Adaptation d'approches connexionnistes non supervisées pour l'analyse de contenus d'images et de sons

Benjamin Chamand

THÈSE

En vue de l'obtention du
DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par l'Université Toulouse 3 - Paul Sabatier

Présentée et soutenue par

Benjamin CHAMAND

Le 16 mai 2023

Adaptation d'approches connexionnistes non supervisées pour l'analyse de contenus d'images et de sons

Ecole doctorale : **EDMITT - Ecole Doctorale Mathématiques, Informatique et Télécommunications de Toulouse**

Spécialité : **Informatique et Télécommunications**

Unité de recherche :
IRIT : Institut de Recherche en Informatique de Toulouse

Thèse dirigée par
Philippe JOLY

Jury

M. Emmanuel DELLANDREA, Rapporteur
M. Jean MARTINET, Rapporteur
Mme Valérie GOUET-BRUNET, Examinatrice
M. Frédéric LERASLE, Examinateur
M. Philippe JOLY, Directeur de thèse

Résumé

Le cerveau humain est capable de traiter temporellement une vaste quantité d'informations sensorielles très efficacement et de les combiner de manières à en tirer des conclusions logiques. Il peut également apprendre et s'adapter en fonction de son expérience. Cette richesse fait que, depuis l'Antiquité, l'Homme est fasciné par les secrets de la cognition humaine et cherche à les comprendre. Ce domaine, plus connu de nos jours sous le terme de neurosciences cognitives, est un terrain de jeu ouvert grâce aux avancées des différentes modélisations mathématiques des mécanismes connexionnistes.

À travers ces travaux de thèse, nous cherchons à étudier le comportement de ces approches connexionnistes, formées de manière non supervisée, sur la génération de représentations de données images et audios. Plus particulièrement, nous nous intéressons aux réseaux de neurones impulsionnels qui présentent certains avantages théoriques comme le traitement fin de données temporelles et une consommation énergétique maitrisée, à l'opposé des réseaux de neurones artificiels très utilisés de nos jours. Nous nous intéressons également à l'adaptation des modèles à différentes échelles : de leur topologie générale à la compréhension d'éléments permettant d'obtenir de meilleures performances sur la classification des représentations.

La première partie présente le fonctionnement du neurone biologique jusqu'à sa modélisation computationnelle afin de s'en inspirer pour générer des représentations de nos données. Cependant, la rareté de ces approches dans le domaine de l'informatique appliquée nous oblige à développer notre propre simulateur pour valider différents modèles développés dans la recherche neurocomputationelle, montrant ainsi la dure réalité d'apprentissage de ces derniers. Enfin, cette difficulté nous a conduits à adapter nos modèles pour accepter des récentes stratégies d'apprentissage présentes dans le domaine de l'apprentissage profond, en particulier l'apprentissage auto-supervisé.

En raison de la diversité et de la sensibilité des hyperparamètres présents dans les modèles connexionnistes, la seconde partie de notre étude se focalise à comprendre comment adapter des composants d'un système pour améliorer ses performances. En d'autres termes, nous cherchons des intuitions permettant de faciliter notre décision sur le choix des valeurs à fixer pour des hyperparamètres quels qu'ils soient ou la définition de nouvelles fonctions de coût. Pour faciliter l'étude, nous nous plaçons dans le cadre d'un modèle linéaire de classification des représentations extraites, pour ensuite étudier d'une part un hyperparamètre important jouant sur la performance finale du modèle : la température, puis sur une tâche plus large telle que la compréhension de la justesse – *accuracy* – d'un modèle. Pour cela, une chaîne de traitement généralisable est proposée afin d'extraire une heuristique sur la température. Puis, en y ajoutant un module de régression symbolique, nous avons pu montrer que nos résultats sont explicables et en adéquation avec des décennies de recherche.

Mots clés : approche connexionniste, réseaux de neurones à impulsions, apprentissage non supervisé, apprentissage de représentations, explicabilité.

Abstract

The human brain is capable of processing a vast amount of sensory information in a very efficient way and combining it in a way that allows it to draw logical conclusions. It can also learn and adapt according to its experience. This wealth of information has fascinated mankind since antiquity, and he has always sought to understand the secrets of human cognition. This field, better known today as cognitive neuroscience, is an open playing field thanks to the advances of various mathematical models of connectionist mechanisms.

Through this thesis work, we aim to study the behavior of these connectionist approaches, trained in an unsupervised manner, on the generation of image and audio data representations. More specifically, we are interested in spiking neural networks, which have some theoretical advantages such as fine-grained processing of temporal data and low energy consumption, as opposed to artificial neural networks, which are widely used nowadays. We are also interested in the adaptation of the models at different scales: from their general topology to the understanding of elements allowing to obtain better performances on the classification of representations.

The first part presents the functioning of the biological neuron up to its computational modeling in order to be inspired to generate representations of our data. However, the scarcity of these approaches in the field of applied computing forces us to develop our own simulator to validate different models developed in neurocomputational research, thus showing the hard reality of learning them. Finally, this difficulty led us to adapt our models to accept recent learning strategies present in the field of deep learning, in particular self-supervised learning.

Due to the diversity and sensitivity of hyperparameters present in connectionist models, the second part of our study focuses on understanding how to adapt components of a system to improve its performance. In other words, we are looking for insights to facilitate our decision on the choice of values to set for any hyperparameters or the definition of new cost functions. To facilitate the study, we place ourselves in the framework of a linear model of classification of the extracted representations, to then study on the one hand an important hyperparameter playing on the final performance of the model: the temperature, and on the other hand a larger task such as the understanding of the accuracy of a model. For this, a generalizable processing chain is proposed to extract a heuristic on the temperature. Then, by adding a symbolic regression module, we were able to show that our results are explainable and consistent with decades of research.

Keywords: connectionist approach, spiking neural networks, unsupervised learning, representation learning, explainability.

Remerciements

Je tiens à exprimer ma plus sincère gratitude à Philippe, mon directeur de thèse, sans qui cette thèse de doctorat n'aurait pas abouti. Il a eu la lourde tâche de me suivre dans cette aventure, explorant ainsi de nombreux axes de recherche. Qu'il s'agisse de surmonter des obstacles, de faire face à des résultats décevants ou de faire preuve de persévérance face aux défis scientifiques, il m'a toujours apporté son soutien inconditionnel. Sa confiance en mes capacités et sa conviction en la valeur de ma recherche ont été des sources de réconfort et de détermination lorsque j'en avais le plus besoin.

Je remercie naturellement l'ensemble des membres de mon jury de thèse pour avoir pris le temps de lire mon manuscrit ainsi que pour leurs retours. Merci aussi à Agnès pour sa précieuse aide durant toutes les démarches administratives de cette thèse.

Je tiens également à exprimer ma reconnaissance envers tous mes collègues de l'équipe SAMoVA, ainsi qu'envers mon épouse, Émilie, ma famille et mes amis, en particulier Olivier et Antoine. Les discussions enrichissantes et les collaborations fructueuses ont été essentielles pour nourrir ma curiosité scientifique en dehors de mes activités de recherches principales.

Merci infiniment !

En essayant continuellement on finit par réussir.
Donc : plus ça rate, plus on a de chance que ça marche.

Devise Shadocks

Table des matières

Abrévations

Pour des raisons de lisibilité, il a été choisi de ne pas traduire les sigles et acronymes. Le tableau ci-dessous récapitule les principaux sigles et acronymes utilisés tout au long du manuscrit.

Sigle ou acronyme	Signification anglophone	Traduction francophone
ANN	*Artificial Neural Network*	Réseau de neurones artificiels
API	*Application Programming Interface*	Interface de programmation
AVC	*Audio-Visual Correspondance*	Correspondance audiovisuelle
BCM	*Bienenstock-Cooper-Munro*	Bienenstock-Cooper-Munro
BP	*BackPropagation*	Rétropropagation du gradient
BP-STDP	*BackPropagation STDP*	STDP avec rétropropagation
BPTT	*Back-Propagation Through Time*	Rétropopagation temporelle
CE	*Cross-Entropy*	Entropie croisée
CNN	*Convolutional Neural Network*	Réseau de neurones convolutifs
DBN	*Deep Belief Network*	Réseau de croyances profonds
DoG	*Difference of Gaussians*	Différence de gaussiennes
EA	*Evolutionary Algorithm*	Algorithme évolutionniste
EBM	*Energy-Based Model*	Modèle fondé sur l'énergie
EIF	*Exponential Integrate-and-Fire*	Intègre-et-tire exponentiel
EPSP	*Excitatory PostSynaptic Potential*	Potentiel post-synaptique excitateur
ESN	*Echo State Network*	Réseau à états échoïques
FC	*Fully Connected*	Couche entièrement connectée
FFNN	*FeedForward Neural Network*	Réseau de neurones à propagation avant
FLOPS	*FLoating-point Operations Per Second*	Opérations en virgule flottante par seconde
GA	*Genetic Algorithm*	Algorithme génétique

Sigle ou acronyme	Signification anglophone	Traduction francophone
GP	*Genetic Programming*	Programmation génétique
IA	*Artificial Intelligence*	Intelligence artificielle
IF	*Integrate-and-Fire*	Intègre-et-tire
IPSP	*Inhibitory PostSynaptic Potential*	Potentiel post-synaptique inhibiteur
kNN	*K-Nearest Neighbors*	K plus proches voisins
LDA	*Linear Discriminant Analysis*	Analyse discriminante linéaire
LI	*Leaky Integrate*	Intègre avec fuite
LIF	*Leaky Integrate-and-Fire*	Intègre-et-tire avec fuite
LSM	*Liquid State Machine*	Machine à état liquide
LTD	*Long-Term Depression*	Dépression à long terme
LTP	*Long-Term Potentiation*	Potentialisation à long terme
MFCC	*Mel-Frequency Cepstral Coefficients*	Coefficients cepstraux en fréquences Mel
ML	*Machine Learning*	Apprentissage machine
MLP	*MultiLayer Perceptron*	Perceptron multicouche
MSE	*Mean Square Error*	Erreur quadratique moyenne
ODE	*Ordinary Differential Equation*	Équation différentielle ordinaire
PCA	*Principal Component Analysis*	Analyse en composantes principales
PSP	*PostSynaptic Potential*	Potentiel post-synaptique
QIF	*Quadratic Integrate-and-Fire*	Intègre-et-tire quadratique
R-STDP	*Reward-modulated STDP*	STDP modulée par la récompense
ReLU	*Rectifier Linear Unit*	Unité de rectification linéaire
ReSuMe	*Remote Supervised Method*	Méthode supervisé à distance
RNN	*Recurrent Neural Network*	Réseau de neurones récurrents
ROC	*Rank-Order Coding*	Codage par rang
SCNN	*Spiking CNN*	Réseau de neurones convolutifs à impulsions
SGD	*Stochastic Gradient Descent*	Algorithme du gradient stochastique
SNN	*Spiking Neural Network*	Réseau de neurones à impulsions
SOL	*Sounding Object Localization*	Localisation d'objets sonores
SR	*Symbolic Regression*	Régression symbolique
SRM	*Spike Response Model*	Modèle de réponse aux impulsions
SRNN	*Spiking RNN*	Réseau de neurones à impulsions récurrents
SSL	*Self-Supervised Learning*	Apprentissage auto-supervisé

Sigle ou acronyme	Signification anglophone	Traduction francophone
STDP	*Spike-Timing-Dependent Plasticity*	Plasticité fonction du temps d'occurrence des impulsions
SVM	*Support Vector Machine*	Machine à vecteurs de support
TTFS	*Time-To-First-Spike*	Temps avant la première impulsion
WTA	*Winner-Take-All*	Le gagnant rafle la mise
ZCA	*Zero Components Analysis*	Analyse des composants nuls

Introduction générale

Introduction

Ce chapitre retrace la réflexion qui m'a conduit de la motivation initiale de cette thèse académique jusqu'aux hypothèses qui m'ont amené à orienter différemment mes travaux. À la suite de cela, différents axes de recherches ont été explorés pour répondre à plusieurs questionnements qui je vais présenter dans la suite.

Motivation initiale

Dans la nature, la manifestation d'un événement visuel est souvent accompagnée d'un événement sonore. Par le fait, nous associons plus ou moins inconsciemment à tout objet un ensemble de sons qu'il est susceptible d'émettre en fonction de l'usage qui en est fait. Si cet usage induit des types de son particulier, la nature même de l'objet – sa composition, ses propriétés, sa forme – influe également sur ces sons. Ils sont ainsi différents si l'objet est jeté, manipulé ou frappé, et dépendent aussi du fait que l'objet soit fait de tissu, de bois ou de métal par exemple.

Afin d'illustrer ces propos, prenons l'exemple d'un objet courant, une bouteille en verre. En fonction de son utilisation, différents sons peuvent être associés à cette bouteille. Si on la manipule doucement, le son produit peut être un léger tintement lorsque les doigts tapotent délicatement sur la surface en verre. Si on la frappe plus fort, le son devient plus fort et résonnant, avec une sorte de claquement aigu. Maintenant, imaginons que la bouteille soit jetée dans une poubelle en métal. Dans ce cas, le son sera complètement différent, avec un bruit de verre qui s'écrase contre le métal, suivi peut être d'un bruit de rebondissement. Ainsi, la nature même de l'objet (la bouteille en verre) et son interaction avec d'autres matériaux (les doigts, le métal) déterminent les sons spécifiques qui lui sont associés, créant une expérience auditive distincte pour chaque utilisation de la bouteille.

Il existe par le fait une relation entre la nature de l'objet, sa manipulation et le son émis de sorte qu'une information sur l'un de ces trois aspects renseigne sur le potentiel des deux autres.

Il est fréquent, dans le domaine de la production audiovisuelle, d'ajouter des bruitages sur la bande son lors de la phase d'édition. Ces bruitages sont censés être directement liés aux objets et aux actions visibles dans une scène. Mais, il arrive que le bruitage vienne soutenir une ambiance ou apporter des éléments diégétiques [1] à la narration. Par exemple, un son de cloche et un chien qui aboie dans le lointain créeront une atmosphère champêtre pour des scènes tournées en studio. Cette partie du son entendu

1. Dans le contexte du cinéma et de la musique de film, le terme « diégétique » se réfère aux sons présents dans l'univers fictionnel d'une histoire, audibles par les personnages, tels que les dialogues, les bruits ambiants et la musique qui fait partie de l'histoire.

ne correspond alors pas à ce qui est donné à voir, mais suffit à créer l'idée de l'existence d'un village proche de la scène.

Il arrive parfois que, pour des raisons techniques, la bande son d'un document audiovisuel soit désynchronisée avec la bande vidéo. Cela suscite dans ce cas une sorte de malaise, de non-compréhension car l'expérience est en contradiction avec une conception bien précise du monde, acquise depuis notre naissance.

Pour aller plus loin, les fausses informations qui prolifèrent aujourd'hui sur différents médias s'appuient souvent sur une remédiation d'enregistrements en les détournant de leur sens d'origine. Si le nouveau sens donné est le plus souvent porté par une narration textuelle ou parlée, il est parfois soutenu par un détournement soit de l'image, soit du son.

Sur la base de ces observations, le sujet initial de cette thèse était :

> Analyse de la consistance de concepts audiovisuels

L'objectif portait sur l'analyse des relations entre un concept visuel et un son de manière à établir la consistance de l'ensemble – ou non – à travers le traitement simultané des deux médias, permettant par la suite de détecter une désynchronisation intentionnelle ou non entre ces deux modalités. Les termes de « concept audiovisuel » et de « consistance » étaient à définir pour cadrer le sujet.

Concept audiovisuel Un concept audiovisuel est une idée ou une vision qui est exprimée à travers l'utilisation combinée de l'audio et de la vidéo. C'est donc un terme très vaste mais dont la connaissance peut facilement être extraite grâce à une disponibilité immense et grandissante de données vidéos disponibles sur le web. Dans notre cas, un concept sera défini par une action, un objet, un élément, etc. qui est fortement représenté dans nos données, comme l'action de jouer d'un instrument ou tout simplement le concept du piano.

Consistance Le second terme important à définir est la « consistance » d'un concept qui désigne l'absence de contradictions ou d'incohérences dans la manière dont le concept est présenté ou utilisé, permettant ainsi de maintenir la compréhension du concept. Dans notre cas, le problème consistait donc à analyser les différents concepts présents dans une séquence audiovisuelle et de mesurer un score de consistance entre les deux modalités.

Ce terme ne doit pas être confondu avec la « cohérence » d'un concept qui désigne la manière dont le concept est relié à d'autres concepts ou idées et la façon dont il s'intègre dans un ensemble plus large de connaissance, permettant ainsi de faciliter la compréhension et l'apprentissage tout en évitant la confusion de la scène. En d'autres termes, la cohérence caractérise le degré d'harmonie globale entre tous les éléments visuels et sonores d'un média permettant de créer une expérience cohésive.

Où se place la recherche actuelle sur ce sujet ? Cette tâche est apparentée à la problématique de fusion multimodale audio-visuelle, et plus particulièrement à deux catégories sous-jacentes : la correspondance audiovisuelle (AVC) et la localisation d'objets

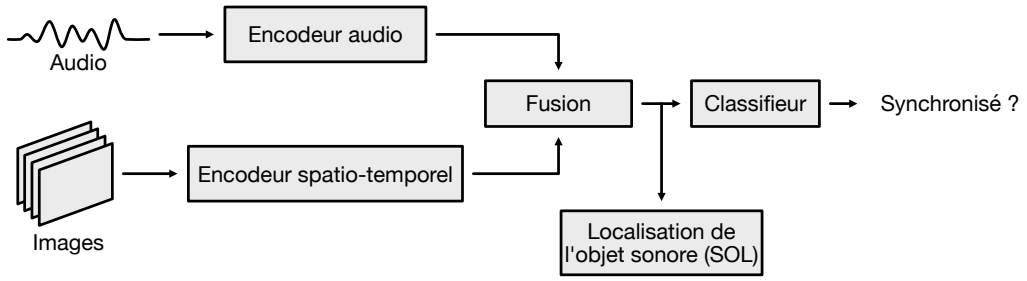


FIGURE 1 – Architecture généralement utilisée pour la tâche de AVC consistant en une classification binaire dont le but est de savoir comment aligner et fusionner les flux audio et images.

sonores (SOL). Le but de cette tâche est de découvrir des relations sémantiques entre les modalités visuelles et sonores, et de localiser ces sources dans le média audiovisuel. Pour ce faire, les méthodes utilisées de nos jours utilisent des techniques d'apprentissage profond afin d'obtenir une mesure de consistance entre l'audio et l'image basé sur la synchronicité des modalités [1, 2, 4, 8]. L'apprentissage est réalisé généralement de manière auto-supervisée, c'est-à-dire que le réseau de neurones utilise ses propres caractéristiques pour apprendre à représenter les données de manière significative en générant, par exemple, des paires positives et négatives depuis les données elles-mêmes. Ainsi, en reprenant l'exemple précédent sur le son de cloche, la cloche qui tinte hors de la scène rend l'événement audio non synchronisé avec l'événement visuel et aura donc une prédiction négative malgré la cohérence générale de celle-ci. Il en ira de même si le clocher de l'église est présent dans la scène accompagné d'un son de musique électronique.

La figure 1 présente la chaîne de traitement générale utilisée dans la tâche de AVC. Chaque modalité est encodée sous la forme d'un vecteur par un extracteur de caractéristiques (généralement basé sur des réseaux de neurones convolutifs (CNNs)), puis la représentation de chaque modalité est fusionnée suivant différentes stratégies.

Vers une approche de connexionnisme impulsionnel

Cette mesure de consistance entre les modalités visuelle et sonore peut donc être facilement modélisée par une extraction sémantique issue d'approches connexionnistes associées à chaque modalité, puis en appliquant une stratégie de fusion entre ces différentes modalités permettant d'en extraire des caractéristiques intrinsèques. L'approche connexionniste la plus populaire de nos jours est celle basé sur l'apprentissage profond dont l'état de l'art évolue rapidement. En 5 ans, nous sommes passés de simples réseaux convolutifs très profonds à des systèmes complexes comme les « transformers », mais ils posent toujours la contrainte d'avoir un jeu de données relativement important pour obtenir une convergence du modèle. Mais – et c'est un point commun entre toutes ces approches – la temporalité des données est toujours gérée à partir d'une taille fixe d'analyse, contrairement au cerveau humain qui a la capacité de traiter l'information en continu. Même si aujourd'hui, les méthodes artificielles avancées, telles que l'appren-

tissage profond [5], sont capables de rivaliser avec les humains dans de nombreuses tâches grâce à la masse de données exploitées, l'exploration de méthodes alternatives inspirées du fonctionnement réel des neurones restent limitée, en raison, entre-autres, de performances faibles et d'une connaissance moins approfondie de leur modélisation. Pourtant, à l'air de la sobriété énergétique, des méthodes basées sur l'apprentissage profond peuvent nécessiter des centaines, voire plusieurs milliers de Watts contre environ 20 Watts pour le cerveau [3]. Cet écart énergétique s'explique par des modes de fonctionnement différents. D'un côté, le cerveau utilise des unités désynchronisées qui fonctionnent localement tant au niveau des calculs que pour la mémoire. De l'autre côté, l'apprentissage profond basé sur l'architecture de von Neumann utilise un mode de fonctionnement synchronisé entre les différents modules centralisés. Ce mode de fonctionnement naturel permettrait aussi de gagner en confidentialité grâce à un traitement local des données sans avoir besoin de les envoyer en ligne sur des machines de calculs performantes. Les réseaux de neurones à impulsions (SNNs), ou connexionnisme impulsionnel, apportent une modélisation plausible du cerveau en simulant différentes propriétés des neurones naturelles. Des chercheurs mettent ainsi au point des plateformes neuromorphiques qui peuvent consommer entre 100k et 300k $\times$ moins d'énergie que les architectures traditionnelles [7].

C'est pour cela que je me suis intéressé au connexionnisme impulsionnel pour le traitement de données temporelles afin de générer des vecteurs caractéristiques de nos données pouvant être exploités a posteriori.

Approche connexionniste impulsionnelle Après une première revue de la littérature, il est rapidement apparu que la création d'un modèle informatique d'analyse audiovisuel inspiré par le cerveau était un sujet quasi-inexploré. Les neuroscientifiques n'ont pas entièrement identifié tous les processus biologiques qui permettent au cerveau de comprendre et d'apprendre des concepts temporels. Ils ont certes formulé diverses hypothèses depuis des décennies pouvant être testées sur nos tâches. Cependant, la transposition de ces modèles sur des données réelles reste encore marginale, et la communauté scientifique active sur cette problématique issue à la base de la neuroscience reste encore réduite.

Pourtant, plusieurs propriétés des SNNs sont intéressantes à explorer telle que l'apprentissage en ligne des données de manière non supervisée. Biologiquement, il existe un système d'augmentation et de diminution des connexions synaptiques, respectivement appelé potentialisation à long terme (LTP) et dépression à long terme (LTD). Ce mécanisme est appelé plasticité fonction du temps d'occurrence des impulsions (STDP). Des études expérimentales ont permis d'observer qu'il permet entre autres de localiser un motif spatio-temporel répétitif [6]. Ce mécanisme, appliqué à des données réelles, apporterait une nouvelle manière de classifier des concepts.

Analyse de contenus d'images et de sons L'utilisation des SNNs est un véritable défi sur des données réelles, et encore plus sur des données audiovisuelles telles que celles utilisés dans le domaine de l'AVC. Il a donc été jugé plus réaliste de commencer par comprendre les mécanismes d'un modèle réseau de neurones à impulsions (SNN) sur des données de types images et sons avant d'attaquer les problèmes de fusion audiovisuelle pour mesurer la consistance. Il aurait pu être intéressant de s'attaquer aux données directement issues de capteur adapté au formalisme d'événements temporels comme des vidéos issus des caméras événementielles, plus couramment appelées DVS –

Dynamic Vision Sensor. Contrairement aux caméras conventionnelles, chaque pixel des caméras DVS réagit indépendamment aux variations de luminosité, et n'envoie aucun signal lorsque aucun changement local n'est détecté. Mais, par la diversité des données collectées depuis des décennies, il apparaît plus logique d'utiliser ces données pour l'apprentissage d'un modèle et de tester le comportement de celui-ci durant l'inférence avec des capteurs événementiels.

Le sujet final présenté dans ce manuscrit traite donc de :

> Adaptation d'approches connexionnistes non supervisées pour l'analyse de contenus d'images et de sons.

Adaptation Le terme d' « adaptation » est utilisé ici pour désigner le processus de modification d'un modèle pour le rendre plus précis et adapté à un ensemble de données ou une tâche spécifique. Pour cela, il existe plusieurs manières de procéder à l'adaptation d'un modèle, par exemple, en y ajoutant ou supprimant des variables, changer la forme de la fonction de perte, ou utiliser une méthode d'optimisation différente pour ajuster les paramètres. Dans mon cas, l'adaptation va concerner la portabilité d'un modèle SNN à travers différentes modalités, mais je vais aussi chercher à comprendre comment on peut modifier un modèle pour mieux répondre une tâche spécifique.

Explicabilité L'explicabilité n'est pas un terme directement lié au sujet de thèse, mais y est sous-entendu dès lors que l'on parle d'adaptation du modèle. L'explicabilité d'un modèle d'apprentissage machine (ML) se réfère à la capacité d'un système à rendre compte de sa décision ou de ses prédictions d'une manière compréhensible pour les utilisateurs, les décideurs et les experts du domaine. Cela inclut la compréhension des mécanismes sous-jacents de l'apprentissage automatique, des hypothèses sur lesquelles le modèle est basé, ainsi que des facteurs qui ont influencé les prédictions finales. Cela permet entre autres de comprendre comment nous pouvons jouer sur les paramètres pour influencer de telle manière le comportement du modèle ou de trouver des heuristiques nous aidant à améliorer certains points de l'apprentissage du modèle.

Approche non supervisée Un autre concept qui m'a toujours fasciné, et que je voulais aussi explorer durant cette thèse, est celui des modèles pouvant ingérer des données sans annotations, générant ainsi des vecteurs de représentation de nos données initiales – *embeddings*. Ces représentations vectorielles capturent de l'information sémantique ou encore des caractéristiques visuelles. Cette approche, appelée « apprentissage non supervisé », permet ainsi de découvrir des structures cachées dans les données initiales. Pour aller plus loin, dans le cas des données multimodales, cela permettrait d'apprendre à combiner les différentes modalités de manière cohérente. Par exemple, l'utilisation de l'apprentissage non supervisé peut permettre de découvrir des motifs – *patterns* – communs à plusieurs modalités de données, ou de détecter des relations entre ces modalités qui ne sont pas explicitement étiquetées.

Pour conclure, cette thèse se place dans le cadre plus général de l'apprentissage de représentations où je cherche dans un premier temps à générer des représentations grâce

à des modèles de neurones impulsionnels, et dans un second temps, à comprendre comment améliorer les performances des modèles basés sur des vecteurs de représentations.

Verrous et enjeux scientifiques

Dans cette thèse, je m'interroge donc sur le fonctionnement des SNNs et questionne l'applicabilité des méthodes, créées initialement pour de la modélisation biologique, au traitement de données réelles telles que des données images et sons. La communauté de recherche travaillant sur l'exploitation des SNNs s'est peu intéressée aux spécificités de ces méthodes pour leur exploitation sur des tâches de fusion multimodale. En d'autres termes, je cherche à comprendre leurs mécanismes en partant de la définition même du neurone biologique jusqu'à sa simulation pour le soumettre à des tâches classiques en reconnaissance des formes sur des données d'image et de sons.

Mes travaux visent à répondre aux questions suivantes, dont certaines sont liées aux connexionnismes impulsionnels, et d'autres à la spécificité des représentations générées par des modèles connexionnistes au sens général :

1. Quel lien existe-t-il entre l'apprentissage profond très utilisé de nos jours et un SNN ? Ce dernier est-il adapté au traitement de données réelles pour générer des représentations discriminantes ?
2. L'apprentissage d'un SNN par des mécanismes bio-inspirés étant relativement complexe, quelle stratégie pouvons-nous employer pour faciliter l'apprentissage d'un SNN ? Les performances sont-elles similaires ?
3. Les architectures de réseaux de neurones présentent actuellement une multitude d'hyperparamètres. Il en existe encore davantage dans le domaine des SNNs. Existe-t-il une méthode pour fixer ces hyperparamètres à l'initialisation du modèle ?
4. Pour aller plus loin sur cette dernière question et en connaissance de la recherche actuelle, pouvons-nous obtenir d'une intelligence artificielle (AI) une heuristique à une tâche donnée pouvant être facilement explicable ?

Pour répondre à ces questions, j'ai commencé par établir un état de l'art général dédié aux SNNs encore peu connu par la communauté scientifique en essayant de créer un parallèle avec l'apprentissage profond tel que nous le connaissons actuellement.

Je me suis ensuite lancé dans le développement d'un simulateur de SNNs basé sur PyTorch pouvant simuler de larges réseaux de neurones utilisant initialement des règles d'apprentissage de type hebbien. Ce dernier m'a permis de tester différentes architectures développées dans la littérature et de me confronter aux différents problèmes que ces méthodes peuvent poser.

Pour pallier certaines difficultés dues à l'apprentissage des SNNs, je me suis ensuite tourné vers le concept d'apprentissage auto-supervisé largement développé dans le domaine de l'apprentissage profond pour l'adapter dans un contexte de connexionnisme impulsionnel. Cela a été rendu possible grâce à la récente démocratisation de l'estimation du gradient – *surrogate gradient* – permettant d'offrir la possibilité d'utiliser l'apprentissage par descente du gradient directement sur des SNNs.

