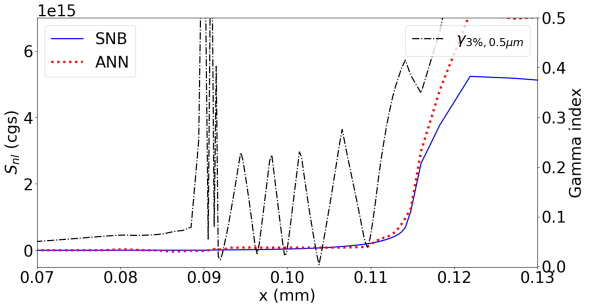


(b) Température

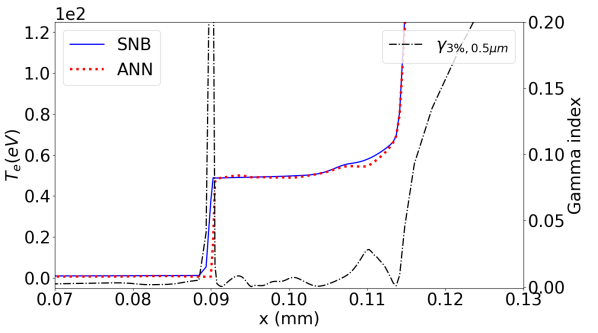


(c) Densité

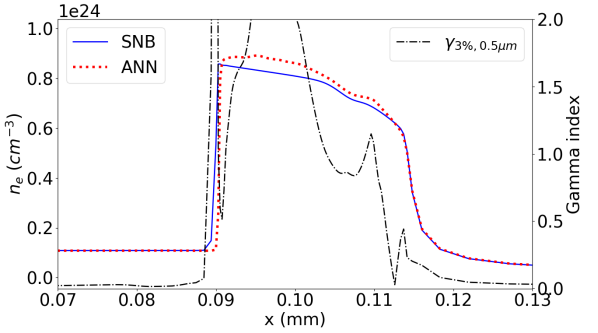
FIGURE 7.36 – Zoom sur les profils hydrodynamiques sans lissage à $t = 500$ps (RNA convolutif)



(a) Source non locale



(b) Température



(c) Densité

FIGURE 7.37 – Zoom sur les profils hydrodynamiques sans lissage à $t = 1000$ps (RNA convolutif)

ment un très léger décollement sur la portion droite du domaine, et quelques oscillations sur le bord gauche du domaine.

Les zooms sur la portion du domaine où se trouvent le pied d'onde de température et la hausse en densité (Figures 7.36 et 7.37) montrent que les profils du réseau convolutif sont plus oscillatoires qu'avec les réseaux denses, précisément là où se trouve le pied d'onde en température (Figures 7.36a et 7.37a), et qu'en conséquence, la simulation RNA convolutif est également moins proche de la simulation SNB sur les profils en température (Figures 7.36b et 7.37b) et en densité (Figures 7.36c et 7.37c).

Ces résultats peuvent s'expliquer par le fait que la convolution a un effet inhérent de lissage. Là où avec les réseaux denses on appliquait un lissage sur la portion du profil de source non locale qui en avait besoin, le lissage interne du réseau convolutif est appliqué sur tout le profil de source non locale. C'est ce qui explique que la source non locale soit aussi différente que d'ordinaire : les gradients ont été lissés et le reste du profil moyenné par la convolution, qui est interprêté comme un phénomène physique par le code

	Passe-avant	Lissage	SNB	Facteur de gain
1D	1.595 ms	0.005 ms	375 ms	233
2D	0.643 ms	0.007 ms	282 ms	433

Tableau 7.5 – Temps de calcul moyen par pas de temps de la passe-avant, du lissage et du modèle SNB , et facteur de gain total

hydrodynamique, d'où les profils en température et en densité altérés.

Il faudrait faire une étude plus poussée sur le sujet afin de s'assurer s'il s'agit d'un défaut d'utilisation des réseaux convolutifs, ou bien si les réseaux convolutifs ne sont simplement pas adaptés à nos données. Si finalement on trouve un réseau convolutif avec une performance de couplage similaire aux réseaux denses présentés ici, alors on pourra évaluer le temps de calcul de la passe-avant de ce réseau lorsqu'il est couplé au code CHIC.

7.5 Conclusion et perspectives

Ce chapitre a présenté une étude sur la capacité des réseaux de neurones artificiels à être entraînés afin de reproduire la physique complexe du transport de chaleur non local, et d'être ensuite couplés à des simulations hydrodynamiques. Les premières expériences réalisées sur des cas simplifiés d'ablation de matière par laser montrent qu'en 1D, un réseau est capable de faire des prédictions avec une erreur relative moyenne inférieure à 3%. Cependant, les résultats ont aussi montré qu'il était nécessaire de lisser la prédiction du réseau, afin d'éviter d'augmenter la propagation des oscillations au reste de la simulation hydrodynamique. La même performance globale a été observée en 2D, où le réseau a fait une erreur relative moyenne inférieure à 3 % après l'entraînement. Le tableau 7.5 résume les temps de calcul observés en 1D et 2D de la passe directe, du processus de lissage, du module SNB, ainsi que du facteur de gain total. Au final, le gain moyen est d'un facteur 233 en 1D, et d'un facteur 433 en 2D, lissage inclus.

Dans cette étude, l'entraînement des RNA a toujours été effectué en dehors du code hydrodynamique, à l'aide d'un script Python, afin d'utiliser la bibliothèque optimisée Tensorflow [96]. Un module Fortran reproduisant la passe-avant d'un réseau dense a été écrit et introduit dans le code hydrodynamique CHIC. Comme cette passe-avant équivaut à une simple multiplication matrice-vecteur effectuée couche après couche, l'utilisation de Fortran, un langage de codage matriciel, a simplifié la reproduction du processus.

Les résultats présentés dans ce chapitre, et publiés dans l'article [136], sont une preuve d'un concept général : ils sont une première étape vers la création d'un RNA qui pourra réduire le temps de modélisation multi-échelle de cibles pour la fusion par confinement inertiel. La conception de cibles pour la FCI implique de nombreuses simulations très similaires les unes aux autres, avec une variation de quelques paramètres, tels que la distribution de l'intensité du laser, la forme temporelle de l'impulsion, la composition de la cible, le maillage, la durée de simulation, etc. Le domaine de variation de ces paramètres est limité par la validité du modèle et les conditions d'optimisation numérique, ce qui en fait un espace fini. Sachant cela, et compte tenu du fait que le principal problème de la modélisation multi-échelle est le temps de calcul, les RNA pourraient s'avérer être

des outils puissants pour réduire le temps de calcul, tout en préservant une bonne précision. Dans l'espace de paramètres qui nous intéresse ici, on pourrait produire une base de données plus étendue et entraîner un RNA avec celle-ci. Qui plus est, cette base de données ne nécessitera pas la génération de chaque cas dans l'espace fini considéré : tant que la variation entre les cas est continue, il suffit de définir un espace de paramètres suffisamment large pour inclure tous les cas d'intérêt possibles, et en générer seulement une partie à utiliser pour l'entraînement de RNA. Par exemple, les RNA couplés au code hydrodynamique dans cette étude sont capables de faire des prédictions précises à des pas de temps sur lesquels ils n'ont jamais été entraînés, et en 1D, nous avons vu que le RNA était même capable de faire des prédictions justes pour une intensité laser qui ne faisait pas partie des intensités des données d'entraînement, mais qui faisait partie de la gamme d'intensités d'entraînement.

Un RNA formé sur une telle base de données serait donc un outil utile pour obtenir des résultats rapides pour des cas connus, mais aussi pour interpoler des cas inconnus dans l'espace d'intérêt défini. Cela donnerait la possibilité de récupérer rapidement des cas connus, sans avoir à stocker une très grande quantité de données : une fois le RNA entraîné, il n'est pas nécessaire de garder en mémoire tous les cas d'entraînement, et la taille du RNA, en termes de stockage, est nettement inférieur à celui de la base de données. Cependant, il faut être conscient du fait qu'un RNA ne peut pas faire de prédictions en dehors de l'espace d'intérêt défini par la base de données : ce n'est pas un outil d'extrapolation, mais plutôt un outil d'interpolation, et qui plus est, très efficace. Le point clé est alors le nombre de cas d'apprentissage nécessaires pour fournir des interpolations suffisamment précises. Limiter l'utilisation de RNA à un espace paramétrique fini est finalement un moyen d'assurer leur validité et d'en faire des outils de modélisation fiables.

Il existe cependant une limite à l'utilisation des réseaux de neurones denses : les dimensions de l'entrée et de la sortie doivent toujours rester fixes. Ceci restreint ainsi l'utilisation de tels réseaux à des maillages avec un nombre de points fixe, qu'ils soient lagrangiens ou eulériens. Il ne serait pas efficace de former un réseau pour chaque nombre de points de maillage, car cela signifierait qu'une base de données doit être générée pour chacun séparément, et que de nombreux réseaux doivent être formés afin que l'utilisateur puisse choisir le maillage dont il a besoin. Une façon de contourner ce problème pourrait être de projeter les profils sur une grille de référence fixe, qui aurait été au préalable utilisée pour tous les profils de la base de données lors de l'entraînement du réseau. Cependant, il faut penser au fait que ce processus devra également être inversé pour la prédiction du réseau avant qu'elle ne soit utilisée par le reste du code hydrodynamique. L'ajout d'une projection de l'entrée, puis d'une projection inverse de la prédiciton, pourrait ne pas améliorer le temps d'exécution global, et même le rendre plus long. Un autre type de réseaux pourrait être utilisé à la place : des réseaux entièrement convolutifs, qui préservent la forme multidimensionnelle de l'entrée à travers les couches, et dont les noyaux convolutifs ne dépendent pas de la taille de l'entrée, ce qui implique que différentes tailles de maillage pourraient être envisagées.

Les résultats préliminaires de l'utilisation de tels réseaux sur la base de données 1D ont montré que contrairement aux réseaux denses, la convolution a un effet de lissage sur la prédiction. Cependant, ce lissage est présent sur tout le profil et le modifie à tel point que la simulation en est impactée de manière néfaste. Les réseaux convolutifs testés étaient bridés de telle manière à garder la dimension du maillage intecte tout au long des couches : une étude plus poussée mérite d'être menée afin de voir s'il est possible de maîtriser le lissage intrinsèque à la convolution. Et si un réseau convolutif avec une performance de couplage similaire aux réseaux denses présentés ici est identifié, alors on

pourra également s'intéresser au gain en temps de calcul de ce réseau. Dans l'idéal, pour chaque base de données, le réseau convolutif comportera également moins de paramètres que le réseau dense (car par essence, la convolution fait que les poids sont partagés par tous les neurones et non propres à chacun d'entre eux), donc nous aurons potentiellement un gain en termes d'espace mémoire.

Le chapitre suivant présente une étude sur l'utilisation de réseaux de neurones pour une autre application : le calcul de dose en radiothérapie.

Chapitre 8

Couplage de réseaux de neurones à un code aux moments pour la radiothérapie

Nous proposons propose maintenant une utilisation des RNA pour une autre application : le calcul de dose en radiothérapie. Le nombre de publications présentant des solution de *deep learning* pour la radiothérapie est en forte augmentation, avec des applications principalement liées à l'imagerie, telles que la segmentation d'images (par exemple pour la détection de tumeur [140] et délimitation des organes à risques [141]), la classification d'images (comme la détection de tumeur [142], ou la détermination du type de tumeur [143]), la génération d'images de CT-scan synthétiques à partir d'IRM [144], la réduction du bruit engendré par des objets métalliques [145], ou encore l'amélioration de la résolution d'images [146]. Depuis plusieurs années, les réseaux de neurones artificiels sont également utilisés pour le calcul de dose [147–153]. On propose ici une nouvelle manière d'utiliser des RNA pour le calcul de dose : en les couplant à un code aux moments.

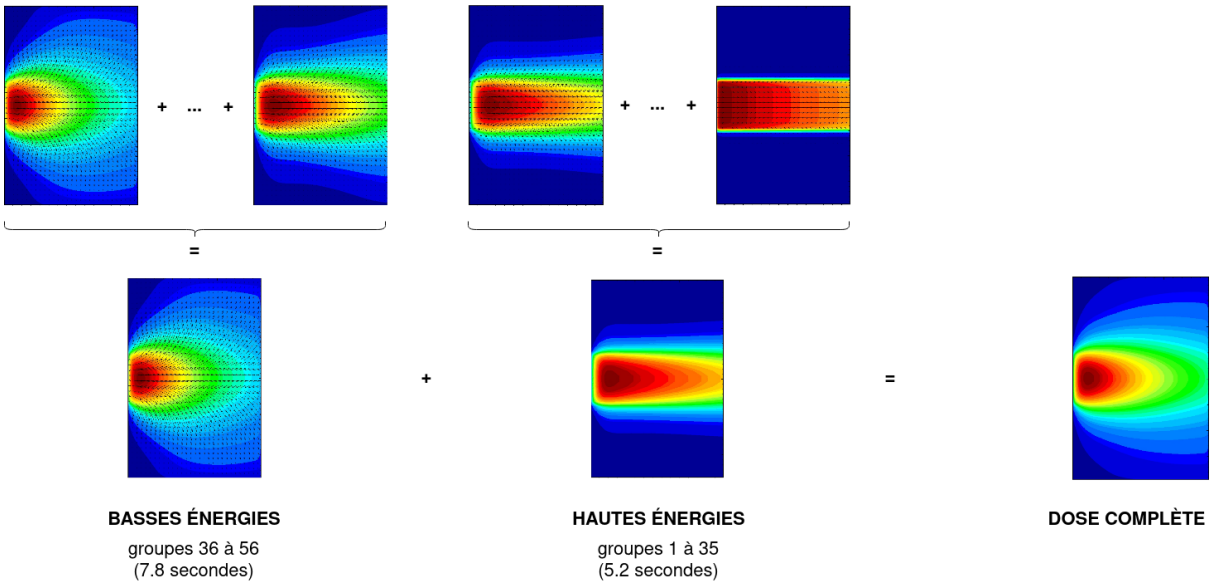
La première partie de ce chapitre présente la stratégie de couplage et la génération des données d'apprentissage. Ensuite, les résultats d'apprentissage de réseaux denses sont exposés dans la deuxième partie. La troisième partie montre les résultats du couplage d'un RNA au code KIDS [38, 39]. Enfin, ce chapitre termine par une conclusion où on aborde également les perpectives de cette étude.

Le travail présenté ici a été mené en collaboration avec Mathilde Haudiquet, étudiante en deuxième année du Master Light Sciences and Technologies de l'Université de Bordeaux, durant l'encadrement de son stage de fin d'études de mars à juin 2022.

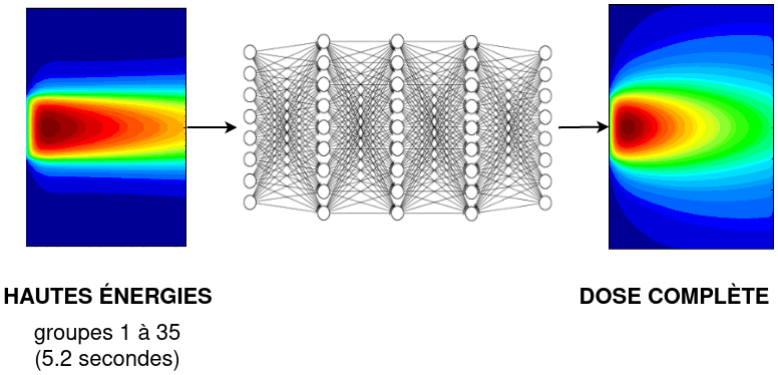
8.1 Stratégie de couplage et génération d'une base de données

Le code KIDS résout le système d'équations aux moments (4.2.4) fermé par l'équation (4.2.5), comme présenté au Chapitre 4. Nous avions également mentionné que ces équations sont résolues sur un maillage en espace 2D, ainsi qu'un maillage en énergie 1D. Les contributions de chaque niveau (ou groupe) d'énergie sont en réalité additives, si bien que l'on peut exprimer la dose complète comme la somme des doses de chaque groupe en énergie (Figure 8.1a).