Pour répondre aux questions 3 et 4, j'ai analysé les représentations issues de plusieurs modèles de réseaux de neurones profonds pré-entrainés sur différents jeux de données pour trouver une méthodologie permettant de produire des heuristiques sur la valeur d'un hyperparamètre d'un modèle. J'ai ensuite étendu cette chaîne de traitement en intégrant un module de régression de telle sorte que ce nouveau modèle génère une heuristique explicable.

Organisation du manuscrit

Ce manuscrit se compose de cinq chapitres, répartis en deux grandes parties.

Première partie La première partie est constituée des trois premiers chapitres et correspond à la définition du connexionnisme impulsionnel, son implémentation et son application sur des données réelles.

Le chapitre 1 établit un état de l'art sur le concept des réseaux impulsionnels allant du principe originel d'un neurone biologique jusqu'à ses différents formalismes mathématiques, en passant par une comparaison entre l'approche du connexionnisme fréquentiel, plus connu sous le nom de réseau de neurones artificiels (ANN), et le connexionnisme temporel, appelé SNN. Nous décrivons ensuite comment l'information transite entre les neurones aussi bien dans les stratégies d'encodage des données que du décodage de l'information. Dans une dernière section, nous passons en revue les principales méthodes existantes pour entrainer ces modèles ainsi que les mécanismes intrinsèques permettant d'aider à sa convergence.

Le chapitre 2 expose dans un premier temps les différents simulateurs existants. Nous décrivons ensuite notre positionnement et le développement d'une bibliothèque permettant d'instancier un SNN avec des mécanismes bio-inspirés pour sa formation. Nous cherchons ensuite un moyen d'accélérer la vitesse d'apprentissage en acceptant le calcul parallèle des données d'entrainement. Puis, nous présentons les performances de notre simulateur sur des tâches de classification de données d'images ou de sons.

Ayant montré dans le chapitre 2 diverses difficultés rencontrées pour l'apprentissage d'un SNN avec les mécaniques bio-inspirés, le chapitre 3 défini une nouvelle manière d'entrainer un SNN de manière non supervisée en transposant des méthodes issues de l'apprentissage profond au SNN. Pour cela, nous nous intéressons plus particulièrement aux méthodes d'apprentissage auto-supervisé qui gagne en popularité grâce à la qualité des représentations générées et sa facilité de mise en œuvre. Pour l'appliquer au domaine des SNNs, nous détaillons les mécanismes permettant l'usage de la descente de gradient. Nous adaptons ensuite un modèle d'apprentissage auto-supervisé dans le contexte des données temporelles. Enfin, nous validons l'expérience sur des données d'images.

Seconde partie La seconde partie se compose de deux chapitres axés sur l'assistance des décisions.

Après avoir noté l'importante influence des différents hyperparamètres d'un SNN et plus largement de tous les hyperparamètres présents dans les modèles connexionnistes, le chapitre 4 se concentre sur l'étude d'un hyperparamètre : la température, afin de gagner un peu d'intuition sur le choix de sa valeur. Pour cela, nous nous plaçons dans un cadre dans lequel nous disposons déjà d'extracteurs de représentations. Nous cherchons à trouver une heuristique sur la valeur de la température présente dans notre classifieur linéaire. Dans un premier temps, nous cherchons à correctement isoler cet hyperparamètre pour éviter toute influence de sa valeur dans la fonction de coût lors de l'optimisation du modèle. Puis, nous proposons une chaine de traitements généralisable à d'autres tâches permettant d'extraire une heuristique sur cet hyperparamètre.

Le chapitre 5 améliore la chaine de traitement proposée dans le chapitre 4 en proposant d'extraire des connaissances par la résolution d'une tâche prétexte : ici, maximiser la justesse – *accuracy* –, en y ajoutant un module de régression symbolique. Ce dernier permet de générer une formule de prédiction de la justesse avec peu de composantes tout en étant interprétable. Des analyses détaillées sont proposées, dans lesquelles nous observons une robustesse de la solution trouvée qui est plus simple que celles proposées dans la littérature, et dont l'interprétation est pleinement conforme aux approches développées par la recherche en reconnaissance des formes.

Références

[1] Relja Arandjelović et Andrew Zisserman. "Look, Listen and Learn". In : *2017 IEEE International Conference on Computer Vision (ICCV)*. 2017, pages 609-617.

[2] Relja Arandjelović et Andrew Zisserman. "Objects that Sound". In : *2018 IEEE European Conference on Computer Vision (ECCV)*. 2018.

[3] György Buzsáki, Kai Kaila et Marcus Raichle. "Inhibition and brain work". In : *Neuron* 56.5 (déc. 2007), pages 771-783.

[4] Ying Cheng et al. "Look, Listen, and Attend : Co-Attention Network for Self-Supervised Audio-Visual Representation Learning". In : *Proceedings of the 28th ACM International Conference on Multimedia*. Association for Computing Machinery, 2020, 3884–3892.

[5] Yann LeCun, Yoshua Bengio et Geoffrey Hinton. "Deep learning". In : *Nature* 521.7553 (mai 2015), pages 436-444.

[6] Timothée Masquelier, Rudy Guyonneau et Simon J. Thorpe. "Spike Timing Dependent Plasticity Finds the Start of Repeating Patterns in Continuous Spike Trains". In : *PLOS ONE* 3 (jan. 2008), pages 1-9.

[7] Paul Merolla et al. "A million spiking-neuron integrated circuit with a scalable communication network and interface". In : *Science* 345 (2014), pages 668 -673.

[8] Andrew Owens et al. "Visually Indicated Sounds". In : *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pages 2405-2413.

Première partie

Connexionnisme impulsionnel

Chapitre 1

De la théorie à la modélisation

Le cerveau humain dispose environ entre 85 à 120 milliards de cellules nerveuses, que l'on appelle couramment neurones, interconnectées entre eux [39]. Ces neurones sont capables de communiquer entre eux grâce à des signaux électriques et chimiques, formant ainsi des réseaux complexes qui permettent au cerveau de traiter l'information sensorielle, de contrôler les mouvements, d'élaborer des pensées et des émotions, de gérer la mémoire, de réaliser des prises de décision et, évidemment, d'assurer les fonctions vitales du corps. Malgré cette incroyable complexité, le fonctionnement du cerveau reste encore mystérieux. Les avancées en neurosciences nous permettent de mieux comprendre certains mécanismes utilisés dans le cerveau, ce qui a ouvert la voie à de nouvelles approches pour la modélisation informatique des systèmes neuronaux, telles que le connexionnisme impulsionnel.

L'informatique inspirée par le cerveau est donc un champ interdisciplinaire qui a entrainé la création de plusieurs modèles de réseaux de neurones. Dans sa version initiale, ces modèles étaient capables de résoudre des problèmes linéaires uniquement, tandis que dans une deuxième version, il permettait la résolution de problèmes non linéaires grâce à des réseaux neuronaux profonds. Actuellement, une troisième génération de réseaux voit le jour, s'inspirant de la biologie, dans le but d'élargir les capacités de modélisation des systèmes neuronaux et s'approcher des avantages qui nous sont offerts par notre cerveau.

Dans ce chapitre, nous présentons un état des connaissances sur le connexionnisme impulsionnel permettant d'introduire le vocabulaire principal présent dans ce milieu. Pour cela, nous définirons le neurone biologique jusqu'à la modélisation numérique de ces réseaux de neurones. Nous en profiterons pour donner quelques points de comparaison avec l'apprentissage profond – *deep learning* .

Sommaire

1.1 Caractéristiques du neurone biologique

Avant de détailler les différentes modélisations numériques d'un neurone, il est intéressant dans un premier temps de comprendre la mécanique biologique d'un neurone.

1.1.1 Qu'est-ce qu'un neurone ?

Le neurone est une cellule spécialisée dans le traitement et la transmission de l'information. Comme le montre mon schéma de la figure 1.1, sa morphologie se compose généralement de trois unités : un corps et deux types d'expansions, les dendrites d'un côté et un axone de l'autre.

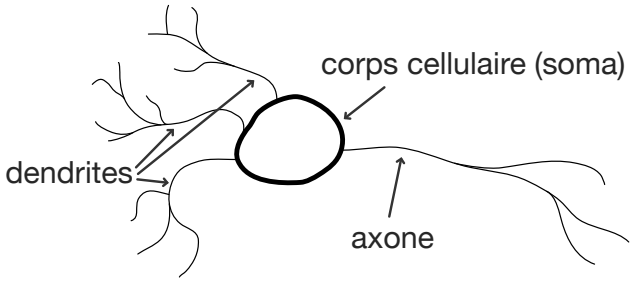


FIGURE 1.1 – Schéma simplifié d'un neurone.

Le corps cellulaire – *ou le nœud de calcul* – Appelé aussi soma, le corps cellulaire est le cœur du neurone permettant principalement d'émettre des potentiels d'action suivant les informations électriques reçues par ses dendrites. J'utiliserai souvent le terme d'« impulsions » ou de « décharge » pour potentiel d'action dans la suite du manuscrit.

Les dendrites – *ou le dispositif d'entrée* – Les dendrites forment de multiples ramifications qui rentrent en contact avec d'autres neurones. Leurs rôles sont de recevoir des informations électriques ou chimiques pour les transmettre ensuite au corps cellulaire.

L'axone – *ou le dispositif de sortie* – L'axone, quant à lui, est un prolongement de la cellule nerveuse qui conduit un signal électrique du corps cellulaire jusqu'à son arborisation terminale permettant de transmettre à d'autres neurones la même information via leurs dendrites associées. La jonction entre l'axone et les dendrites est appelée synapse. Par la suite, j'utiliserai le terme « connexions synaptiques », pour qualifier l'ensemble dendrites, synapses et axone.

Bien que ce schéma s'applique à la plupart des neurones, il convient de noter que le cerveau contient une très grande variété de neurones qui n'ont pas la même structure. Cela peut aller de l'ablation d'un axone à une forme spatiale (arbre dendritique) différente, sachant que ce dernier influe sur la fonction du neurone.

Concernant la transmission de l'information, les neurones communiquent entre eux par le biais d'impulsions électriques produites et transmises par une combinaison d'évènements chimiques et électriques, grâce aux ions présents autour et dans la membrane cellulaire. Ces ions vont permettre de créer deux types de potentiel que nous allons détailler dans la suite.

1.1.2 Potentiel de repos

Lorsque le neurone est au repos, une différence de potentiel de -70 mV est généralement observée entre l'intérieur et l'extérieur de la cellule. Cela est dû à une différence de concentration ionique et à la perméabilité sélective de la membrane cellulaire à certaines espèces d'ions. Afin de garantir cet équilibre, la membrane du neurone dispose de plusieurs protéines (canaux et pompes ioniques) permettant de stabiliser presque instantanément le potentiel à une valeur définie par l'équation de Nersnt :

$$V = \frac{k_B T}{q} \ln(\frac{C_e}{C_i}) \tag{1.1}$$

où k_B correspond à la constante de Boltzmann, T est la température en Kelvin, ainsi que pour chaque ion considéré, q est la charge de l'ion, C_e définit la concentration à l'extérieur de la cellule de l'ion considéré, et C_i, la concentration à l'intérieur de la cellule de l'ion considéré.

1.1.3 Potentiel d'action

Le potentiel d'action, souvent désigné par les termes d'influx nerveux ou de décharge neuronale, correspond à l'impulsion générée par le neurone lorsque le courant envoyé par les dendrites dépasse un certain seuil (figure 1.2). Cette impulsion se caractérise par une montée (dépolarisation) jusqu'à une valeur typique de +40 mV, suivie d'une redescente rapide (repolarisation/hyperpolarisation) à sa valeur de repos.

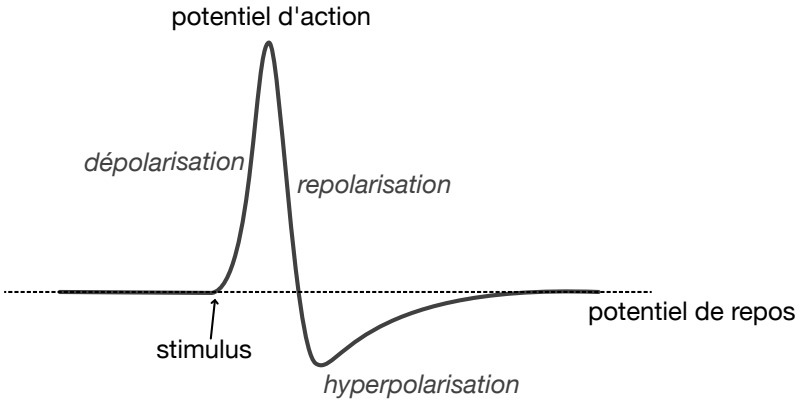


FIGURE 1.2 – Schéma simplifié d'un potentiel d'action.

Les premières expériences réalisées sur les axones du calmar montrent que lorsque l'on augmente suffisamment le potentiel de membrane en y injectant du courant électrique, de nouveaux canaux ioniques s'ouvrent afin que le potentiel se déplace vers un nouveau point d'équilibre. Une fois ce nouvel équilibre atteint, ces canaux se bloquent permettant de diminuer, pendant une courte période, le potentiel à un niveau inférieur au potentiel de repos (période réfractaire) interdisant toute réception de nouveaux signaux électriques.

Ce potentiel d'action ainsi généré se propage tout le long de l'axone par ce mécanisme de dépolarisation/repolarisation.

1.1.4 Synapses

Une fois que le signal électrique atteint la terminaison de l'axone, l'information est envoyée aux autres neurones par la jonction axone/dendrites que l'on appelle synapse. Il existe deux types de synapses : les synapses chimiques, majoritairement présentes, et les synapses électriques.

Synapse chimique Dans le cas des synapses chimiques, le neurone présynaptique libère des molécules appelées neurotransmetteurs qui vont se lier aux récepteurs pré-

sents sur la membrane des dendrites du neurone postsynaptique, ce qui aura pour conséquence de générer un courant ionique dans le neurone postsynaptique. Il existe plusieurs neurotransmetteurs ayant chacun leurs rôles et, par exemple, auront comme effet de rendre la synapse excitatrice ou inhibitrice. De plus, il est intéressant de noter qu'un neurone n'est pas limité à la libération d'un neurotransmetteur, il est capable d'en relâcher plusieurs, c'est-à-dire qu'il peut aussi bien être inhibiteur qu'excitateur suivant son potentiel.

Synapse électrique Les synapses électriques sont beaucoup moins présentes que les synapses chimiques bien qu'elles soient plus rapides. Elles utilisent une jonction communicante pour propager leurs ions directement, et ainsi transmettre l'impulsion.

1.2 Vers une représentation formelle

Le neurone biologique est considéré comme l'unité fondamentale du système nerveux. Comme nous avons pu le voir précédemment, il joue un rôle crucial dans la transmission de l'information électrique codant une information temporelle. Cependant, comprendre comment le neurone biologique fonctionne peut être complexe et requiert une représentation formelle.

Cette section se concentre sur la représentation formelle du neurone biologique, qui utilise des modèles mathématiques pour décrire et comprendre son fonctionnement. Cette approche permet de tester et de simuler des hypothèses sur la manière dont les neurones collaborent pour transmettre l'information.

1.2.1 Deux modélisations distinctes

Nous avons vu que pour communiquer, un neurone pouvait échanger plusieurs impulsions électriques. Par définition, cette génération successive est appelée un train d'impulsions témoignant ainsi d'une structure temporelle encodant de l'information. C'est à travers l'interprétation de ce train d'impulsions que dans la littérature deux types de modèles computationnels sont distincts : les modèles fréquentiels d'une part et les modèles impulsionnels de l'autre [12].

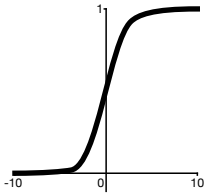
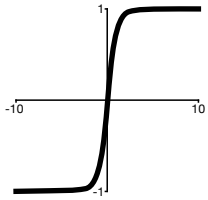
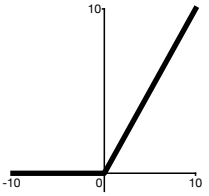
Les modèles fréquentiels Appelé aussi modèle à taux de décharge, cette modélisation se caractérise par la simplification de la représentation du train d'impulsions en une probabilité de génération d'impulsions sous la forme d'un processus de Poisson indépendant. Dans ce cas-ci, les neurones ne transmettent plus un train d'impulsions, mais s'échangent simplement des fréquences d'impulsions indépendantes. La fréquence d'impulsions postsynaptiques peut s'écrire comme une somme pondérée des fréquences d'impulsions présynaptiques :

$$p(t) = f(\sum w_i p_i(t)) \tag{1.2}$$

où $p(t)$ est la fréquence des impulsions du neurone postsynaptique, $p_i(t)$ celles des neurones présynaptiques, w_i est le poids de la connexion modélisant la propriété de la synapse, et f une fonction réelle appelée fonction d'activation. Cette fonction d'activation f doit être croissante, positive et majorée à une valeur fixant la fréquence maximale d'impulsions. Le tableau 1.1 présente les principales fonctions d'activations. Dans le cas

des architectures modernes, d'autres mécanismes de régularisation permettent de ne pas respecter la dernière propriété de la fonction d'activation.

TABLE 1.1 – Exemple de fonctions d'activation.

Nom	logistique	tangente hyperbolique	ReLU
Courbe			
Équation	$f(x) = \frac{1}{1+\exp^{-x}}$	$f(x) = \tanh(x)$	$f(x) = \max(0, x)$

Les modèles fréquentiels sont de nos jours la modélisation la plus répandue pour les différentes tâches de traitement numérique en raison de sa simplicité de mise en œuvre, de ses performances atteintes dans les différentes applications, ainsi que de son formalisme théorique bien décrit [33] permettant de les étudier à l'aide de méthodes d'analyse mathématique des fonctions continues.

Les modèles impulsionnels Appelé aussi modèle « intègre-et-tire (IF) », cette modélisation va se rapprocher un peu plus du neurone biologique en décrivant le train d'impulsions binaires sous forme d'un enchainement de Diracs. La modélisation se caractérise par une équation différentielle ordinaire (ODE) et un mécanisme de réinitialisation afin de produire une série d'impulsions en fonction du potentiel en entrée. L'ODE du potentiel membranaire du neurone peut s'écrire de la manière suivante :

$$\frac{\mathrm{d}v}{\mathrm{d}t} = f(v, t) \tag{1.3}$$

où v est le potentiel de la membrane du neurone, et f est la fonction de transfert synaptique qui définit la dynamique du potentiel membranaire du neurone. Lorsque ce potentiel dépasse un certain seuil, il est réinitialisé à sa valeur de repos, et une impulsion est générée.

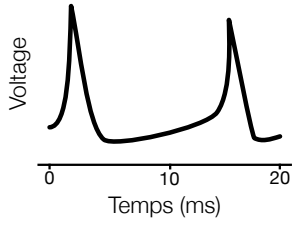
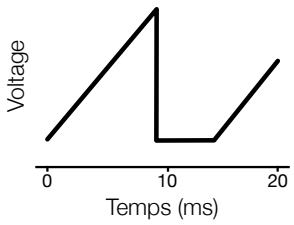
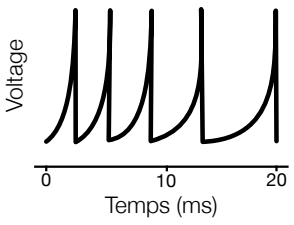
Les modèles impulsionnels sont de plus en plus explorés pour leurs propriétés de coût énergétique grâce à une représentation parcimonieuse des données et montrent des résultats prometteurs.

1.2.2 Les neurones impulsionnels

L'idée de la modélisation fréquentielle selon laquelle l'impulsion neuronale est poissonienne et indépendante, interdit certaines possibilités comme la synchronisation temporelle des neurones. Dans le cadre de cette thèse, nous nous intéressons donc à l'utilisation des modèles impulsionnels pour leur propriété de synchronisation et d'analyse de phénomènes temporels fins qui restent à ce jour très peu explorés.

Dans cette section, nous allons énumérer les principaux modèles de neurones impulsionnels allant du modèle biophysique au modèle computationnel simpliste.

TABLE 1.2 – Réaction du potentiel membranaire v de différents modèles de neurone pour un courant d'entrée I donné.

Modèle Hodgkin-Huxley	Modèle IF	Modèle Izhikevich
		

Modèle Hodgkin-Huxley Les travaux d'Alan Hodgkin et Andrew Huxley de 1952 mettant en évidence la génération des potentiels d'action dans l'axone géant du calmar [40] sont importants en neuroscience et leur ont valu un prix Nobel en 1963. À travers leur étude, ils ont proposé un modèle sous la forme d'un système d'ODEs modélisant les différents éléments du neurone naturel vus dans la première section, dont l'ODE principale décrit la membrane d'une cellule nerveuse sous la forme d'un condensateur :

$$v(t) = \frac{1}{C}\, Q(t) \tag{1.4}$$

où Q est la charge électrique, C est la capacitance et v représente la tension aux pôles du condensateur modélisant le potentiel de membrane du neurone.

En dérivant l'équation 1.4 par rapport au temps, nous obtenons :

$$\frac{\mathrm{d}v(t)}{\mathrm{d}t} = \frac{1}{C}\, I_{total}(t) \tag{1.5}$$

où $I_{total}(t) = dQ(t)/dt$ est le courant électrique total traversant le dipôle.

Sur la base d'expérience, les deux auteurs de l'étude ont émis l'hypothèse que I_{total} était composé de quatre éléments : deux courants issus d'ions présents dans la cellule notés I_{Na} et I_K [1], un courant de fuite I_L ainsi que le courant qu'ils injectaient I. L'équation 1.5 peut donc être réécrite comme :

$$\frac{\mathrm{d}v(t)}{\mathrm{d}t} = \frac{1}{C}(I_{Na}(t) + I_K(t) + I_L(t) + I(t)) \tag{1.6}$$

avec, en respectant la loi d'Ohm, les courants I_{Na}, I_K et I_L définis par :

$$I_{Na}(t) = g_{Na}(t)(v_{Na} - v(t)),\ I_K(t) = g_K(t)(v_K - v(t)),\ I_L(t) = g_L(v_L - v(t))$$

où v_{Ba} et v_K sont les potentiels d'équilibre défini par l'équation de Nernst (équation 1.1).

Ils mettent aussi en lumière l'évolution du comportement des conductances ioniques g_{Na}, g_K, g_L suivant le changement de tension v :

$$g_{Na}(t) = \bar{g}_{Na} m(t)^3 h(t),\ g_K(t) = \bar{g}_K n(t)^4,\ g_L = \bar{g}_L \tag{1.7}$$

1. Na et K

où $\bar{g}_{Na}$, $\bar{g}_K$, $\bar{g}_L$ [2] sont des constantes et m , h, et n sont des variables dépendantes du temps comprises entre 0 à 1.

En fusionnant les équations 1.6 et 1.7, l'ODE générale du modèle Hodgkin-Huxley définissant l'évolution temporelle du potentiel membranaire du neurone s'écrit de la manière suivante :

$$\frac{\mathrm{d}v(t)}{\mathrm{d}t} = \frac{1}{C}(\bar{g}_{Na}m(t)^3h(t)(v_{Na}-v(t))+\bar{g}_K n(t)^4(v_K-v(t))+\bar{g}_L(v_L-v(t))+I(t)) \quad (1.8)$$

Comme nous pouvons le voir, le modèle composé de quatre ODEs est extrêmement complexe à manipuler aussi bien analytiquement qu'en simulation, mais permet d'offrir une description détaillée proche d'un neurone biologique. Cette difficulté a motivé l'introduction de modèles simplifiés comme celui de FitzHugh-Nagumo [25, 77] qui réduit le modèle à seulement deux ODEs.

Modèle intègre-et-tire (IF) Ce modèle est à ma connaissance le premier à introduire la dynamique du potentiel du neurone. L'étude sortie en 1907 par Louis Édouard Lapicque cherchait à étudier l'évolution de la contraction d'une patte de grenouille en fonction d'un courant injecté dans une fibre nerveuse [53]. L'auteur conclut que le modèle de la réaction ressemblait à un circuit de filtre passe-bas composé d'une résistance R et d'un condensateur C (figure 1.3).

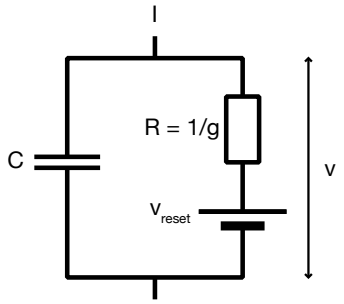


FIGURE 1.3 – Circuit RC modélisant le fonctionnement d'un neurone.

Un parallèle peut être fait avec le modèle Hodgkin-Huxley en supposant que les conductances ioniques g_{Na} et g_K présentes dans l'équation 1.6 sont constantes, c'est-à-dire que les variables n, m et h sont indépendantes du temps t. Cela donne l'équation suivante :

$$\frac{\mathrm{d}v(t)}{\mathrm{d}t} = \frac{1}{C}(g_{Na}(v_{Na}-v(t)) + g_K(v_K-v(t)) + g_l(v_L-v(t)) + I(t)) \quad (1.9)$$

qui peut être simplifié en :

$$\frac{\mathrm{d}v(t)}{\mathrm{d}t} = \frac{1}{\tau_m}(v_{eq}-v(t)) + \frac{I(t)}{C} \quad (1.10)$$

où I est un courant injecté, τ_m est la constante de temps membranaire s'écrivant :

$$\tau_m = \frac{C}{g_{Na}+g_K+g_L},$$

2. La notation $\bar{g}_L$ est utilisée ici pour uniformiser la formule avec les notations $\bar{g}_{Na}$ et $\bar{g}_K$

et v_{eq} :

$$v_{eq} = \frac{g_{Na}v_{Na} + g_K v_K + g_L v_L}{g_{N_a} + g_K + g_L}$$

L'équation 1.10 est complétée par une condition de réinitialisation :

$$v(t) = v_{reset} \text{ si } v(t) > v_{threshold} \tag{1.11}$$

où v_{reset} est le potentiel de repos, et $v_{threshold}$ est le seuil du potentiel membranaire du neurone. Lorsque le seuil $v_{threshold}$ est atteint, le potentiel v est réinitialisé à son état de repos v_{reset}. On dit alors qu'une impulsion est générée à l'instant t.

Afin de simplifier l'écriture de la modélisation présenté par l'équation 1.10, sans en changer son fonctionnement, nous fixons généralement $v_{reset} = 0$ et $v_{threshold} = 1$, puis nous posons un changement de variable afin d'obtenir :

$$\frac{\mathrm{d}\tilde{v}(t)}{\mathrm{d}t} = \frac{1}{\tau_m}\,\tilde{v}(t) + \tilde{I}(t) \tag{1.12}$$

Suivant la valeur de τ_m, deux modélisations se distinguent. Lorsque $\tau_m = \infty$, le modèle devient un intégrateur parfait que l'on appelle *Integrate-and-Fire* (IF), il intègre simplement le courant d'entrée I. Quand $\tau_m < \infty$, une modélisation du courant de fuite permet aux neurones de revenir à leur état de repos en l'absence d'activité. Ce modèle est appelé *Leaky Integrate-and-Fire* (LIF).

Ce modèle connaît un vrai intérêt pour sa simplicité de mise en œuvre et sa rapidité de calcul.

> *« While Lapicque, because of the limited knowledge of his time, had no choice but to model the action potential in a simple manner, the stereotypical character of action potentials allows us, even today, to use the same approximation to avoid computation of the voltage trajectory during an action potential. This allows us to focus both intellectual and computation resources on the issues likely to be most relevant in neural computation, without expending time and energy on modeling a phenomenon, the generation of action potentials, that is already well understood. »*
>
> **Laurence F. Abbott** [1]

Des extensions de ce modèle ont été créées pour obtenir un comportement plus proche des observations biologiques tout en gardant la simplicité du modèle de Lapicque comme le modèle intègre-et-tire quadratique (QIF) [22] qui ajoute un point d'inflexion à la trace du neurone IF, ou encore le modèle intègre-et-tire exponentiel (EIF) [26].

J'aimerais détailler une autre variante du neurone IF que j'ai utilisé durant mes expériences en particulier durant les expérimentations du chapitre 3, incorporant une modélisation temporelle du courant I en plus du potentiel de membrane v. Pour ce faire, la modélisation temporelle du courant I utilise une notion de récurrence, c'est-à-dire que les impulsions de sortie du neurone sont réinjectés dans le calcul du potentiel de membrane en plus du courant injecté. Pour simuler la dynamique de la membrane, le modèle de neurone est divisé en 2 parties. La première partie modélise la dynamique du potentiel de membrane v_i du neurone i dans la couche l :

$$\frac{d}{dt}v_i^{(l)}(t) = \frac{1}{\tau^{\mathrm{mem}}}((v^{\mathrm{rest}} - v_i^{(l)}(t-1)) + RI_i^{(l)}(t-1)) \tag{1.13}$$

La dernière partie modélise la dynamique du courant d'entrée I_i du neurone i :

$$\frac{d}{dt}I_i^{(l)}(t) = -\frac{I_i(t-1)}{\tau^{\text{syn}}} + \sum_j W_{ij}S_j^{(l-1)}(t-1) + \sum_j X_{ij}S_j^{(l)}(t-1) \tag{1.14}$$

avec $S_j^{(l)}(t)$ est le train d'impulsions émis par le neurone j de la couche l. R (généralement fixé à 1), v^{rest}, τ^{mem}, τ^{syn} sont des hyperparamètres du modèle, W et X sont des matrices de poids synaptiques. Ensuite, comme dans les réseaux neuronaux classiques et la modélisation intègre-et-tire avec fuite (LIF), une fonction d'activation Θ, plus précisément la fonction de pas de Heaviside, est appliquée à la tension $v(t+1)$: $\Theta(v(t) - v^{\text{threshold}})$.