Une rapide analyse montre cependant que le calcul est plus rapide pour les groupes de haute énergie. Par exemple, dans le cas montré Figure 8.1a, le calcul sur les 35 groupes



(a) Méthode classique



(b) Couplage avec un RNA

FIGURE 8.1 – Calcul de dose avec le code KIDS

de haute énergie prends 5.2 secondes, contre 7.8 secondes pour les 21 groupes de basse énergie. C'est pour cela que l'on propose de remplacer le calcul des doses des énergies faibles par un RNA, comme illustré Figure 8.1b : la dose imputable aux hautes énergies est calculée par le code KIDS, puis passée par un RNA qui calcule la dose complète associée.

On construit ainsi une base de données pour des cas simplifiés à l'aide du code KIDS. Le maillage est fixé à 40 mailles dans la direction de propagation du faisceau (axe x) et 100 mailles dans la direction transversale (axe y). Le nombre de groupes en énergies est également fixé à 56, dont 35 groupes de haute énergie dont nous avons cumulé les contributions pour former les données d'entrée. Les doses complètes font, elles, partie des données de sortie. Le milieu considéré est homogène et constitué d'eau. Le faisceau laser est un faisceau de photons de forme carrée, avec des côtés de 1 cm, et appliqué sur le bord gauche du domaine. On fait varier l'intensité du faisceau initial entre 0.5 et 20 MeV avec un pas de 0.01, et pour chaque intensité, une simulation du code KIDS produit une donnée d'entrée et une donnée cible. En tout, 1 496 couples entrée-cible ont été formés, dont 1200 pour l'ensemble d'entraînement, 150 pour l'ensemble de validation et 146 pour l'ensemble test.

8.2 Entraînement de réseaux denses

Pour cette étude, nous avons fait varier le nombre de couches cachées (1, 2, 3, 5 ou 10), ainsi que le nombre de neurones par couche cachée (10, 50, 100, 250, 500, 750 ou 1000). Nous avons également fait varier le taux d'apprentissage initial α_0 (0.01, 0.001 ou 0.0001) et nous avons fixé la taille de *batch* à 30. Un récapitulatif des résultats pour $\alpha_0 = 0.001$ est présenté dans les Tableaux 8.1 et 8.2. Les résultats pour $\alpha_0 = 0.01$ et $\alpha_0 = 0.0001$ sont disponibles dans l'annexe D. Globalement, ce sont les réseaux pour lesquels α_0 vaut 0.001 qui ont les meilleures performances, et ceux pour lesquels α_0 vaut 0.0001 qui ont les pires.

Le Tableau 8.1 présente les valeurs du taux de réussite à 1% à la fin de l'apprentissage, sur les ensembles d'entraînement (valeurs bleues) et de validation (valeurs orange). De manière générale, les résultats sont très bons, voir excellents, même pour les plus petits réseaux. Une majorité des réseaux atteint 100% de réussite sur les deux ensembles, notamment le réseau de 1 couche à 10 neurones. Les mêmes observations sont valables pour les performances sur l'ensemble test, résumées dans le Tableau 8.2, où les valeurs bleues représentent le taux de réussite à 1% d'erreur et les valeurs orange représentent le taux de réussite pour un gamma index inférieur à 1.

On ne s'attendait pas à de si bonnes performances pour de très petits réseaux. Cependant, elles peuvent s'expliquer par deux choses :

- d'abord, le fait que les données d'entrée et les données cibles sont, somme toute, très proches : quand bien même les valeurs sont différentes, les formes des profils sont identiques, c-à-d de hautes valeurs au centre du domaine dans le sens longitudinal, avec une diminution progressive de gauche à droite dans le sens longitudinal, et du centre vers les bords dans le sens transversal ;
- enfin, le fait que les cas considérés sont très simples : le milieu est homogène et le faisceau se propage toujours la même direction, à savoir dans le sens longitudinal, du bord gauche au bord droit du domaine.

Pour les réseaux présentés dans les Tableaux 8.1 et 8.2, les tendances des courbes d'apprentissage sont également semblables d'un réseau à l'autre. Un exemple d'entraînement est présenté Figures 8.2 à 8.4 : le réseau considéré est un réseau à 1 couche cachée de 10 neurones avec activation sigmoïd, $\alpha_0 = 0.001$ et un *batch* de taille 30.

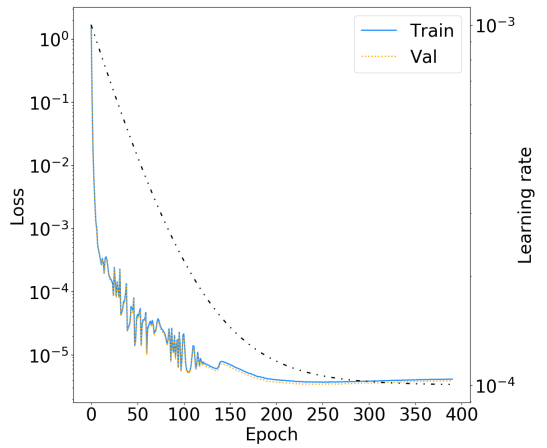
Tout d'abord, les évolutions de la *loss* (Figure 8.2a) et du taux de réussite à 1% (Figure 8.2b) ont les mêmes tendances sur les ensembles d'entraînement et de validation, et terminent sur des plateaux, notamment un plateau à 100% sur les deux ensembles pour le taux de réussite. Ensuite, les performances sur l'ensemble test sont également excellentes, avec 100% de réussite à 1% (Figure 8.3a) et 100% de réussite pour une gamma index inférieur 1 avec un critère de 1% d'erreur sur un rayon de 1mm (Figure 8.3b). Enfin, un exemple de profil est présenté Figure 8.4 : on ne distingue pas de différence entre la prédiction du réseau (ANN) et la cible (M1) (Figure 8.4a), et le profil de gamma index (Figure 8.4b) montre qu'en effet, le critère est respecté en tout point du maillage.

	nb couches cachées									
nb neurones/couche	1		2		3		5		10	
10	100,00 %	100,00 %	98,25 %	97,33 %	94,42 %	92,67 %	81,42 %	80,67 %	71,08 %	68,00 %
50	100,00 %	100,00 %	99,67 %	99,33 %	97,75 %	96,67 %	97,92 %	97,33 %	97,92 %	97,33 %
100	99,92 %	100,00 %	100,00 %	100,00 %	100,00 %	100,00 %	99,75 %	100,00 %	99,92 %	100,00 %
250	99,17 %	98,67 %	100,00 %	100,00 %	100,00 %	100,00 %	100,00 %	100,00 %	98,83 %	98,67 %
500	98,33 %	97,33 %	100,00 %	100,00 %	100,00 %	100,00 %	99,83 %	100,00 %	100,00 %	100,00 %
750	99,92 %	100,00 %	100,00 %	100,00 %	99,33 %	98,67 %	87,08 %	89,33 %	99,50 %	99,33 %
1000	99,75 %	99,33 %	100,00 %	100,00 %	98,42 %	98,00 %	93,42 %	90,67 %	99,83 %	100,00 %

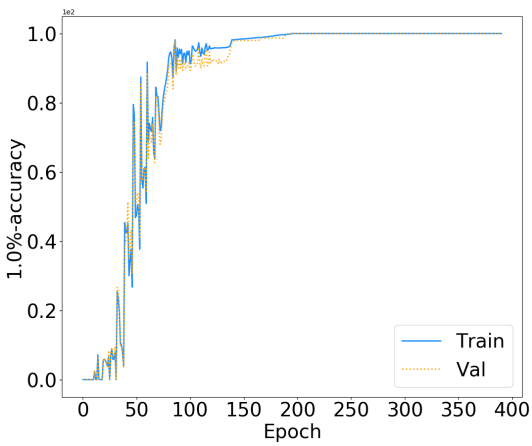
Tableau 8.1 – Taux de réussite à 1% sur les ensembles d'entraînement (valeurs bleues) et de validation (valeurs orange)

	nb couches cachées									
nb neurones/couche	1		2		3		5		10	
10	100,00 %	100,00 %	99,32 %	99,32 %	96,58 %	98,63 %	81,51 %	92,47 %	63,70 %	71,92 %
50	100,00 %	100,00 %	100,00 %	100,00 %	98,63 %	99,32 %	99,32 %	99,32 %	99,32 %	99,32 %
100	100,00 %	100,00 %	100,00 %	100,00 %	100,00 %	100,00 %	100,00 %	100,00 %	100,00 %	100,00 %
250	99,32 %	100,00 %	100,00 %	100,00 %	100,00 %	100,00 %	100,00 %	100,00 %	99,32 %	99,32 %
500	99,32 %	100,00 %	100,00 %	100,00 %	100,00 %	100,00 %	100,00 %	100,00 %	100,00 %	100,00 %
750	100,00 %	100,00 %	100,00 %	100,00 %	99,32 %	100,00 %	85,62 %	94,52 %	100,00 %	100,00 %
1000	100,00 %	100,00 %	100,00 %	100,00 %	99,32 %	100,00 %	94,52 %	96,58 %	100,00 %	100,00 %

Tableau 8.2 – Taux de réussite à 1% d'erreur (valeurs bleues) et taux de réussite pour un gamma index < 1 (valeurs orange) sur l'ensemble test

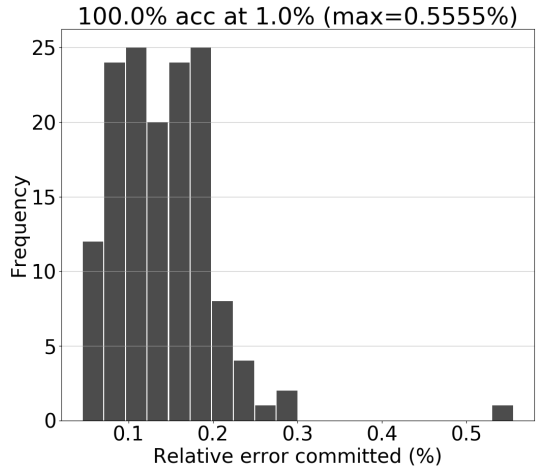


(a) Évolution de la *loss*

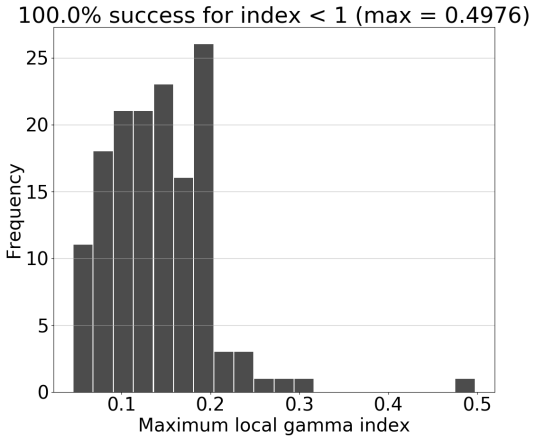


(b) Évolution du taux de réussite à 3%

FIGURE 8.2 – Résultats d'entraînement sur la base CHIC2D : réseau de 4 couches cachées à 150 neurones avec activation sigmoïd, *batch* de taille 60 et α_0=0.01

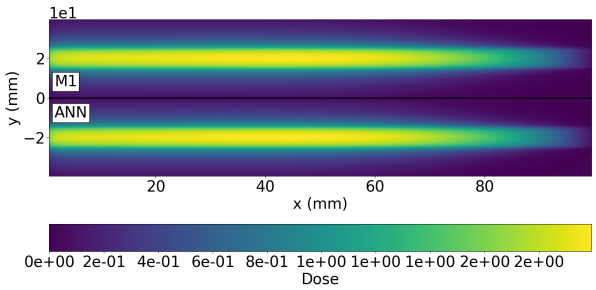


(a) Histogramme des erreurs relatives

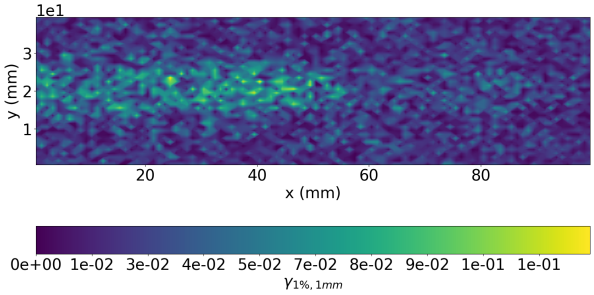


(b) Histogramme des gamma index

FIGURE 8.3 – Performances sur l'ensemble test de la base CHIC2D : réseau de 4 couches cachées à 150 neurones avec activation sigmoïd, *batch* de taille 60 et α_0=0.01

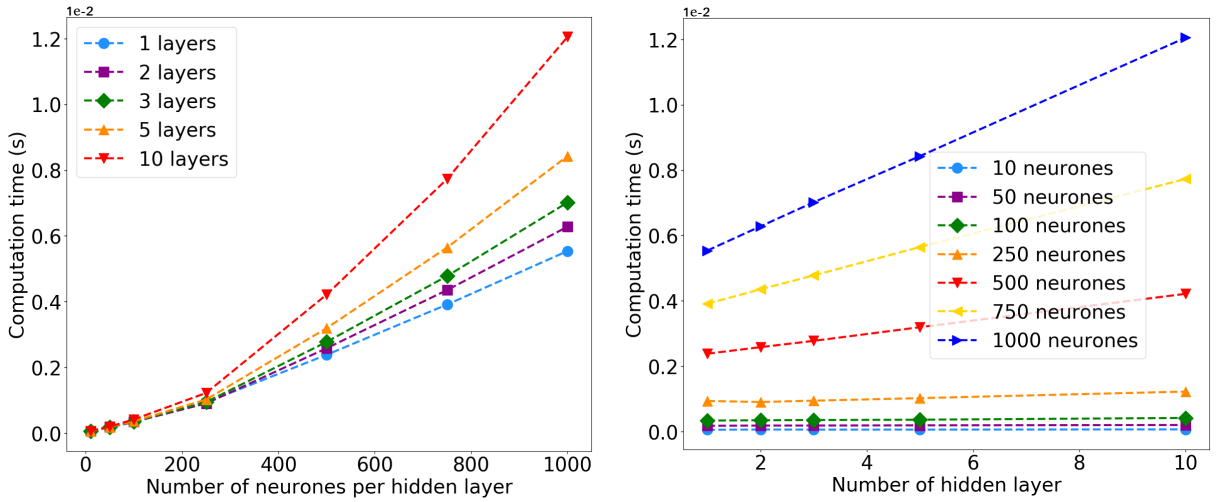


(a) Cible (M1) vs prédiction (ANN)



(b) Carte 2D du gamma index

FIGURE 8.4 – Exemple de profil : réseau de 4 couches cachées à 150 neurones avec activation sigmoïd, *batch* de taille 60 et α_0=0.01



(a) Temps d'exécution en fonction du nombre de neurones par couche cachée

(b) Temps d'exécution en fonction du nombre de couches cachées

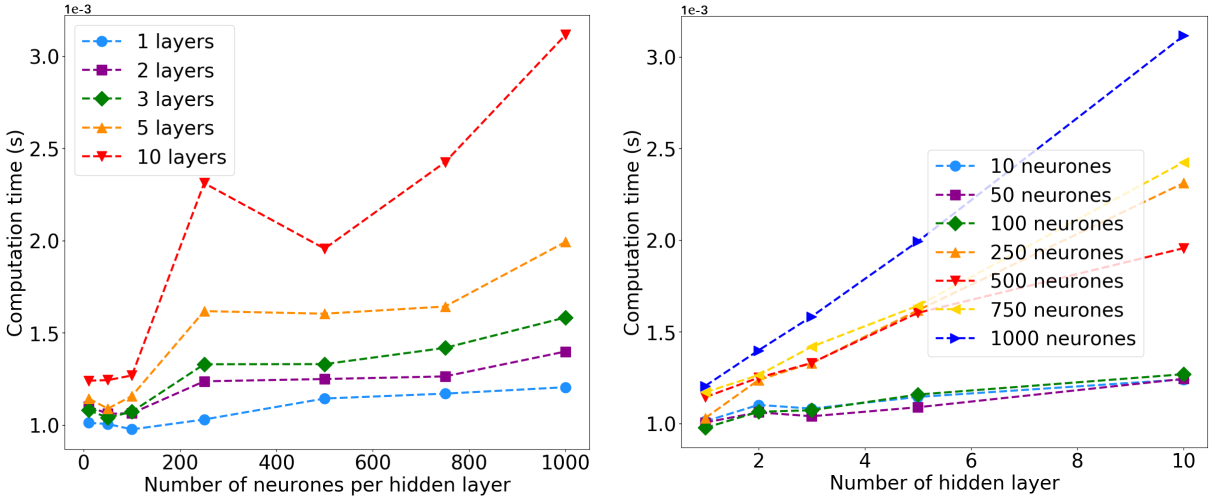
FIGURE 8.5 – Analyse du temps d'exécution du module Fortran

8.3 Couplage de réseaux denses : analyse du temps d'exécution

On rappelle que l'objectif principal est de réduire le temps de calcul de dose, point clé pour une utilisation clinique. C'est pour cela que l'on cherche des architectures de réseaux les plus rapides possible. Dans un premier temps, on analyse donc le temps d'exécution de la passe avant de notre module Fortran sur une donnée test pour chaque réseau présenté aux Tableaux 8.1 et 8.2.

La Figure 8.5a montre le temps d'exécution des différentes architectures en fonction du nombre de neurones par couche cachée. Pour obtenir ces courbes, nous avons effectué 100 lancements pour chaque architecture et utilisé la moyenne du temps d'exécution. Ce graphe illustre tout d'abord la proportionnalité entre le nombre de neurones par couche cachée et le temps d'exécution. Pour un réseau à 1 couche cachée, le temps d'exécution augmente linéairement avec l'augmentation de la taille de la couche cachée. Mais plus on ajoute des couches cachées, plus l'augmentation du temps d'exécution devient non linéaire, voire exponentielle. On note cependant que pour de très petites couches (de 10, 50 et 100 neurones), on ne distingue pas de différence notable entre les architectures avec plus ou moins de couches cachées. Ces architectures commencent à se distinguer seulement à partir de 200 neurones par couche cachée, jusqu'à atteindre une différence notable entre les architectures à 1 et 10 couces cachées. En effet, à partir de 750 neurones par couche cachée, l'architecture à 10 couches cachées a un temps d'exécution environ 2.5 fois plus important que l'architecture à 1 couche cachée, et cet écart se creuse d'autant plus en augmentant la taille des couches cachées à 1000 neurones.

Sur le graphe de la Figure 8.5b, nous avons cette fois-ci tracé le temps d'exécution du module Fortran en fonction du nombre de couches cachées. On note tout d'abord que pour de petites couches (entre 10 et 250 neurones), le temps d'exécution est quasiment constant, tandis que pour les architectures avec de plus grandes couches (500 neurones et plus), le temps d'exécution augmente linéairement avec le nombre de couches cachées. Enfin, on observe que globalement, plus une architecture a des grandes couches, plus le



(a) Temps d'exécution en fonction du nombre de neurones par couche cachée

(b) Temps d'exécution en fonction du nombre de couches cachées

FIGURE 8.6 – Analyse du temps d'exécution de la librairie Tensorflow

temps d'exécution augmente. Finalement, l'augmentation de la taille de la matrice des poids se répercute sur le temps d'exécution, ce qui est également le cas pour la librairie Tensorflow, mais dans une moindre mesure.

Les Figures 8.6a et 8.6b présentent la même analyse du temps d'exécution que pour le module Fortran, mais cette fois-ci pour la librairie Tensorflow. Tout d'abord, on constate que l'ordre de grandeur des temps d'exécution est de 10^{-3} secondes pour la librairie Tensorflow, contre 10^{-2} secondes pour le module Fortran. Ensuite, on observe les mêmes tendances générales, à savoir une croissance du temps d'exécussion en augmentant le nombre de couches cachées et leurs tailles. Cependant on note quelques différences avec le module Fortran. En effet, sur la Figure 8.6a, la croissance du temps d'exécution en fonction du nombre de neurones par couches cachées est plus progressive. Quant aux courbes de la Figure 8.6b, elles montrent que le temps d'exécution n'est plus quasi constant quel que soit le nombre de couches cachées lorsque leurs tailles sont petites.

Finalement, cette différence dans les performances de notre module Fortran est de la librairie Tensorflow est justifiée, puisque la librairie Tensorflow est optimisée et parallélisée. Dans notre cas précis, Tensorflow est uniquement parallélisé avec des threads sur CPU, mais Tensorflow peut également user d'un parallélisme sur GPU si des GPUs sont disponibles. C'est cette parallélisation qui lui permet d'avoir un temps d'exécution de l'ordre de 10^{-3} secondes, au lieu de 10^{-2} secondes pour le module Fortran, qui est non parallélisé. L'ordre de grandeur de ces valeurs ne justifie pas cependant d'utiliser Tensorflow pour le couplage, puisqu'il sera plus aisé de garder un code uniforme implémenté dans un seul et même langage. Cependant, le parallélisme et les algorithmes implémentés dans Tensorflow en font un outil idéal pour l'apprentissage de réseaux.

Néanmoins, on voit ici la limite du module Fortran que nous avons développé et la nécessité de soit le paralléliser, soit utiliser une des nouvelles librairies de *deep learning* pour Fortran mentionnée au Chapitre 2, afin de diminuer d'autant plus le temps d'exécution tout en maintenant l'intégralité du code dans un unique langage de programmation. Dans l'exemple de la Figure 8.1b, le temps de calcul de la dose complète est de 13 secondes. Si l'on prend un réseau à 1 couche cachée de 10 neurones, qui est le plus petit des meilleurs réseaux présentés dans les Tableaux 8.1 et 8.2, alors ce temps de calcul sera environ égal

à celui de la dose dûe aux hautes énergies, soit 5.2 secondes, car le temps d'exécution du module RNA pour ce réseau est de moins d'1 milliseconde.

8.4 Conclusion et perspectives

Ce chapitre présente les premiers résultats de l'utilisation de réseaux denses pour le calcul de dose en radiothérapie. Dans le chapitre précédent sur le calcul de flux non local pour la FCI, nous avions couplé un RNA à un code hydrodynamique de telle manière que le RNA était appelé à chaque pas de temps, et sa prédiction au temps t était utilisée par le code hydrodynamique pour calculer l'état du système physique, afin de passer au pas de temps suivant. Ici, le couplage se fait différemment. En effet, le code considéré est un code aux moments, où on a un maillage en énergie au lieu d'un maillage en temps, de telle sorte que la dose complète est la somme des doses à chaque pas en énergie. Une rapide analyse nous avait montré que les énergies pouvaient être séparées en deux sortes : les hautes énergies, dont les temps de calcul sont plus rapides, et les faibles énergies, dont les temps de calcul sont plus élevés. En conséquence, nous avons défini un couplage de telle sorte qu'un RNA soit utilisé pour calculer la dose complète à partir de la dose imputée aux hautes énergies.

Une base de données a été générée à partir de cas simples, où le milieu condéré est homogène (eau) et un faisceau de photons se propage dans la direction longitudinale à partir de la face gauche du domaine 2D. Le réglage des hyperparamètres a constitué la plus importante partie du travail. Une multitude de réseaux ont été entraînés, avec une variation de plusieurs hyperparamètres, afin de trouver la meilleure architecture possible. Les premiers résultats de cette étude paramétrique montrent finalement que de nombreuses architectures de réseaux sont capables d'approcher la dose complète avec moins d'1% d'erreur sur un rayon de 1 mm. En fait, cet ensemble de données est facile à prédire pour une majorité des réseaux testés car pour tous les couples entrée-cible, la donnée d'entrée est très similaire à la donnée cible. Finalement, même une très petite architecture, avec seulement 10 neurones sur 1 couche cachée, est capable de faire des prédictions avec moins d'1% d'erreur sur un rayon de 1 mm.

En plus d'être très précis, les architectures testées sont également rapides, avec des temps d'exécution avec le module Fortran de l'ordre de 10^{-2} secondes, comparé au temps de calcul du code KIDS sur les basses énergies et qui est de l'ordre de quelques secondes. Une analyse du temps d'exécusion du module Fortran a montré que ce dernier augmente presque linéairement avec le nombre de couches cachées et leurs tailles. Cependant, lorsqu'on compare ces performances à celles de la librairie Tensorflow, on constate que le manque de parallélisme du module Fortran le rend nettement moins efficace que la librairie Tensorflow, qui est optimisée et parallélisée.

Néanmoins, la facilité avec laquelle de nombreux réseaux ont obtenu de très bons résultats laisse penser qu'il serait possible d'augmenter la complexité de la base de données. La taille de la base de données peut être augmentée par différents moyens :

- en changeant la composition du milieu traversé par le faisceau (on pourrait alterner des couches d'eau, d'air ou d'os) ;
- en changeant la direction du faisceau (il pourrait venir n'importe quelle face du domaine, à n'importe quel angle) ;
- en changeant le nombre de faisceaux traversant le milieu.

On pourrait également modifier la délimitation entre les hautes et basses énergies, et former une base de données où la donnée d'entrée est la dose du niveau d'énergie le plus

élevé, ce qui diminuerait encore plus le temps de calcul total de la dose complète lorsqu'on couple le code KIDS à un RNA.

Un ensemble de données plus important pourrait nécessiter de plus grandes architectures que celles utilisées au cours de ce travail, impliquant d'autant plus la nécessité de soit optimiser le module Fortran déjà développé, soit utiliser une des librairies de *deep learning* citées au Chapitre 2.

Conclusion générale et perspectives

La problématique abordée dans ce travail de thèse est à la fois de déterminer si un réseau de neurones artificiels peut se substituer à un modèle physique, et comment mettre en place cette substitution. Plus particulièrement, on s'est intéressé à des modèles de transport de sources énergétiques en physique des plasmas, présentés aux Chapitre 3 et 4 de la deuxième partie du manuscrit.

Avant d'entrer dans le vif du sujet, une première étape a été d'aborder l'utilisation de réseaux de neurones artificiels sur des cas d'école, comme présenté au Chapitre 5 de ce manuscrit. Trois explorations distinctes ont été menées afin d'utiliser différentes architectures de réseaux pour résoudre différents types de problèmes. Tout d'abord, nous nous sommes intéressés à la classification binaire avec des réseaux à propagation avant. Nous avons pu voir à cette occasion qu'un neurone simple était suffisant pour résoudre un problème de classification binaire lorsque la frontière de décision séparant les données est linéaire. Dès lors qu'elle ne l'est plus, il était cependant nécessaire d'utiliser des réseaux avec au moins 1 couche cachée. Nous avons également vu l'intérêt de la descente de gradient avec moment, et comment elle permet de sortir des minimums locaux. Ensuite, on s'est intéressé aux *Physics Informed Neural Networks*, pour lesquels la fonction perte n'est pas une fonction classique mais une équation différentielle à résoudre. Nous avons choisi l'équation de la chaleur 1D, et les résultats ont montré la facilité avec laquelle ces réseaux peuvent résoudre une équation différentielle. Enfin, on s'est intéressé aux réseaux à propagation avant pour résoudre un problème de régression linéaire : la modélisation du flux non local de Lucciani-Mora-Virmont. Cette dernière étude a servi de preuve de concept pour les études présentées dans la quatrième et dernière partie du manuscrit, puisqu'elle a montré qu'un réseau était capable de modéliser un phénomène physique complexe avec une erreur inférieure à 3% en tout point de maillage.

Ces explorations ont également mené à une réflexion sur le lien théorique entre fonction perte et fonction d'activation, présentée au Chapitre 6. En effet, le choix de ces deux fonctions est toujours motivé par des lois empiriques, approuvées par la communauté du *deep learning*. On s'est en particulier intéressé au cas d'un neurone simple pour une classification binaire, dont on ne détermine pas la fonction d'activation, mais dont on définit l'entropie croisée comme fonction perte. L'apprentissage de ce neurone peut être défini comme un problème de minimisation avec contraintes d'égalité et d'inégalité : l'apprentissage revient à minimiser l'entropie croisée, sachant que la valeur cible est égale à 0 ou 1, et que la prédiction est un réel compris entre 0 et 1. Finalement, la recherche de minimum pour ce problème entraîne que la prédiction du neurone est le résultat d'une fonction sigmoïd. Une rapide étude numérique a de plus confirmé que c'est cette association entropie croisée-sigmoïd qui donne les meilleurs résultats d'apprentissage.

À la suite des études préliminaires présentées dans la troisième partie du manuscrit,

les principales études sur le couplage de réseaux de neurones à des codes de calculs sont présentées dans la quatrième et dernière partie. Le Chapitre 7 traite notamment du couplage de réseaux entraînés à modéliser le phénomène de transport de chaleur non local au code hydrodynamique CHIC, qui est un outil servant à la simulation d'expériences de fusion par confinement inertiel.

Dans ce chapitre, la stratégie de couplage est définie de telle sorte qu'un réseau est appelé à chaque pas de temps pour le calcul du gradient de la somme des contributions des groupes en énergie, provenant du modèle de flux de chaleur non local de Schurtz-Nicolaï-Busquet, et servant à résoudre une équation de transport de chaleur pour fermer le système d'équations d'un modèle hydrodynamique de plasma bitempérature. Nous avons généré deux bases de données à l'aide du code CHIC, l'une avec des profils 1D et l'autre avec des profils 2D, sur des cas simplifiés d'ablation laser d'une cible de plastique. Dans ces deux bases, un profil en température constitue une donnée en entrée, et le profil de source non locale correspondant (i.e. le gradient de la somme des contributions des groupes en énergie mentionné précédemment) constitue la cible en sortie. Afin de pouvoir coupler les réseaux entraînés au code CHIC, nous avons également implémenté un module en Fortran (langage dans lequel est programmé le code hydrodynamique), où nous avons défini d'une part une fonction pour la lecture des poids en début de simulation, et d'autre part une fonction pour reproduire la passe avant d'un réseau dense quelconque. Finalement, nous avons entraîné des réseaux sur chacune des bases.

Afin de trouver le meilleur réseau pour chacune des bases, une étude paramétrique a été menée afin de trouver les hyperparamètres engendrant la meilleure performance en entraînement. Les hyperparamètres que nous avons fait varier sont le nombre de couches cachées, le nombre de neurones cachés, la fonction d'activation des couches cachées, le taux d'apprentissage et sa variation : le nombre d'entraînements effectués étant très grand, seuls certains sont présentés dans ce manuscrit (dans le chapitre et en annexe). Pour les deux bases de données, les résultats d'apprentissage ont montré que comme avec le modèle de Lucciani-Mora-Virmont vu au Chapitre 5, les réseaux à propagation avant étaient capables de faire des prédictions avec une erreur inférieure à 3% en tout point du maillage pour le modèle Schurtz-Nicolaï-Busquet. Mais une fois couplé au code hydrodynamique, il s'est avéré que de faibles oscillations présentes sur la prédiction étaient capables d'engendrer des oscillations encore plus fortes sur le reste de la simulation, notamment sur les profils en température et en densité.

Pour évaluer l'étendue de ces oscillations, nous avons notamment utilisé un outil couramment rencontré dans les application médicales, le gamma index, qui n'est autre qu'un moyen d'évaluer si une fonction calculée est dans un voisinage satisfaisant d'une fonction de référence : si le gamma index est inférieur à 1, alors la fonction calculée a une erreur inférieure à un seuil de notre choix, dans un rayon d'espace de notre choix également. Pour cette étude, nous avons défini un critère d'erreur de 1% dans un rayon de 0.5μm. Pour les simulations 1D, le calcul du gamma index à un pas de temps donné en tout point du maillage pour la prédiction, la température et la densité, a prouvé que les oscillations sur la prédiction ont lieu là où la température a une première hausse (que l'on appelle pied d'onde), mais aussi là où la densité est maximum. S'agissant d'un problème instationnaire, les faibles erreurs commises par le RNA s'accumulent à chaque pas de temps, et les oscillations de la prédiction sont en fait interprétées par le code hydrodynamique comme des oscillations physiques, qui se répercutent ensuite sur le reste du profil hydrodynamique (température, pression, densité, etc.).