Spike Response Model (SRM) Ce modèle, dont le nom peut être traduit en français par « modèle de réponse aux impulsions », a été défini par Gerstner et van Hemmen [29, 30] et donne une généralisation du modèle IF dont l'équation est formulée sans ODE mais avec des fonctions paramétriques (appelées aussi noyaux –*kernel*–) :

$$v_i(t) = \sum_f \mu(t - t_i^{(f)}) + \sum_j w_{ij} \sum_f \epsilon_{ij}(t - \hat{t}_i, t - t_j^{(f)}) + \int_0^\infty \kappa(t - \hat{t}_i, s)I(t-s)ds + v_{reset} \tag{1.15}$$

où $v_i(t)$ est le potentiel de membrane du neurone i au temps t, $\hat{t}$ correspond au temps de la dernière impulsion, et $t^{(f)}$ est le temps de la f-ème impulsion du neurone. L'équation se compose de 3 parties : $\sum_j w_{ij} \sum_f \epsilon_{ij}(t - \hat{t}_i, t - t_j^{(f)})$ correspondant à l'influence des impulsions des neurones pré-synaptiques j pondérés par w, $\sum_f \mu(t - t_i^{(f)})$ étant l'influence des impulsions post-synaptiques sur le potentiel de membrane à travers, et $\int_0^\infty \kappa(t - \hat{t}_i, s)I(t-s)ds$ modélisant l'incorporation d'un courant externe. Les noyaux – *kernels* – μ, ϵ, κ sont à définir.

C'est un modèle assez flexible permettant de modéliser le temps réfractaire qui contrôle l'inactivité après la génération d'une impulsion, mais également de considérer l'ensemble des impulsions passées émis du neurone en plus de la dernière impulsion. Comme le modèle d'Izhikevich, le modèle de réponse aux impulsions (SRM) est capable de rivaliser avec le modèle biophysique d'Hodgkin-Huxley tout en étant plus simple en termes de temps de calcul.

Modèle d'Izhikevich Le modèle d'Izhikevich permet de reproduire un grand nombre de schémas d'impulsions observés in vivo [45]. C'est une représentation biologiquement plausible et fortement inspiré du modèle biophysique d'Hodgkin-Huxley avec une complexité de calcul comparable à un modèle IF. Le modèle réduit le système d'Hodgkin-Huxley aux deux ODEs suivantes :

$$\frac{\mathrm{d}v(t)}{\mathrm{d}t} = 0.04v(t)^2 + 5v(t) + 140 - u(t) + I(t) \tag{1.16}$$

$$\frac{\mathrm{d}u(t)}{\mathrm{d}t} = a(bv(t) - u(t)) \tag{1.17}$$

avec la remise à zéro après l'impulsion générée :

$$\text{si } v(t) >= 30 \text{ mV, alors } \begin{cases} v(t) \leftarrow c \\ u(t) \leftarrow u(t) + d \end{cases}$$

où v est le potentiel de membrane du neurone, u est une membrane additionnelle appelée « *recovery variable* », et a, b, c et d sont les paramètres qui contrôlent la dynamique du potentiel membranaire.

Dans cette représentation, le modèle ajoute la possibilité de générer des oscillations proches du seuil sans générer d'impulsions. Cette caractéristique permet d'avoir un impact sur les résultats des potentiels d'action des neurones voisins.

En résumé, nous avons vu les principaux modèles computationnels d'un neurone que nous pouvons classer selon deux usages bien distincts : des modèles biophysiques permettant de simuler des potentiels d'action naturels, c'est-à-dire très proche de la nature, comme le modèle d'Hodgkin-Huxley ou d'Izhikevich ; et des modèles numériques réduisant les contraintes temporelles et les ressources de calcul nécessaires, pouvant servir aux calculs d'apprentissage machine (ML) comme les modèles IF. Le tableau 1.3 présente l'ensemble des modèles impulsionnels implémentés durant cette thèse ainsi que leur nombre d'opérations en virgule flottante par seconde (FLOPS) associé.

TABLE 1.3 – Nombre d'opérations en virgule flottante par seconde (FLOPS) (addition, multiplication, etc.) nécessaires pour simuler chaque modèle pendant une durée de 1 ms, ainsi que sa plausibilité biologique associée due à ses propriétés neurocomputationnelles (modèle biophysique ou non). Inspiré de [46].

Modèle de neurones	**IF**	**LIF**	**QIF**	**Izhikevich**	**Hodgkin-Huxley**
Nombre de FLOPS (en ms)	4	5	7	13	1200
Forte plausibilité biologique				✓	✓

1.2.3 Topologie

Afin de créer un réseau capable d'effectuer différentes tâches de calcul, les neurones doivent être connectés à d'autres neurones. La manière dont ces neurones sont connectés et agencés spatialement est appelée la topologie. Fondamentalement, un réseau neuronal est un ensemble de neurones N, reliés par un ensemble de synapses S. Chaque synapse s_{ij} relie un neurone présynaptique n_i à un neurone postsynaptique n_j. Une synapse ou « poids synaptique » est une connexion unidirectionnelle, c'est-à-dire qu'elle va d'un neurone vers un autre. L'optimisation de cette connexion, en phase d'apprentissage, consiste à trouver un poids optimal permettant de répondre à une tâche donnée. Dans la majorité des topologies en ML, les neurones sont rassemblés en groupes, appelés couches L, chaque couche étant définie comme un sous-ensemble des neurones N.

La topologie des SNNs est identique à celle des réseaux profonds classiques. Dans le domaine des SNNs, nous identifions principalement trois types d'architectures : le réseau de neurones à propagation avant (FFNN), le réseau de neurones convolutifs (CNN) et le réseau de neurones récurrents (RNN). Pour des questions de lisibilité, j'ai décidé de reprendre les mêmes notations schématiques des unités que dans [59] reportés dans le tableau 1.4, avec le terme « cellule » qui correspond à la modélisation du neurone.

TABLE 1.4 – Unités utilisées pour décrire les différentes architectures.

Cellule d'entrée	Cellule cachée	Cellule de sortie	Kernel	Convolution ou Pool
				

Réseau de neurones à propagation avant (FFNN) Les FFNNs sont l'archétype du réseau neuronal, avec des couches « verticales » constituées de neurones d'entrée, cachés ou de sortie (figure 1.4). Les neurones d'une couche l_i sont connectés aux neurones des couches supérieures l_j avec $i < j$ afin de s'assurer qu'aucun cycle n'est possible. Généralement, les neurones d'une couche l_i sont entièrement connectés à tous les neurones de la couche l_{i+1}. Dans un tel cas, on parle de couche dense.

Cette architecture est mise en oeuvre principalement dans les tâches de classification [7, 65, 82, 95].

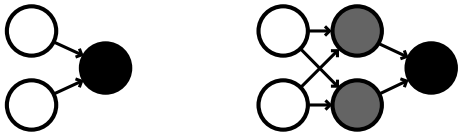


FIGURE 1.4 – Exemple d'architectures FFNNs.

Réseau de neurones convolutifs (CNN) Les CNNs, ou réseaux de neurones convolutifs à impulsions (SCNNs), sont une extension de la topologie vue précédemment en y ajoutant des neurones pouvant extraire de l'information locale à l'aide de filtres convolutifs appelés aussi *kernel* (figure 1.5). Cette architecture permet à un neurone d'une couche l_i de réagir à des informations envoyées par un sous-ensemble de neurones de la couche précédente l_{i-1}. Le concept derrière cette topologie consiste à considérer que la covariance locale des observations est un support caractéristique à la décision. Les filtres linéaires qu'elle implémente sont utilisés pour mettre en évidence des schémas de covariance spécifiques.

Cette architecture est très utilisée dans diverses tâches telles que la classification d'images [15, 44, 51, 81], de sons [75] ou encore dans le traitement du flot optique [36, 57, 72].

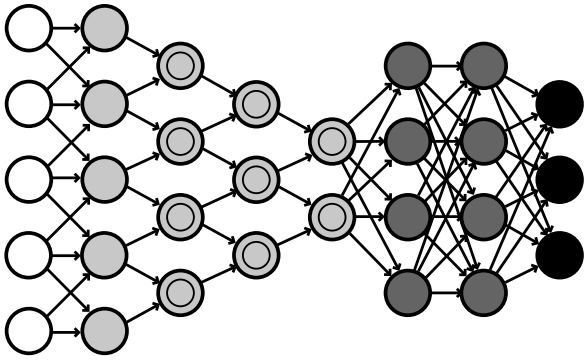


FIGURE 1.5 – Exemple d'une architecture CNN.

Réseau de neurones récurrents (RNN) Un RNNs, ou réseau de neurones à impulsions récurrents (SRNN), utilise une topologie légèrement différente de celles vues précédemment en y ajoutant un facteur de récurrence, c'est-à-dire que l'architecture peut présenter des cycles (figure 1.6). Une des architectures les plus connues est la machine à état liquide (LSM) [61] composée d'une couche d'entrée, une couche de population de neurones connectés de manière aléatoire entre eux que l'on appelle « réservoir » et une couche de sortie. Ce modèle permet de projeter l'entrée dans un nouvel espace de dimension qui sera ensuite utilisé dans d'autres méthodes d'apprentissage plus traditionnelles. D'autres modèles récurrents vont utiliser un modèle de décroissance du potentiel membranaire pour représenter des cellules mémoires [97], ou des approches hybrides contenant plusieurs populations de neurones qui peuvent communiquer entre elles [5].

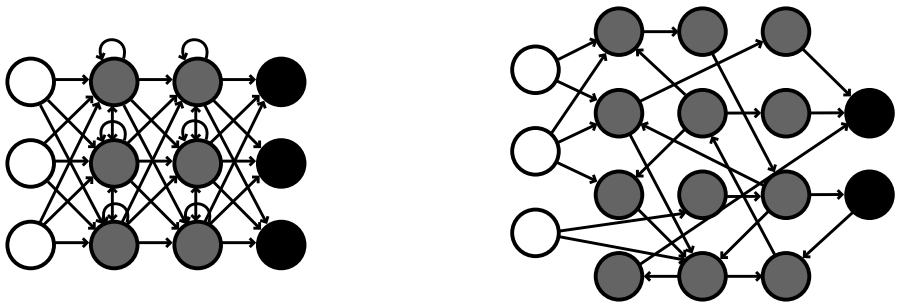


FIGURE 1.6 – Exemple d'architectures RNNs.

1.3 Code neural

Le code neural correspond au langage utilisé par les neurones du cerveau pour communiquer entre eux et qui code précisément les informations [11]. D'un côté, la périphérie sensorielle transmet les signaux au cerveau sous forme d'impulsions de l'autre côté, le décodage de ces impulsions permet de comprendre comment le cerveau traite ces signaux codés. C'est dans cette optique que David Hubel et Torsten Wiesel ont été les précurseurs sur ce décodage en réalisant des expériences peu éthiques sur un chat afin de décrypter le code neural du cortex visuel primaire [42]. Dans ce travail qui leur

valu un prix Nobel en 1981, ils ont conclu que les informations en provenance des yeux étaient reconstruites sous la forme d'une carte de contraste de neurones codant une orientation préférentielle.

Dans le contexte de l'entraînement d'un modèle de réseau de neurones à impulsions (SNN), deux mécanismes entrent donc en jeux : l'encodage des données d'entrée et le décodage des trains d'impulsions en sortie. L'encodage de l'entrée correspond à la conversion en train d'impulsions des données d'entrée pour alimenter le SNN, équivalent à nos capteurs sensoriels, tandis que le décodage de la sortie permet d'interpréter les trains d'impulsions de manière significative.

1.3.1 Encodage de l'information

Il existe deux grandes classes d'encodages : l'encodage fréquentiel et l'encodage temporel. Dans ce dernier cas, l'ordre et le moment exacts des impulsions dans les différents neurones représentant l'information à transmettre joue un rôle essentiel au codage contrairement à l'encodage fréquentiel qui utilise la fréquence des impulsions pour transmettre l'information.

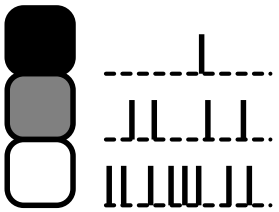
Il est couramment admis l'existence d'une troisième catégorie : le codage par population, qui ajoute une information de dépendance à d'autres neurones permettant d'encoder et de transmettre une information. Plus précisément, elle combine les informations provenant de plusieurs neurones pour représenter une information complexe. Cette méthode ressemble donc à une sorte de méta encodage, pouvant prendre des trains d'impulsions issus d'un encodage temporel ou fréquentiel.

Encodage fréquentiel

Le schéma de codage fréquentiel – *rate coding* – le plus courant est appelé le codage par taux de décharge. Il est défini par son taux d'impulsions moyen N_{spike}/T où N_{spike} est le nombre d'impulsions, et T est la fenêtre temporelle, c'est-à-dire la durée du train d'impulsions. Ce schéma a déjà été observé in vivo sur certains neurones émettant des impulsions à une fréquence proportionnelle à l'intensité d'un stimulus [4, 42].

La conversion d'une donnée en entrée en trains d'impulsions I de longueur temporelle T peut se faire soit par des temps exacts d'impulsions utilisant simplement le taux d'impulsions moyen pour les générer, soit par des temps d'impulsions aléatoires qui sont généralement modélisés à travers une loi uniforme. C'est cette dernière qui est toujours utilisée dès que nous parlons d'encodage fréquentiel des données.

TABLE 1.5 – Visualisation des techniques d'encodage fréquentiel.

Codage par taux de décharge


• **Codage par taux de décharge** Ce formalisme a été défini par Dayan et Abbott [18] pour modéliser l'encodage d'une donnée. À chaque instant t, chaque donnée de l'entrée normalisé e est respectivement comparée à un nombre aléatoire $Rand_i(t)$ défini par une distribution uniforme de 0 à 1. Si le nombre aléatoire généré est inférieur à ses données d'entrée, $I_i(t)$ est mis à 1, ou sinon à 0. Ce qui nous donne le formalisme suivant :

$$I_i(t) = \begin{cases} 1 & \text{si } e_i > Rand_i(t) \\ 0 & \text{sinon} \end{cases} \tag{1.18}$$

où $Rand_i(t) \sim \mathcal{U}(0,1), i = 1, \ldots, n, t = 1, \ldots, T$ et $\mathcal{U}$ est la loi de distribution uniforme.

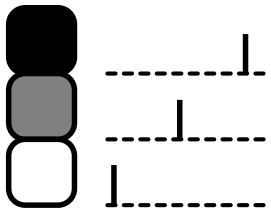
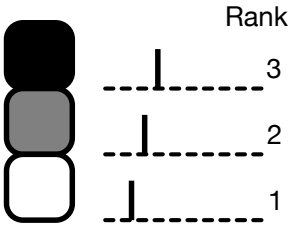
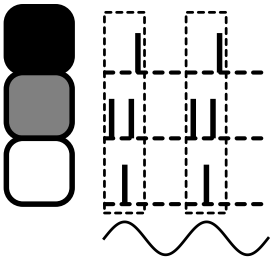
Cette méthode d'encodage permet de simplifier l'utilisation d'un processus de Poisson qui aurait demandé dans un premier temps de définir le nombre d'occurences (nombre d'impulsions ici) en fonction de l'entrée normalisée e, puis de générer le temps associé à l'occurence par une fonction uniforme.

Encodage temporel

Contrairement au codage fréquentiel, un codage temporel – *latency coding* – s'intéresse au moment où se produit une impulsion et attribue plus de signification à chaque impulsion individuelle. Elle apporte donc plus d'informations avec moins d'impulsions.

Il existe plusieurs catégories de codages temporels qui se distinguent par une référence différente pouvant être, par exemple, un signal périodique (codage par la phase) ou un point unique (codage TTFS). Les principaux encodages temporels utilisés que je présente dans la suite sont illustrés dans le tableau 1.6.

TABLE 1.6 – Visualisation des techniques d'encodage temporel.

TTFS	Codage par rang	Codage par phase
		

• **Temps avant la première impulsion (TTFS)** Le codage TTFS est le schéma de codage temporel le plus élémentaire. L'information est codée par la différence de temps Δt entre l'apparition du courant d'entrée I et la première impulsion du neurone. Dans le cas le plus simple, le temps t de l'impulsion est défini par l'inverse de l'amplitude du signal $1/I$. Ainsi, les impulsions les plus rapides codent des valeurs du signal élevées, tandis que les impulsions plus tardives représentent des valeurs plus faibles (voir première colonne du tableau 1.6). À partir de cet encodage, il est donc tout à fait possible de retrouver le signal d'entrée.

• **Codage par rang (ROC)** Le ROC est une variante du codage TTFS défini par Simon Thorpe *et al.*, où l'information est codée par l'ordre des impulsions d'une population

de neurones sans tenir compte du moment précis des impulsions [27, 90], il est donc impossible de reconstruire l'amplitude du signal d'entrée. Il suppose aussi que chaque neurone n'émet qu'une seule impulsion sur leur train d'impulsions respectif. Cette méthode a permis aux auteurs de développer un système capable de catégoriser des images statiques avec une vitesse de traitement comparable à celle observée chez l'homme [89].

Les valeurs d'entrée sont triées par une heuristique comme la valeur de l'amplitude du signal, et cette base d'indices ainsi générée permettra de définir l'ordre des neurones qui vont envoyer une impulsion (voir deuxième colonne du tableau 1.6).

• **Codage par la phase** Des études biologiques ont mis en évidence la présence d'oscillations dans des trains d'impulsions du cerveau d'un chat [34]. Partant de la base de ce constat, l'information est codée par la différence de temps relative entre les impulsions et une oscillation de référence. [41, 50]. C'est un codage cyclique, c'est-à-dire que l'information se répète périodiquement, car les neurones se déclenchent à un signal périodique de référence (voir troisième colonne du tableau 1.6).

• **Autres codages temporels** J'ai cité ici les principaux codages temporels que j'ai utilisé dans mes travaux. Ils existent cependant plusieurs autres codages avec des variantes associées comme le codage binaire où une valeur est codée sous forme binaire par un train d'impulsions dont la présence ou l'absence encode respectivement un 1 ou 0, le codage de la latence où l'information est intégrée dans la différence de temps relative entre les impulsions d'une population de neurones, ou encore le codage par synchronie où des neurones d'une population codant une information se déclenchent de manière synchrone.

1.3.2 Décodage de l'information

Jusqu'à présent, nous avons vu la modélisation d'un SNN ainsi que les différents types d'encodage du train impulsionnel en entrée et en sortie de ce réseau. La question que l'on peut se poser à ce stade est, dans un problème de classification des données, comment le train impulsionnel en sortie peut-il être interprété pour une tâche donnée ? À cette question, il ressort deux solutions suivant la variable interprétée : un score au niveau du train impulsionnel ou un score au niveau du potentiel d'action.

Au niveau des impulsions de sortie Afin de déterminer quel neurone en sortie est pertinent pour le codage de l'information en entrée, la méthode la plus simple est d'analyser le train impulsionnel. Dans le cas du codage fréquentiel où le train d'impulsions présentent de nombreuses impulsions, le calcul de la fréquence impulsionnelle ou du nombre total d'impulsions de chaque neurone en sortie permet de déterminer quel neurone contribue le plus au transfert de l'information. Le neurone ayant donc le score le plus élevé est identifié comme celui représentant le mieux l'information d'entrée. Dans le cas d'un train impulsionnel encodé temporellement, le décodage se fait généralement en déterminant le neurone qui a envoyé la première impulsion en sortie.

Potentiel d'action comme représentation Une autre méthode consiste, si cela est possible, à transformer le modèle du neurone en intégrateur parfait en supprimant le

seuil de la membrane. Une représentation finale est ensuite extraite en prenant la valeur des niveaux du potentiel d'action à la fin de l'inférence d'une donnée d'entrée. Ces représentations ainsi obtenues sont utilisées en entrée d'autres approches de ML afin d'opérer une classification par exemple.

1.3.3 Quel type d'encodage ?

Chacun des deux encodages présentent des avantages et des inconvénients. Par sa multitude d'impulsions et sa redondance à coder l'information, le codage fréquentiel sera plus tolérant aux erreurs et évitera aux règles d'apprentissage, que nous verrons par la suite, d'avoir des problèmes du type « neurone mort » en cas d'absence d'impulsions. Ces avantages qui font les faiblesses principales du codage temporel, ne sont pas adaptés pour des traitements rapides. C'est sur ce dernier point que se caractérise réellement le codage temporel, ce qui permet aussi un gain en consommation d'énergie dû aux données très éparses qui transitent dans le réseau.

1.4 Méthode d'apprentissage

Un algorithme d'apprentissage est une méthode utilisée pour traiter les données d'entrée afin d'en extraire des modèles appropriés pour une tâche spécifique. En particulier, dans le cas des SNNs, l'algorithme cherche à optimiser les poids synaptiques suivant un critère qui doit prendre en compte l'aspect dynamique des informations.

Ils existent trois principaux paradigmes d'apprentissage automatique utilisés dans le domaine de l'apprentissage profond que l'on va retrouver dans les SNNs.

Apprentissage supervisé L'apprentissage supervisé – *supervised learning* – consiste à construire une fonction de prédiction à partir d'exemples annotés. Les poids synaptiques vont donc être ajustés pour que la différence entre la sortie du modèle et la sortie souhaitée soit aussi faible que possible, optimisant le modèle pour trouver la plus forte corrélation entre le signal d'entrée et de sortie sortie. L'inconvénient est de devoir fournir les données annotées et en grande quantité.

Apprentissage non supervisé Contrairement à l'apprentissage supervisé, l'apprentissage non supervisé – *unsupervised learning* – utilise des données brutes (non annotés) pour lequel l'objectif est d'en extraire des propriétés structurelles intéressantes sans information sur les données d'entrée. Contrairement à la méthode d'apprentissage précédente, ces modèles sont plus compliqués à optimiser pour une tâche spécifique, mais permettent d'entrainer sur un plus grand ensemble de données collectées automatiquement. Il existe plusieurs manières de construire ce modèle mathématique, par exemple, avec des règles hebbiennes que je détaillerai dans la suite de ce chapitre, ou encore par l'utilisation de modèle auto-supervisé – *self-supervised* – que j'approfondirai dans le chapitre 3.

Apprentissage par renforcement Dans ce tout autre paradigme d'apprentissage, l'apprentissage par renforcement – *reinforcement learning* – s'intéresse à la façon dont des agents intelligents devraient réaliser des actions dans un environnement afin de maximiser une récompense. Il se distingue des deux autres méthodes par le fait que

son apprentissage se faire uniquement par des interactions sans aucune supervision. En revanche, le signal de récompense contient généralement peu d'informations pour actualiser correctement l'ensemble des poids synaptiques. Ce type d'approche ne sera pas exploré dans le cadre de cette thèse.

Dans ces trois paradigmes, l'apprentissage d'un SNN est généralement fondé sur des méthodes d'optimisation dont le but est de minimiser ou de maximiser une règle, ou fonction de coût, comme une heuristique sur la fréquence d'impulsions en sortie ou un train d'impulsions spécifiques (voir la partie sur le décodage de l'information à la section 1.3.2).

Ils existent à ma connaissance deux catégories principales de méthodes d'apprentissage : les modèles d'apprentissage basés sur le gradient comme la conversion d'un réseau de neurones artificiels (ANN) en SNN ou l'adaptation de la descente du gradient (section 1.4.1) et les méthodes utilisant des règles d'apprentissage locales biologiquement plausibles (section 1.4.2). Pour finir, la section 1.4.3 référence d'autres méthodes d'apprentissage présentes dans la littérature comme les méthodes utilisant des EAs, des modèles hybrides, etc.

1.4.1 Méthode basé sur la descente du gradient

La rétropropagation du gradient (BP) est l'algorithme de descente de gradient [3] par excellence pour la formation des ANNs permettant d'ajuster les poids synaptiques du réseau [55]. Fort de son succès dans le domaine de l'apprentissage profond, elle en fait une cible de recherche pour l'appliquer dans le domaine des SNNs. Son fonctionnement consiste à calculer tous les gradients du modèle à chaque passage d'une donnée d'entrée afin d'actualiser les poids synaptiques pour que l'erreur en sortie soit minimisée [79]. Ainsi, si un gradient vaut « 0 », le poids synaptique associé ne reçoit aucune mise à jour. Au vu de l'aspect dynamique des SNNs et la parcimonie des informations circulant dans le réseau, cela peut conduire au problème connu sous le nom de « neurone mort ».

Problème du « neurone mort » Nous avons vu que les informations transitent dans un SNN sous forme d'impulsions déclenchées dès qu'un neurone dépasse un seuil défini. Cependant, cette fonction de seuillage n'est pas différentiable, car le gradient d'une fonction seuil vaut « 0 », sauf lorsque le test est respecté où elle vaut « 1 ». Cela aura pour conséquence qu'un neurone ne produit pas de donnée en sortie. Comme il n'est pas possible de mesurer une erreur entre donnée produite et donnée attendue, aucune correction des poids synaptiques des connexions en entrée du neurone n'intervient lors du processus d'apprentissage. Par le fait, celui-ci restera indéfiniment insensible aux impulsions qu'il reçoit. On se retrouve donc avec un « neurone mort » – *dead neuron* .

Par analogie, dans le domaine de l'apprentissage profond avec des neurones utilisant l'unité de rectification linéaire (ReLU), ce problème se retrouve sous le terme de *dying ReLU* où un neurone avec un biais trop négatif produira toujours « 0 » lors de sa propagation avant et un gradient toujours nul, les poids synaptiques de ces neurones ne seront plus jamais actualisés.

Ce problème ne doit pas être confondu avec la disparition du gradient, bien connu de l'apprentissage profond.

3. La descente de gradient est un algorithme d'optimisation minimisant une fonction réelle différentiable de manière itérative.

Différence avec la disparition/explosion du gradient La disparition du gradient – *vanishing gradient* – est causée par la multiplication de gradients inférieurs à 1 en valeur absolue au fil des couches lors de la propagation arrière de l'erreur. Dans le cas de réseaux très profonds ou les RNNs, ce gradient peut systématiquement devenir proche de 0 pour les premières couches et ainsi ne pas être exploitable pour la correction des poids. À l'inverse, des gradients très élevés en valeur absolue peuvent conduire à une explosion du gradient – *exploding gradient* – empêchant la convergence des valeurs des poids des premières couches d'un réseau.

Pour palier à ce problème du « neurone mort », la recherche s'est orientée vers deux types de méthodes : la conversion d'un ANN en SNN ou trouver une parade sur le calcul du gradient pour contourner le problème du neurone mort.

Conversion d'un ANN à un SNN De par les avancées spectaculaires dans le domaine de l'apprentissage profond et par le fait que les ANNs peuvent être vus comme des modèles fréquentielles indiquant la probabilité de décharge des neurones (voir section 1.2.1), la méthode la plus naïve consisterait donc à convertir un modèle ANN (appelé *shadow network*) en SNN.

C'est ce qui a été proposé par des premiers travaux portant sur la conversion d'un réseau de croyances profonds (DBN) en SNN [70], facilité alors par le fait que ces réseaux utilisent des fonctions d'activations binaires pouvant être vues comme des impulsions émises par les neurones impulsionnels et permettant ainsi de minimiser l'erreur entre l'activité des deux modèles. Dans cette architecture, la première phase consiste à entrainer le DBN composé de neurones artificiels définis par Siegert [86], unités qui approximent le neurone LIF en donnant en sortie le taux moyen de décharges. Puis, lors de la phase de conversion, chaque neurone génère un train impulsionnel en codage fréquentiel utilisant un processus de Poisson proportionnel au taux moyen de décharges calculé.

Cette idée a ensuite été étendue aux ANNs avec leurs différentes unités spécifiques (*« pooling, softmax, batch-norm »*), en générant des trains impulsionnels en codage aussi bien fréquentiel [20, 21, 23, 44, 78] que temporel [87] en sortie des neurones. Le principe ici consiste à entrainer un ANN avec une fonction d'activation ReLU puis, lors de la conversion, de remplacer cette fonction par des neurones impulsionnels en y ajoutant des opérations de mise à l'échelle comme la normalisation des poids et l'équilibrage des seuils.

Les dernières techniques de conversion basées sur l'approximation entre les valeurs d'activation des neurones artificiels et la fréquence d'impulsion des neurones impulsionnels obtiennent des performances comparables à un ANN. Cependant, ces méthodes font face à un compromis entre précision et latence, c'est-à-dire qu'un SNN converti a besoin de plus de temps pour obtenir la même performance qu'un SNN original en raison de l'utilisation du codage fréquentiel [37].

Cette méthode souffre principalement de l'incapacité d'apprentissage de représentation temporelle des SNNs que l'on aurait avec un entrainement direct.

Entrainement direct Les méthodes d'entrainement direct basées sur le gradient peuvent être classées en deux catégories [52] : les méthodes basées sur le temps de décharge [9, 16, 67], et les méthodes basées sur l'activation des neurones [48, 49, 56, 58, 84, 94, 102].

Dans la première catégorie, les méthodes se concentrent sur le temps des impulsions et calculent les gradients par rapport aux temps de décharges. Un des premiers algorithmes à entrainer un SNN par la descente de gradient a été proposé par Sander Bohte *et al.* sous le nom de « SpikeProp ». Il utilise une approximation linéaire pour surmonter le mécanisme de déclenchement à un seuil [9], c'est-à-dire que la fonction de seuil générant les impulsions en sortie, donc une fonction non différentiable, est remplacée par une expression différentiable exprimée par temps de décharges, sur laquelle une BP peut ensuite être effectuée. Bien que l'approche originale soit limitée à une seule impulsion par neurone, des extensions de cet algorithme ont été proposées améliorant également ses propriétés de convergence [10, 16, 66, 83, 85, 96]. Par exemple, dans [10], les auteurs rendent compatible l'algorithme d'apprentissage aux multiples impulsions en sortie, ou encore, Comsa *et al.* ajoute une pénalité pour éviter l'inactivité des neurones, ce qui a pour conséquence de geler leurs poids synaptiques [16].