Pour pallier à ce phénomène, nous avons donc ajouté une stratégie de lissage de la prédiction, que nous avons appelé lissage zéro ou *zero smoothing*, car il consiste à changer

la valeur de la prédiction en zéro sur la zone d'intérêt, soit jusqu'à la fin du pied d'onde. Avec cette stratégie, les simulations 1D ne présentent plus les fortes oscillations observées jusquelà, et le critère de gamma index est respecté en tout point du maillage pour tous les profils hydrodynamiques à chaque pas de temps. Les mêmes oscillations sont visibles sur les simulations 2D, et la même stratégie de lissage a également permis de nettement améliorer les résultats et assure une certaine robustesse de ce nouvel outil de modélisation. La garantie de la fiabilité des outils de *deep learning* reste cependant une question importante à résoudre.

Le gain en temps de calcul de ces réseaux est déjà non négligeable pour ces cas simplifiés : il est d'un facteur 233 en 1D, et 433 en 2D. Pour des bases de données plus complexes, on s'attend à utiliser des réseaux plus grands, mais le gain en temps de calcul devrait être acceptable. Il est important de noter que le couplage a également été testé pour des profils qui n'étaient pas dans les bases de données mais faisaient néanmoins partie de la gamme d'intensités des données d'apprentissage. Même dans ce cas, les réseaux sont capables de faire des prédictions précises, prouvant ainsi la capacité d'interpolation des réseaux de neurones.

Nous avons également effectué de premiers tests sur les réseaux convolutifs sur la base de données 1D. Pour ce faire, le module Fortran a été enrichi d'une fonction pour reproduire la passe avant d'un réseau convolutif quelconque. Les résultats d'apprentissage sont très satisfaisants, avec un taux de réussite à 100% pour une erreur inéfieure à 3%. Mais les résultats de couplage ont montré un effet de lissage inhérent à la convolution, et qui modifie de manière trop importante la simulation hydrodynamique. Cette étude est donc à approfondir, notamment en passant à une base de données englobant un plus grand nombre de cas (par exemple en prennant différents matériaux pour la cible, ou encore plus d'intensités lasers dans le cas 2D).

Le Chapitre 8 présente les résultats d'une étude sur l'utilisation de réseaux pour le calcul de dose en radiothérapie. La stratégie de couplage est différente dans ce cas, puisque nous avons choisi de substituer un réseau à la totalité de la fin du calcul. En effet, le modèle considéré est un modèle aux moments où les contributions des énergies sont additives et c'est donc la somme des doses imputables à chaque groupe en énergie qui forment la dose complète. Le système d'équations est également considéré comme stationnaire, contrairement au modèle de transport de chaleur non local vu au Chapitre 7. Il n'y aura donc pas d'accumulation d'erreurs sur la prédiction qui se répercuteraient sur le reste de la simulation numérique et nécessiteraient la mise en place d'un lissage. Il se trouve par ailleurs que les groupes en énergie peuvent être divisés en deux : les hautes énergies, qui se calculent rapidement, et les faibles énergies, qui prennent plus de temps à être calculées. Il est donc plus intéressant, dans ce cas particulier, de remplacer le calcul des faibles énergies par un réseau de neurones, afin d'accélérer le temps de calcul global. Nous avons ainsi généré une base de données 2D, où un faisceau de photons traverse par la face gauche un milieu homogène constitué d'eau. Ici, la dose imputée aux hautes énergies constitue la donnée d'entrée, et la dose complète correspondante constitue la cible en sortie.

Ici encore, une étude paramétrique a été menée : de nombreux réseaux ont été entraînés en faisant varier plusieurs hyperparamètres, à savoir le nombre de couches cachées, le nombre de neurones par couche, la fonction d'activation des couches cachées, le taux d'apprentissage et sa variation. Les résultats les plus intéressants sont présentés dans le chapitre et en annexe. Les résultats d'entraînement ont montré que des réseaux à propagation avant sont capables de faire des prédictions avec une erreur inférieure à 1%, et que même les plus petites architectures (seulement 1 couche cachée de 10 neurones)

sont capables de telles performances. Ceci est certainement dû au fait que l'entrée et la sortie ont des formes très similaires, mais aussi au fait que les cas considérés sont très simples (toujours le même type de faisceau, traversant un milieu homogène).

Enfin, une étude sur le temps d'exécution a montré que les performances du module Fortran que nous avons développé pour faire nos couplages sont moins bonnes que celles d'une librairie comme Tensorflow. En effet, le module Fortran n'est pas parallélisé, d'où des temps de calcul de l'ordre de 10^{-2} secondes, contre 10^{-3} secondes pour la librairie Tensorflow. Les réseaux étudiés sont relativement petits (moins d'1 million de paramètres) comparé aux tailles de réseaux utilisés pour des tâches plus complexes (par exemple, l'architecture Unet, développée pour la segmentation d'images médicales, en possède près de 8 millions). Les performances du module Fortran pourraient donc nettement se détériorer pour de très grandes architectures.

Plusieurs perspectives transparaissent des résultats présentés dans ce manuscrit. Tout d'abord, la question de la taille des maillages des profils. En effet, la majorité des réseaux étudiés sont des réseaux denses, donc les dimensions de l'entrée et de la sortie doivent rester fixes. Or, pour une utilisation dans un code de calcul multidimensionnel de production, il faudrait pouvoir prendre en compte n'importe quelle taille de maillage, selon le besoin des utilisateurs. C'est pour cela que nous avons testé les réseaux convolutifs à la fin du Chapitre 7 : en effet, pour des phénomènes à même échelle, leurs noyaux de convolution ne sont pas sensibles à la dimension des images. De plus, ces réseaux préservent la notion d'espace 2D, contrairement aux réseaux denses, pour lesquels nous avions transformé les données 2D en vecteurs 1D. Les résultats préliminaires sur l'utilisation de réseaux convolutifs pour la modélisation du transport de chaleur non local sont pour l'instant non concluant dans le cas 1D. Il faudrait faire une étude plus poussée afin de s'assurer qu'il ne s'agit pas d'une erreur de mise en œuvre de notre part, ou bien si ces réseaux ne se prêtent simplement pas à ce genre de données. Un autre moyen de contourner le problème de la taille des maillages et qui mérite une étude serait de projeter les profils sur un maillage de référence. Dans ce cas, il faudrait s'assurer que les étapes de projection de l'entrée, et de projection inverse de la prédiction, ne ralentissent pas le temps d'exécution.

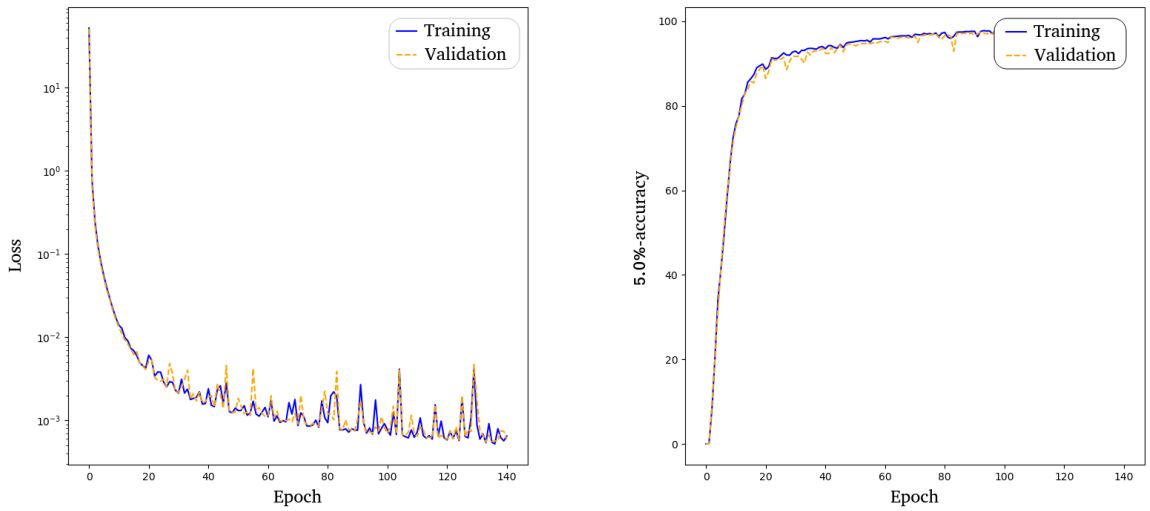
Ensuite, la question du lissage de la prédiction semble également une question importante. Même si nous avons proposé une solution avec le lissage zéro, il serait intéressant de voir si on ne pourrait pas agir sur l'apprentissage du réseau, par exemple en rajoutant une contrainte sur la prédiction dans la définition de la fonction perte. Ou alors, on pourrait éventuellement faire un apprentissage en ligne : lors du couplage, si un nouveau profil avec des oscillations est identifié, il serait rajouté à la base de données de base, et le réseau devra faire plusieurs itérations d'apprentissage avec cette base de données enrichie avant de revenir dans la simulation numérique. Cette dernière idée implique néanmoins un changement dans l'implémentation du module Fortran que nous avons développé.

En effet, on a fait le choix d'implémenter un module Fortran pour pouvoir coupler des réseaux à des codes de calcul. Ce choix était justifié par le fait qu'il s'agissait ici d'une première exploration, et non le développement d'un outil industriel. L'étude du Chapitre 8 a cependant montré que les performances de ce module sont moins bonnes qu'une librairie optimisée comme Tensorflow, et que la mise en place d'une parallélisation serait nécessaire pour les plus grandes architectures de réseaux. Une autre solution serait plutôt d'utiliser une des nouvelles librairies de *deep learning* développées par une partie de la communauté du calcul scientifique qui s'intéresse à l'utilisation des réseaux de neurones dans les grands codes de calcul. L'utilisation d'une telle librairie pourrait notamment permettre de mettre en place l'apprentissage en ligne mentionné précédemment.

Enfin, il serait intéressant d'étendre l'étude sur le lien théorique entre fonction d'activation et fonction de perte du Chapitre 6 à d'autres cas. Nous avons montré le lien entre l'activation sigmoïd et la fonction perte d'entropie croisée, et il serait donc intéressant d'en faire de même pour d'autres associations de fonctions d'activation et de perte. De plus, la preuve a été faite pour un neurone simple, or les réseaux couramment utilisés aujourd'hui sont des réseaux dits profonds. Il serait donc judicieux dans un premier temps d'étendre la démonstration aux réseaux à 1 seule couche cachée, puis aux réseaux à plusieurs couches cachées.

Annexe A

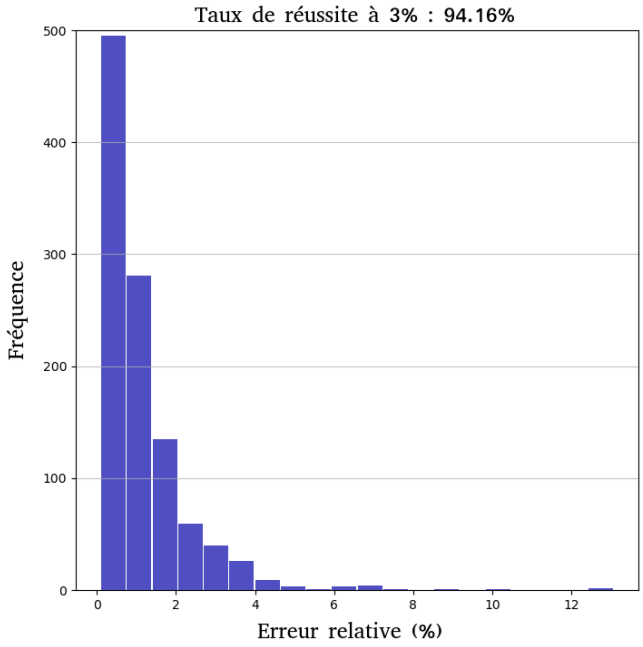
Couplage de RNA au code CHIC : base de données CHIC1D-prélim



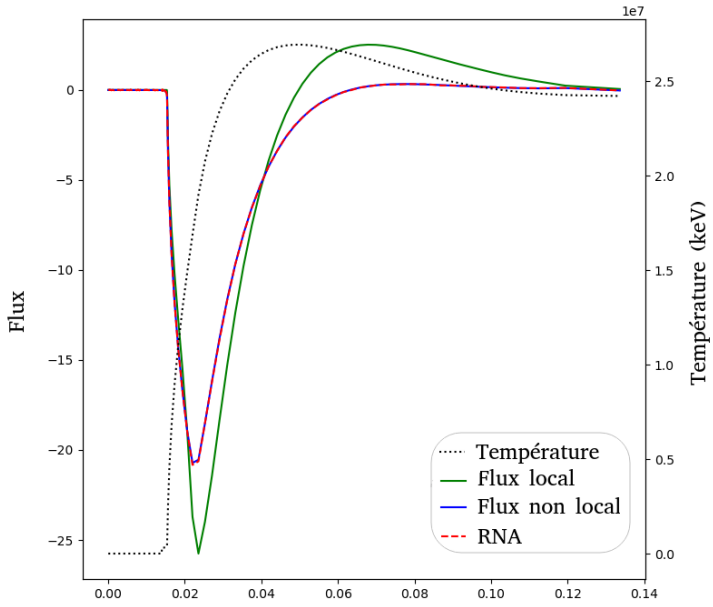
(a) Évolution de la *loss*

(b) Évolution du taux de réussite à 5%

FIGURE A.1 – Résultats de l'entraînement sur la base CHIC1D-prélim sans les 5 premières intensités laser

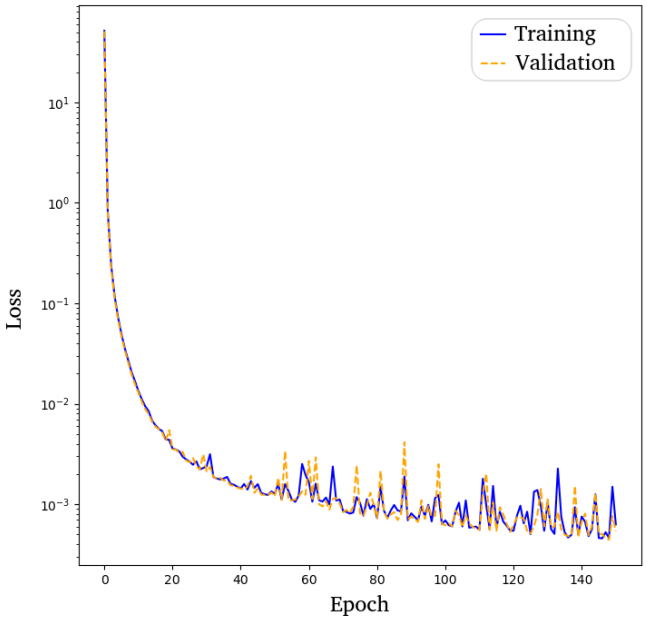


(a) Histogramme des erreurs relatives

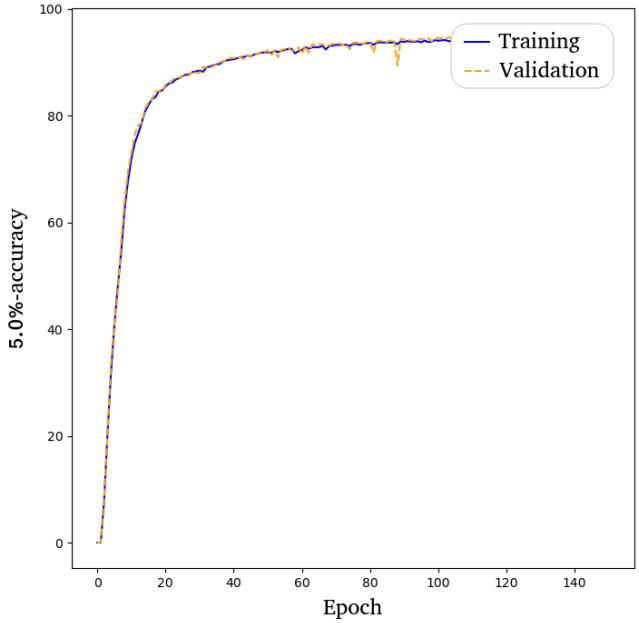


(b) Exemple de profil

FIGURE A.2 – Performances sur l'ensemble test de la base CHIC1D-prélim sans les 5 premières intensités laser

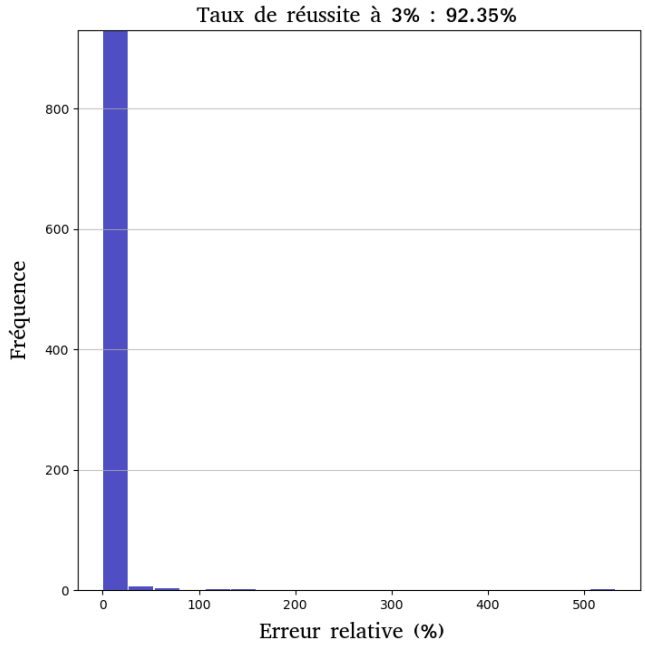


(a) Évolution de la *loss*

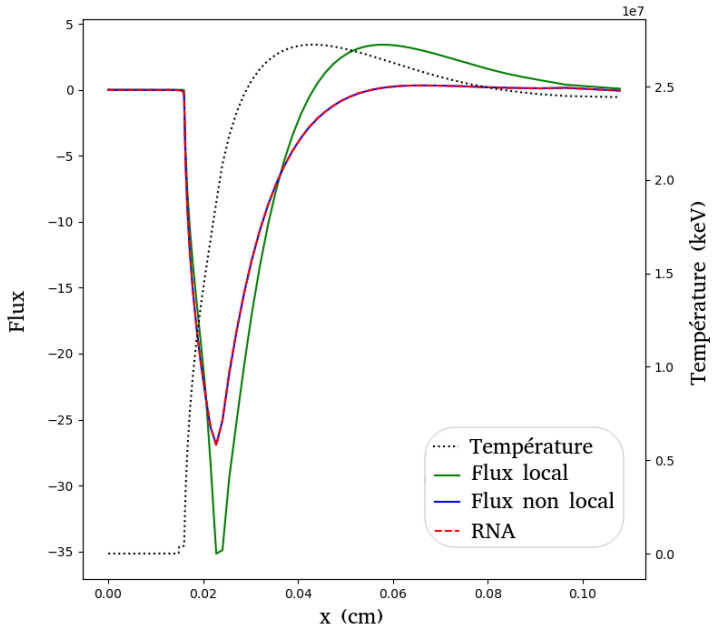


(b) Évolution du taux de réussite à 5%

FIGURE A.3 – Résultats de l'entraînement sur la base CHIC1D-prélim sans les 2 premiers pas de temps



(a) Histogramme des erreurs relatives



(b) Exemple de profil

FIGURE A.4 – Performances sur l'ensemble test de la base CHIC1D-prélim sans les 2 premiers pas de temps

Annexe B

Couplage de RNA au code CHIC : base de données CHIC1D

	nb de neurones cachés					
nb de couches cachées	240		480		960	
1	64,44 %	54,34 %	60,00 %	61,79 %	67,78 %	61,65 %
2	88,89 %	85,09 %	96,67 %	89,84 %	85,56 %	79,27 %
3	96,67 %	97,56 %	97,78 %	98,51 %	100,00 %	97,97 %
4	98,89 %	98,24 %	100,00 %	99,19 %	100,00 %	99,46 %
5	98,89 %	99,32 %	100,00 %	99,46 %	100,00 %	99,19 %
6	100,00 %	99,05 %	100,00 %	99,46 %	100,00 %	99,46 %
8	7,78 %	6,10 %	100,00 %	99,46 %	100,00 %	99,32 %
10	5,56 %	3,39 %	100,00 %	98,92 %	100,00 %	99,59 %

Tableau B.1 – Taux de réussite à 3% sur les ensembles d'entraînement (valeurs bleues) et de validation (valeurs orange) - α_0=0.01 et *batchsize*=90

	nb de neurones cachés					
nb de couches cachées	240		480		960	
1	54,80 %	23,00 %	57,80 %	22,20 %	63,20 %	36,20 %
2	85,00 %	63,00 %	89,40 %	74,40 %	79,80 %	55,60 %
3	96,00 %	81,20 %	98,20 %	94,80 %	98,00 %	93,00 %
4	99,40 %	96,20 %	99,40 %	98,80 %	99,00 %	97,80 %
5	99,20 %	98,20 %	98,80 %	98,40 %	99,60 %	97,80 %
6	98,00 %	91,60 %	99,20 %	98,60 %	99,40 %	99,00 %
8	4,80 %	1,20 %	99,80 %	98,80 %	99,60 %	99,00 %
10	3,80 %	0,60 %	98,80 %	96,40 %	99,40 %	99,00 %

Tableau B.2 – Taux de réussite à 3% d'erreur (valeurs bleues) et taux de réussite pour un gamma index < 1 (valeurs orange) sur l'ensemble test - α_0=0.01 et *batchsize*=90

	nb de neurones cachés					
nb de couches cachées	240		480		960	
1	55,56 %	51,36 %	42,22 %	52,17 %	48,89 %	51,22 %
2	72,22 %	67,21 %	82,22 %	79,67 %	85,56 %	80,62 %
3	84,44 %	88,35 %	96,67 %	94,58 %	100,00 %	97,15 %
4	88,89 %	91,46 %	100,00 %	98,92 %	98,89 %	99,19 %
5	95,56 %	93,63 %	98,89 %	99,46 %	100,00 %	99,46 %
6	94,44 %	97,70 %	100,00 %	99,59 %	100,00 %	99,86 %
8	78,89 %	73,17 %	98,89 %	99,59 %	100,00 %	99,46 %
10	5,56 %	5,28 %	100,00 %	98,92 %	100,00 %	99,59 %

Tableau B.3 – Taux de réussite à 3% sur les ensembles d'entraînement (valeurs bleues) et de validation (valeurs orange) - α_0=0.005 et *batchsize*=90

	nb de neurones cachés					
nb de couches cachées	240		480		960	
1	51,40 %	30,60 %	49,80 %	29,80 %	47,20 %	26,40 %
2	73,40 %	47,60 %	79,20 %	58,60 %	79,60 %	61,40 %
3	88,40 %	69,20 %	93,40 %	84,80 %	97,20 %	92,00 %
4	87,40 %	69,60 %	98,00 %	94,60 %	98,80 %	98,20 %
5	95,00 %	81,40 %	98,20 %	96,40 %	99,80 %	99,00 %
6	95,40 %	84,20 %	99,20 %	98,00 %	100,00 %	99,60 %
8	72,00 %	37,00 %	99,00 %	97,60 %	100,00 %	99,80 %
10	2,80 %	0,60 %	98,20 %	96,20 %	99,60 %	99,20 %

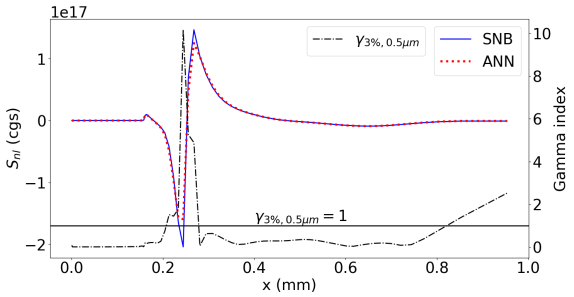
Tableau B.4 – Taux de réussite à 3% d'erreur (valeurs bleues) et taux de réussite pour un gamma index < 1 (valeurs orange) sur l'ensemble test - α_0=0.005 et *batchsize*=90

	nb de neurones cachés					
nb de couches cachées	240		480		960	
1	56,44 %	54,34 %	58,00 %	49,59 %	52,89 %	47,97 %
2	71,11 %	68,29 %	80,67 %	78,18 %	86,22 %	80,49 %
3	82,67 %	77,64 %	93,11 %	92,68 %	97,56 %	95,53 %
4	91,56 %	88,89 %	98,00 %	97,43 %	99,11 %	98,64 %
5	93,33 %	91,06 %	98,44 %	96,75 %	99,78 %	99,19 %
6	95,11 %	95,12 %	97,56 %	98,51 %	100,00 %	99,86 %
8	91,78 %	89,57 %	99,56 %	98,51 %	100,00 %	99,59 %
10	5,56 %	7,99 %	98,44 %	98,37 %	99,78 %	99,73 %

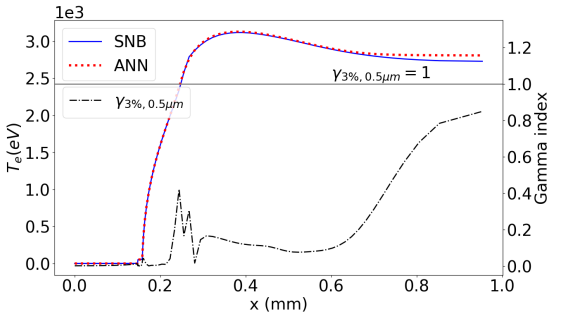
Tableau B.5 – Taux de réussite à 3% sur les ensembles d'entraînement (valeurs bleues) et de validation (valeurs orange) - α_0=0.005 et *batchsize*=450

	nb de neurones cachés					
nb de couches cachées	240		480		960	
1	53,20 %	30,60 %	52,40 %	33,00 %	47,40 %	26,80 %
2	70,40 %	55,00 %	77,60 %	59,00 %	83,80 %	66,00 %
3	78,40 %	60,00 %	90,40 %	75,00 %	94,80 %	85,20 %
4	89,20 %	69,00 %	98,20 %	91,20 %	98,00 %	94,60 %
5	92,80 %	72,00 %	98,60 %	91,80 %	99,20 %	96,60 %
6	93,40 %	77,80 %	98,40 %	91,40 %	99,40 %	98,40 %
8	88,20 %	70,80 %	98,80 %	95,00 %	99,80 %	99,00 %
10	6,60 %	1,80 %	98,80 %	94,20 %	99,80 %	99,00 %

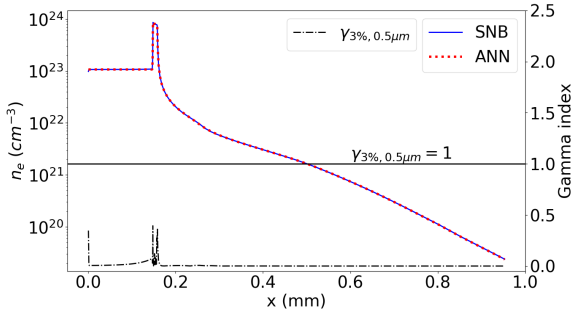
Tableau B.6 – Taux de réussite à 3% d'erreur (valeurs bleues) et taux de réussite pour un gamma index < 1 (valeurs orange) sur l'ensemble test - α_0=0.005 et *batchsize*=450



(a) Source non locale

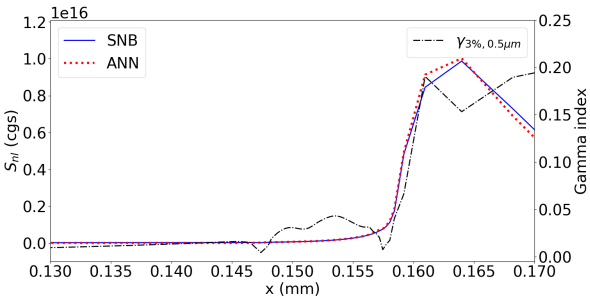


(b) Température

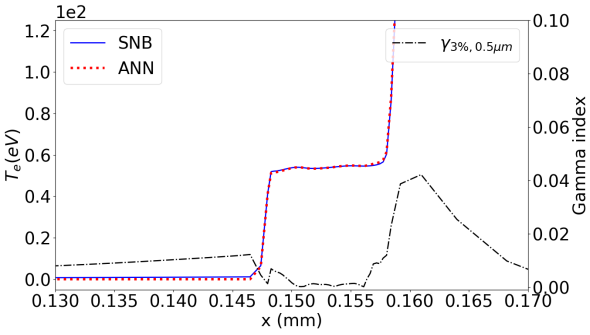


(c) Densité

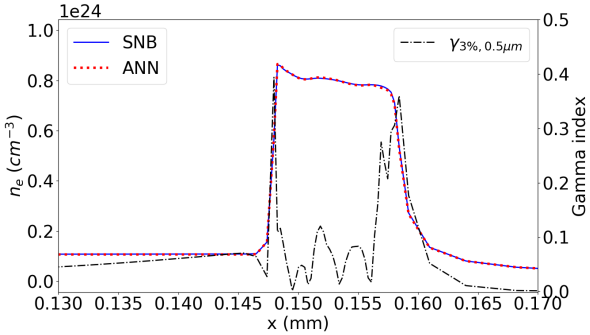
FIGURE B.1 – Profils hydrodynamiques avec lissage à $t = 500$ps (intensité laser inconnue du RNA)



(a) Source non locale



(b) Température

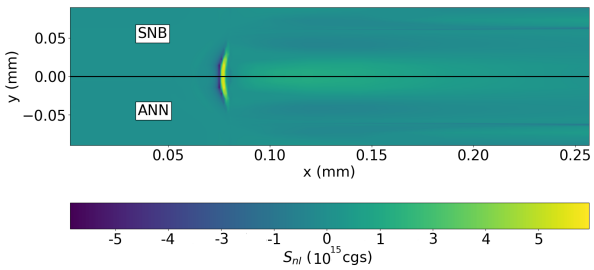


(c) Densité

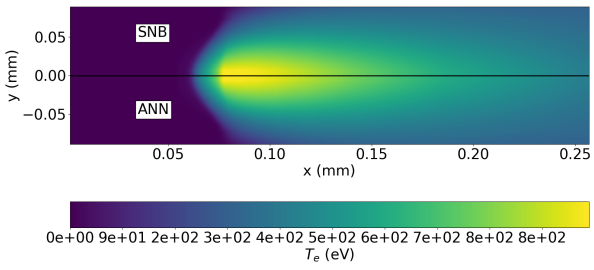
FIGURE B.2 – Zoom sur les profils hydrodynamiques avec lissage à $t = 500$ps (intensité laser inconnue du RNA)

Annexe C

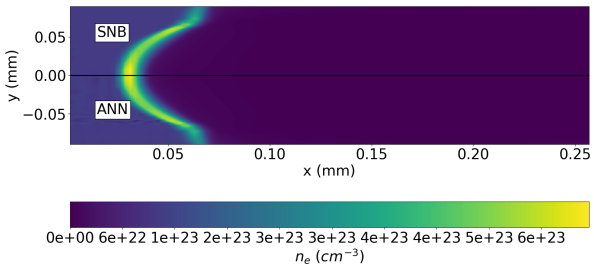
Couplage de RNA au code CHIC : base de données CHIC2D



(a) Source non locale

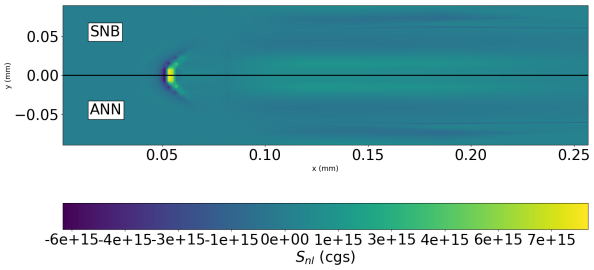


(b) Température

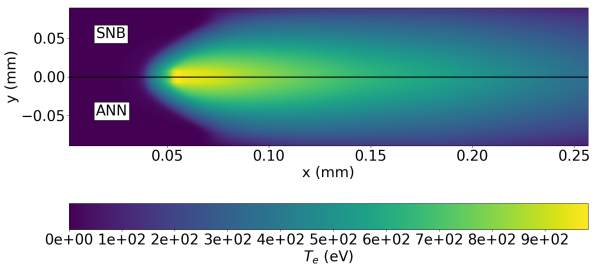


(c) Densité

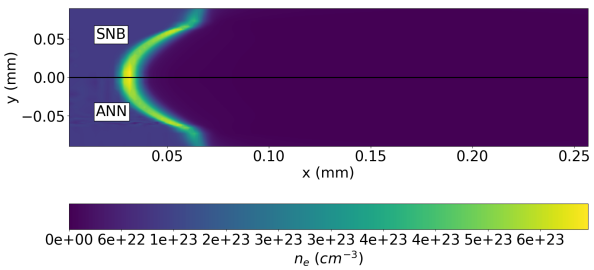
FIGURE C.1 – Profils hydrodynamiques sans lissage à $t = 500$ps