La seconde catégorie est l'approche la plus couramment adoptée ces dernières années. Elle consiste à contourner la non-différentiabilité de la fonction seuil du neurone impulsionnel par l'utilisation d'un gradient de substitution – *surrogate gradient* – [68] appliqué sur la fonction dérivée du mécanisme de seuillage. En d'autres termes, le but de ce mécanisme est de remplacer le gradient de la fonction de seuil du neurone impulsionnel par un gradient estimé. La figure 1.7 illustre cette approche. L'algorithme de BP, utilisé traditionnellement dans les ANNs, permettent d'optimiser les SNNs en calculant tous les gradients par rapport à l'activation de l'impulsion en déroulant le graphe de calcul, idée similaire à la rétropopagation temporelle (BPTT) [48, 49, 56, 58, 84, 94, 100, 102]. Zenke et al. [68, 101] ont aussi étudié la robustesse de l'apprentissage grâce à cette méthode et ont montré que les SNNs optimisés par les méthodes de gradient de substitution peuvent atteindre des performances compétitives avec les ANNs.

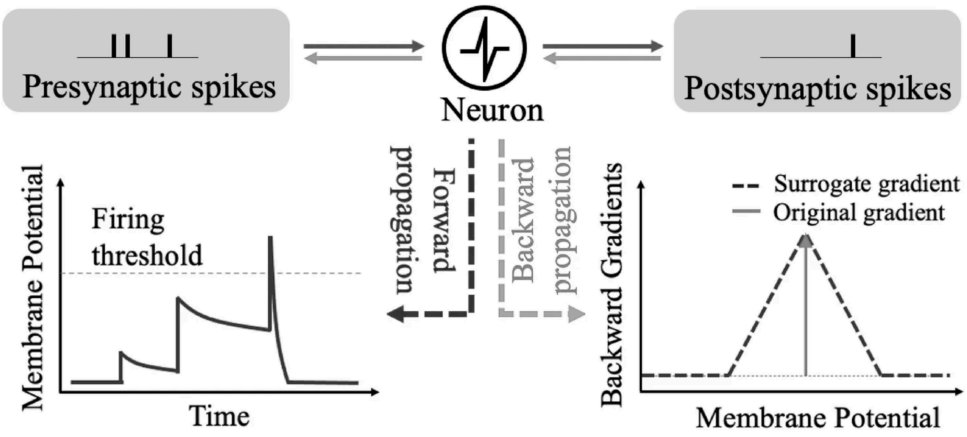


FIGURE 1.7 – Illustration de Youngeun *el al.* [98] présentant la propagation avant et arrière du neurone LIF en utilisant un gradient estimé – *surrogate gradient*.

1.4.2 Plasticité synaptique

Contrairement aux méthodes utilisant la descente de gradient pour optimiser le modèle, des approches à motivation biologique remplacent le signal d'erreur globale, résultant du calcul d'une fonction de coût dans un contexte supervisé, par des erreurs locales issues de règles de plasticité synaptique. Cette dernière représente la capacité essentielle

au bon fonctionnement du cerveau, mais également, est à la base de l'apprentissage et de la mémoire. Elle suit généralement le postulat de Hebb suivant :

> *« Let us assume then that the persistence or repetition of a reverberatory activity (or "trace") tends to induce lasting cellular changes that add to its stability. The assumption can be precisely stated as follows : When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased. »*
>
> **Donald O. Hebb** [38]

Cette règle suggère que lorsque deux neurones sont excités conjointement, il se crée ou se renforce un lien les unissant.

Suivant ce principe hebbien, une des méthodes d'apprentissage les plus étudiées est la plasticité fonction du temps d'occurrence des impulsions (STDP) observée initialement par Bi & Poo [6] et largement utilisé dans la littérature [13, 19, 63] avec de multiples variantes listées par Vigneron *et al.* [92]. La règle STDP, hautement interprétable d'un point de vue biologique [63], suggère que les poids synaptiques dépendent de la différence temporelle des impulsions pré/post synaptiques. C'est-à-dire que le poids synaptique est renforcé lorsque le neurone postsynaptique se déclenche peu après le neurone présynaptique, et il est affaibli lorsque le neurone postsynaptique se déclenche peu avant le neurone présynaptique [3]. La quantité de changement dans le poids synaptique dépend exponentiellement de $t_{post} - t_{pre}$, où t_{post} est le temps de tir du pic postsynaptique, et t_{pre} désigne le temps de tir du pic présynaptique.

La quantité de mise à jour suivante Δw_{ji} pour le poids w_{ji} du neurone présynaptique j vers le neurone postsynaptique i est donnée par :

$$\Delta w_{ji} = \sum_f \sum_n W(t_i^{(n)} - t_j^{(f)}) \tag{1.19}$$

où $t_i^{(n)}$ est le n-ème temps d'impulsion du neurone i, $t_j^{(f)}$ est le f-ème temps d'impulsion du neurone j, et $W()$ est une fonction STDP qui est définie de la manière suivante :

$$W(x) = \begin{cases} A_+ \exp(-x/\tau_+) \text{si} x > 0 \\ A_- \exp(x/\tau_-) \text{sinon} \end{cases} \tag{1.20}$$

avec A_+ et A_- sont des constantes qui controlent la hauteur des courbes, et τ_+ et τ_- sont des constantes de temps qui contrôlent la raideur de la fonction.

Il y a d'autres variantes qui n'utilisent pas la fonction exponentielle, comme par exemple l'expression suivante :

$$\Delta w = \eta(x_{pre} - x_{tar})(w_{max} - w)^\mu \tag{1.21}$$

où x_pre est la trace présynaptique qui modélise l'historique des pics présynaptiques récents, dont la valeur est augmentée de 1 à l'arrivée d'un pic présynaptique, et diminuée exponentiellement en l'absence de pic. x_tar est la valeur cible de la trace présynaptique au moment d'un pic postsynaptique, où plus la valeur cible est élevée, plus la synapse sera faible. η est le taux d'apprentissage, w_{max} est le poids maximal, et μ détermine la dépendance de la mise à jour par rapport au poids précédent.

La figure 1.8 présente une illustration générale du fonctionnement de cette règle d'apprentissage. Lorsque le neurone postsynaptique dépasse un certain seuil, la règle STDP va calculer une différence de temps entre l'impulsion postsynaptique émis et chaque impulsion présynaptique. Ce gradient temporel va ensuite servir à renforcer ou affaiblir la connexion synaptique w_{ji} associée.

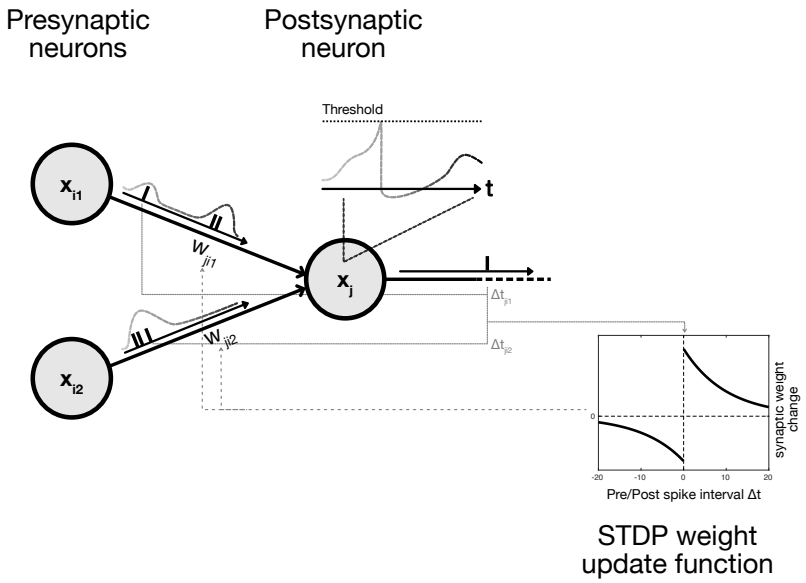


FIGURE 1.8 – Application de la règle STDP.

1.4.3 Autres méthodes

Plusieurs autres méthodes ont été formalisées ne rentrant pas dans les 2 principales catégories que nous avons définies ci-dessus.

SNN composé d'une seule couche Parmi elles, nous pouvons citer la méthode supervisé à distance (ReSuMe) de Ponulak [76] qui se compose de deux sous-ensembles : d'un sous-réseau à impulsions non entrainable suivi d'une couche de sortie. Le sous-réseau peut être associé à un LSM qui va générer une nouvelle représentation de nos données d'entrée en train d'impulsions qui va alimenter une couche de neurones. L'algorithme utilisant une règle de STDP et anti-STDP [4] afin de minimiser l'erreur entre le train d'impulsions en sortie et celui généré par des neurones enseignants.

Ce type de règle est aussi utilisé par Gutig et Sompolinsky dans leur tempotron [35]. Leur modèle est composé d'un unique neurone, dont la règle d'apprentissage modifie les connexions synaptiques en cas d'erreur. Plus précisément, lorsque le train impulsionnel entrant dans le neurone ne produit pas la réponse attendue en sortie, la connexion synaptique est renforcée. En revanche, si l'impulsion entrante ne doit pas engendrer de réponse en sortie mais que cela se produit néanmoins, la connexion synaptique est réduite. À noter que dans ce dernier cas, la règle STDP aurait renforcé la connexion synap-

4. Permet d'affaiblir les connexions synaptiques qui ont contribué négativement à la sortie correcte du réseau, contrairement à la règle STDP qui les renforce.

tique, car l'impulsion du neurone présynaptique et suivie d'une impulsion du neurone postsynaptique. Tout comme le perceptron, le tempotron doit itérativement apprendre à classifier les trains d'impulsions.

SNN multicouche Dans le même style que le tempotron, la méthode de STDP avec rétropropagation (BP-STDP) proposé par Tavanaei et Maida [88] permet d'entraîner un SNN multicouche avec les règles locales STDP et anti-STDP incorporé dans une méthode inspirée de la descente de gradient classique.

Algorithme évolutionniste (EA) Initialement inspirés de la théorie de l'évolution, les EAs sont des métaheuristiques basées sur les populations. Ces algorithmes peuvent être utilisés pour optimiser les poids synaptiques, la topologie du réseau, ainsi que les hyper-paramètres [17, 74, 80, 99]. Cependant, les EAs prennent beaucoup de temps à converger, notamment parce que la fonction de fitness est coûteuse en calcul [28].

1.5 Mécanismes intrinsèques aux SNNs

En plus de ces mécanismes d'apprentissage qui peuvent paraître plus ou moins bio inspirés, il est intéressant de noter qu'il existe d'autres mécanismes biologiques qui viennent compléter le comportement du neurone comme l'inhibition des neurones adjacents, ou encore le concept d'homéostasie. Ces mécanismes permettent entre autres de stabiliser le transit d'informations dans le système nerveux durant la phase de plasticité synaptique. Elle est comparable aux méthodes de régularisations utilisées dans l'apprentissage profond.

Dans le cas des SNN, ces mécanismes sont très utilisés lors de l'utilisation d'algorithmes d'apprentissage non supervisé de type hebbiens comme la STDP qui vont régir la mise à jour des poids synaptiques, car ils vont permettre de forcer les neurones à apprendre différents motifs [19, 24, 51].

1.5.1 Inhibition

Un des deux mécanismes intrinsèques du neurone impulsionnel est l'inhibition. Biologiquement, en plus des neurones excitateurs, il existe dans le cerveau un faible pourcentage de neurones inhibiteurs. Ces derniers agissent de manière opposée aux neurones excitateurs, c'est-à-dire que leur influx nerveux va inhiber le potentiel membranaire des neurones connectés. En modélisation informatique, cela revient simplement à avoir des valeurs de poids synaptiques [5] négatifs en plus de valeurs positives. Les neurones inhibiteurs vont donc empêcher d'autres neurones à réagir aux stimulus. En poussant ce concept plus loin, ce mécanisme permet d'ajouter une notion de compétition neuronale : lorsqu'un neurone d'une couche l_i dépasse son seuil d'activation, il émet une impulsion aux neurones postsynaptiques et inhibe certains neurones adjacents à cette couche l_i. Lorsque seul un neurone dans une couche l_i peut envoyer une impulsion, on parle d'une politique de *Winner-Take-All (WTA)*.

Pour faire une mise en relation avec les modèles fréquentiels très utilisés actuellement, ce mécanisme peut être comparé à certaines méthodes de « pooling » ou encore de « dropout ». Dans le cas du « pooling », plus particulièrement au « max-pooling », le

5. Pour rappel, le terme « poids » correspond à la connexion synaptique entre deux neurones.

but est de sous-échantillonner des données en gardant seulement une seule valeur (ici la valeur maximale), ce qui a pour conséquence d'inhiber la sortie de certains neurones adjacents d'une même couche l_i. Quant au « dropout », cette méthode désactive temporairement certains neurones du réseau. Ces 2 mécanismes sont généralement utilisés pour limiter le sur-apprentissage [6].

1.5.2 Homéostasie

> *« L'homéostasie est un phénomène par lequel un facteur clé (par exemple, la température) est maintenu autour d'une valeur bénéfique pour le système considéré, grâce à un processus de régulation. »*
>
> **Wikipédia**

Cette définition donnée sur Wikipédia résume parfaitement le concept de l'homéostasie. En d'autres termes, l'homéostasie est un mécanisme essentiel pour assurer l'équilibre du réseau de neurones lors de sa formation [14]. L'objectif est d'avoir un processus d'apprentissage qui soit stable, ce qui signifie que les poids des synapses et l'activité des neurones ne doivent pas diverger ni devenir totalement nuls, et qui soit sélectif, ce qui signifie qu'en fonction de l'activité d'entrée et de la topologie, certaines synapses sont potentialisées tandis que d'autres sont dépréciées afin d'obtenir des réponses spécifiques à différents stimuli.

Lors de l'apprentissage avec des règles locales de type hebbiennes, comme la règle STDP, les connexions synaptiques de certains neurones peuvent majoritairement être renforcées, et par conséquent envoyer beaucoup plus d'impulsions que les autres. Ce constat est dû au fait que ces règles ont une propriété générale de renforcement positif. Ce problème est encore plus marquant lorsque le modèle utilise un mécanisme d'inhibition comme le WTA dans lequel le réseau peut se retrouver bloqué dans un état où un seul neurone émet des impulsions, renforce ses connexions synaptiques associées tout en inhibant les autres neurones. Une méthode assez simple consiste à ajouter dans la règle hebbienne une nouvelle heuristique qui induit plus de dépression à long terme (LTD) que de potentialisation à long terme (LTP) en l'absence de corrélation [43, 64].

Le concept de l'homéostasie intervient donc en s'assurant que les neurones d'une même couche l_i ont la même opportunité d'apprendre de nouvelles données présentées. Elle se classe en deux catégories de modèles : les régularisations globales appelées « plasticité synaptique homéostatique » (régulation de l'excitation et l'inhibition), et les régularisations locales appelées « plasticité homéostatique intrinsèque » (excitabilité du neurone) [91].

Modèles de plasticité synaptique homéostatique

Ces modèles ont pour base la règle de Hebb sur laquelle des méthodes sont appliquées afin d'empêcher des poids synaptiques de croître indéfiniment. Pour cela, les modèles mettent en œuvre soit un mécanisme de normalisation des poids synaptique comme la règle d'Oja [71] ou simplement une mise à l'échelle des synapses [43, 54, 62], soit un mécanisme permettant de contrôler l'augmentation des poids synaptiques

6. *Overfitting* en anglais, cela caractérise un modèle prédictif très adapté aux données d'apprentissage et n'est donc plus assez généralisable sur des données nouvelles.

comme dans la théorie Bienenstock-Cooper-Munro (BCM) où l'accroissement est plafonné.

Mise à l'échelle des synapses Ce mécanisme permet de normaliser les poids synaptiques w lors de chaque mise à jour en appliquant une mise à l'échelle globale utilisant la norme l_1 [54] :

$$w_i = \frac{w_i}{\sum_j |w_j|} \tag{1.22}$$

Règle d'Oja Contrairement à la mise à l'échelle globale, la règle d'apprentissage d'Oja [71] propose un mécanisme local tel que l'accroissement permanent des poids synaptiques soit limité par un terme représentant l'oubli. L'informaticien Erkki Oja proposa que ce terme soit proportionnel à la valeur du poids synaptique et à la sortie du neurone :

$$\Delta w_i = \alpha(x_i y - y^2 w_i) \tag{1.23}$$

où x_i est le courant d'entrée de la i-ème synapse, w_i est le poids synaptique de i-ème synapse, y est la somme des courants d'entrée pondérés par les poids synaptiques ($y = \sum_i x_i w_i$), et α est le taux d'apprentissage – *learning rate* . Grâce au terme quadratique, le terme d'oubli $y^2 w_i$ permet d'équilibrer la croissance du poids synaptique en ayant un impact de plus en plus important dès que la sortie du neurone y est de plus en plus grande.

BCM La théorie BCM est un modèle de synapse utilisant un seuil mobile pour réguler la LTD et la LTP [8, 47]. Elle est modélisée par un produit de type hebbien de l'activité présynaptique couplé à une fonction non-linaire définissant le changement de signe en fonction d'un seuil mobile θ :

$$\Delta w_i = y(y - \theta)x_i - \epsilon w_i \tag{1.24}$$

où x_i est le courant d'entrée de la i-ème synapse, w_i est le poids synaptique de i-ème synapse, y est la somme des courants d'entrée pondérés par les poids synaptiques ($y = \sum_i x_i w_i$), et ϵ est la constante de temps de la décroissance uniforme de toutes les synapses. Le seuil de modification θ est variable et se calcule comme la puissance de la sortie du neurone, ou dit autrement comme la moyenne temporelle de y.

Modèles de plasticité homéostatique intrinsèque

Ce mécanisme a été pour la première fois modélisé par LeMasson *et al.* après leur découverte du maintien d'une activité électrique caractéristique sur des neurones des crustacés [2, 60]. Ils ont proposé un mécanisme de détection intégré aux neurones qui surveille l'activité électrique et ajuste la conductance G [7] pour maintenir un niveau d'activité cible. Ce mécanisme homéostatique permet donc de modifier directement l'excitabilité du neurone lui-même sans changer les courants synaptiques.

7. L'inverse de la résistance R = 1/G.

Seuil adaptatif La modélisation la plus simple consiste à ajuster le seuil du neurone en fonction de son activité afin qu'il atteigne une cible de taux d'impulsion moyen. [14, 54, 93]. Une fois que le neurone se décharge, le seuil est augmenté, puis, au bout d'un certain temps sans émission d'impulsion de la part du neurone, le seuil diminue. Par ce procédé, le seuil d'excitation du neurone va s'ajuster à une valeur d'équilibre.

Ces principales méthodes permettent d'imiter la plasticité homéostatique du cortex cérébral, mais l'une des grandes questions encore non résolues est comment choisir un mécanisme homéostatique approprié pour un schéma d'impulsions donné, une méthode d'apprentissage non supervisée choisie, etc. afin d'avoir une architecture d'un SNN relativement stable.

1.6 Conclusion

Dans ce chapitre, j'ai présenté un état de l'art des principes fondamentaux du connexionnisme impulsionnel en adéquation avec la première partie de cette thèse. Dans un premier temps, j'ai détaillé la manière avec laquelle le fonctionnement du neurone biologique a été traduit dans ses deux principales modélisations numériques actuelles, pour ensuite définir les différentes modélisations des neurones impulsionnels. Dans un second temps, j'ai développé les différents concepts propres aux réseaux de neurones à impulsions (SNNs), comme l'encodage et décodage des données, les différentes approches d'entrainement, ainsi que ses mécanismes de plasticité homéostatique.

Ce résumé des modèles impulsionnels va servir de base à une importante partie de mon travail présentée dans le chapitre 2 sur la création d'un outil pour construire ce type de réseaux, ainsi que dans le chapitre 3 qui présente une approche d'apprentissage d'un réseau de manière non supervisée.

Pour aller plus loin, j'invite le lecteur à lire les livres de Wulfram Gerstner [31, 32] présentant de manière complète les concepts biologiques et ses modélisations mathématiques des SNNs. Des états de l'art plus succincts existent aussi présentant une introduction plus générale des SNNs comme j'ai essayé de le faire ici [69, 73].

Je souhaiterais conclure ce chapitre par le tableau 1.7 permettant de donner un aperçu des principales différences de définition de concepts entre la communauté d'apprentissage profond et celle des neurosciences.

TABLE 1.7 – Comparaison entre un réseau de neurones artificiels (ANN) et un réseau de neurones à impulsions (SNN).

	ANN	**SNN**
Neurone (Section 1.2.1)	Neurone artificiel (sigmoid, tanh, ReLU, etc.)	Neurone impulsionnel (Hodgkin-Huxley, IF, Izhikevich, etc.)
Mode de calcul	Fonctions d'activation (Tableau 1.1)	Équations différentielles (Section 1.2.2)
Représentation de l'information	Scalaires	Trains impulsionnels (Section 1.3)
Caractéristiques principales	Optimisation des calculs, parallélisation des calculs, communauté vaste d'experts	Plus proche de la biologie, traitement des données en temps réel de manière non supervisée, peu énergivore

Références

[1] L. F. Abbott. “Lapicque’s introduction of the integrate-and-fire model neuron (1907)”. In : *Brain Research Bulletin* 50.5 (1999), pages 303-304.

[2] L. F. Abbott et Gwendal LeMasson. “Analysis of Neuron Models with Dynamically Regulated Conductances”. In : *Neural Computation* 5.6 (1993), pages 823-842.

[3] L. F. Abbott et Sacha B. Nelson. “Synaptic plasticity : taming the beast”. In : *Nature Neuroscience* 3.11 (2000), pages 1178-1183.

[4] E. D. Adrian et Yngve Zotterman. “The impulses produced by sensory nerve-endings : Part II. The response of a Single End-Organ”. In : *The Journal of Physiology* 61.2 (avr. 1926), pages 151-171.

[5] Guillaume Bellec et al. “Long short-term memory and Learning-to-learn in networks of spiking neurons”. In : *Advances in Neural Information Processing Systems*. Sous la direction de S. Bengio et al. Tome 31. Curran Associates, Inc., 2018.

[6] Guo-qiang Bi et Mu-ming Poo. “Synaptic Modifications in Cultured Hippocampal Neurons : Dependence on Spike Timing, Synaptic Strength, and Postsynaptic Cell Type”. In : *Journal of Neuroscience* 18.24 (1998), pages 10464-10472.

[7] Olivier Bichler et al. “Extraction of temporally correlated features from dynamic vision sensors with spike-timing-dependent plasticity”. In : *Neural Networks* 32 (2012), pages 339-348.

[8] E L Bienenstock, L N Cooper et P W Munro. “Theory for the development of neuron selectivity : orientation specificity and binocular interaction in visual cortex”. In : *Journal of Neuroscience* 2.1 (jan. 1982), pages 32-48.

[9] Sander M. Bohte, Joost N. Kok et Han La Poutré. “Error-backpropagation in temporally encoded networks of spiking neurons”. In : *Neurocomputing* 48.1 (2002), pages 17-37.

[10] Olaf Booij et Hieu Tat Nguyen. “A gradient descent rule for spiking neurons emitting multiple spikes”. In : *Information Processing Letters* 95.6 (2005), pages 552-558.

[11] Alexander Borst et Frédéric E. Theunissen. “Information theory and neural coding”. In : *Nature Neuroscience* 2 (1999), pages 947-958.

[12] Romain Brette. “Modèles Impulsionnels de Réseaux de Neurones Biologiques”. Theses. Université Pierre et Marie Curie - Paris VI, déc. 2003.

[13] Natalia Caporale et Yang Dan. “Spike timing-dependent plasticity : a Hebbian learning rule”. In : *Annual Reviews of Neuroscience* 31.1 (2008), pages 25-46.

[14] Kristofor D. Carlson et al. “Biologically plausible models of homeostasis and STDP : Stability and learning in spiking neural networks”. In : *The 2013 International Joint Conference on Neural Networks (IJCNN)*. 2013, pages 1-8.

[15] Xiang Cheng et al. “LISNN : Improving Spiking Neural Networks with Lateral Interactions for Robust Object Recognition”. In : *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*. Sous la direction de Christian Bessiere. Main track. International Joint Conferences on Artificial Intelligence Organization, juill. 2020, pages 1519-1525.

[16] Iulia M. Comsa et al. “Temporal Coding in Spiking Neural Networks with Alpha Synaptic Function”. In : *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2020, pages 8529-8533.

[17] J. David Schaffer. “Evolving spiking neural networks : A novel growth algorithm corrects the teacher”. In : *2015 IEEE Symposium on Computational Intelligence for Security and Defense Applications (CISDA)*. 2015, pages 1-8.

[18] Peter Dayan et L. F. Abbott. *Theoretical Neuroscience : Computational and Mathematical Modeling of Neural Systems*. The MIT Press, 2005.

[19] Peter U. Diehl et Matthew Cook. “Unsupervised learning of digit recognition using spike-timing-dependent plasticity”. In : *Frontiers in Computational Neuroscience* 9 (2015).

[20] Peter U. Diehl et al. “Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing”. In : *2015 International Joint Conference on Neural Networks (IJCNN)*. 2015, pages 1-8.

[21] Peter U. Diehl et al. “Conversion of artificial recurrent neural networks to spiking neural networks for low-power neuromorphic hardware”. In : *2016 IEEE International Conference on Rebooting Computing (ICRC)*. 2016, pages 1-8.

[22] G. Bard Ermentrout et Nancy Kopell. “Parabolic Bursting in an Excitable System Coupled with a Slow Oscillation”. In : *SIAM Journal on Applied Mathematics* 46.2 (1986), pages 233-253.

[23] Steve K Esser et al. “Backpropagation for Energy-Efficient Neuromorphic Computing”. In : *Advances in Neural Information Processing Systems*. Sous la direction de C. Cortes et al. Tome 28. Curran Associates, Inc., 2015.

[24] Pierre Falez et al. “Multi-layered Spiking Neural Network with Target Timestamp Threshold Adaptation and STDP”. In : *2019 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2019, 1–8.

[25] Richard FitzHugh. “Impulses and Physiological States in Theoretical Models of Nerve Membrane”. In : *Biophysical Journal* 1.6 (1961), pages 445-466.

[26] Nicolas Fourcaud-Trocmé et al. “How Spike Generation Mechanisms Determine the Neuronal Response to Fluctuating Inputs”. In : *Journal of Neuroscience* 23.37 (2003), pages 11628-11640.

[27] Jacques Gautrais et Simon J. Thorpe. “Rate coding versus temporal order coding : a theoretical approach”. In : *Biosystems* 48.1 (1998), pages 57-65.

[28] Andrey V. Gavrilov et Konstantin O. Panchenko. “Methods of learning for spiking neural networks. A survey”. In : *2016 13th International Scientific-Technical Conference on Actual Problems of Electronics Instrument Engineering (APEIE)*. Tome 02. 2016, pages 455-460.

[29] Wulfram Gerstner. “A framework for spiking neuron models : The spike response model”. In : *Neuro-Informatics and Neural Modelling*. Sous la direction de F. Moss et S. Gielen. Tome 4. Handbook of Biological Physics. North-Holland, 2001, pages 469-516.

[30] Wulfram Gerstner et J Leo van Hemmen. “Associative memory in a network of ‘spiking’ neurons”. In : *Network : Computation in Neural Systems* 3.2 (1992), pages 139-164.

[31] Wulfram Gerstner et Werner M. Kistler. *Spiking Neuron Models : Single Neurons, Populations, Plasticity*. Cambridge University Press, 2002.

[32] Wulfram Gerstner et al. *Neuronal Dynamics : From Single Neurons to Networks and Models of Cognition*. USA : Cambridge University Press, 2014.

[33] Ian Goodfellow, Yoshua Bengio et Aaron Courville. *Deep Learning*. MIT Press, 2016.

[34] C. M. Gray et al. "Oscillatory responses in cat visual cortex exhibit intercolumnar synchronization which reflects global stimulus properties." In : *Nature* 338 (1989), pages 334-337.

[35] Robert Gütig et Haim Sompolinsky. "The tempotron : a neuron that learns spike timing-based decisions". en. In : *Nat. Neurosci.* 9.3 (mars 2006), pages 420-428.

[36] Jesse Hagenaars, Federico Paredes-Vallés et Guido de Croon. "Self-Supervised Learning of Event-Based Optical Flow with Spiking Neural Networks". In : *Advances in Neural Information Processing Systems* 34 (2021).

[37] B. Han, G. Srinivasan et K. Roy. "RMP-SNN : Residual Membrane Potential Neuron for Enabling Deeper High-Accuracy and Low-Latency Spiking Neural Network". In : *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. Los Alamitos, CA, USA : IEEE Computer Society, 2020, pages 13555-13564.

[38] Donald O. Hebb. *The organization of behavior : A neuropsychological theory*. Wiley, juin 1949.

[39] Suzana Herculano-Houzel. "The human brain in numbers : a linearly scaled-up primate brain". In : *Front. Hum. Neurosci.* 3 (nov. 2009), page 31.

[40] Alain L. Hodgkin et Andrew F. Huxley. "A quantitative description of membrane current and its application to conduction and excitation in nerve". In : *J. Physiol.* 117.4 (août 1952), pages 500-544.