(a) Source non locale

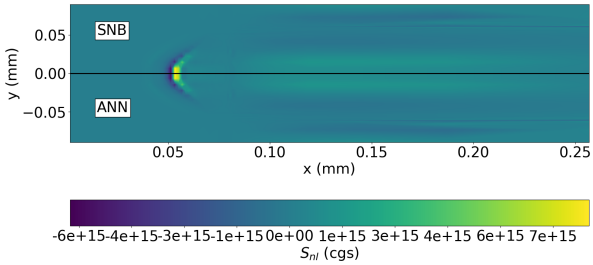


(b) Température

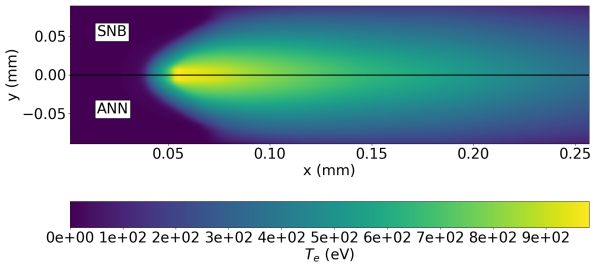


(c) Densité

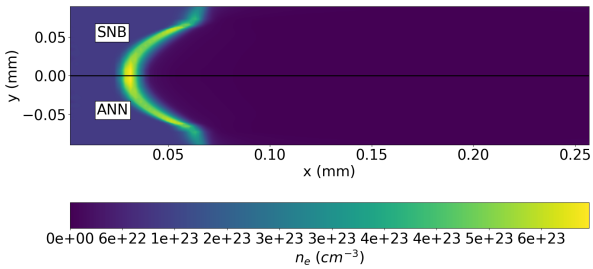
FIGURE C.2 – Profils hydrodynamiques sans lissage à $t = 1000$ps



(a) Source non locale

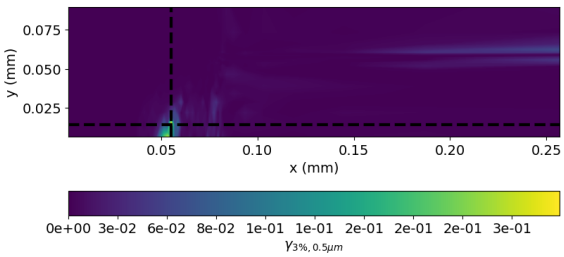


(b) Température

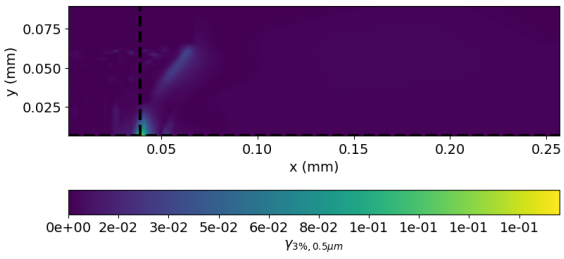


(c) Densité

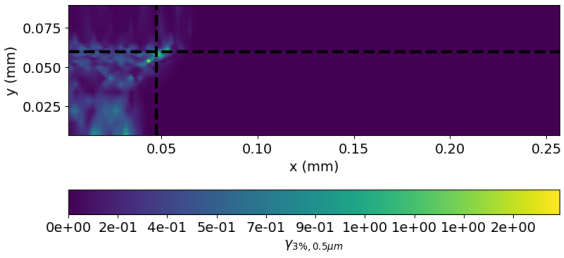
FIGURE C.3 – Profils hydrodynamiques avec lissage à $t = 1000$ps



(a) Source non locale

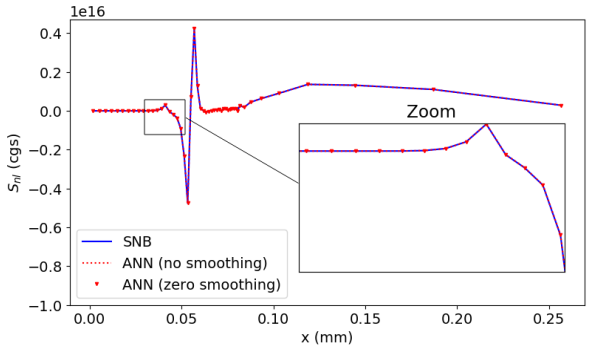


(b) Température

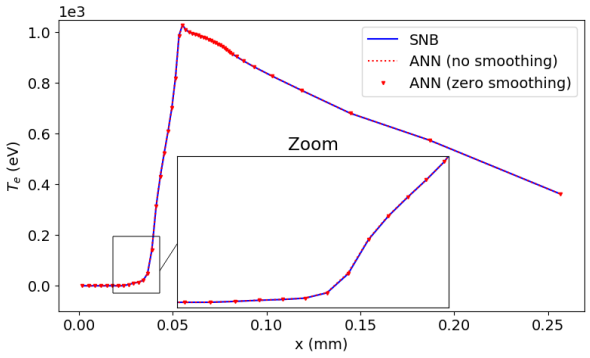


(c) Densité

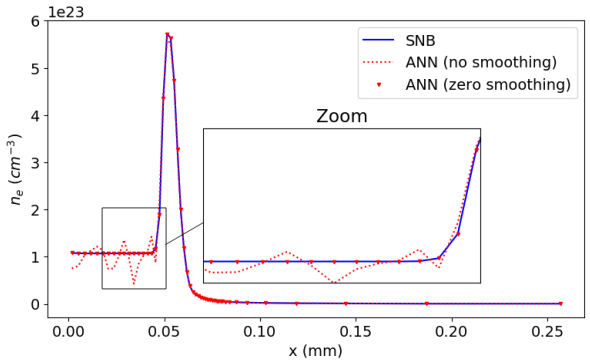
FIGURE C.4 – Gamma index des profils hydrodynamiques avec lissage à $t = 1000$ps



(a) Source non locale

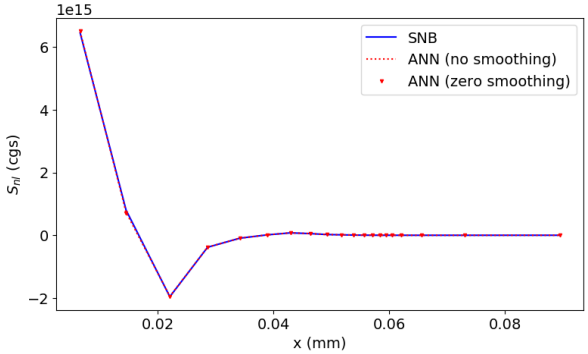


(b) Température

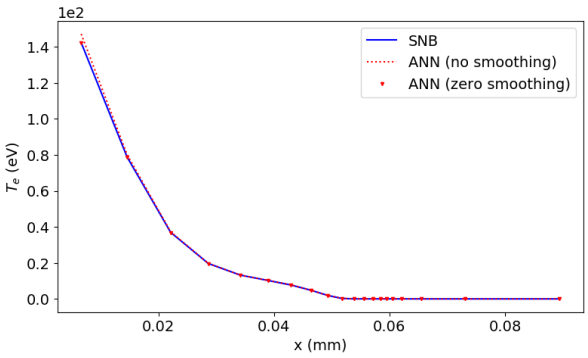


(c) Densité

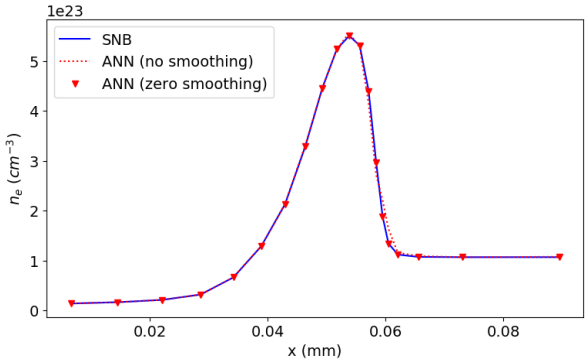
FIGURE C.5 – Coupe longitudinale des profils hydrodynamiques à $t = 1000$ps



(a) Source non locale



(b) Température



(c) Densité

FIGURE C.6 – Coupe transversale des profils hydrodynamiques à $t = 1000$ps

Annexe D

Couplage de RNA au code KIDS

	nb couches cachées									
nb neurones/couche	1		2		3		5		10	
10	0,00 %	0,00 %	0,00 %	0,00 %	0,00 %	0,00 %	0,00 %	0,00 %	92,42 %	88,67 %
50	0,00 %	0,00 %	98,83 %	98,00 %	98,92 %	98,67 %	98,67 %	98,00 %	0,00 %	0,00 %
100	19,67 %	17,33 %	99,58 %	98,67 %	100,00 %	100,00 %	100,00 %	100,00 %	0,00 %	0,00 %
250	81,25 %	80,67 %	100,00 %	100,00 %	100,00 %	100,00 %	0,00 %	0,00 %	0,00 %	0,00 %
500	100,00 %	100,00 %	98,92 %	98,67 %	0,00 %	0,00 %	0,00 %	0,00 %	0,00 %	0,00 %
750	100,00 %	100,00 %	93,67 %	93,33 %	69,50 %	64,67 %	0,00 %	0,00 %	0,00 %	0,00 %
1000	100,00 %	100,00 %	0,00 %	0,00 %	0,00 %	0,00 %	0,00 %	0,00 %	0,00 %	0,00 %

Tableau D.1 – Taux de réussite à 1% sur les ensembles d'entraînement (valeurs bleues) et de validation (valeurs oranges) - $\alpha_0 = 0.01$

	nb couches cachées									
nb neurones/couche	1		2		3		5		10	
10	0,00 %	0,00 %	0,00 %	0,00 %	0,00 %	0,00 %	0,00 %	0,00 %	93,15 %	94,52 %
50	0,00 %	0,00 %	99,32 %	99,32 %	99,32 %	99,32 %	99,32 %	99,32 %	0,00 %	0,00 %
100	21,23 %	21,23 %	99,32 %	100,00 %	100,00 %	100,00 %	100,00 %	100,00 %	0,00 %	0,00 %
250	79,45 %	80,14 %	100,00 %	100,00 %	100,00 %	100,00 %	0,00 %	0,00 %	0,00 %	0,00 %
500	100,00 %	100,00 %	99,32 %	99,32 %	0,00 %	0,00 %	0,00 %	0,00 %	0,00 %	0,00 %
750	100,00 %	100,00 %	91,78 %	95,21 %	63,01 %	63,70 %	0,00 %	0,00 %	0,00 %	0,00 %
1000	100,00 %	100,00 %	0,00 %	0,00 %	0,00 %	0,00 %	0,00 %	0,00 %	0,00 %	0,00 %

Tableau D.2 – Taux de réussite à 1% d'erreur (valeurs bleues) et taux de réussite pour un gamma index < 1 (valeurs orange) sur l'ensemble test - $\alpha_0 = 0.01$

	nb couches cachées									
nb neurones/couche	1		2		3		5		10	
10	73,75 %	76,00 %	71,25 %	66,67 %	15,75 %	14,67 %	15,25 %	14,00 %	13,00 %	10,67 %
50	98,83 %	98,67 %	78,67 %	75,33 %	74,08 %	70,00 %	73,25 %	68,00 %	74,42 %	70,00 %
100	98,67 %	98,00 %	97,75 %	96,67 %	95,25 %	93,33 %	93,92 %	92,00 %	94,50 %	93,33 %
250	96,08 %	96,00 %	99,67 %	99,33 %	98,67 %	98,00 %	98,25 %	98,00 %	98,17 %	97,33 %
500	96,67 %	95,33 %	99,50 %	99,33 %	98,00 %	97,33 %	98,00 %	97,33 %	97,75 %	96,67 %
750	95,08 %	94,67 %	98,17 %	97,33 %	92,50 %	92,00 %	92,08 %	88,00 %	86,83 %	86,67 %
1000	96,00 %	94,67 %	86,92 %	83,33 %	93,83 %	92,67 %	77,00 %	75,33 %	67,08 %	62,00 %

Tableau D.3 – Taux de réussite à 1% sur les ensembles d'entraînement (valeurs bleues) et de validation (valeurs orange) - $\alpha_0 = 0.0001$

	nb couches cachées									
nb neurones/couche	1		2		3		5		10	
10	72,60 %	80,14 %	66,44 %	68,77 %	17,12 %	20,55 %	17,12 %	19,18 %	15,07 %	16,44 %
50	99,32 %	100,00 %	71,92 %	84,25 %	67,81 %	76,71 %	67,12 %	73,29 %	69,18 %	80,82 %
100	99,32 %	100,00 %	98,63 %	99,32 %	96,58 %	99,32 %	96,58 %	98,63 %	96,58 %	98,63 %
250	97,26 %	98,63 %	100,00 %	100,00 %	99,32 %	99,32 %	99,32 %	99,32 %	99,32 %	99,32 %
500	97,95 %	98,63 %	99,32 %	100,00 %	99,32 %	99,32 %	99,32 %	99,32 %	99,32 %	99,32 %
750	97,26 %	98,63 %	98,63 %	100,00 %	95,21 %	99,32 %	94,52 %	97,26 %	88,36 %	96,58 %
1000	97,95 %	98,63 %	86,99 %	89,73 %	97,26 %	98,63 %	75,34 %	96,58 %	58,22 %	67,12 %

Tableau D.4 – Taux de réussite à 1% d'erreur (valeurs bleues) et taux de réussite pour un gamma index < 1 (valeurs orange) sur l'ensemble test - $\alpha_0 = 0.0001$

Bibliographie

[1] W. Gropp, W. D. Gropp, E. Lusk, A. Skjellum, and A. D. F. E. E. Lusk, *Using MPI : portable parallel programming with the message-passing interface*, vol. 1. MIT press, 1999.

[2] L. Dagum and R. Menon, "Openmp : an industry standard api for shared-memory programming," *IEEE computational science and engineering*, vol. 5, no. 1, pp. 46–55, 1998.

[3] A. Zylstra, O. Hurricane, D. Callahan, A. Kritcher, J. Ralph, H. Robey, J. Ross, C. Young, K. Baker, D. Casey, *et al.*, "Burning plasma achieved in inertial fusion," *Nature*, vol. 601, no. 7894, pp. 542–548, 2022.

[4] J. Ongena, R. Koch, R. Wolf, and H. Zohm, "Magnetic-confinement fusion," *Nature Physics*, vol. 12, no. 5, pp. 398–410, 2016.

[5] R. Betti and O. Hurricane, "Inertial-confinement fusion with lasers," *Nature Physics*, vol. 12, no. 5, pp. 435–448, 2016.

[6] W. Manheimer and D. Colombant, "Analytic insights into nonlocal energy transport. III. steady state Fokker Planck theory in spherical and planar geometry," *Physics of Plasmas*, vol. 28, p. 012701, Jan. 2021. Publisher : American Institute of Physics.

[7] M. R. K. Wigram, C. P. Ridgers, B. D. Dudson, J. P. Brodrick, and J. T. Omotani, "Incorporating nonlocal parallel thermal transport in 1D ITER SOL modelling," *Nucl. Fusion*, vol. 60, p. 076008, June 2020. Publisher : IOP Publishing.

[8] N. R. Council and P. S. Committee, *Frontiers in High Energy Density Physics : The X-Games of Contemporary Science*. Washington, DC : The National Academic Press, 2003.

[9] R. Schneider, X. Bonnin, K. Borrass, D. P. Coster, H. Kastelewicz, D. Reiter, V. A. Rozhansky, and B. J. Braams, "Plasma Edge Physics with B2-Eirene," *Contributions to Plasma Physics*, vol. 46, no. 1-2, pp. 3–191, 2006. _eprint : https ://onlinelibrary.wiley.com/doi/pdf/10.1002/ctpp.200610001.

[10] M. D. Rosen, H. A. Scott, D. E. Hinkel, E. A. Williams, D. A. Callahan, R. P. J. Town, L. Divol, P. A. Michel, W. L. Kruer, L. J. Suter, R. A. London, J. A. Harte, and G. B. Zimmerman, "The role of a detailed configuration accounting (DCA) atomic physics package in explaining the energy balance in ignition-scale hohlraums," *High Energy Density Physics*, vol. 7, pp. 180–190, Sept. 2011.

[11] A. R. Bell, R. G. Evans, and D. J. Nicholas, "Electron energy transport in steep temperature gradients in laser-produced plasmas," *Phys. Rev. Lett.*, vol. 46, p. 243, 1981.

[12] J. F. Luciani, P. Mora, and J. Virmont, "Nonlocal Heat Transport Due to Steep Temperature Gradients," *Phys. Rev. Lett.*, vol. 51, pp. 1664–1667, Oct. 1983. Publisher : American Physical Society.

[13] G. P. Schurtz, P. D. Nicolaï, and M. Busquet, "A nonlocal electron conduction model for multidimensional radiation hydrodynamics codes," *Physics of Plasmas*, vol. 7, pp. 4238–4249, Oct. 2000. Publisher : American Institute of Physics.

[14] W. Manheimer, D. Colombant, and V. Goncharov, "The development of a Krook model for nonlocal transport in laser produced plasmas. I. Basic theory," *Physics of Plasmas*, vol. 15, p. 083103, Aug. 2008. Publisher : American Institute of Physics.

[15] G. Schurtz, S. Gary, S. Hulin, C. Chenais-Popovics, J.-C. Gauthier, F. Thais, J. Breil, F. Durut, J.-L. Feugeas, P.-H. Maire, P. Nicolaï, O. Peyrusse, C. Reverdin, G. Soullié, V. Tikhonchuk, B. Villette, and C. Fourment, "Revisiting Nonlocal Electron-Energy Transport in Inertial-Fusion Conditions," *Phys. Rev. Lett.*, vol. 98, p. 095002, Feb. 2007. Publisher : American Physical Society.

[16] D. Cao, G. Moses, and J. Delettrez, "Improved non-local electron thermal transport model for two-dimensional radiation hydrodynamics simulations," *Physics of Plasmas*, vol. 22, p. 082308, Aug. 2015. Publisher : American Institute of Physics.

[17] A. Marocchino, M. Tzoufras, S. Atzeni, A. Schiavi, P. D. Nicolaï, J. Mallet, V. Tikhonchuk, and J.-L. Feugeas, "Comparison for non-local hydrodynamic thermal conduction models," *Physics of Plasmas*, vol. 20, p. 022702, Feb. 2013. Publisher : American Institute of Physics.

[18] J. Breil, S. Galera, and P.-H. Maire, "Multi-material ALE computation in inertial confinement fusion code CHIC," *Computers & Fluids*, vol. 46, pp. 161–167, July 2011.

[19] P. D. Nicolai, J.-L. A. Feugeas, and G. P. Schurtz, "A practical nonlocal model for heat transport in magnetized laser plasmas," *Phys. Plasmas*, vol. 13, p. 032701, 2006.

[20] E. Buresi, J. Coutant, R. Dautray, M. Decroisette, B. Duborgel, P. Guillaneux, J. Launspach, P. Nelson, C. Patou, J. Reisse, *et al.*, "Laser program development at cel-v : overview of recent experimental results," *Laser and Particle Beams*, vol. 4, no. 3-4, pp. 531–544, 1986.

[21] S. Atzeni, A. Marocchino, and A. Schiavi, "Improved robustness study of a shock ignited target, with dued code including non-local electron transport and 3d laser ray-tracing," in *Journal of Physics : Conference Series*, vol. 688, p. 012005, IOP Publishing, 2016.

[22] H. Nagatomo, N. Ohnishi, K. Mima, K. Nishihara, S. Yamada, K. Sawada, and H. Takabe, "Numerical simulation of non-spherical implosion related to fast ignition," in *AIP Conference Proceedings*, vol. 669, pp. 253–256, American Institute of Physics, 2003.

[23] M. M. Marinak, G. Kerbel, N. Gentile, O. Jones, D. Munro, S. Pollaine, T. Dittrich, and S. Haan, "Three-dimensional hydra simulations of national ignition facility targets," *Physics of Plasmas*, vol. 8, no. 5, pp. 2275–2280, 2001.

[24] D. Cao, G. Moses, and J. Delettrez, "Improved non-local electron thermal transport model for two-dimensional radiation hydrodynamics simulations," *Physics of Plasmas*, vol. 22, no. 8, p. 082308, 2015.

[25] J. Dutreix, A. Desgrez, B. Bok, and J. Vinot, "Biophysique des radiations et imagerie médicale," *Abrégés de médecine*, 1993.

[26] P. Mayles, A. Nahum, and J.-C. Rosenwald, *Handbook of radiotherapy physics : theory and practice*. CRC Press, 2007.

[27] M. G. Stabin, *Radiation protection and dosimetry.* Springer, 2007.

[28] E. B. Podgorsak *et al.*, *Radiation oncology physics.* IAEA Vienna, 2005.

[29] N. Metropolis and S. Ulam, "The monte carlo method," *Journal of the American statistical association*, vol. 44, no. 247, pp. 335–341, 1949.

[30] G. Battistoni, F. Cerutti, A. Fasso, A. Ferrari, S. Muraro, J. Ranft, S. Roesler, and P. Sala, "The fluka code : Description and benchmarking," in *AIP Conference proceedings*, vol. 896, pp. 31–49, American Institute of Physics, 2007.

[31] S. Agostinelli, J. Allison, K. a. Amako, J. Apostolakis, H. Araujo, P. Arce, M. Asai, D. Axen, S. Banerjee, G. Barrand, *et al.*, "Geant4—a simulation toolkit," *Nuclear instruments and methods in physics research section A : Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 506, no. 3, pp. 250–303, 2003.

[32] A. Ahnesjö and M. M. Aspradakis, "Dose calculations for external photon beams in radiotherapy," *Physics in Medicine & Biology*, vol. 44, no. 11, p. R99, 1999.

[33] A. Gustafsson, B. K. Lind, and A. Brahme, "A generalized pencil beam algorithm for optimization of radiation therapy," *Medical physics*, vol. 21, no. 3, pp. 343–356, 1994.

[34] R. Mohan, C. Chui, and L. Lidofsky, "Differential pencil beam dose computation model for photons," *Medical Physics*, vol. 13, no. 1, pp. 64–73, 1986.

[35] H. Hensel, R. Iza-Teran, and N. Siedow, "Deterministic model for dose calculation in photon radiotherapy," *Phys. Med. Biol.*, vol. 51, pp. 675–693, Jan. 2006. Publisher : IOP Publishing.

[36] M. W. K. Kan, P. K. N. Yu, and L. H. T. Leung, "A Review on the Use of Grid-Based Boltzmann Equation Solvers for Dose Calculation in External Photon Beam Treatment Planning," *BioMed Research International*, vol. 2013, p. e692874, Aug. 2013. Publisher : Hindawi.

[37] K. A. Gifford, J. L. Horton, T. A. Wareing, G. Failla, and F. Mourtada, "Comparison of a finite-element multigroup discrete-ordinates code with Monte Carlo for radiotherapy calculations," *Phys. Med. Biol.*, vol. 51, pp. 2253–2265, Apr. 2006. Publisher : IOP Publishing.

[38] J. Caron, J. L. Feugeas, B. Dubroca, G. Kantor, C. Dejean, G. Birindelli, T. Pichard, P. Nicolaï, E. d'Humières, M. Frank, and V. Tikhonchuk, "Deterministic model for the transport of energetic particles : Application in the electron radiotherapy," *Physica Medica*, vol. 31, pp. 912–921, Dec. 2015.

[39] G. Birindelli, J.-L. Feugeas, J. Caron, B. Dubroca, G. Kantor, J. Page, T. Pichard, V. Tikhonchuk, and P. Nicolaï, "High performance modelling of the transport of energetic particles for photon radiotherapy," *Physica Medica*, vol. 42, pp. 305–312, Oct. 2017.

[40] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *Bulletin of Mathematical Biophysics*, vol. 5, pp. 115–133, Dec. 1943.

[41] F. F. Rosenblatt, "The perceptron : a probabilistic model for information storage and organization in the brain.," *Psychological review*, 1958.

[42] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, pp. 84–90, May 2017.

[43] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *Int J Comput Vis*, vol. 115, pp. 211–252, Dec. 2015.

[44] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics-informed neural networks : A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," *Journal of Computational physics*, vol. 378, pp. 686–707, 2019.

[45] D. Maturana and S. Scherer, "Voxnet : A 3d convolutional neural network for real-time object recognition," in *2015 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pp. 922–928, IEEE, 2015.

[46] Y. LeCun, B. Boser, J. Denker, D. Henderson, R. Howard, W. Hubbard, and L. Jackel, "Handwritten Digit Recognition with a Back-Propagation Network," in *Advances in Neural Information Processing Systems*, vol. 2, Morgan-Kaufmann, 1989.

[47] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-Based Learning Applied to Document Recognition," *Proceedings of the IEEE*, vol. 86, pp. 2278–2324, Dec. 1998.

[48] Y. N. Dauphin, R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli, and Y. Bengio, "Identifying and attacking the saddle point problem in high-dimensional non-convex optimization," in *Advances in Neural Information Processing Systems*, vol. 27, Curran Associates, Inc., 2014.

[49] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv :1609.04747 [cs]*, June 2017. arXiv : 1609.04747.

[50] N. Qian, "On the momentum term in gradient descent learning algorithms," *Neural Networks*, vol. 12, pp. 145–151, Jan. 1999.

[51] D. P. Kingma and J. Ba, "Adam : A Method for Stochastic Optimization," *arXiv :1412.6980 [cs]*, Jan. 2017. arXiv : 1412.6980.

[52] Y. Nesterov, "A method for solving the convex programming problem with convergence rate O(1/k^2)," *undefined*, 1983.

[53] J. Duchi, E. Hazan, and Y. Singer, "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization," p. 39, 2011.

[54] M. D. Zeiler, "ADADELTA : An Adaptive Learning Rate Method," *arXiv :1212.5701 [cs]*, Dec. 2012. arXiv : 1212.5701.

[55] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533–536, Oct. 1986. Number : 6088 Publisher : Nature Publishing Group.

[56] Y. Lecun, "A Theoretical Framework for Back-Propagation," 1988.

[57] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. The MIT Press, 2016.

[58] A. Burkov, *The Hundred-page Machine Learning Book*. 2019. OCLC : 1089445188.

[59] K. You, M. Long, J. Wang, and M. I. Jordan, "How Does Learning Rate Decay Help Modern Neural Networks ?," *arXiv :1908.01878 [cs, stat]*, Sept. 2019. arXiv : 1908.01878.

[60] Y. Bengio, "Practical recommendations for gradient-based training of deep architectures," *arXiv :1206.5533 [cs]*, Sept. 2012. arXiv : 1206.5533.

[61] J. Bergstra and Y. Bengio, "Random Search for Hyper-Parameter Optimization," p. 25, 2012.

[62] L. Deng, G. Hinton, and B. Kingsbury, "New types of deep neural network learning for speech recognition and related applications : an overview," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 8599–8603, May 2013. ISSN : 2379-190X.

[63] A. Roy, J. Sun, R. Mahoney, L. Alonzi, S. Adams, and P. Beling, "Deep learning detecting fraud in credit card transactions," in *2018 Systems and Information Engineering Design Symposium (SIEDS)*, pp. 129–134, Apr. 2018.

[64] E. Angelini, G. di Tollo, and A. Roli, "A neural network approach for credit risk evaluation," *The Quarterly Review of Economics and Finance*, vol. 48, pp. 733–755, Nov. 2008.

[65] A. I. Maqueda, A. Loquercio, G. Gallego, N. García, and D. Scaramuzza, "Event-Based Vision Meets Deep Learning on Steering Prediction for Self-Driving Cars," pp. 5419–5427, 2018.

[66] A. S. Miller, B. H. Blott, and T. K. hames, "Review of neural network applications in medical imaging and signal processing," *Med. Biol. Eng. Comput.*, vol. 30, pp. 449–464, Sept. 1992.

[67] J. Citrin, S. Breton, F. Felici, F. Imbeaux, T. Aniel, J. F. Artaud, B. Baiocchi, C. Bourdelle, Y. Camenen, and J. Garcia, "Real-time capable first principle based modelling of tokamak turbulent transport," *Nucl. Fusion*, vol. 55, p. 092001, Sept. 2015. arXiv : 1502.07402.

[68] G. B. Goh, N. O. Hodas, and A. Vishnu, "Deep learning for computational chemistry," *Journal of Computational Chemistry*, vol. 38, no. 16, pp. 1291–1307, 2017. _eprint : https ://onlinelibrary.wiley.com/doi/pdf/10.1002/jcc.24764.

[69] J. N. Kutz, "Deep learning in fluid dynamics," *Journal of Fluid Mechanics*, vol. 814, pp. 1–4, Mar. 2017. Publisher : Cambridge University Press.

[70] M. F. McCracken, "Artificial Neural Networks in Fluid Dynamics : A Novel Approach to the Navier-Stokes Equations," in *Proceedings of the Practice and Experience on Advanced Research Computing*, (Pittsburgh PA USA), pp. 1–4, ACM, July 2018.

[71] L. Bois, E. Franck, L. Navoret, and V. Vigon, "A neural network closure for the Euler-Poisson system based on kinetic simulations." Oct. 2020.

[72] R. Maulik, N. A. Garland, X.-Z. Tang, and P. Balaprakash, "Neural network representability of fully ionized plasma fluid model closures," *Physics of Plasmas*, vol. 27, p. 072106, July 2020. arXiv : 2002.04106.

[73] P. G. Breen, C. N. Foley, T. Boekholt, and S. P. Zwart, "Newton versus the machine : solving the chaotic three-body problem using deep neural networks," *Monthly Notices of the Royal Astronomical Society*, vol. 494, pp. 2465–2470, May 2020.

[74] K. L. van de Plassche, J. Citrin, C. Bourdelle, Y. Camenen, F. J. Casson, V. I. Dagnelie, F. Felici, A. Ho, and S. Van Mulders, "Fast modeling of turbulent transport in fusion plasmas using neural networks," *Physics of Plasmas*, vol. 27, p. 022310, Feb. 2020. Publisher : American Institute of Physics.

[75] G. Kluth, K. Humbird, B. Spears, J. Peterson, H. Scott, M. Patel, J. Koning, M. Marinak, L. Divol, and C. Young, "Deep learning for nlte spectral opacities," *Physics of Plasmas*, vol. 27, no. 5, p. 052707, 2020.

[76] A. Seltz, P. Domingo, and L. Vervisch, "Solving the population balance equation for non-inertial particles dynamics using probability density function and neural networks : Application to a sooting flame," *Physics of Fluids*, vol. 33, p. 013311, Jan. 2021. Publisher : American Institute of Physics.

[77] A. Mendizabal, P. Márquez-Neila, and S. Cotin, "Simulation of hyperelastic materials in real-time using deep learning," *Medical Image Analysis*, vol. 59, p. 101569, 2019.

[78] P. Testolina, M. Lecci, M. Rebato, A. Testolin, J. Gambini, R. Flamini, C. Mazzucco, and M. Zorzi, "Enabling Simulation-Based Optimization Through Machine Learning : A Case Study on Antenna Design," *2019 IEEE Global Communications Conference (GLOBECOM)*, pp. 1–6, Dec. 2019. arXiv : 1908.11225.

[79] T. Bai, B. Wang, D. Nguyen, and S. Jiang, "Deep Dose Plugin Towards Real-time Monte Carlo Dose Calculation Through a Deep Learning based Denoising Algorithm," *arXiv :2011.14959 [cs]*, Dec. 2020. arXiv : 2011.14959.

[80] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics Informed Deep Learning (Part I) : Data-driven Solutions of Nonlinear Partial Differential Equations," *arXiv :1711.10561 [cs, math, stat]*, Nov. 2017. arXiv : 1711.10561.

[81] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics Informed Deep Learning (Part II) : Data-driven Discovery of Nonlinear Partial Differential Equations," *arXiv :1711.10566 [cs, math, stat]*, Nov. 2017. arXiv : 1711.10566.