[41] John J. Hopfield. "Pattern recognition computation using action potential timing for stimulus representation". In : *Nature* 376.6535 (1995), pages 33-36.

[42] David H. Hubel et Torsten N. Wiesel. "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex". In : *The Journal of Physiology* 160.1 (jan. 1962), pages 106-154.

[43] James Humble, Susan Denham et Thomas Wennekers. "Spatio-temporal pattern recognizers using spiking neurons and spike-timing-dependent plasticity". In : *Front. Comput. Neurosci.* 6 (oct. 2012), page 84.

[44] Eric Hunsberger et Chris Eliasmith. "Spiking Deep Networks with LIF Neurons". In : *ArXiv* abs/1510.08829 (2015).

[45] Eugene M. Izhikevich. "Simple model of spiking neurons". In : *IEEE Transactions on Neural Networks* 14.6 (2003), pages 1569-1572.

[46] Eugene M. Izhikevich. "Which model to use for cortical spiking neurons ?" In : *IEEE Transactions on Neural Networks* 15.5 (2004), pages 1063-1070.

[47] Eugene M. Izhikevich et Niraj S. Desai. "Relating STDP to BCM". In : *Neural Computation* 15.7 (juill. 2003), pages 1511-1523.

[48] Yingyezhe Jin, Wenrui Zhang et Peng Li. “Hybrid Macro/Micro Level Backpropagation for Training Deep Spiking Neural Networks”. In : *Proceedings of the 32nd International Conference on Neural Information Processing Systems*. NIPS’18. Montréal, Canada : Curran Associates Inc., 2018, 7005–7015.

[49] Jacques Kaiser, Hesham Mostafa et Emre Neftci. “Synaptic Plasticity Dynamics for Deep Continuous Local Learning (DECOLLE)”. In : *Frontiers in Neuroscience* 14 (2020).

[50] Christoph Kayser et al. “Spike-Phase Coding Boosts and Stabilizes Information Carried by Spatial and Temporal Spike Patterns”. In : *Neuron* 61.4 (2009), pages 597-608.

[51] Saeed Reza Kheradpisheh et al. “STDP-based spiking deep convolutional neural networks for object recognition”. In : *Neural Networks* 99 (2018), pages 56-67.

[52] Jinseok Kim, Kyungsu Kim et Jae-Joon Kim. “Unifying Activation- and Timing-Based Learning Rules for Spiking Neural Networks”. In : *Proceedings of the 34th International Conference on Neural Information Processing Systems*. NIPS’20. Vancouver, BC, Canada : Curran Associates Inc., 2020.

[53] Louis Lapicque. “Recherches quantitatives sur l’excitation électrique des nerfs traitée comme une polarisation”. In : *J. Physiol. Pathol. Gen.* 9 (1907), pages 620-635.

[54] Andreea Lazar, Gordon Pipa et Jochen Triesch. “SORN : a self-organizing recurrent neural network”. In : *Frontiers in Computational Neuroscience* 3 (2009).

[55] Yann LeCun, Yoshua Bengio et Geoffrey Hinton. “Deep learning”. In : *Nature* 521.7553 (mai 2015), pages 436-444.

[56] Chankyu Lee et al. “Enabling Spike-Based Backpropagation for Training Deep Neural Network Architectures”. In : *Frontiers in Neuroscience* 14 (2020).

[57] Chankyu Lee et al. “Spike-FlowNet : Event-based Optical Flow Estimation with Energy-Efficient Hybrid Neural Networks”. In : *European Conference on Computer Vision*. Springer. 2020, pages 366-382.

[58] Jun Haeng Lee, Tobi Delbruck et Michael Pfeiffer. “Training Deep Spiking Neural Networks Using Backpropagation”. In : *Frontiers in Neuroscience* 10 (2016).

[59] Stefan Leijnen et Fjodor van Veen. “The Neural Network Zoo”. In : *Proceedings* 47.1 (2020).

[60] Gwendal LeMasson, Eve Marder et L. F. Abbott. “Activity-Dependent Regulation of Conductances in Model Neurons”. In : *Science* 259.5103 (1993), pages 1915-1917.

[61] Wolfgang Maass, Thomas Natschläger et Henry Markram. “Real-time computing without stable states : a new framework for neural computation based on perturbations”. In : *Neural Comput.* 14.11 (nov. 2002), pages 2531-2560.

[62] Christoph von der Malsburg. “Self-organization of orientation sensitive cells in striate cortex”. In : *Biological Cybernetics* 14 (jan. 1973), pages 85-100.

[63] Henry Markram, Wulfram Gerstner et Per Jesper Sjöström. “Spike-Timing-Dependent Plasticity : A Comprehensive Overview”. In : *Frontiers in Synaptic Neuroscience* 4 (2012).

[64] Timothée Masquelier, Rudy Guyonneau et Simon J. Thorpe. “Spike Timing Dependent Plasticity Finds the Start of Repeating Patterns in Continuous Spike Trains”. In : *PLOS ONE* 3 (jan. 2008), pages 1-9.

[65] Timothée Masquelier et Simon J. Thorpe. “Learning to recognize objects using waves of spikes and Spike Timing-Dependent Plasticity”. In : *The 2010 International Joint Conference on Neural Networks (IJCNN)*. 2010, pages 1-8.

[66] Sam McKennoch, Dingding Liu et Linda G. Bushnell. “Fast Modifications of the SpikeProp Algorithm”. In : *The 2006 IEEE International Joint Conference on Neural Network Proceedings*. 2006, pages 3970-3977.

[67] Hesham Mostafa. “Supervised Learning Based on Temporal Coding in Spiking Neural Networks”. In : *IEEE Transactions on Neural Networks and Learning Systems* 29.7 (2018), pages 3227-3235.

[68] Emre O. Neftci, Hesham Mostafa et Friedemann Zenke. “Surrogate Gradient Learning in Spiking Neural Networks : Bringing the Power of Gradient-Based Optimization to Spiking Neural Networks”. In : *IEEE Signal Processing Magazine* 36.6 (2019), pages 51-63.

[69] João D. Nunes et al. “Spiking Neural Networks : A Survey”. In : *IEEE Access* 10 (2022), pages 60738-60764.

[70] Peter O’Connor et al. “Real-time classification and sensor fusion with a spiking deep belief network”. In : *Frontiers in Neuroscience* 7 (oct. 2013), page 178.

[71] Erkki Oja. “A simplified neuron model as a principal component analyzer”. In : *J. Math. Biol.* 15.3 (1982), pages 267-273.

[72] Chethan M. Parameshwara et al. “SpikeMS : Deep Spiking Neural Network for Motion Segmentation”. In : *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2021, pages 3414-3420.

[73] Hélène Paugam-Moisy et Sander M. Bohte. “Computing with Spiking Neuron Networks”. In : *Handbook of Natural Computing*. Sous la direction de G. Rozenberg, T. Back et J. Kok. Springer-Verlag, sept. 2012, pages 335-376.

[74] N.G. Pavlidis et al. “Spiking neural network training using evolutionary algorithms”. In : *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.* Tome 4. 2005, 2190-2194 vol. 4.

[75] Thomas Pellegrini, Romain Zimmer et Timothée Masquelier. “Low-Activity Supervised Convolutional Spiking Neural Networks Applied to Speech Commands Recognition”. In : *2021 IEEE Spoken Language Technology Workshop (SLT)*. 2021, pages 97-103.

[76] Filip Ponulak. “ReSuMe - New Supervised Learning Method for Spiking Neural Networks”. In : 2005.

[77] John Rinzel et G. Bard Ermentrout. “Analysis of Neural Excitability and Oscillations”. In : *Methods in Neuronal Modeling : From Synapses to Networks*. MIT Press, 1989, 135–169.

[78] Bodo Rueckauer et al. “Conversion of Continuous-Valued Deep Networks to Efficient Event-Driven Networks for Image Classification”. In : *Frontiers in Neuroscience* 11 (2017).

[79] David E. Rumelhart, Geoffrey E. Hinton et Ronald J. Williams. “Learning Representations by Back-propagating Errors”. In : *Nature* 323.6088 (1986), pages 533-536.

[80] Abdulrazak Yahya Saleh et al. “A Novel hybrid algorithm of Differential evolution with Evolving Spiking Neural Network for pre-synaptic neurons Optimization”. In : tome 6. Mars 2014, pages 1-16.

[81] Ali Samadzadeh et al. *Convolutional Spiking Neural Networks for Spatio-Temporal Feature Extraction*. 2020.

[82] Alexander Sboev et al. “Solving a classification task by spiking neurons with STDP and temporal coding”. In : *Procedia Computer Science* 123 (2018), pages 494-500.

[83] Benjamin Schrauwen et Jan Van Campenhout. “Extending SpikeProp”. In : *2004 IEEE International Joint Conference on Neural Networks*. Tome 1. 2004, pages 471-475.

[84] Sumit Bam Shrestha et Garrick Orchard. “SLAYER : Spike Layer Error Reassignment in Time”. In : *Advances in Neural Information Processing Systems*. Sous la direction de S. Bengio et al. Tome 31. Curran Associates, Inc., 2018.

[85] Sumit Bam Shrestha et Qing Song. “Robustness to Training Disturbances in SpikeProp Learning”. In : *IEEE Transactions on Neural Networks and Learning Systems* 29.7 (2018), pages 3126-3139.

[86] Arnold J. F. Siegert. “On the First Passage Time Probability Problem”. In : *Phys. Rev.* 81 (4 1951), pages 617-623.

[87] Christoph Stöckl et Wolfgang Maass. “Optimized spiking neurons can classify images with high accuracy through temporal coding with two spikes”. In : *Nature Machine Intelligence* 3.3 (mars 2021), pages 230-238.

[88] Amirhossein Tavanaei et Anthony S. Maida. “BP-STDP : Approximating back-propagation using spike timing dependent plasticity”. In : *Neurocomputing* 330 (2019), pages 39-47.

[89] Simon J. Thorpe et Jacques Gautrais. “Rapid Visual Processing using Spike Asynchrony”. In : *Advances in Neural Information Processing Systems*. Sous la direction de M.C. Mozer, M. Jordan et T. Petsche. Tome 9. MIT Press, 1996.

[90] Simon J. Thorpe et Jacques Gautrais. “Rank Order Coding”. In : *Computational Neuroscience : Trends in Research, 1998*. Sous la direction de James M. Bower. Boston, MA : Springer US, 1998, pages 113-118.

[91] Gina Turrigiano. “Too Many Cooks ? Intrinsic and Synaptic Homeostatic Mechanisms in Cortical Circuit Refinement”. In : *Annual Review of Neuroscience* 34.1 (2011), pages 89-103.

[92] Alex Vigneron et Jean Martinet. “A critical survey of STDP in Spiking Neural Networks for Pattern Recognition”. In : *2020 International Joint Conference on Neural Networks (IJCNN)*. 2020, pages 1-9.

[93] Alanna J. Watt et Niraj S. Desai. “Homeostatic plasticity and STDP : keeping a neuron’s cool in a fluctuating world”. In : *Frontiers in Synaptic Neuroscience* 2 (2010).

[94] Yujie Wu et al. “Spatio-Temporal Backpropagation for Training High-Performance Spiking Neural Networks”. In : *Frontiers in Neuroscience* 12 (2018).

[95] Shuiying Xiang et al. “Spiking VGG7 : Deep Convolutional Spiking Neural Network with Direct Training for Object Recognition”. In : *Electronics* 11.13 (2022).

[96] Yan Xu et al. “A supervised multi-spike learning algorithm based on gradient descent for spiking neural networks”. In : *Neural Networks* 43 (2013), pages 99-113.

[97] Bojian Yin, Federico Corradi et Sander M. Bohté. “Effective and Efficient Computation with Multiple-Timescale Spiking Recurrent Neural Networks”. In : *International Conference on Neuromorphic Systems 2020*. ICONS 2020. Oak Ridge, TN, USA : Association for Computing Machinery, 2020.

[98] Kim Youngeun et Panda Priyadarshini. “Visual explanations from spiking neural networks using inter-spike intervals”. en. In : *Sci. Rep.* 11.1 (sept. 2021), page 19037.

[99] Zulhairi Mi Yusuf et al. “Evolving spiking neural network (ESNN) and harmony search algorithm (HSA) for parameter optimization”. In : *2017 6th International Conference on Electrical Engineering and Informatics (ICEEI)*. 2017, pages 1-6.

[100] Friedemann Zenke et Surya Ganguli. “SuperSpike : Supervised Learning in Multilayer Spiking Neural Networks”. In : *Neural Computation* 30.6 (avr. 2018), pages 1514-1541.

[101] Friedemann Zenke et Tim P. Vogels. “The Remarkable Robustness of Surrogate Gradient Learning for Instilling Complex Function in Spiking Neural Networks”. In : *Neural Computation* 33.4 (mars 2021), pages 899-925.

[102] Wenrui Zhang et Peng Li. “Spike-Train Level Backpropagation for Training Deep Recurrent Spiking Neural Networks”. In : *Advances in Neural Information Processing Systems*. Sous la direction de H. Wallach et al. Tome 32. Curran Associates, Inc., 2019.

Chapitre 2

PyTorch SNN, un simulateur SNN

Le développement d'un simulateur de réseau de neurones à impulsions (SNN) est essentiel dans le but de pouvoir comprendre les phénomènes biologiques qui entourent les neurones biologiques. Ces modèles représentent une approche alternative à la modélisation des réseaux de neurones plus traditionnels utilisée dans l'apprentissage profond – *deep learning* – et largement adoptée par la communauté scientifique. Pourtant, de nombreuses études de recherches s'attèlent à utiliser ces *nouveaux* modèles pour des applications telles que la reconnaissance de formes, la classification de signaux, la reconnaissance de la parole et la robotique.

C'est pour cela que nous proposons un nouvel outil simple à prendre en main : PyTorch SNN. PyTorch SNN se veut être une bibliothèque Python contenant différents modules afin de modéliser et de simuler des SNNs tout en gardant les mêmes niveaux d'abstraction qui caractérisent PyTorch : flexibilité et performance. Cette bibliothèque est spécifiquement orientée pour l'apprentissage automatique de modèles grâce à des règles de type hebbien comme la plasticité fonction du temps d'occurrence des impulsions (STDP), permettant d'assurer un apprentissage non supervisé.

Ce chapitre décrit l'architecture de la bibliothèque ainsi que les outils développés pour faciliter la boucle d'apprentissage. PyTorch SNN atteint des performances compétitives tant sur des tâches de classification provenant de la littérature, que sur la rapidité de calcul comparée à d'autres simulateurs. Ce dernier point est facilité grâce à l'utilisation de PyTorch en arrière-plan qui assure la compatibilité de la bibliothèque sur de multiples architectures (CPU et GPU).

Sommaire

2.1 Introduction

Les SNNs sont parfois appelés la « troisième génération » de réseaux de neurones [23], car ils représentent une nouvelle approche, plus biologique, pour simuler le fonctionnement des neurones par l'utilisation d'impulsions ponctuelles modélisant les signaux électriques qui circulent entre eux. Par cette caractéristique intrinsèque, les SNNs sont considérés comme plus à même à traiter des données contenant une composante temporelle. Ce fonctionnement binaire permettrait donc une simulation rapide et efficace énergétiquement sur un dispositif matériel.

2.1.1 Contexte

Ces différents attraits font qu'un SNN est un bon candidat pour un projet d'analyse temporelle embarquée. Cependant, ils ne présentent actuellement aucun cadre adapté à l'apprentissage machine (ML) car l'exploration de modèles reste compliquée et utilisable à travers des simulateurs développés pour de l'exploration d'hypothèses biologiques. Pourtant, des travaux depuis plusieurs années montrent qu'il est possible d'entrainer ce type de modèle afin d'extraire une nouvelle représentation des données d'entrée pour ensuite les classifier [15, 19].

Ce projet de développement se positionne sur deux axes : une collaboration avec le ministère de l'Intérieur et le développement d'une base expérimentale pour mes recherches.

Projet RAVI Dans le cadre de ce partenariat, une équipe du ministère de l'Intérieur spécialisée dans la vision par ordinateur cherchait à se doter d'une technologie utilisant les SNNs pour explorer les capacités d'apprentissage et par la suite, la possibilité d'utiliser des algorithmes sur cartes électroniques embarquées. Le système développé dans le cadre du projet RAVI – Reconnaissance Automatique de Véhicules dans les Images – doit pouvoir reconnaitre des objets simples dans une image telle que des visages ou des voitures, de manière non supervisée. Pour cela, une règle simulant la plasticité synaptique, comme la STDP, a été choisie car elle présente des applications intéressantes dans la reconnaissance d'image depuis quelques années [12, 17, 19]. L'objectif final du projet étant de prendre en main ce type de modèles et de les intégrer dans un projet interne de R&D, l'équipe du ministère cherchait une plateforme aisément exploitable par des ingénieurs, tout en étant facilement configurable pour différents tests, efficiente en temps de calcul, et facilitant l'implémentation de nouveaux modules.

2.1.2 Simulateurs existants

La simulation consiste à observer les performances d'un système neuronal impulsionnel selon le choix de différents éléments que nous avons présentés dans le chapitre 1 : la modélisation neuronale, la topologie de réseau, le code neural utilisé ainsi que le mécanisme d'apprentissage. Il existe actuellement deux méthodes pour travailler sur la simulation en temps discret des SNNs : les simulateurs logiciels ou le matériel neuromorphique dédié.

Simulation à partir d'une bibliothèque numérique L'approche logicielle est la solution la plus simple et la moins coûteuse à mettre en place. Elle consiste à installer une bibliothèque dans un langage défini, les plus récentes étant en Python, puis de coder le modèle avant de lancer la simulation. En raison de la nature discrète des impulsions, la conception d'une simulation efficace est un problème non trivial. Deux grandes catégories se distinguent dans ce domaine : les simulateurs pilotés par événements – *event-driven* – et ceux pilotés par l'horloge – *clock-driven* . Brette *et al.* [6] ont écrit un bon état de l'art sur ce sujet en 2007, qui reste toujours d'actualité.

La simulation événementielle est basée sur la gestion d'un ensemble d'événements. Ces événements peuvent se produire à des moments arbitraires, c'est-à-dire qu'ils sont associés à un temps spécifique. Les simulateurs traitent ces évènements dans l'ordre temporel et les exécutent au moment approprié. De ce fait, la quantité de données circulant dans le modèle reste très limitée. Par conséquent, ce type de modèle se comporte davantage comme de la simulation neuromorphique dans laquelle les composants sont activés lors d'événements entrants. Des simulateurs comme N2S3 [4] et NEURON [7] réalisent ce genre de simulation.

En revanche, la simulation pilotée par l'horloge, que j'appellerai aussi simulation séquentielle, implique l'exécution d'une séquence de tâches dans un ordre déterminé. À chaque pas de temps, les équations différentielles sont intégrées pour donner les valeurs au temps suivant. Cette approche est généralement la plus utilisée, car elle se rapproche des concepts de ML habituels. Elle est plus simple en mettre en œuvre et peut profiter des optimisations de calcul comme l'implémentation des routines sur GPU. Les simulateurs comme Bindsnet [16], Brian [40] ou encore Nengo [3] réalisent ce genre de simulation.

En plus des deux modélisations possibles, les simulateurs numériques peuvent se classer en trois catégories différentes :

1. les approches bio-réalistes visant à reproduire le comportement des neurones et des connexions synaptiques comme Brian [40], N2S3 [4] ou NEURON [7] ;
2. les approches de haut niveau axées sur la simulation du comportement neuronal comme Nengo [3] ;
3. et les approches plus récentes conçues comme outils d'intelligence artificielle (AI).

Dans cette dernière catégorie, nous retrouvons Bindsnet [16], Norse [35] et SpikingJelly [14]. Ces trois dernières bibliothèques sont basées sur PyTorch et sont plus simplement utilisables que toutes les autres mentionnées précédemment grâce à un équilibre entre des APIs de haut niveau et une modularité d'implémentation. Cela se rapproche des expériences que nous avons avec l'apprentissage profond.

Simulation à partir du matériel Si l'objectif est de vérifier les propriétés du modèle en termes de performances et de consommations d'énergie dans un contexte embarqué, l'approche logicielle n'est plus suffisante et doit donc être complétée par la simulation sur du matériel neuromorphique. Ce type de simulation peut être subdivisé en deux catégories : l'approche analogique et l'approche numérique. Une revue détaillée en a été proposé par Schuman *et al.* [39].

Sans trop rentrer dans les détails, l'approche analogique exploite des processus physiques pour modéliser des opérations de calculs comme l'utilisation d'un condensateur pour modéliser un neurone intègre-et-tire avec fuite (LIF) [28]. Nous pouvons citer en particulier la plateforme TrueNorth [2].

L'approche numérique quant à elle utilise une quantification des signaux qui sont traités par des puces électroniques, de manière similaire à ce que ferait une architecture de Von Neumann. Cependant, ces puces sont développées spécifiquement pour cette utilisation, comme cela est proposé par Intel en 2017 avec leur plateforme Loihi [9]. Ce type de système totalement dédié à la simulation, voire l'apprentissage de SNNs, étant très couteux, d'autres solutions ont été développées pour réduire ces coûts comme SpiNNaker [36] qui utilise des processeurs plus classiques de type ARM.

En général, ces méthodes sont limitées par les contraintes matérielles, ce qui a pour conséquence de ne pas autoriser la définition de modèles atypiques. De plus, ces plateformes sont dépendantes exclusivement de leurs bibliothèques et n'utilisent pas de langages de programmation compatibles entre eux.

Interface de programmation de haut niveau Afin de développer aussi bien sur une plateforme logicielle que matérielle, il existe un langage indépendant du simulateur pour produire un modèle SNN appelé PyNN [11]. Ce langage est très intéressant, mais ne permet pas de coder de nouveaux types de modules car l'interface de progammation (API) de haut niveau se traduit ensuite par un ensemble d'appels de fonctions propres aux simulateurs utilisés en arrière-plan.

2.1.3 Contribution

Comme nous avons pu voir, il existe actuellement une multitude de bibliothèques Python permettant de simuler ce type de réseau de neurones. Mais une question se pose :

Pourquoi développer un nouveau simulateur ?

Par abus de langage, j'utiliserai souvent dans la suite du manuscrit le terme « simulateur » à la place de « bibliothèque logicielle », là où un simulateur est utilisé pour imiter le comportement d'un système réel alors qu'une bibliothèque logicielle est utilisée pour fournir aux développeurs des fonctionnalités à intégrer dans leurs propres applications.

Contexte et argumentation des choix Le projet RAVI débute en 2017, et le choix de développer une bibliothèque devait se faire en fonction de l'état de l'art du moment. Deux points étaient alors à prendre en considération.

D'un côté, l'apprentissage profond – *deep learning* – était en plein essor avec des performances qui se révélaient, même à l'heure actuelle, toujours plus impressionnantes les unes que les autres. Ce succès, dû en partie à l'accélération des primitives sur GPU, a conduit au développement de plusieurs bibliothèques, généralement en Python, permettant d'implémenter et d'entraîner des modèles d'apprentissage profond de manière assez simple [1, 5, 8, 32].

De l'autre, les simulateurs précédemment mentionnés étaient développés par une communauté issue de la neuroscience pour des neuroscientifiques, ce qui pouvait en limiter l'usage. Par exemple, ces simulateurs pouvaient demander d'implémenter les équations différentielles. La définition d'un modèle particulier s'avérait compliquée, et la prise en main de ces outils était généralement fastidieuse pour les novices. Après le commencement de ma thèse, quelques bibliothèques sont apparues telles que Bindsnet en 2018 [16] suivi quelques années plus tard du développement de Norse [35] et plus particulièrement SpikingJelly [14] en 2020. Ces bibliothèques sont intéressantes, car elles montrent d'une part la volonté d'un développement commun par une communauté scientifique interdisciplinaire et d'autre part, un accroissement de l'intérêt pour les SNNs.

Une plateforme d'expérimentation a donc été développée à partir de l'ensemble des connaissances apportées par l'état de l'art du chapitre précédent et ma connaissance personnelle des différents outils de ML appelée PyTorch SNN.

PyTorch SNN Afin d'associer un cadre expérimental pour ma thèse au projet partenarial avec le Ministère, nous avons opté pour le développement d'un nouveau simulateur. En basant le développement de ce simulateur sur PyTorch [32], une bibliothèque de calcul tensoriel, nous arrivons à lier l'utilisation des progrès apportés par l'apprentissage profond aux SNNs. Sous le terme progrès, j'entends aussi bien toutes les accélérations matérielles, que la manière de concevoir les APIs d'une bibliothèque moderne en ML utilisée par une communauté scientifique d'horizons variés. Ces éléments devaient répondre aux principales problématiques de l'équipe du Ministère concernant l'efficience de la bibliothèque par l'abstraction matérielle et la facilité de développement de nouveaux modules à travers le langage Python et des APIs similaires à celles de PyTorch.

Notre bibliothèque, appelée PyTorch SNN, permet aux utilisateurs de construire et d'entraîner des SNNs. L'apprentissage de ces modèles fait appel seulement à différentes implémentations de la plasticité synaptique pour respecter notre contrainte d'apprentissage non supervisé. Afin de faciliter son usage, un script d'apprentissage permettant de charger automatiquement les éléments nécessaires à la formation d'un SNN est fourni.

Ce développement a été réalisé en même temps que celui de Bindsnet [16] basé, lui aussi, sur PyTorch et utilisant des règles d'apprentissage exploitant la plasticité synaptique, mais présentant toute une mécanique interne différente.

Structure du chapitre Ce chapitre est structuré en 3 principales parties :

1. nous commençons par décrire PyTorch SNN de manière générale en présentant ses principes de conception, les différents modules ainsi que des exemples d'implémentation ;
2. nous évaluons la bibliothèque par l'implémentation de différents modèles issus de la littérature aussi bien dans le traitement de contenus audios que d'images ;
3. nous cherchons à comparer les performances du simulateur à des simulateurs existants sur la rapidité d'exécution.

2.2 PyTorch SNN

Dans cette section, je vais présenter PyTorch SNN, une bibliothèque de SNNs utilisant des méthodes d'apprentissage basées sur la STDP, en partant de l'idée de sa conception initiale jusqu'à son implémentation à travers des exemples de modèles de la littérature.

FIGURE 2.1 – Logo de PyTorch SNN.

2.2.1 Principe de conception

J'ai développé PyTorch SNN afin de fournir une solution flexible et puissante pour simuler et entraîner des SNNs utilisant des règles hebbiennes. Afin de répondre à ces exigences, les principes de conception reposent sur les 3 critères suivants :

Be Pythonic La communauté des chercheurs et des ingénieurs en R&D connaissent bien le langage Python qui leur permet de développer rapidement de nouvelles fonctionnalités à l'aide d'une syntaxe claire et concise sans avoir besoin de connaissances approfondies en programmation bas niveau. Être « pythonique » consiste donc à respecter les conventions et les pratiques de développement de cette communauté, leur offrant ainsi une facilité de prise en main et une intégration efficace à leurs structures de données et aux bibliothèques standards. Dans la communauté de l'apprentissage machine, 2 bibliothèques Python sortent du lot comme des réussites de développement : PyTorch [32] et scikit-learn [34]. La première joue sur l'abstraction du calcul tensoriel et différentiel sur différents types de processeur (CPU, GPU, TPU) permettant une définition simpliste d'un modèle complexe sans se soucier des optimisations sous-jacentes, tandis que la seconde vise la simplicité d'utilisation autorisant l'exploration d'algorithmes existants en seulement quelques lignes de codes. Ainsi, PyTorch SNN se place dans la même optique que celles-ci en laissant l'utilisateur libre de définir manuellement un modèle SNN complexe avec des APIs simples d'utilisation tout en ayant la possibilité d'avoir un

système « boîte noire » pour la reproductibilité des modèles issus de la littérature à la manière de scikit-learn.

Modularité Le deuxième principe concerne la modularité de la bibliothèque, c'est-à-dire la possibilité d'ajouter de nouvelles fonctionnalités aussi simplement que possible. PyTorch SNN utilise pour cela la même logique que PyTorch avec un module Python par concept (chargement des données, définition du modèle, apprentissage, etc.). Une classe abstraite est définie dans la plupart de ces modules permettant de gérer en interne les parties complexes. L'utilisateur peut ainsi définir un nouveau type de neurones sans se soucier de la gestion de la trace des dernières impulsions par exemple.

Performance Le dernier point touche la performance de la bibliothèque afin d'utiliser de manière efficace les ressources systèmes et de minimiser les temps d'exécution. Cela est accompli grâce à la plateforme PyTorch [32] utilisée en arrière-plan – *backend* – qui offre des fonctionnalités avancées pour les opérations tensorielles sur divers processeurs ainsi que la gestion de la parallélisation des différentes tâches. Un autre avantage d'utiliser ce type de plateforme est qu'elle est maintenue par une large communauté permettant d'être à jour sur toutes les dernières optimisations possibles.

2.2.2 Architecture

D'un point de vue architectural, PyTorch SNN fonctionne seulement à partir de Python 3.6 et peut être vu de deux manières différentes :

- comme une bibliothèque disposant d'une collection de routines allant du chargement des données à l'utilisation de règles d'apprentissage ;
- comme un simulateur clé en main avec un script principal qui charge un fichier de configuration au format YAML.

Comme le montre le tableau 2.1, notre bibliothèque dispose de ces 2 niveaux d'abstractions dont les APIs sont directement inspirées de PyTorch [32] et scikit-learn [34]. Les utilisateurs habitués à PyTorch ne seront pas déstabilisés dans le démarrage d'un nouveau projet présentant une structure du paquet similaire. L'utilisateur se retrouve à définir manuellement toutes les étapes de formation d'un SNN en plus du modèle. Une abstraction de plus haut niveau a aussi été implémentée via la classe `Trainer` afin de cacher toute la boucle d'apprentissage.

PyTorch SNN repose sur 6 modules Python :

1. `torch_snn.datasets` contenant les jeux de données utilisés dans un format permettant de charger aussi bien le jeu d'apprentissage que le jeu d'évaluation dans une seule classe sans devoir charger les 2 ensembles séparément ;
2. `torch_snn.transforms` permettant la transformation des données en train d'impulsions ;
3. `torch_snn.nn` regroupant l'ensemble des opérateurs nécessaires à la formulation d'un SNN ;
4. `torch_snn.model` regroupant des modèles de la littérature ;
5. `torch_snn.optim` définissant les différentes stratégies de mises à jour des connexions synaptiques du modèle, en particulier des méthodes de type hebbien ;
6. `torch_snn.utils` qui contient toutes les fonctionnalités utiles telles que les fonctions de visualisations.

TABLE 2.1 – Comparaison des paquets – *packages* – à importer pour commencer un nouveau script entre PyTorch et PyTorch SNN.

API	PyTorch	PyTorch SNN

Low-level	`import torch`
	`import torch.nn as nn`
	`import torch.nn.functional as F`
	`import torch.optim as optim`
	
	`from torchvision import datasets`
	`from torchvision import transforms`
		`import torch`
		`import torch_snn.nn as nn`
		`import torch_snn.nn.functional as F`
		`import torch_snn.optim as optim`
		
		`from torchvision import datasets`
		`from torch_snn import transforms`

High-level	–	`from torch_snn import datasets`
		`from torch_snn import models`
		`from torch_snn import Trainer`

Dans les paragraphes suivants, nous explorons plus en détails l'anatomie d'un script d'apprentissage pour former un SNN qui exploite les 6 modules ci-dessus.

Hyperparamètres Comme dans tout modèle de ML, les modèles de SNNs reposent sur la définition d'hyperparamètres. Ceux-ci sont généralement vus comme les paramètres du modèle fixés avant l'entraînement tels que le nombre de couches, le nombre de neurones, le taux d'apprentissage – *learning rate* –, etc. Dans notre cas, élargissons la définition de ce terme à tous les éléments permettant la construction d'un modèle dans son intégralité tels que le choix de la fonction d'optimisation, le corpus à utiliser, etc.

Pour cela, deux manières de les définir sont proposées dans PyTorch SNN : soit directement depuis le fichier script comme n'importe quelle autre modèle en Python, soit depuis un fichier de configuration qui sera automatiquement chargé au lancement du script. Dans ce dernier cas, PyTorch SNN s'appuie sur le format YAML [1] et la bibliothèque Hydra [44] [2]. Cette dernière nous permet d'instancier une configuration hiérarchique à partir de plusieurs sources, de surcharger les paramètres en ligne de commande et d'exécuter plusieurs tâches avec des arguments différents à l'aide d'une seule commande.

Le code source 1 présente un exemple de définition du modèle impulsionnel de Kheradpisheh *et al.* [19]. La structure de fichier est composée de 4 parties : le corpus – *dataset* – à utiliser, une liste de transformations – *transform* – à appliquer sur le jeu de données, le modèle – *model* – à instancier pour notre tâche, et l'optimiseur – *optim* – à lancer pour l'apprentissage du modèle à chaque inférence des données. Par sa structure simpliste, ce type de fichier de configuration permet une lecture plus claire d'une expérimentation d'un modèle facilitant ainsi sa reproductibilité.

1. YAML (Yet Another Markup Language) est un format de données en texte simple utilisé pour stocker et transmettre des informations structurées. Sa principale particularité est qu'il est facile à lire et à écrire pour les humains.

2. Plateforme de configuration flexible pour les applications Python ajoutant de nouvelles possibilités au format de fichier YAML

```yaml
dataset:
    _target_: "torch_snn.datasets.MNIST"
    data_dir: /data/

transform:
  - _target_: "torch_snn.transforms.DoG"
    size: 7
    sigma1: 1
    sigma2: 2
  - _target_: "torch_snn.transforms.RankOrderCoding"
    bins: 15

model:
    _target_: "torch_snn.models.Kheradpisheh"

optim:
    _target_: "torch_snn.optim.STDP"
    a_plus: 0.004
    a_minus: 0.003

batch_size: 1
epoch: 1
```

Code source 1 – Extrait du fichier de configuration `kheradpisheh_model.yaml` pour la spécification des hyperparamètres de l'expérience.

Le code source 2 montre la commande permettant d'exécuter le simulateur à partir de notre fichier présenté par le code source 1 à travers l'argument d'entrée *experiment*. La modification d'une clé du fichier de configuration peut se faire simplement par une nouvelle association en ligne de commande, ce qui est utile lors de l'optimisation du modèle via une recherche par grille – *grid search* –, c'est-à-dire en testant toutes les combinaisons des valeurs pour trouver les meilleurs hyperparamètres.

```
$ python train.py experiment=kheradpisheh_model dataset.data_dir="./data"
```

Code source 2 – Ligne de commande pour exécuter le script.

Chargement et transformation des données Les modules `torch_snn.datasets` et `torch_snn.transforms` gèrent d'une part le chargement de certains jeux de données utilisés pendant la thèse et d'autre part, les transformations qui peuvent être appliquées sur le corpus chargé, qu'il soit dans le domaine de l'audio, de l'image ou de la vidéo. La procédure de chargement et de transformation des données est identique à PyTorch avec quelques améliorations concernant la transformation des données. Cette dernière est généralement appliquée à chaque requête d'obtention d'une nouvelle image. Comme la transformation en train d'impulsions peut s'avérer être une tâche longue, nous autorisons la transformation des données comme une partie intégrante du réseau de neurones. Cela permet d'utiliser le traitement sur GPU en plus du CPU, mais également de traiter directement, en un seul passage, des lots – *batches* – de données. Chaque nouveau module de transformation doit hériter de la classe `torch.transforms.TransformModule` qui permet de détecter automatiquement si nous utilisons la méthode classique ou si les transformations s'appliquent en tête du réseau. Cela est rendu possible par la détection

de la présence de la dimension correspondant à la taille du lot – *batch*. Actuellement, le module `torch_snn.transforms` implémente plusieurs types d'encodage fréquentiel ainsi qu'un encodage temporel basé sur la loi Poisson.

Gestion de la temporalité Le concept le plus important dans les SNNs est le « temps ». Cette temporalité est modélisée de la même manière que pour les réseaux récurrents, c'est-à-dire qu'elle correspond à la première dimension du vecteur de nos données. Contrairement aux autres simulateurs qui utilisent le temps exact, PyTorch SNN travaille avec des pas de temps unitaires.

La dimension du vecteur des données qui transitent dans tout le modèle s'écrit sous la forme d'un triplet (T, B, H) où T est la taille de la séquence du train d'impulsions, B est la taille du lot – *batch* – de données, et H correspond à la dimension des données à un instant t de T.

Modélisation des couches neuronales impulsionnelles PyTorch SNN fournit un ensemble de modèles de neurones dans le module `torch_snn.nn` tels que la modélisation LIF et Izhikevich dont les détails ont été présentés dans le chapitre précédent. Les neurones sont représentés sous forme de cellules comme présenté dans la figure 2.2. La cellule prend en entrée des états cachés ainsi que le train d'impulsion à un instant t. Les états cachés correspondent aux représentations internes qui capturent l'information importante à partir de l'entrée à un instant t pour l'utiliser dans la prédiction future $t + 1$. Cette représentation est comparable à celle des cellules récurrentes que l'on retrouve dans le domaine de l'apprentissage profond.

Toutes les nouvelles modélisations doivent hériter de la classe `torch_snn.nn.Module` qui contient les mécanismes nécessaires à la trace des impulsions des neurones pré- et postsynpatiques servant aux calculs de la mise à jour des poids du modèle.

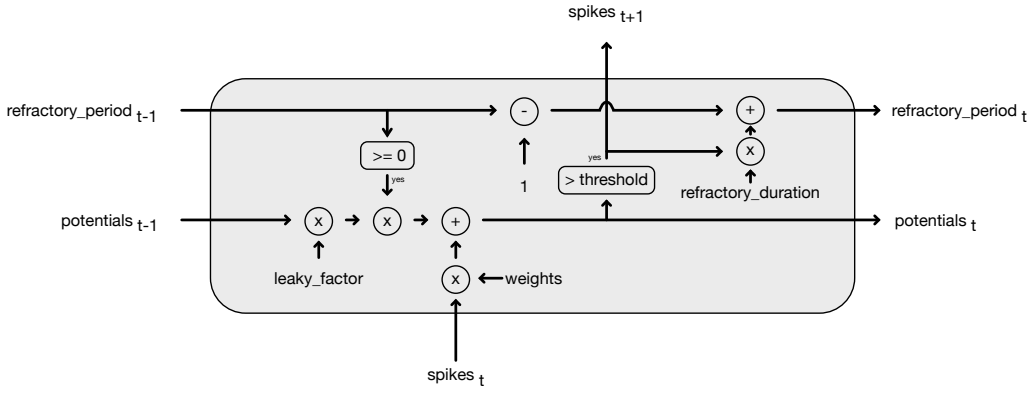


FIGURE 2.2 – Schéma de la modélisation d'une cellule correspondant au neurone LIF.

Définition de la topologie du modèle La topologie se définit de manière identique à PyTorch et comme la modélisation d'une couche neuronale. La classe du modèle doit hériter de `torch_snn.nn.Module`, puis l'utilisateur est libre de mélanger les modules de PyTorch dans Pytorch SNN.

Optimisation L'optimisation d'un modèle consiste à ajuster les paramètres du modèle pour améliorer ses performances, généralement en minimisant une mesure de perte. Dans notre cas, nous cherchons à optimiser les poids du modèle en fonction de règles locales basées sur les activités neuronales. Plus précisément, la STDP cherche à renforcer la connexion synaptique si l'impulsion générée par le neurone postsynaptique est une conséquence directe du neurone présynaptique, sinon la connexion synaptique est atténuée. Pour utiliser ce principe, nous devons calculer un gradient temporel entre les neurones pré- et post-synaptiques. Le gradient temporel vise à exploiter les informations de temporalité des impulsions neuronales pour optimiser les performances du réseau. Concrètement, il s'agit d'estimer comment un changement donné dans le poids d'une synapse affecte la variation temporelle des impulsions neuronales, et d'ajuster le poids en fonction de cette estimation. Cette méthode permet de tenir compte du décalage temporel entre les impulsions neuronales et de prendre en compte les interactions entre les neurones sur une échelle temporelle fine.

Le module `torch_snn.optim` entre en jeu ici pour relever ce défi. Lors du développement de la bibliothèque, le principal enjeu fut de trouver un moyen de représenter efficacement la trace des neurones. Initialement, pour modéliser le gradient temporel entre les impulsions des neurones pré- et post-synaptiques, nous avons adapté la classe `torch.autograd.Function` qui gère le calcul *forward* et *backward* de tous les modules PyTorch. Mais, au fur et à mesure des mises à jour de PyTorch, de multiples contraintes ont été imposées afin de garantir une sécurité sur le calcul du gradient dans le contexte d'un apprentissage profond et de moins en moins adaptées à notre situation. Nous avons donc abandonné ce mécanisme qui devenait compliqué à maintenir.

L'implémentation finale utilise la méthode `register_buffers` incorporée dans `torch.nn.Module` pour sauvegarder une trace temporelle des dernières impulsions des neurones.

La définition d'un nouvel optimiseur est identique à celle de PyTorch (la nouvelle classe hérite de `torch.optim.Optimizer`) excepté qu'elle prend en entrée l'ensemble des paramètres ainsi que l'ensemble des buffers.

Plusieurs règles d'apprentissage issues de la littérature sont implémentées dans le module `torch_snn.optim` telles que la règle de Hebb, plusieurs formules de la STDP [12, 19], et une variante de la STDP dédiée à l'apprentissage par renforcement : la STDP modulée par la récompense (R-STDP) [29, 30].

Extraction des représentations L'usage final de cette bibliothèque est de pouvoir extraire de nouvelles représentations de nos données grâce à des modèles de connexionnismes temporels en appliquant les différents codages vu dans la section 1.3.2. Une fois l'apprentissage du modèle réalisé, 3 méthodes sont proposées dans le module `torch_snn.utils` afin de générer notre représentation finale.

- la première méthode calcule la fréquence d'émissions des impulsions de chaque neurone de la couche de sortie du modèle ;
- la deuxième technique dévoile la durée requise pour chaque neurone afin de produire leurs premières impulsions ;
- la dernière méthode désactive le seuil interne des neurones impulsionnels de la dernière couche pour les considérer comme des intégrateurs purs, puis extrait le potentiel de membrane final.

La décision peut être prise suivant une heuristique que l'utilisateur aura définie avant ou tout simplement par un classifieur entrainé à partir de ces nouvelles représentations.

La chaîne de traitement Finalement, pour entraîner un modèle, notre système de ML a besoin de seulement 3 éléments : un jeu de données, un modèle, et la méthode permettant d'optimiser les connexions synaptiques. Pour simplifier sa prise en main et éviter les tâches répétitives, la bibliothèque dispose d'un script de base qui charge automatiquement le fichier de configuration et qui en extrait les différents éléments. Ces éléments sont ensuite instanciés et envoyés à la classe `Trainer` qui implémente la boucle d'entraînement générale. Dans le même esprit que scikit-learn, la classe `Trainer` dispose des méthodes `.fit()`, `.predict()` et `.score()`. La classe `Trainer` gère aussi la mise à jour des outils de visualisation, la sauvegarde et le chargement des modèles.

Concernant l'apprentissage multicouche, apprendre toutes les couches simultanément est possible, mais inutile, car la première couche n'est pas encore formée au début de l'apprentissage. Afin d'éviter de générer inutilement des calculs, 2 possibilités s'offrent à nous : entrainer manuellement couche par couche, ou utiliser une heuristique qui permette de changer automatiquement de couche durant l'apprentissage. La classe `Trainer` implémente l'heuristique suivante pour entraîner la couche C :

$$C = \sum_f \sum_i \frac{1}{n_f} w_{f,i}(1 - w_{f,i}) \tag{2.1}$$

où $w_{f,i}$ correspond à la i-ème connexion synaptique associé au f-ème vecteur de caractéristiques – *features* –[3], et n_f est le nombre de connexions synaptiques associé au f-ème vecteur de caractéristiques – *features* – et permet ainsi de normaliser le calcul. Le calcul tend vers 0 si la somme des valeurs des connexions synaptiques associées à chaque vecteur de caractéristiques – *features* – f converge vers 1 ou 0. L'arrêt de l'apprentissage est réalisé si cette mesure est suffisamment proche de 0.

Visualisation Des méthodes sont présentes dans le module `torch_snn.utils` pour récupérer et formater des données du modèle afin de les rendre visualisables par différentes bibliothèques graphiques comme matplotlib. L'utilisateur peut ainsi observer l'image d'entrée, le train d'impulsions généré, les connexions synaptiques et la sortie du modèle.

Lors de l'utilisation de la classe `Trainer`, la visualisation des données pendant l'apprentissage est pré-configurée pour une utilisation avec les biblothèques visdom ou tensorboard.

2.2.3 Exemples d'implémentation

Illustrons maintenant l'implémentation d'un SNN sur la base de 2 modèles adaptés de la littérature : un modèle dense proposé par Diehl & Cook [12] et un modèle convolutif proposé par Kheradpisheh *et al.* [19]. Ces deux modèles ont été entrainés sur une seule époque – *epoch* – avec le jeu de données MNIST que nous présenterons dans la section suivante. L'implémentation des modèles utilise les hyperparamètres par défaut définis par leurs auteurs respectifs.

Réseau de neurones denses Le modèle de Diehl & Cook [12] est initialement composé de 2 couches associées entre elles : une couche de neurones excitateurs et une

3. Un seul vecteur de caractéristiques – *features* – dans le cas d'une couche dense, mais plusieurs dans une couche convolutive correspondant au nombre de noyaux – *kernel* – de convolution.

couche de neurones inhibiteurs. Pour simplifier le modèle, nous proposons de supprimer cette dernière pour appliquer une inhibition latérale selon un principe de Winner-Take-All (WTA) qui permet de maintenir un comportement similaire sans le délai d'impulsions inhibitrices issues de la couche de neurones inhibiteurs. Plus précisément, lorsqu'un neurone excitateur émet une impulsion, au lieu d'envoyer l'information à un neurone inhibiteur, l'information d'inhibition est directement reliée aux neurones latéraux. La figure 2.3 présente ce modèle adapté. Son implémentation est illustré par le code source 3. Ce dernier se comprend facilement et correspond aux habitudes de développement en ML.

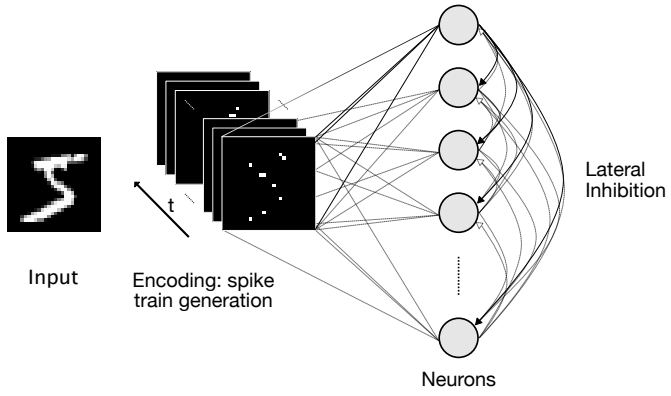


FIGURE 2.3 – Architecture adaptée du modèle proposé par Diehl & Cook [12].

La figure 2.4 présente l'exécution de la première partie du code concernant le chargement et la transformation du corpus sur une image. La figure 2.5 montre les connexions synaptiques formées (les poids du réseau).

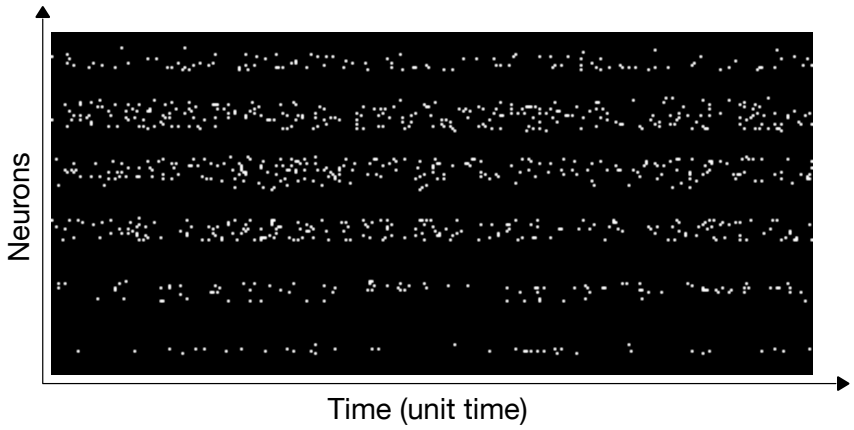


FIGURE 2.4 – Encodage du chiffre « 5 » en trains d'impulsions.

```python
import torch
import torch_snn.nn as nn
import torch_snn.optim as optim
import torch_snn.nn.functional as F

from torchvision import datasets
from torch_snn import transforms

transform = transforms.Compose([
    transforms.ToTensor(),  # transformation linéaire entre 0 et 1
    transform.FrequencyCoding(350)
])

train_dataset = datasets.MNIST("data/", train=True, download=True,
    transform=transform)
train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=1,
    shuffle=True)

test_dataset = datasets.MNIST('data/', train=False, download=False,
    transform=transform)
test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=1)

model = nn.LinearLIF(input_size=28*28, threshold=-52.0, reset=-65,
    refractory_period=5, wta=True)

optimizer = optim.PrePostSTDP(model.stdp_parameters(),
    a_plus=0.01, a_minus=0.0001, w_min=0, w_max=1.0)

for batch_idx, (spike_trains, _) in enumerate(train_loader):
    voltages = torch.zeros(28*28)
    for data in spike_trains:
        spikes, voltages = model(data, hidden=voltages)
        optimizer.step()
```

Code source 3 – Implémentation du modèle adapté de Diehl & Cook [12] avec PyTorch SNN.

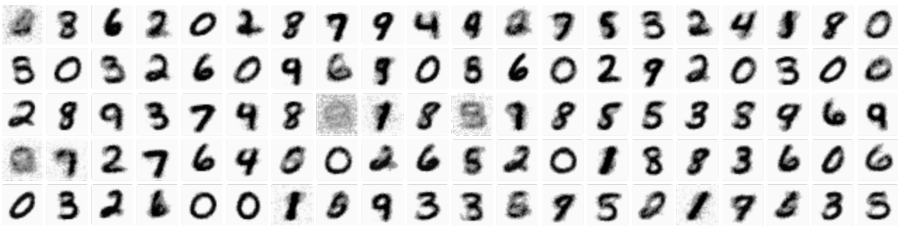


FIGURE 2.5 – visualisation des poids des 28x28 connexions synaptiques pour chacun des 100 neurones du modèle de Diehl & Cook [12] après apprentissage.

Réseau de neurones convolutifs impulsionnels Pour ce second exemple d'implémentation, je propose une autre manière de charger les données et d'entrainer le modèle grâce à des fonctions spécifiques à PyTorch SNN. Le modèle utilisé ici est un modèle convolutif proposé par Kheradpisheh *et al.* [19] qui est construit sur une détection de contours suivi d'un codage par rang (ROC) sur une durée de 15 unités de temps et d'une architecture composée de 2 couches de convolution. La première couche se compose de 15 filtres de taille 5x5 et une seconde couche de 60 filtres de taille 7x7. Afin que chaque filtre apprenne un motif spécifique, des mécanismes d'inhibition sont implémentés sur les cartes caractéristiques – *feature map* – afin que chaque filtre – *kernel* – apprenne un motif spécifique. Le concept de l'inhibition ici consiste à avoir une seule impulsion par carte caractéristique et différente spatialement entre les cartes connexes pour ensuite lancer la règle d'apprentissage STDP. Cette dernière va permettre de modifier les poids des filtres.

La figure 2.6 présente le schéma du modèle. Son implémentation est illustré par le code source 4. Le module de transformation des données n'est plus associé comme un paramètre du module `torch.utils.data.Dataset` mais comme partie intégrante du modèle. Cela permet d'appliquer le pré-traitement sur plusieurs données en parallèle et sur un GPU.

Le second changement vient de l'utilisation du module clé en main `Trainer`. Ce module récupère tous les éléments nécessaires à l'apprentissage du modèle et propose une multitude de paramètres pour gérer le nombre d'époques – *epochs* –, la visualisation, etc. Ces paramètres peuvent aussi être instanciés depuis le fichier de configuration vu précédemment.

La figure 2.7 montre les poids des deux couches extraits après apprentissage.

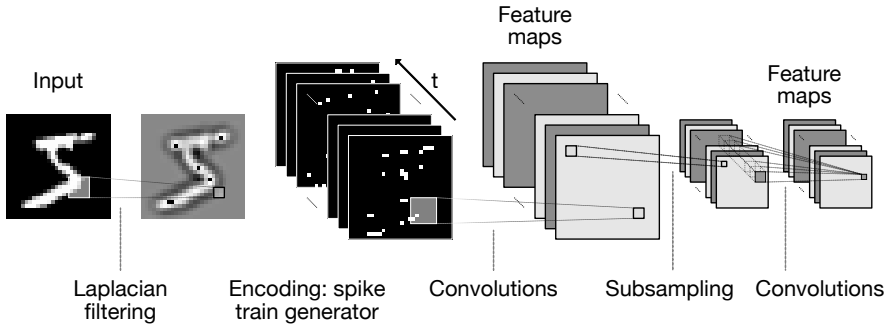


FIGURE 2.6 – Architecture adaptée du modèle proposé par Kheradpisheh *et al.* [19].

```
import torch
import torch_snn.nn as nn
import torch_snn.nn.functional as F

from torch_snn import datasets, transforms, Trainer


train_loader, test_loader = datasets.MNIST("data/",
    transform=transforms.ToTensor())

class Transform(nn.Module):
    def __init__(self):
        super().__init__()
        self.dog = transforms.DoG(7, sigma1=1, sigma2=2)
        self.roc = transform.RankOrderCoding(15)
    def forward(self, input):
        x = self.dog(input)
        x = self.roc(x)
        return x

class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.if1 = nn.Conv2dLIF(1, 15, kernel_size=5,
            threshold=10, decay=0)
        self.if2 = nn.Conv2dLIF(15, 60, kernel_size=8,
            threshold=20, decay=0)
        self.reset()
    def forward(self, input):
        x, self.potential1 = self.if1(input, self.potential1)
        x = F.max_pooling2d(x, 2)
        x, self.potential2 = self.if2(x, self.potential2)
        return x
    def reset(self):
        self.potential1 = torch.zeros(24, 24, 15)
        self.potential2 = torch.zeros(5, 5, 60)

model = nn.Sequential(Transform(), Net())

trainer = Trainer(model, optimizer='STDP',
    optimizer_params={"a_plus": 0.004, "a_minus": -0.003})
trainer.fit(train_loader)
```

Code source 4 – Implémentation du modèle de Kheradpisheh *et al.* [19] avec PyTorch SNN.

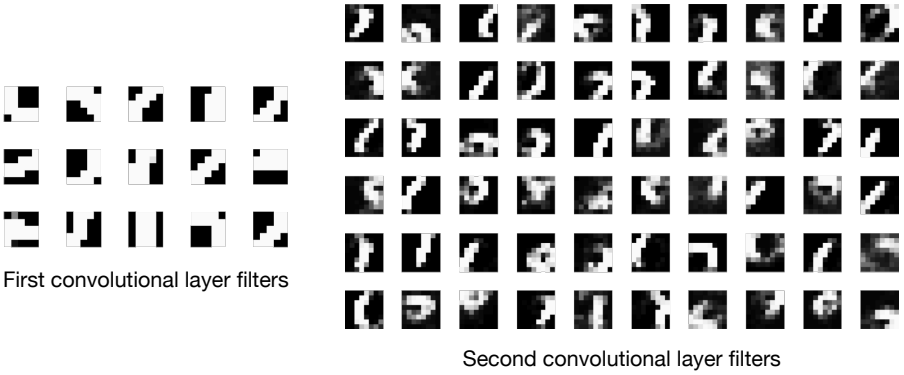


FIGURE 2.7 – Visualisation des filtres de convolutions extraits après apprentissage du modèle de Kheradpisheh *et al.* [19].

Bilan Les implémentations présentées ici sont des exemples-type de ML. Dans les deux implémentations, le code est relativement simple à mettre en place. Une ancienne version de la bibliothèque a fait l'objet d'un test sur un panel de 4 personnes : 2 ingénieurs de développement web et 2 doctorants avec un profil IA. Cette évaluation m'a permis de trouver les points faibles à corriger pour la rendre plus accessible à des novices du domaine de la neuroscience. Le test consistait à analyser des fichiers de code et relever les incompréhensions après une initiation au connexionnisme impulsionnel. La principale remarque de tous les participants concernait la boucle d'apprentissage, en particulier la mise à jour des poids qui était peu intuitive (par exemple, utiliser une pseudo-fonction de perte pour ensuite employer la méthode `.backward()`). Ce problème a été corrigé en abandonnant l'utilisation du gradient proposé par PyTorch mais aussi en proposant des outils pour cacher cette partie via le module `Trainer`.

À la fin de l'apprentissage d'un modèle, nous pouvons extraire les nouvelles représentations de nos données pour les exploiter pour une autre tâche comme la classification, la segmentation, etc.

2.3 Évaluations

Dans cette section, différentes expérimentations sont présentées pour évaluer les capacités de ma bibliothèque à traiter des données de type image et son en se fondant sur des méthodes de la littérature.

2.3.1 Sur des données de type image

Pour la partie image, nous utilisons 4 modèles issus de la littérature qui utilise des règles d'apprentissage basées sur la plasticité synaptique – STDP – permettant une approche d'apprentissage non supervisé avec des SNNs. Seul un des modèles exploite une forme de supervision durant l'apprentissage appelée la R-STDP que je détaillerai dans la suite. Le modèle de Sanders *et al.* [37] est une extension du modèle de Diehl & Cook [12] en y ajoutant des filtres de convolution.

TABLE 2.2 – Modèles de référence choisis pour évaluer notre bibliothèque sur des données de type image.

	Model	Architecture	Learning method	Coding scheme
Supervised	Mozafari *et al.* [30]	Three-layer SCNN	Hebbian/anti-Hebbian STDP	Latency coding
Unsupervised	Diehl & Cook [12]	Two-layer SNN	STDP	Rate coding
	Kheradpisheh *et al.* [19]	Three-layer SCNN	STDP	Latency coding
	Sanders *et al.* [37]	Two-layer SCNN	STDP	Rate coding

Le tableau 2.2 présente les spécifications des modèles originaux. Pour la reproduction de ces modèles avec notre simulateur, nous avons respecté autant que possible les paramètres fournis par leurs auteurs.

Corpus De même que dans les travaux principaux en reconnaissance d'images avec un SNN présentés dans le tableau 2.2, nous utilisons le jeu de données MNIST (Modified National Institute of Standards and Technology) [20] composé d'un ensemble de données de chiffres manuscrits réels, collectés auprès d'élèves et d'employés du gouvernement des États-Unis. Il comprend $60k$ images d'entraînement et $10k$ images de test, toutes en niveaux de gris et de taille 28×28 pixels. Chaque image est associée à une étiquette numérique allant de 0 à 9, correspondant au chiffre représenté sur l'image. Ce jeu de données est fréquemment utilisé en apprentissage automatique comme base pour tester de nouveaux algorithmes, car il est relativement simple à utiliser et à comprendre, mais néanmoins suffisamment complexe pour mettre en évidence les limites des modèles en apprentissage automatique.