[82] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Math. Control Signal Systems*, vol. 2, pp. 303–314, Dec. 1989.

[83] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, pp. 359–366, Jan. 1989.

[84] K. Hornik, "Some new results on neural network approximation," *Neural Networks*, vol. 6, pp. 1069–1072, Jan. 1993.

[85] M. Leshno, V. Y. Lin, A. Pinkus, and S. Schocken, "Multilayer feedforward networks with a nonpolynomial activation function can approximate any function," *Neural Networks*, vol. 6, pp. 861–867, Jan. 1993.

[86] I. Lagaris, A. Likas, and D. Fotiadis, "Artificial neural networks for solving ordinary and partial differential equations," *IEEE Transactions on Neural Networks*, vol. 9, pp. 987–1000, Sept. 1998.

[87] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind, "Automatic differentiation in machine learning : a survey," *arXiv :1502.05767 [cs, stat]*, Feb. 2018. arXiv : 1502.05767.

[88] Y. Zhu, N. Zabaras, P.-S. Koutsourelakis, and P. Perdikaris, "Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data," *Journal of Computational Physics*, vol. 394, pp. 56–81, 2019.

[89] L. Sun, H. Gao, S. Pan, and J.-X. Wang, "Surrogate modeling for fluid flows based on physics-constrained deep learning without simulation data," *Computer Methods in Applied Mechanics and Engineering*, vol. 361, p. 112732, 2020.

[90] S. Markidis, "The Old and the New : Can Physics-Informed Deep-Learning Replace Traditional Linear Solvers ?," *Frontiers in Big Data*, vol. 4, 2021.

[91] S. Jin, Z. Ma, and K. Wu, "Asymptotic-preserving neural networks for multiscale time-dependent linear transport equations," *arXiv preprint arXiv :2111.02541*, 2021.

[92] J. He and J. Xu, "Mgnet : a unified framework of multigrid and convolutional neural network," *Science china mathematics*, vol. 62, no. 7, pp. 1331–1354, 2019.

[93] B. Kim, V. C. Azevedo, N. Thuerey, T. Kim, M. Gross, and B. Solenthaler, "Deep fluids : A generative network for parameterized fluid simulations," in *Computer graphics forum*, vol. 38, pp. 59–70, Wiley Online Library, 2019.

[94] H. Gao, L. Sun, and J.-X. Wang, "Phygeonet : Physics-informed geometry-adaptive convolutional neural networks for solving parameterized steady-state pdes on irregular domain," *Journal of Computational Physics*, vol. 428, p. 110079, 2021.

[95] Kaggle, "State of machine learning and data science 2021," tech. rep., 2021.

[96] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow : A system for large-scale machine learning," p. 21.

[97] F. Chollet and others, "Keras : The Python Deep Learning library," *Astrophysics Source Code Library*, p. ascl :1806.022, June 2018.

[98] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch : An Imperative Style, High-Performance Deep Learning Library," in *Advances in Neural Information Processing Systems*, vol. 32, Curran Associates, Inc., 2019.

[99] B. J. Erickson, P. Korfiatis, Z. Akkus, T. Kline, and K. Philbrick, "Toolkits and libraries for deep learning," *Journal of digital imaging*, vol. 30, no. 4, pp. 400–405, 2017.

[100] H. M. Pandey and D. Windridge, "A comprehensive classification of deep learning libraries," in *Third international congress on information and communication technology*, pp. 427–435, Springer, 2019.

[101] G. Nguyen, S. Dlugolinsky, M. Bobák, V. Tran, Á. López García, I. Heredia, P. Malík, and L. Hluchỳ, "Machine learning and deep learning frameworks and libraries for large-scale data mining : a survey," *Artificial Intelligence Review*, vol. 52, no. 1, pp. 77–124, 2019.

[102] B. Fryxell, K. Olson, P. Ricker, F. Timmes, M. Zingale, D. Lamb, P. MacNeice, R. Rosner, J. Truran, and H. Tufo, "Flash : An adaptive mesh hydrodynamics code for modeling astrophysical thermonuclear flashes," *The Astrophysical Journal Supplement Series*, vol. 131, no. 1, p. 273, 2000.

[103] T. D. Arber, K. Bennett, C. S. Brady, A. Lawrence-Douglas, M. G. Ramsay, N. J. Sircombe, P. Gillies, R. G. Evans, H. Schmitz, A. R. Bell, and C. P. Ridgers, "Contemporary particle-in-cell approach to laser-plasma modelling," *Plasma Physics and Controlled Fusion*, vol. 57, pp. 1–26, Nov. 2015.

[104] G. Madec, R. Bourdallé-Badie, P.-A. Bouttier, C. Bricaud, D. Bruciaferri, D. Calvert, J. Chanut, E. Clementi, A. Coward, D. Delrosso, *et al.*, "Nemo ocean engine," 2017.

[105] A. Wallcraft, H. Hurlburt, E. Metzger, E. Chassignet, J. Cummings, and O. M. Smedstad, "Global ocean prediction using hycom," in *2007 DoD High Performance Computing Modernization Program Users Group Conference*, pp. 259–262, IEEE, 2007.

[106] E. Anderson, Z. Bai, C. Bischof, L. S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, *et al.*, *LAPACK users' guide*. SIAM, 1999.

[107] S. Balay, S. Abhyankar, M. Adams, J. Brown, P. Brune, K. Buschelman, L. Dalcin, A. Dener, V. Eijkhout, W. Gropp, *et al.*, "Petsc users manual," 2019.

[108] L. Lu, X. Meng, Z. Mao, and G. E. Karniadakis, "DeepXDE : A deep learning library for solving differential equations," *SIAM Review*, vol. 63, no. 1, pp. 208–228, 2021.

[109] S. Behnel, R. Bradshaw, C. Citro, L. Dalcin, D. S. Seljebotn, and K. Smith, "Cython : The best of both worlds," *Computing in Science & Engineering*, vol. 13, no. 2, pp. 31–39, 2010.

[110] L. D. Dalcin, R. R. Paz, P. A. Kler, and A. Cosimo, "Parallel distributed computing using python," *Advances in Water Resources*, vol. 34, no. 9, pp. 1124–1139, 2011.

[111] M. Curcic, "A parallel Fortran framework for neural networks and deep learning," *arXiv :1902.06714 [cs, stat]*, Mar. 2019. arXiv : 1902.06714.

[112] J. Ott, M. Pritchard, N. Best, E. Linstead, M. Curcic, and P. Baldi, "A Fortran-Keras Deep Learning Bridge for Scientific Computing," *arXiv :2004.10652 [cs]*, Aug. 2020. arXiv : 2004.10652.

[113] P. Ukkonen, R. Pincus, R. J. Hogan, K. Pagh Nielsen, and E. Kaas, "Accelerating radiation computations for dynamical models with targeted machine learning and code optimization," *Journal of Advances in Modeling Earth Systems*, vol. 12, no. 12, p. e2020MS002226, 2020.

[114] H. D. Pasinato, F. D. Gerosa, and E. A. Krumrick, "Reynolds stresses prediction using deep neural networks," *Mecánica Computacional*, vol. 38, no. 23, pp. 905–914, 2021.

[115] C. Badiali, P. J. Bilbao, F. Cruz, and L. O. Silva, "Machine learning-based models in particle-in-cell codes for advanced physics extensions," *arXiv preprint arXiv :2206.02937*, 2022.

[116] P. Ukkonen, "Exploring pathways to more accurate machine learning emulation of atmospheric radiative transfer," *Journal of Advances in Modeling Earth Systems*, vol. 14, no. 4, p. e2021MS002875, 2022.

[117] S. Partee, M. Ellis, A. Rigazzi, S. Bachman, G. Marques, A. Shao, and B. Robbins, "Using machine learning at scale in hpc simulations with smartsim : An application to ocean climate modeling," *arXiv preprint arXiv :2104.09355*, 2021.

[118] T. van der Heide, J. Kullgren, P. Broqvist, V. Bačić, T. Frauenheim, and B. Aradi, "Fortnet, a software package for training behler-parrinello neural networks," *arXiv preprint arXiv :2202.03118*, 2022.

[119] M. Tzoufras, A. R. Bell, P. A. Norreys, and F. S. Tsung, "A Vlasov–Fokker–Planck code for high energy density physics," *Journal of Computational Physics*, vol. 230, pp. 6475–6494, July 2011.

[120] E. M. Epperlein, G. J. Rickard, and A. R. Bell, "A code for the solution of the Vlasov-Fokker-Planck equation in 1-D or 2-D," *Computer Physics Communications*, vol. 52, pp. 7–13, Dec. 1988.

[121] L. Spitzer Jr and R. Härm, "Transport phenomena in a completely ionized gas," *Physical Review*, vol. 89, no. 5, p. 977, 1953.

[122] J.-L. Feaugeas, *L'entropie comme argument.* Habilitation à diriger la recherche, Université de Bordeaux, 2015. 186 pages.

[123] P. Mora and J. Luciani, "Nonlocal electron transport in laser created plasmas," *Laser and Particle Beams*, vol. 12, no. 3, pp. 387–400, 1994.

[124] N. F. Mott, "The scattering of fast electrons by atomic nuclei," *Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character*, vol. 124, no. 794, pp. 425–442, 1929.

[125] H. Bethe and W. Heitler, "On the stopping of fast particles and on the creation of positive electrons," *Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character*, vol. 146, no. 856, pp. 83–112, 1934.

[126] C. Møller, "Zur theorie des durchgangs schneller elektronen durch materie," *Annalen der Physik*, vol. 406, no. 5, pp. 531–585, 1932.

[127] H. Bhabha, "The scattering of positrons by electrons with exchange on dirac's theory of the positron," *Proceedings of the Royal Society of London. Series A-Mathematical and Physical Sciences*, vol. 154, no. 881, pp. 195–206, 1936.

[128] F. Sauter, "Über den atomaren photoeffekt in der k-schale nach der relativistischen wellenmechanik diracs," *Annalen der Physik*, vol. 403, no. 4, pp. 454–488, 1931.

[129] O. Klein and Y. Nishina, "Über die streuung von strahlung durch freie elektronen nach der neuen relativistischen quantendynamik von dirac," *Zeitschrift für Physik*, vol. 52, no. 11, pp. 853–868, 1929.

[130] B. Dubroca and J.-L. Feugeas, "Etude théorique et numérique d'une hiérarchie de modèles aux moments pour le transfert radiatif," *Comptes Rendus de l'Académie des Sciences - Series I - Mathematics*, vol. 329, pp. 915–920, Nov. 1999.

[131] R. Duclous, B. Dubroca, and M. Frank, "A deterministic partial differential equation model for dose calculation in electron radiotherapy," *Phys. Med. Biol.*, vol. 55, pp. 3843–3857, June 2010. Publisher : IOP Publishing.

[132] D. Wilson and T. Martinez, "The general inefficiency of batch training for gradient descent learning," *Neural networks : the official journal of the International Neural Network Society*, vol. 16, pp. 1429–51, Jan. 2004.

[133] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," *Journal of Machine Learning Research - Proceedings Track*, vol. 9, pp. 249–256, Jan. 2010.

[134] K. He, X. Zhang, S. Ren, and J. Sun, "Delving Deep into Rectifiers : Surpassing Human-Level Performance on ImageNet Classification," *IEEE International Conference on Computer Vision (ICCV 2015)*, vol. 1502, Feb. 2015.

[135] L. Guillopé, "Optimisation sous contrainte," p. 91.

[136] C. Lamy, B. Dubroca, P. Nicolaï, V. Tikhonchuk, and J.-L. Feugeas, "Modeling of electron nonlocal transport in plasmas using artificial neural networks," *Phys. Rev. E*, vol. 105, p. 055201, May 2022.

[137] J.-L. Feugeas, P. Nicolai, X. Ribeyre, G. Schurtz, V. Tikhonchuk, and M. Grech, "Modeling of two-dimensional effects in hot spot relaxation in laser-produced plasmas," *Physics of Plasmas*, vol. 15, no. 6, p. 062701, 2008.

[138] M. Hussein, C. H. Clark, and A. Nisbet, "Challenges in calculation of the gamma index in radiotherapy – Towards good practice," *Physica Medica*, vol. 36, pp. 1–11, Apr. 2017.

[139] M. Chen, W. Lu, Q. Chen, K. Ruchala, and G. Olivera, "Efficient gamma index calculation using fast Euclidean distance transform," *Phys. Med. Biol.*, vol. 54, pp. 2037–2047, Apr. 2009.

[140] M. Havaei, A. Davy, D. Warde-Farley, A. Biard, A. Courville, Y. Bengio, C. Pal, P.-M. Jodoin, and H. Larochelle, "Brain tumor segmentation with deep neural networks," *Medical image analysis*, vol. 35, pp. 18–31, 2017.

[141] B. Ibragimov and L. Xing, "Segmentation of organs-at-risks in head and neck ct images using convolutional neural networks," *Medical physics*, vol. 44, no. 2, pp. 547–557, 2017.

[142] H.-C. Shin, H. R. Roth, M. Gao, L. Lu, Z. Xu, I. Nogues, J. Yao, D. Mollura, and R. M. Summers, "Deep convolutional neural networks for computer-aided detection : Cnn architectures, dataset characteristics and transfer learning," *IEEE transactions on medical imaging*, vol. 35, no. 5, pp. 1285–1298, 2016.

[143] A. Esteva, B. Kuprel, R. A. Novoa, J. Ko, S. M. Swetter, H. M. Blau, and S. Thrun, "Dermatologist-level classification of skin cancer with deep neural networks," *nature*, vol. 542, no. 7639, pp. 115–118, 2017.

[144] X. Han, "Mr-based synthetic ct generation using a deep convolutional neural network method," *Medical physics*, vol. 44, no. 4, pp. 1408–1419, 2017.

[145] L. Gjesteby, Q. Yang, Y. Xi, Y. Zhou, J. Zhang, and G. Wang, "Deep learning methods to guide ct image reconstruction and reduce metal artifacts," in *Medical Imaging 2017 : Physics of Medical Imaging*, vol. 10132, p. 101322W, International Society for Optics and Photonics, 2017.

[146] C. Dong, C. C. Loy, K. He, and X. Tang, "Image super-resolution using deep convolutional networks," *IEEE transactions on pattern analysis and machine intelligence*, vol. 38, no. 2, pp. 295–307, 2015.

[147] A. Akhavanallaf, R. Mohammadi, I. Shiri, Y. Salimi, H. Arabi, and H. Zaidi, "Personalized brachytherapy dose reconstruction using deep learning," *Computers in biology and medicine*, vol. 136, p. 104755, 2021.

[148] Y. Song, J. Hu, Y. Liu, H. Hu, Y. Huang, S. Bai, and Z. Yi, "Dose prediction using a deep neural network for accelerated planning of rectal cancer radiotherapy," *Radiotherapy and Oncology*, vol. 149, pp. 111–116, 2020.

[149] M. S. Lee, D. Hwang, J. H. Kim, and J. S. Lee, "Deep-dose : a voxel dose estimation method using deep convolutional neural network for personalized internal dosimetry," *Scientific reports*, vol. 9, no. 1, pp. 1–9, 2019.

[150] S. Shiraishi and K. L. Moore, "Knowledge-based prediction of three-dimensional dose distributions for external beam radiotherapy," *Medical Physics*, vol. 43, pp. 378–387, 2016.

[151] A. Vasseur, L. Makovicka, E. Martin, M. Sauget, S. Contassot-Vivier, and J. Bahi, "Dose calculations using artificial neural networks : A feasibility study for photon beams," *Nuclear Instruments and Methods in Physics Research Section B : Beam Interactions with Materials and Atoms*, vol. 266, pp. 1085–1093, 2008.

[152] J. Bahi, S. Contassot-Vivier, L. Makovicka, E. Martin, and M. Sauget, "Neural Network Based Algorithm for Radiation Dose Evaluation in Heterogeneous Environments," vol. 4132, pp. 777–787, 2006.

[153] R. Mathieu, E. Martin, R. Gschwind, L. Makovicka, S. Contassot-Vivier, and J. Bahi, "Calculations of dose distributions using a neural network model," *Physics in medicine and biology*, vol. 50, pp. 1019–28, 2005.