Prétraitement Deux méthodes d'encodage des données ont été utilisées pour convertir le signal d'entrée correspondant à des images statiques du jeu de données MNIST en évènement impulsionnel discret dans le domaine temporel : le codage fréquentiel – *rate coding* – et le codage temporel – *latency coding*. Nous avons respecté les indications des auteurs pour chaque modèle.

Globalement, la technique utilisée dans ces expérimentations pour le codage temporel consiste en un filtrage avec une différence de gaussiennes (DoG) sur l'image en niveau de gris duquel un codage par rang (ROC) est ensuite appliqué [19, 30].

Concernant le codage fréquentiel – *rate coding* –, la technique utilisée suit la procédure définie par O'Connor *et al.* [31] qui consiste à générer un train impulsionnel d'une certaine durée de temps selon une distribution de Poisson avec une fréquence d'impulsions proportionnelle à la valeur de l'intensité du pixel [12, 37].

La principale différence entre ces deux types de méthodes d'encodage concerne la quantité d'informations temporelles générées. Avec le codage temporel, nous aurons au

plus une seule impulsion émise par neurone d'entrée à un instant t, tandis qu'avec le codage fréquentiel, chaque neurone d'entrée génère de nombreuses impulsions. Par le terme « neurone d'entrée », nous entendons ici « pixel de l'image ». Par conséquent, la quantité d'informations temporelles que le modèle ingère dans le cas d'un codage fréquentiel est largement plus importante que dans le cas d'un codage temporel (la séquence d'impulsions étant plus longue dans le codage fréquentiel).

Architecture Tous les modèles du tableau 2.2 utilisent deux topologies distinctes que j'ai présentées dans la section précédente (figures 2.3 et 2.6) avec quelques variantes entre-elles.

Le modèle de Diehl & Cook [12] est composé de 2 couches : une couche excitatrice de neurones LIF et une couche inhibitrice de neurones LIF. Tous les neurones d'entrée, correspondant à chaque pixel de l'image, sont totalement connectés à tous les neurones de la couche excitatrice. Chaque neurone de la couche excitatrice est connecté respectivement à un neurone de la couche inhibitrice, lui-même connecté à tous les autres neurones de la couche excitatrice. Lors de l'apprentissage, le modèle cherche à optimiser les connexions synaptiques entre la couche d'entrée et la première couche excitatrice. Cette topologie permet de réguler le potentiel d'action des neurones afin que seul un neurone se spécialise sur un motif d'entrée en renforçant ses poids. Sanders *et al.* [37] proposent une extension de ce modèle en remplaçant la couche dense en entrée par une couche convolutive. Le principe reste le même, le champ réceptif – *receptive field* – de cette convolution ainsi obtenue correspond à la couche excitatrice à laquelle est associée une couche inhibitrice connectée aux autres champs réceptifs issus de la convolution des données d'entrée avec d'autres noyaux – *kernel*.

Le modèle de Kheradpisheh *et al.* [19] est, à ma connaissance, le premier réseau convolutif profond utilisant des neurones impulsionnels. Il est constitué de 3 couches convolutives de neurones intègre-et-tire (IF) (sans fuite) [4] avec une couche de regroupement – *pooling* – interposée entre elles utilisant l'opérateur « max ». Une couche de *global max-pooling* finale est utilisée pour l'étape de classification afin de créer un vecteur de caractéristiques présentant les cartes de caractéristiques – *feature maps* – les plus importantes. Mozafari *et al.* [30] proposent une approche de classification des images différentes sur ce même modèle. Au lieu d'utiliser un classifieur sur la couche de *global max-pooling*, les auteurs ont proposé d'appliquer un algorithme de renforcement appelé R-STDP.

Règle d'apprentissage Deux règles d'apprentissages sont utilisées ici : la STDP et, dans le cas du modèle de Mozafari *et al.* [30], la R-STDP.

Concernant la STDP, l'implémentation utilisée pour ces expérimentations est une forme simplifiée [25] :

$$\Delta w_{ij} = \begin{cases} \alpha_+ \, w_{ij}(1 - w_{ij}) & \text{if } t_j - t_i \leq 0 \\ \alpha_- \, w_{ij}(1 - w_{ij}) & \text{if } t_j - t_i > 0 \end{cases} \tag{2.2}$$

où w représente le poids de la connexion synaptique, l'indice i correspond au neurone présynaptique et j au neurone postsynaptique, t est la donnée temporelle de la génération d'une impulsion par un neurone, et α_+ et α_- spécifient le taux d'apprentissage – *learning rate* .

4. Correspondant à un intégrateur pur.

Au sujet de la R-STDP, cette technique consiste à appliquer un usage particulier de la STDP sur la dernière couche de neurones du réseau de telle sorte que la modification des poids synaptiques maximise la performance globale du réseau. Pour cela, la méthode utilise une mise à jour des poids basée sur les corrélations entre les entrées et les sorties du réseau. La première étape associe un ou des neurones à un label spécifique de manière pré-déterminée ou via une heuristique durant l'apprentissage (association d'un label au neurone qui a généré la première impulsion de sortie, ou qui émet le plus d'impulsions sur une durée donnée). Dans une seconde étape, si la classification de la donnée d'entrée par les neurones de la couche de sortie du réseau est correcte, une récompense est générée et se modélise par une règle de STDP sur les connexions synaptiques du neurone post-synaptique associé à cette réussite, tandis qu'une pénalité lors d'une mauvaise classification se modélise par une règle d'anti-STDP. Cette méthode peut être assimilée à un système d'apprentissage par renforcement. Sa formulation ajoute un terme de pénalité comparé à l'équation 2.2 :

$$\Delta w_{ij} = \begin{cases} \alpha\alpha_+ + \beta\alpha_- \, w_{ij}(1 - w_{ij}) & \text{if } t_j - t_i <= 0 \\ \alpha\alpha_- + \beta\alpha_+ \, w_{ij}(1 - w_{ij}) & \text{if } t_j - t_i > 0 \end{cases} \tag{2.3}$$

où α et β sont deux paramètres contrôlant le signal de renforcement. Si le signal correspond à une récompense, la STDP classique est activée par $\alpha = 1$ et $\beta = 0$, tandis que si le signal correspondant à une punition, alors $\alpha = 0$ et $\beta = 1$ activant ainsi une anti-STDP.

Résultats Pour le calcul de la justesse – *accuracy* – des différents modèles, 2 types de décisions sont mis en œuvre : une décision basée sur le résultat d'un classifieur linéaire, généralement un modèle de machine à vecteurs de support (SVM), ou une décision basée sur la sortie des neurones de la dernière couche du modèle. Dans ce dernier cas, le décodage dépend aussi de l'encodage utilisé. Lorsque le modèle utilise un système de codage temporel, la décision est inférée grâce au neurone de sortie envoyant le premier une impulsion, tandis que dans le cas d'un système de codage fréquentiel, la décision correspond au neurone ayant réalisé le plus grand nombre d'impulsions.

Le tableau 2.3 illustre les scores obtenus avec nos implémentations des modèles sur notre simulateur PyTorch SNN, en comparaison avec les scores des modèles originaux présentés dans les papiers de chaque auteur.

Les performances sont généralement similaires, la légère fluctuation entre les résultats des différents modèles est due à l'implémentation algorithmique de chaque module d'un SNN. Il est intéressant de noter que durant ces expériences, le temps d'apprentissage était extrêmement long dû à l'utilisation d'une taille de lot – *batch* – à 1, surtout pour les architectures multicouches « Three-layer SCNN » où chaque couche est entrainée l'une après l'autre. Même si le ROC est très efficace pour encoder nos données dans un contexte de traitement rapide, il n'en demeure pas moins que les modèles utilisant cet encodage sont plus complexes que celui de Diehl & Cook.

TABLE 2.3 – Comparaison des résultats obtenus sur différents modèles de classification d'images de chiffres manuscrits.

Model	Decision	Original score	TorchSNN score
Diehl & Cook - 100 neurons [12]	Neuron-based	**82.9**	82.1
Diehl & Cook - 400 neurons [12]	Neuron-based	87	**88.9**
Diehl & Cook - 1600 neurons [12]	Neuron-based	91.9	**92.5**
Diehl & Cook - 6400 neurons [12]	Neuron-based	95	**95.1**
Kheradpisheh *et al.* [19]	SVM	**98.4**	**98.4**
Mozafari *et al.* [30]	Neuron-based	**97.2**	96.1
Sanders *et al.* [37]	Neuron-based	**68.24**	67.5

2.3.2 Sur des données de type audio

Dans cette partie, nous proposons de tester le comportement de notre bibliothèque sur des modèles de la littérature dédiés à la reconnaissance de données audio via des modèles de SNNs. Nous avons sélectionné 4 modèles qui utilisent la règle d'apprentissage STDP dont 1, le modèle de Diehl & Cook [12], adapté au traitement de l'audio en entrée.

Le tableau 2.4 présente les spécifications des modèles originaux. De même que précédemment, pour la reproduction de ces modèles avec notre simulateur, nous avons respecté autant que possible les indications des auteurs.

TABLE 2.4 – Modèles de référence choisis pour évaluer notre bibliothèque sur des données de type audio.

	Model	Architecture	Learning method	Dataset	Coding scheme
Supervised	Tavanaei *et al.* [41]	Single layer SNN	Hebbian/anti-Hebbian STDP	Aurora-2	5 frequency bands + Izhikevich neurons
Unsupervised	Dong *et al.* [13]	SCNN	STDP	TIDigits	MFCC + Temporal coding
	Diehl & Cook [12]	Two-layer SNN	STDP	MNIST	Rate coding

Corpus Dans le même style que l'évaluation sur des données de type image, les modèles dédiés au traitement audio avec des SNNs utilisent un corpus composé de chiffres de 0 à 9 lus. Les expérimentations sélectionnées utilisent 3 corpus différents : TIDigits [21], Aurora-2 [33] et TI 46-Word [24].

Le jeu de données TIDigits [21] est un ensemble de données de reconnaissance vocale de chiffres et de lettres. Il contient des enregistrements audio de locuteurs in-

dividuels qui disent des chiffres et des lettres de manière claire et articulée. Les enregistrements ont été effectués dans un environnement contrôlé et sont accompagnés de transcriptions écrites pour chaque parole prononcée.

Aurora-2 [33] est une extension de TIDigits incluant une procédure pour ajouter artificiellement du bruit afin de rendre les systèmes de reconnaissance de mots isolés plus robustes au bruit réel.

Le corpus TI 46-Word [24] a été conçu à la même époque que TIDigits et présente la même description que celui-ci.

Cependant, ces 3 corpus n'étant pas libres et ne disposant pas de licences commerciales pour les utiliser, nous avons cherché d'autres données similaires pour notre implémentation. Deux corpus ressortent de nos recherches : FSDD [24] et Speech Commands Dataset [43].

Le premier, Free Spoken Digits Dataset (FSDD) [18], couvre les dix chiffres de 0 à 9 prononcés par seulement 6 locuteurs d'une durée moyenne de moins d'une seconde.

Le second, Speech Commands Dataset [43], est un corpus plus large que le précédent incorporant un ensemble d'enregistrements étiquetés de commandes vocales simples telles que *stop*, *go*, *left*, *one*, etc. Ce jeu de données créé par Google comprend plus de $65k$ enregistrements audio provenant de plus de $30k$ utilisateurs à profil très varié. Les enregistrements audio durent environ une seconde.

Afin de tester notre bibliothèque sur les modèles sélectionnées (tableau 2.4), nous avons utilisé un sous-ensemble de ce dernier corpus correspondant aux caractéristiques des données exploitées par les auteurs : les chiffres de 0 à 9.

Prétraitement Dans le cadre d'un système de reconnaissance de mot isolé grâce à un SNN, la première étape consiste à encoder le signal audio en format impulsionnel comme le fait notre système auditif afin d'extraire une nouvelle représentation du signal brut. Le système auditif humain est composé de plusieurs parties dont l'unité principale est la cochlée dans lequel l'énergie acoustique va être transcrit en signaux électriques grâce à des cellules sensorielles sensibles à des fréquences spécifiques. Cette transformation peut être modélisée par un ensemble de filtres passe-bande.

Le modèle de Lyon [22] est un exemple connu de modèle cochléen imitant le plus précisément possible cette transcription et est très souvent utilisé dans les papiers traitant la classification de fichiers audio avec un modèle SNN [42]. Cette transduction est modélisée à l'aide de filtres du second ordre afin d'obtenir en sortie un cochléogramme, c'est-à-dire une représentation temps/fréquence.

D'un autre côté, la communauté traitement du signal utilise conventionnellement les MFCCs [10] présentant de manière simplifiée le même intérêt et qui ont montré de très bonnes performances sur les tâches de reconnaissance de la parole depuis plusieurs décennies. Le but principal des MFCCs est de capturer les caractéristiques acoustiques les plus importantes de la parole tout en éliminant les informations non pertinentes et permettre ainsi une réduction de la dimensionnalité des données. Pour cela, le signal audio en entrée est transformé en cosinus discrète (DCT) d'un spectre de puissance logarithmique sur une échelle de fréquence Mel non linéaire.

C'est cette méthode que nous allons utiliser dans la suite avec une modélisation en 20 bandes de fréquences utilisant les paramètres de génération par défaut défini dans la bibliothèque librosa [27].

Afin de transformer l'audio en train d'impulsions, Tavanaei *et al.* [41] propose d'associer à chaque valeur du vecteur caractéristique extrait, un neurone modélisé par Iz-

hikevich. Le courant d'entrée de ce neurone correspond donc à la valeur d'une bande de fréquence d'un vecteur caractéristique à un instant t. Le modèle de Dong *et al.* [13] propose de générer un codage temporel, en particulier un codage temps avant la première impulsion (TTFS), à chaque pas de temps sur chaque bande de fréquences. Dans le contexte de l'adaptation d'un modèle proposé par Diehl & Cook [12] à des données audio, la méthode choisie est similaire à celle utilisée pour les données d'images, à ceci près qu'après la transformation de l'audio en bandes de fréquences, un codage fréquentiel (utilisant un processus de Poisson) est relié à chaque valeur de cette représentation, en vue de produire un train d'impulsions proportionnel à cette même valeur.

Architecture Les expérimentations sur l'audio couvre 2 types de topologies différentes : une topologie de couche entièrement connectée (FC), et une avec des couches convolutives.

Le modèle supervisé de Tavanaei *et al.* [41] est un modèle dense où tous les neurones d'entrée représentant chaque bande de fréquences de chaque vecteur caractéristique du signal est totalement connecté aux neurones de sorties. Dans la même veine, notre adaptation du modèle de Diehl & Cook [12] est identique à celui présenté dans la partie sur l'image à l'exception des neurones d'entrée qui ne représentent plus les pixels de l'image mais toutes les bandes de fréquences de tous les vecteurs caractéristiques.

Dans le modèle de Dong *et al.* [13], les auteurs utilisent des couches convolutives dont la taille du noyau – *kernel* – correspond aux nombres de bandes de fréquences extraites. Contrairement aux méthodes convolutives que l'on trouve en apprentissage profond, la couche convolutive est ici subdivisée en plusieurs sous-couches convolutives. Plus précisément, l'ensemble des noyaux d'une des sous-couches vont seulement parcourir une partie du signal, par exemple, le début du signal audio sur 1 seconde. La 2ème seconde du signal audio sera parcourue par d'autres noyaux et ainsi de suite.

Règle d'apprentissage Nous utilisons les mêmes règles STDP, et R-STDP dans le cas d'un apprentissage supervisé, que celles appliquées précédemment sur les données images.

Résultats La représentation finale de chaque donnée d'entrée est extraite de plusieurs manières différentes. Dans le modèle de Tavanaei *et al.* [41], on récupère la signature finale du modèle après simulation sur une donnée d'entrée. Pour le modèle de Diehl & Cook [12], nous procédons de la même manière que pour l'image via un comptage d'impulsions de sorties. Dans le cas du modèle de Dong *et al.* [13], le seuil des neurones IF est supprimé de telle sorte que la signature correspond à la valeur des potentiels de membranes de la dernière couche.

Le tableau 2.5 illustre la performance de notre bibliothèque dans nos implémentations de modèles issus de la littérature. Contrairement aux données images, notre simulateur montre une justesse – *accuracy* – en baisse de plusieurs points par rapport aux scores originaux des implémentations officielles. Deux facteurs sont en cause, l'utilisation d'un jeu de données différent et une méthode différente d'extraction de caractéristiques depuis le signal brut.

TABLE 2.5 – Comparaison des résultats obtenus sur différents modèles de classification de sons isolés.

Model	Decision	Original score	TorchSNN score
Dong *et al.* [13]	SVM	**97.5**	92
Tavanaei *et al.* [41]	SVM	91	81.1
Diehl & Cook - 100 neurons [12]		-	81.2
Diehl & Cook - 400 neurons [12]		-	85

2.4 Performances

Dans cette section, nous cherchons à discuter des performances computationnelles de PyTorch SNN. La première expérience concerne l'utilisation du calcul par lot – *batch* – dans un SNN utilisant la STDP. La seconde expérience vise à comparer notre bibliothèque aux autres sur une simulation d'un modèle.

Les expérimentations ont été effectuées sur 2 machines différentes : une machine utilisée pour ses capacités de calcul graphique (machine GPU), et une autre pour son CPU tournant à plus haute fréquence que la première (machine CPU). Les caractéristiques sont les suivantes :

- [Machine GPU] Ubuntu 18.04 LTS – Intel Xeon CPU E5-2620 v4 @ 2.10GHz x32 – 64 Go RAM – Nvidia GeForce GTX 1080 Ti ;
- [Machine CPU] macOS 11 – Intel Core i7 2.6GHz x8 – 16 Go RAM

Dans les expériences ci-après le modèle de Diehl & Cook utilisé est celui composé de 100 neurones.

2.4.1 Calcul par lots

Par l'utilisation de PyTorch et notre modélisation sous un format de cellules récurrentes, PyTorch SNN présente des capacités de calculs parallèles.

Définition Le calcul par lots – *batch processing* – est un moyen de traiter les données en groupe plutôt que de les traiter de manière individuelle. En ML, lorsque les données sont traitées par lots, les mises à jour des paramètres du modèle sont effectuées sur un échantillon de données. Cela peut présenter de nombreux avantages tels que :

1. **La réduction du temps de calcul :** le calcul par lots permet une utilisation plus efficace des ressources computationnelles en effectuant plusieurs opérations simultanément. Par exemple, dans le domaine de l'apprentissage machine, il est courant d'utiliser la capacité d'un GPU an matière de calcul parallèle.
2. **La stabilité du modèle :** lorsque les données sont traitées par lots, les mises à jour des paramètres du modèle sont effectuées sur un échantillon de données plutôt que sur un seul exemple. Cela permet de stabiliser les paramètres du modèle en tenant compte de la variabilité des données sur un grand nombre d'exemples.
3. **La régularisation des données :** la régularisation est un moyen d'éviter le surapprentissage – *overfitting* –, qui se produit lorsqu'un modèle s'adapte trop bien

aux données d'entraînement, ce qui peut conduire à une performance médiocre sur les données de test. Lorsque les données sont traitées par lots, l'utilisation d'algorithmes de régularisation tels que la régularisation $\mathcal{L}_1$ ou $\mathcal{L}_2$ peuvent aider à limiter l'adaptation du modèle aux données d'entraînement et à améliorer la performance sur les données de test. En somme, le calcul par lots peut être utilisé en conjonction avec des algorithmes de régularisation pour répondre à cette problématique.

L'utilisation de ce type de méthode n'est pas courant dans l'apprentissage de SNNs utilisant la plasticité synaptique. Cela peut s'expliquer par la non-plausibilité biologique d'un tel calcul.

Stratégie de réduction Lorsque nous alimentons notre réseau de données avec une certaine taille de lots, les états cachés tels que le potentiel de membrane du neurone sont dupliqués à l'initialisation afin qu'ils évoluent indépendamment à travers le temps. Lors du processus d'optimisation, avec une règle de plasticité synaptique dans notre cas, une règle de réduction des données doit être appliquée permettant de combiner toutes les mises à jour obtenues pour chaque donnée dans le lot – *batch* –, avant de mettre à jour les paramètres du modèle, ici les connexions synaptiques. De manière analogue, en apprentissage profond, lorsque tous les gradients du modèle sont calculés, l'optimiseur applique généralement une réduction des données basées sur la moyenne. En considérant cela, nous avons décidé de tester 3 méthodes de réduction naïves : la moyenne, le maximum et la somme.

La figure 2.8 présente le résultat de ces 3 stratégies de réduction tout au long de l'apprentissage. Le modèle utilisé est celui de Diehl & Cook [12] présenté précédemment sur le corpus de données MNIST. Pour cette expérience, nous fixons une taille de lots – *batch size* – à 32 (l'influence de ce paramètre sur les résultats sera étudié ensuite). Étonnamment, le choix d'une réduction basé sur la moyenne n'est absolument pas adapté dans notre situation, contrairement à la somme ou au maximum.

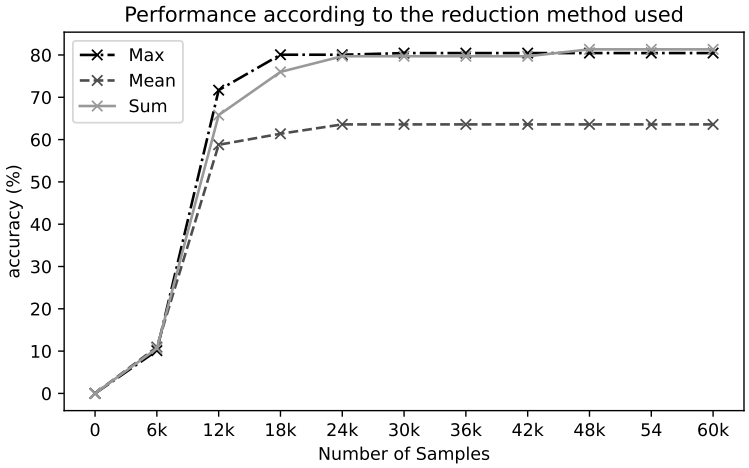


FIGURE 2.8 – Performance du modèle durant l'apprentissage.

Comportement des résultats Cependant, est-ce que l'utilisation d'un calcul par lot est stable dans un environnement comme le nôtre ? Pour répondre à cela,

nous avons entrainé le modèle sur différentes tailles de lots – *batch size* – $\in \{1, 8, 16, 32, 64, 128, 256, 512\}$ en utilisant la réduction des données basé sur la somme. Nous avons fait le choix de la somme au détriment du maximum, car elle a obtenu de meilleurs résultats sur les données de tests. Différentes mesures d'utilisation des ressources computationnelles sont évaluées dans le tableau 2.6. Nous pouvons constater que dès que la taille du lot est très élevé, les résultats chutent considérablement.

TABLE 2.6 – Performance de la réduction par somme en fonction de la taille du batch.

		Batch size							
	Measures	1	8	16	32	64	128	256	512
CPU	Accuracy	0.82	0.78	0.78	0.77	0.76	0.75	0.17	0.08
	Time	02:02:00	23:34	14:59	14:06	20:16	19:24	19:33	29:12
	RAM	0.5	0.51	0.52	0.55	0.55	0.56	0.62	0.69
GPU	Accuracy	0.82	0.79	0.77	0.79	0.79	0.78	0.29	0.20
	Time	02:54:45	28:34	17:10	09:15	09:03	07:36	06:54	06:43
	RAM	1.3	1.3	1.3	1.3	1.3	1.3	1.3	1.3
	VRAM	0.94	0.97	0.98	1.01	1.03	1.10	1.21	1.72

Bilan Cette étude montre que l'utilisation du calcul par lot est possible avec une stratégie de réduction basée sur le maximum et la somme. Cela nous permet de réduire significativement le temps de calcul au détriment d'une légère perte de justesse – *accuracy* . Le bon compromis serait d'utiliser une taille de lot – *batch size* – égale à 32 ou 64. Ces résultats entrent donc en adéquation avec l'étude de Masters [26] qui avait avancé que l'utilisation de lots de petites tailles – *mini-batch* – permet d'obtenir une meilleure stabilité durant l'apprentissage ainsi qu'une meilleure performance du modèle. Cependant, les résultats montrent que l'utilité d'instancier un modèle sur un GPU reste discutable. Le temps de calcul se gagne au détriment de la performance. Il serait intéressant d'approfondir le travail sur un autre modèle comme celui de Kheradpisheh *et al.* [19] où les premières expérimentations ont montré une baisse encore plus significative des résultats.

Il est intéressant de noter que les auteurs de Bindsnet [38] ont proposé un papier plus complet sur l'utilisation des *mini-batchs* dans le même contexte que nous. Leur étude a été diffusée en septembre 2019 tandis que la nôtre a été réalisée en juin de l'année 2019 lors d'un séjour scientifique au NII à Tokyo.

2.4.2 Comparaison avec d'autres simulateurs

Dans cette partie, nous comparons PyTorch SNN à d'autres simulateurs, en particulier à Brian [40] qui est le plus connu dans le domaine, et Bindsnet [16] développé au même moment que le nôtre et qui se positionne plus ou moins sur la même problématique.

Nous proposons 2 comparaisons : (1) une comparaison du temps de calcul et des ressources mémoire utilisées par l'implémentation du modèle de Diehl & Cook [12] sur 3 simulateurs ; (2) une comparaison du temps de calcul en fonction du nombre de neurones d'un modèle en mode inférence.

Sur l'apprentissage du modèle Diehl & Cook Nous reprenons le modèle de Diehl&Cook déjà présenté précédemment. Nous utilisons l'implémentation officielle des auteurs du modèle [12] développé sur Brian [40], l'implémentation proposée dans les exemples de code de Bindsnet [16] ainsi que la nôtre.

Le tableau 2.7 montre que PyTorch SNN offre une performance très raisonnable grâce à l'apprentissage par lot – *batch* .

TABLE 2.7 – Comparaison des performances d'un modèle implémenté sur 3 simulateurs différents dont le nôtre.

		CPU		GPU		
	Batch size	Time	RAM	Time	RAM	VRAM
Brian [40]	1	06:39:15	2.8 Gb	–	–	–
Bindsnet [16]	1	02:41:00	0.61	03:34:41	1.52	0.83
Bindsnet [16]	64	00:31:01	0.65	00:17:43	1.52	0.89
Ours	1	02:02:00	0.5	02:54:45	1.52	0.94
Ours	64	00:20:16	0.55	00:09:32	1.52	0.97

Sur l'inférence des données Nous avons aussi évalué la bibliothèque sur un usage plus élémentaire qui consiste à simuler un réseau artificiel sans mécanisme de plasticité synaptique. Cette simulation suit le même protocole proposé par les auteurs des bibliothèques Bindsnet [16] et Norse [35]. Le modèle est composé d'une couche d'entrée entièrement connectée à une couche de neurones LIF. La taille de la population de neurones d'entrée et de sortie varient à chaque exécution. L'encodage utilisé pour la simulation est un codage fréquentiel basé sur la loi de Poisson. Chaque simulation est réalisée sur 1000 unités de temps.

Les résultats obtenus montrent simplement que plus la population de neurones augmentent, plus PyTorch SNN obtient de meilleures performances en termes de temps de simulation grâce à l'utilisation du GPU. Plus précisément, pour une population de 10000 neurones, nous observons une multiplication d'un facteur 10 entre une simulation sur GPU et sur CPU. Sur le plan d'une comparaison avec les autres simulateurs, nous obtenons des résultats équivalents à ceux obtenus par Norse [35] et Bindsnet [16] aussi bien sur CPU que sur GPU. Cela s'explique par l'utilisation de la même bibliothèque en arrière-plan : PyTorch.

Bilan PyTorch SNN offre des performances comparables à d'autres simulateurs à travers les deux expérimentations réalisées.

2.5 Conclusion

Dans ce chapitre, nous avons détaillé l'implémentation d'un nouveau simulateur nommé « PyTorch SNN » basé sur la bibliothèque tensorielle « PyTorch ». Cette bibliothèque présente l'intérêt d'être facile à prendre en main par la communauté scientifique travaillant sur l'apprentissage profond comme nous l'avons montré à travers des exemples de code. Elle complète la liste des simulateurs existants en proposant une approche différente. Nous avons montré que ce simulateur avait des performances honorables dans différentes tâches de reconnaissance d'images et de sons malgré des temps de simulation extrêmement longs en comparaison avec d'autres méthodes de ML. Pour remédier à ce problème, nous avons testé l'utilisation du calcul par lots – *batch* – rendu possible par notre interface similaire à PyTorch. Les résultats montrent qu'il peut être intéressant d'utiliser des tailles de lot de 32 ou 64 permettant de garantir un score tout a fait convenable au vu du temps de calcul gagné.

Ces travaux ont servi de support à deux collaborations distinctes. La bibliothèque était l'objet d'un livrable dans le cadre d'un projet partenarial avec le Ministère de l'Intérieur, incluant des clauses de confidentialité. La seconde collaboration, plus brève, portait sur le traitement de données audio avec un SNN avec une équipe de recherche du NII – *National Institute of Informatics* – à Tokyo. Un poster scientifique portant sur les résultats expérimentaux a été présenté à une journée d'étude.

Pour aller plus loin, durant les différentes expériences, nous avons remarqué que les hyperparamètres d'un modèle SNN étaient ultra-sensibles. Par exemple, un mauvais choix de α_- et α_+ dans la règle STDP affecte grandement la convergence du modèle. Ajouté à cela, l'entraînement des architectures complexes utilisant des règles de plasticité synaptique reste assez compliqué. Aussi, nous proposons dans la suite, une nouvelle façon d'extraire des représentations de manière non supervisée ainsi qu'une manière de trouver des heuristiques sur des hyperparamètres du modèle.

Références

[1] Martín Abadi et al. *TensorFlow : Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015.

[2] Filipp Akopyan et al. “TrueNorth : Design and Tool Flow of a 65 mW 1 Million Neuron Programmable Neurosynaptic Chip”. In : *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 34.10 (2015), pages 1537-1557.

[3] Trevor Bekolay et al. “Nengo : a Python tool for building large-scale functional brain models”. In : *Frontiers in Neuroinformatics* 7.48 (2014), pages 1-13.

[4] Pierre Boulet et al. *N2S3, an Open-Source Scalable Spiking Neuromorphic Hardware Simulator*. Research Report. Université de Lille 1, Sciences et Technologies ; CRIStAL UMR 9189, jan. 2017.

[5] James Bradbury et al. *JAX : composable transformations of Python+NumPy programs*. Version 0.3.13. 2018.

[6] Romain Brette et al. “Simulation of networks of spiking neurons : a review of tools and strategies”. In : *J. Comput. Neurosci.* 23.3 (déc. 2007), pages 349-398.

[7] Nicholas T. Carnevale et Michael L. Hines. *The NEURON Book*. Cambridge University Press, 2006.

[8] Tianqi Chen et al. “MXNet : A Flexible and Efficient Machine Learning Library for Heterogeneous Distributed Systems.” In : *CoRR* abs/1512.01274 (2015).

[9] Mike Davies et al. “Loihi : A Neuromorphic Manycore Processor with On-Chip Learning”. In : *IEEE Micro* 38.1 (2018), pages 82-99.

[10] S. Davis et P. Mermelstein. “Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences”. In : *IEEE Transactions on Acoustics, Speech, and Signal Processing* 28.4 (1980), pages 357-366.

[11] Andrew Davison et al. “PyNN : a common interface for neuronal network simulators”. In : *Frontiers in Neuroinformatics* 2 (2009).

[12] Peter U. Diehl et Matthew Cook. “Unsupervised learning of digit recognition using spike-timing-dependent plasticity”. In : *Frontiers in Computational Neuroscience* 9 (2015).

[13] Meng Dong, Xuhui Huang et Bo Xu. “Unsupervised speech recognition through spike-timing-dependent plasticity in a convolutional spiking neural network”. In : *PLOS ONE* 13 (nov. 2018), pages 1-19.

[14] Wei Fang et al. *SpikingJelly*. 2020.

[15] Paul Ferré, Franck Mamalet et Simon J Thorpe. “Unsupervised feature learning with Winner-Takes-All based STDP”. In : *Front. Comput. Neurosci.* 12 (avr. 2018), page 24.

[16] Hananel Hazan et al. “BindsNET : A Machine Learning-Oriented Spiking Neural Networks Library in Python”. In : *Frontiers in Neuroinformatics* 12 (2018), page 89.

[17] Taras Iakymchuk et al. “Simplified spiking neural network architecture and STDP learning algorithm applied to image classification”. In : *EURASIP Journal on Image and Video Processing* 2015.1 (2015), pages 1-11.

[18] Zohar Jackson et al. *Free Spoken Digit Dataset (FSDD)*. Version v1.0.8. Août 2018.

[19] Saeed Reza Kheradpisheh et al. "STDP-based spiking deep convolutional neural networks for object recognition". In : *Neural Networks* 99 (2018), pages 56-67.

[20] Y. Lecun et al. "Gradient-based learning applied to document recognition". In : *Proceedings of the IEEE* 86.11 (1998), pages 2278-2324.

[21] R. Gary Leonard et George R. Doddington. *TIDIGITS*. 1993.

[22] R. Lyon. "A computational model of filtering, detection, and compression in the cochlea". In : *ICASSP '82. IEEE International Conference on Acoustics, Speech, and Signal Processing*. Tome 7. 1982, pages 1282-1285.

[23] Wolfgang Maass. "Networks of spiking neurons : The third generation of neural network models". In : *Neural Networks* 10.9 (1997), pages 1659-1671.

[24] Ken Church Ed Fox Carole Hafner Judy Klavans Mitch Marcus Bob Mercer Jan Pedersen Paul Roossin Don Walker Susan Warwick Antonio Zampolli Mark Liberman Robert Amsler. *TI 46-Word*. 1993.

[25] Timothée Masquelier et Simon J Thorpe. "Unsupervised Learning of Visual Features through Spike Timing Dependent Plasticity". In : *PLOS Computational Biology* 3.2 (fév. 2007), pages 1-11.

[26] Dominic Masters et Carlo Luschi. "Revisiting Small Batch Training for Deep Neural Networks". In : *ArXiv* abs/1804.07612 (2018).

[27] Brian McFee et al. "librosa : Audio and music signal analysis in python". In : *Proceedings of the 14th python in science conference*. Tome 8. 2015.

[28] C. Mead. "Neuromorphic electronic systems". In : *Proceedings of the IEEE* 78.10 (1990), pages 1629-1636.

[29] Milad Mozafari et al. "First-Spike-Based Visual Categorization Using Reward-Modulated STDP". In : *IEEE Transactions on Neural Networks and Learning Systems* 29.12 (2018), pages 6178-6190.

[30] Milad Mozafari et al. "Bio-inspired digit recognition using reward-modulated spike-timing-dependent plasticity in deep convolutional networks". In : *Pattern Recognition* 94 (2019), pages 87-95.

[31] Peter O'Connor et al. "Real-time classification and sensor fusion with a spiking deep belief network". In : *Frontiers in Neuroscience* 7 (oct. 2013), page 178.

[32] Adam Paszke et al. "PyTorch : An Imperative Style, High-Performance Deep Learning Library". In : *Advances in Neural Information Processing Systems*. Sous la direction de H. Wallach et al. Tome 32. Curran Associates, Inc., 2019.

[33] David Pearce et Hans-Günter Hirsch. "The aurora experimental framework for the performance evaluation of speech recognition systems under noisy conditions." In : *INTERSPEECH*. ISCA, 2000, pages 29-32.

[34] F. Pedregosa et al. "Scikit-learn : Machine Learning in Python". In : *Journal of Machine Learning Research* 12 (2011), pages 2825-2830.

[35] Christian Pehle et Jens Egholm Pedersen. *Norse - A deep learning library for spiking neural networks*. Version 0.0.7. Documentation : https ://norse.ai/docs/. Jan. 2021.

[36] Luis A. Plana et al. “SpiNNaker : Design and Implementation of a GALS Multicore System-on-Chip”. In : *J. Emerg. Technol. Comput. Syst.* 7.4 (2011).

[37] Daniel J. Saunders et al. “STDP Learning of Image Patches with Convolutional Spiking Neural Networks”. In : *2018 International Joint Conference on Neural Networks (IJCNN)* (2018), pages 1-7.

[38] Daniel J. Saunders et al. “Minibatch Processing in Spiking Neural Networks”. In : *ArXiv* abs/1909.02549 (2019).

[39] Catherine D. Schuman et al. “A Survey of Neuromorphic Computing and Neural Networks in Hardware”. In : *ArXiv* abs/1705.06963 (2017).

[40] Marcel Stimberg, Romain Brette et Dan FM Goodman. “Brian 2, an intuitive and efficient neural simulator”. In : *eLife* 8 (août 2019). Sous la direction de Frances K Skinner, e47314.

[41] Amirhossein Tavanaei et Anthony S. Maida. “A spiking network that learns to extract spike signatures from speech signals”. In : *Neurocomputing* 240 (2017), pages 191-199.

[42] D. Verstraeten, B. Schrauwen et D. Stroobandt. “Reservoir-based techniques for speech recognition”. In : *The 2006 IEEE International Joint Conference on Neural Network Proceedings*. 2006, pages 1050-1053.

[43] P. Warden. “Speech Commands : A Dataset for Limited-Vocabulary Speech Recognition”. In : *ArXiv e-prints* (avr. 2018).

[44] Omry Yadan. *Hydra - A framework for elegantly configuring complex applications*. Github. 2019.

Chapitre 3

SNN en apprentissage auto-supervisé

Le paradigme de l'apprentissage auto-supervisé (SSL) est un domaine de recherche en plein essor ces dernières années, avec des résultats prometteurs pour de la représentation des données, et notamment en vision par ordinateur. Le SSL suppose que les données d'apprentissage contiennent des informations facilement extractibles servant d'objectif de prédiction au modèle. Pour que ces modèles convergent vers la création de représentations discriminantes, une augmentation des données est appliquée aux données d'entrée qui alimentent ensuite un réseau de neurones composé de deux branches.

D'autre part, les SNNs attirent une communauté grandissante en raison de leur capacité à traiter les informations temporelles, de leur faible consommation d'énergie et de leur grande plausibilité biologique. Mais, par la difficulté d'entrainer les SNNs de manière non supervisée, nous cherchons dans cette étude une autre manière de bénéficier des avantages de l'apprentissage profond couplé à un réseau de neurones à impulsions (SNN).

Grâce à la stochasticité du processus de Poisson qui permet d'encoder une donnée dans différentes représentations temporelles, et au succès de l'utilisation du gradient de substitution – *surrogate gradient* – sur l'apprentissage, nous proposons une méthode de SSL appliquée à un SNN en travaillant sur une étude préliminaire des représentations générées. Nous avons montré sa faisabilité en entraînant une architecture que nous proposons pour le traitement d'un jeu de données de chiffres manuscrits (MNIST), puis nous avons évalué les représentations avec deux méthodes de classification.

Sommaire

3.1 Introduction

L'un des principaux arguments en faveur de la mise en œuvre des SNNs est la possibilité d'implémenter les modèles sur des cartes neuromorphiques, réduisant ainsi les coûts énergétiques de fonctionnement par rapport aux réseaux de neurones classiques tout en ayant un traitement plus rapide de l'information [22]. Cela est rendu possible par la nature du neurone impulsionnel qui ne traite l'information que lorsque cela est nécessaire. Néanmoins, la réalisation de l'apprentissage d'un réseau est une opération complexe en raison du nombre d'hyperparamètres à régler, et de la nature éparse des données circulant dans le réseau. Parmi les principales formes d'apprentissage, on peut distinguer les approches non supervisées (règle de Hebb), les approches supervisées (utilisant l'estimation du gradient), et les méthodes permettant de transformer les réseaux entraînés ANNs en SNNs [21].

Notre contribution dans ce chapitre exploite le gradient de substitution utilisé notamment dans les formes supervisées d'apprentissage SNN, mais cette fois-ci selon une approche auto-supervisée SSL. Pour surmonter le coût élevé d'augmentation des données requises par cette dernière approche, nous proposons d'utiliser la propriété d'une représentation temporelle stochastique [5, 7, 14, 23].

Comme le concept d'apprentissage auto-supervisé (SSL) est un sujet de recherche assez récent, nous examinons d'abord l'état de l'art des stratégies déployées par la communauté de l'apprentissage profond, en particulier dans le domaine de la vision par ordinateur.

3.1.1 Apprentissage auto-supervisé (SSL)

Les modèles utilisant le SSL sont capables d'apprendre des représentations discriminantes sans recourir à des annotations humaines. Des travaux récents dans le domaine

de la vision par ordinateur [3, 4, 8, 10, 26] ont montré qu'il est possible que des modèles apprennent des représentations efficaces. Les architectures proposées relèvent généralement de la même topologie, c'est-à-dire un réseau à deux branches alimenté par des images augmentées qui sont ensuite fusionnées à travers une fonction de perte modélisant la correspondance ou non de la sortie des deux branches. Les modèles ont connu différentes évolutions notables allant de l'utilisation d'une fonction de perte contrastive [1] avec la génération de paires positives et négatives [3], au modèle de *Barlow Twins* [26] utilisant une fonction de perte basée sur la corrélation croisée d'un réseau symétrique. En revanche, les modèles *BYOL* [8] et *SimSiam* [4] ont introduit la notion de réseau « prédicteur » sur l'une des deux branches, ce qui a pour conséquence de rendre le réseau asymétrique. Ils appliquent une fonction de perte contrastive sur les représentations des deux branches et optimisent le modèle sur une seule des branches.

3.1.2 Estimation du gradient

Afin d'adapter les méthodes vues précédemment sur un modèle SNN, il est nécessaire d'adapter la fonction seuil afin qu'elle ait une dérivée non nulle car cette absence de gradient interdit tout mécanisme de convergence vers un optimum. Une méthode consiste à utiliser un proxy appelé Straight-Through Estimator (STE) [1] qui consiste à remplacer la dérivée de la fonction seuil par un substitut apparenté. Cette idée a depuis été largement utilisée dans le domaine des SNNs [17, 24, 27, 28] où le gradient de la fonction *Heaviside* est remplacé par le gradient d'une fonction d'activation similaire à dérivée non nulle comme la *softmax*.

3.1.3 SNN auto-supervisé

Concernant l'implémentation d'une stratégie de SSL par le moyen d'un SNN, seuls quelques travaux existent. Des travaux tentent de prédire le flot optique à partir d'une caméra événementielle – *event-based camera* – [9, 15]. Pour cela, ils entraînent un SNN couplé ou non à un réseau de neurones artificiels (ANN) et appliquent un gradient de substitution pour utiliser les mêmes optimiseurs qu'un ANN classique.

3.1.4 SNN non supervisé

Nous avons précédemment abordé à travers les chapitres 1 et 2, et sans entrer dans les détails, la possibilité de réaliser un apprentissage non supervisé grâce à l'utilisation de la plasticité synaptique, dont la modélisation se base notamment sur la plasticité fonction du temps d'occurrence des impulsions (STDP) pour optimiser les poids des connexions synaptiques [6, 11, 18]. Diehl & Cook [6] utilisent une architecture à 2 couches avec une première couche de neurones excitateurs et une seconde couche de neurones inhibiteurs, tandis que Kheradpisheh *et al.* [11] utilisent une architecture multicouche avec des stratégies d'inhibition et d'excitation sur chaque couche afin de classer les images. Les travaux de Paredes-Vallés *et al.* [18] sont similaires aux travaux présentés précédemment en cherchant à estimer le flot optique [9, 15], mais en adaptant l'approche donnée par Kheradpisheh [11]. Le défaut de cette méthode d'apprentissage

1. Une fonction de perte contrastive a pour but de maximiser la similitude des représentations associées à des données similaires et de minimiser la similitude entre les représentations associées à des données dissimilaires.

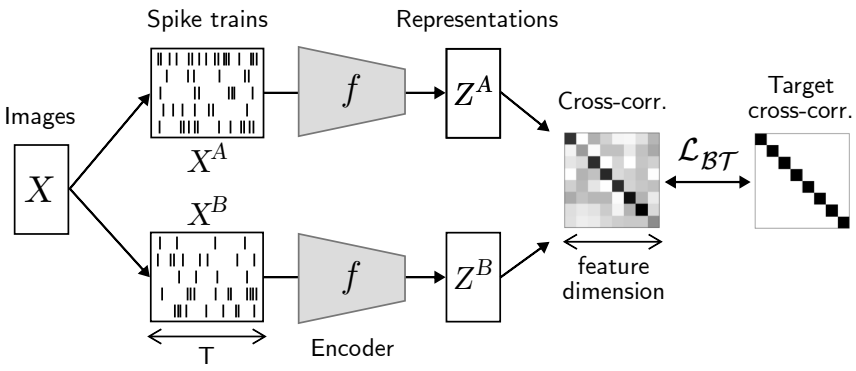


FIGURE 3.1 – Architecture proposée pour extraire des représentations d'une image.

Algorithm 1: Main learning algorithm.

Data: dataset $\boldsymbol{x}$, batch size N, structure of the model f, simulation length T
Result: encoder network $f(\cdot)$

for *sampled minibatch* $\{\boldsymbol{x}_k\}_{k=1}^{N}$ **do**
  `// encode data in temporal view`
  $\boldsymbol{x}_A \leftarrow poisson\_process\_encoder(\boldsymbol{x}_k, T)$
  $\boldsymbol{x}_B \leftarrow poisson\_process\_encoder(\boldsymbol{x}_k, T)$
  `// compute embeddings`
  $\boldsymbol{z}_A \leftarrow f(\boldsymbol{x}_A)$
  $\boldsymbol{z}_B \leftarrow f(\boldsymbol{x}_B)$
  `// apply Barlow Twins Loss`
  $\boldsymbol{c} \leftarrow cosine\_similiraty(\boldsymbol{z}_A, \boldsymbol{z}_B)$
  $loss \leftarrow \sum_i (1 - \boldsymbol{c}_{ii})^2 + \lambda \sum_i \sum_{j \neq i} \boldsymbol{c}_{ij}{}^2$
  `// apply optimization step`
  update network f to minize $loss$
end

FIGURE 3.2 – Pseudo-code de l'apprentissage du modèle proposé.

basée sur des règles hebbiennes est le grand nombre de stratégies et d'hyperparamètres à fixer.

3.2 Méthodologie

Motivés par les performances d'apprentissage des modèles SNN grâce à l'estimation du gradient – *surrogate gradient* – et par l'émergence du paradigme de SSL, nous proposons une méthode inspirée du modèle *Barlow Twins* [26] pour l'architecture générale et du modèle de Diehl & Cook [6] pour la partie du connexionnisme impulsionnel.

La chaîne de traitement globale de notre extracteur de caractéristiques est illustrée par la figure 3.1. Un pseudo-code de l'implémentation est donné par l'algorithme 1.

3.2.1 Modèle

Comme nous l'avions expliqué dans le chapitre 1, le modèle SNN peut être décomposé en 3 éléments : une stratégie d'encodage de données en format impulsionnel, une topologie du modèle et un décodeur des neurones de la couche de sortie. Nous allons les détailler dans la suite. Le modèle, désigné par f dans la figure 3.1, sera appelé « encodeur » par la suite, car il vise à générer une représentation de nos images d'entrée.

Représentation temporelle en entrée Dans le domaine du SSL, les modèles en vision par ordinateur créent généralement deux vues augmentées x_A et x_B d'une image d'entrée x grâce à une stratégie d'augmentation de données [3, 4, 8, 26]. Inspirés par ces travaux, nous proposons une adaptation de cette stratégie aux SNNs. Pour cela, l'image d'entrée x est transformée en deux vues temporelles stochastiques x_A et x_B d'une durée T grâce à un codage fréquentiel. La première partie de la figure 3.1 montre cette transformation. Pour produire le train d'impulsions de chaque pixel de l'image x, nous utilisons un processus de génération d'événements par une loi de Poisson avec un taux moyen d'émission d'impulsions compris entre 0 et $100Hz$ proportionnels à l'intensité des pixels de l'image. Ce choix est comparable à celui du modèle de Diehl & Cook [7]. L'utilisation du codage fréquentiel, en particulier en utilisant un processus de Poisson, cela nous permet d'obtenir pour une image x, une multitude de représentations possibles sous forme de trains d'impulsions qui peuvent être vues comme étant notre augmentation de données. Les deux vues temporelles x_A et x_B alimentent ensuite la fonction f : l'encodeur.

L'utilisation du codage par rang (ROC), générant une représentation temporelle plus rapide, n'est pas étudié ici en raison de sa nature déterministe. Une piste pour utiliser ce type d'encodage avec notre méthode serait d'appliquer une stratégie d'augmentation de données comme Li *et al.* le proposent dans leur étude [16] et qui consiste à réaliser le même type d'augmentations que celles mises en oeuvre en audio. Ces augmentations peuvent être du décalage temporel sur les impulsions, l'ajout de bruit sur les données d'entrée ou la représentation temporelle, ou encore l'étirement du temps – *time stretching* .

Définition du modèle Notre SNN, défini sous le terme « encodeur » ou de fonction f, est constitué de 3 couches (voir la figure 3.1). La première couche est la couche d'entrée, contenant d neurones où d est la dimension de l'image d'entrée (un neurone par pixel d'image). Les deuxième et troisième couches sont les couches de traitement, contenant n neurones excitateurs et inhibiteurs. Pour modéliser la dynamique des neurones, nous avons choisi le modèle intègre-et-tire avec fuite (LIF) plus précisément la variante basée sur le courant et introduisant une récurrence au niveau de la sortie du neurone qui reboucle sur l'entrée. La modélisation se compose des équations suivantes :

$$\frac{dv(t)}{dt} = \frac{1}{\tau_{\text{mem}}}(v_{\text{leak}} - ((1 - z(t)) \cdot v(t) + z(t) \cdot v_{reset}) + i(t)) \tag{3.1}$$

$$\frac{di(t)}{dt} = -\frac{1}{\tau_{\text{syn}}} i(t-1) + s(t) \cdot W_s + z(t) \cdot W_z \tag{3.2}$$

et la fonction d'activation est la fonction de pas *Heaviside* Θ :

$$z(t) = \Theta(v(t) - v_{\text{threshold}}) = \begin{cases} 0, & \text{si } v(t) - v_{\text{threshold}} <= 0 \\ 1, & \text{sinon} \end{cases} \tag{3.3}$$

s est l'impulsion à l'entrée du neurone et z est l'impulsion à la sortie du neurone. W_s et W_z sont les connexions synaptiques (c'est-à-dire les poids) du modèle à optimiser.

Contrairement à la couche intermédiaire, la dernière couche utilise des neurones LIF avec un seuil $v_{threshold}$ fixé à l'infini. Les neurones de cette couche intègrent uniquement les impulsions d'entrée sans générer d'impulsions de sortie. Ces neurones sont appelés intègre avec fuite (LI).

Pour éviter le problème du neurone mort causé par la fonction de pas Heaviside Θ, nous utilisons la méthode du gradient de substitution définie dans les travaux de Zenke *et al.* [27] sous le nom de modèle « superspike ». Le calcul du gradient est ainsi remplacé par la dérivée partielle de la fonction sigmoïde rapide [2] définie par :

$$\frac{d}{dv}\Theta(v - v_{\text{threshold}}) \approx \frac{1}{(\alpha|v - v_{\text{threshold}}| + 1)^2} \tag{3.4}$$

avec un coefficient α qui donne la forme de la courbe. Plus α est grand, plus la courbe va approximer une fonction Dirac qui correspond à la dérivée de notre fonction Heaviside Θ.

Notre architecture présentée par la figure 3.1 prend donc en entrée deux représentations temporelles x_A et x_B d'une image x. Les deux vues temporelles, où trains d'impulsions, sont traitées par un SNN f composé de trois couches appelé encodeur. L'encodeur f, même s'il est dupliqué sur les deux branches, partage les mêmes poids entre les deux vues. La sortie du réseau enregistre la trace du potentiel de la dernière couche à chaque instant t pour une durée totale de T.

Décodage de l'information La sortie de l'encodeur f des 2 branches produit une trace du potentiel membranaire. À partir de cette trace, nous devons appliquer une stratégie pour interpréter la sortie comme une distribution de probabilité que nous utiliserons pour appliquer une fonction de perte afin d'optimiser le modèle. Nous proposons de conserver uniquement le potentiel de membrane final y du modèle. Nous appelons la sortie de notre décodage la « représentation » de nos données. Il aurait été possible d'extraire d'autres représentations, comme le potentiel membranaire maximal, ou de compter le nombre d'impulsions si nous avions un neurone de sortie avec un seuil permettant la génération d'impulsions. Les 2 représentations extraites sont introduites dans la fonction de perte de *Barlow Twins*. À la fin de l'apprentissage, la représentation d'une seule des 2 branches suffit pour être utilisée dans une autre tâche.

3.2.2 Fonction de perte

Afin d'entrainer le modèle à générer des représentations discriminantes, nous proposons d'utiliser la fonction de perte de *Barlow Twins* $\mathcal{L}_{\mathcal{BT}}$ telle qu'elle est définie par Zbontar *et al.* [26] pour ensuite optimiser le SNN, c'est-à-dire l'encodeur f. Cette fonction de perte permet d'entraîner un modèle de manière auto-supervisée sans la génération de paires positives/négatives et sans les grands lots – *batches* – de données que l'on retrouve dans d'autres méthodes.

La fonction de perte est calculée à partir de la matrice de corrélation croisée $\mathcal{C} \in \mathbb{R}^{d \times d}$, où d est la taille de la représentation, calculée entre les sorties des deux réseaux

2. Pour rappel, la fonction sigmoïde rapide s'écrit : $\Theta(v - v_{\text{threshold}}) = (v - v_{\text{threshold}})/(1 + |v - v_{\text{threshold}}|)$

identiques le long de la dimension du batch :

$$\mathcal{C}_{ij} = \frac{\sum_b z^A_{b,i} z^B_{b,j}}{\sqrt{\sum_b \left(z^A_{b,i}\right)^2} \sqrt{\sum_b \left(z^B_{b,j}\right)^2}}, \tag{3.5}$$

où b est l'indice du lot – *batch* – et i, j sont les indices des représentations A et B.

Une fois la matrice de corrélation croisée calculée, nous rendons la représentation invariante au processus de Poisson appliqué à l'entrée du réseau SNN en maximisant la diagonale (terme d'invariance) et en décorrélant les composantes des vecteurs de représentation (terme de réduction de la redondance). L'optimisation de la fonction de perte *Barlow Twins* $\mathcal{L}_{\mathcal{BT}}$ conduit donc à rechercher une matrice de corrélation croisée proche de l'identité :

$$\mathcal{L}_{\mathcal{BT}} = \underbrace{\sum_i (1 - \mathcal{C}_{ii})^2}_{\text{invariance term}} + \lambda \underbrace{\sum_i \sum_{j \neq i} {\mathcal{C}_{ij}}^2}_{\text{redundancy reduction term}} \tag{3.6}$$

Le paramètre λ dans la fonction de coût $\mathcal{L}_{\mathcal{BT}}$ défini par l'équation 3.6 contrôle l'importance relative des termes de maximisation et de minimisation, c'est-à-dire que ce terme permet de réguler l'équilibre entre la similarité et la diversité dans les représentations apprises par le modèle. Une valeur plus élevée de λ augmentera l'importance de la maximisation de la corrélation entre les paires d'images similaires, ce qui renforcera la similarité entre les représentations correspondantes. D'autre part, une valeur plus faible de λ diminuera l'importance de la minimisation de la corrélation pour les paires d'images différentes, permettant ainsi plus de diversité dans les représentations.

3.2.3 Paramètres d'entraînement

Afin de lancer l'apprentissage du modèle, il nous reste à définir une fonction d'optimisation et les hyperparamètres utilisés.

Choix de la fonction d'optimisation Nous utilisons l'optimiseur Adam [12] et nous entraînons le modèle pendant 1000 époques – *epochs* – avec une taille de lot – *batch size* – de 256. Nous utilisons un taux d'apprentissage – *learning rate* – de 10^{-3} pour tous les paramètres.

Choix des hyperparamètres Le modèle comporte plusieurs hyperparamètres à définir lors de l'initialisation du SNN. Par exemple, la modélisation d'un neurone LIF possède 5 hyperparamètres. Nous avons décidé de laisser les paramètres par défaut du modèle de neurones implémenté dans la bibliothèque Norse [20] afin de nous concentrer sur la stratégie de SSL sur un SNN. Les paramètres sont présentés dans la colonne *Value* du tableau 3.1.

TABLE 3.1 – Valeurs des paramètres du modèle de neurone utilisé.

Parameter	Value	Bio value	Description
$v_{threshold}$	1.0	-50	threshold potential (mV)
v_{reset}	0.0	-65.0	reset potential (mV)
v_{leak}	0.0	-65.0	leak potential (mV)
τ_{syn}	0.005	0.5	synaptic time constant (mS)
τ_{mem}	0.01	20.0	membrane time constant (mS)

La colonne *Bio value* permet de contraster les valeurs utilisées avec celles issues de la recherche en neurosciences et utilisées dans les expérimentations de Diehl & Cook [7]. Les connexions synaptiques sont initialisées aléatoirement grâce à la méthode par défaut de la bibliothèque PyTorch qui nous permet d'avoir indirectement des neurones inhibiteurs et excitateurs. Nous avons choisi d'avoir 100 neurones LIF dans la couche intermédiaire et 100 neurones LI dans la couche finale. Concernant le coefficient α du gradient de substitution défini par l'équation 3.4, nous avons essayé différentes valeurs de α au-delà de 1 sans obtenir de grandes différences dans les résultats. Nous avons donc décidé de laisser le paramètre par défaut du modèle fixé à 100.

Un autre paramètre important du modèle est la durée du train d'impulsions générées par le processus de Poisson lors de la transformation de nos images en vue temporelle. Une séquence trop petite ne fournirait pas assez d'informations à notre encodeur f et une séquence trop grande allongerait les temps de calcul et la latence. Pour notre étude, nous utilisons une longueur de séquence de $150ms$.

Pour le paramètre λ présent dans l'équation 3.6, nous avons essayé différentes valeurs et trouvé de bons résultats pour $\lambda = 0.1$.

3.3 Étude expérimentale

Nous commençons la procédure d'évaluation par l'entraînement non supervisé du modèle proposé. Nous évaluons notre modèle à chaque époque – *epoch* – en suivant la méthode proposée dans les travaux de Wu *et al.* [25] qui consiste à appliquer l'algorithme des k plus proches voisins (kNN) qui est un moyen simple et rapide de calculer la performance de notre modèle sur les représentations générées à partir des données d'entrée. Une fois l'entraînement terminé, nous chargeons le meilleur modèle obtenu, figeons les connexions synaptiques (les poids du modèle) et extrayons les représentations de nos données. Nous implémentons notre expérience en utilisant les bibliothèques PyTorch [19] et Norse [20].

3.3.1 Protocole d'évaluation

Nous présentons ici le protocole de notre étude en détaillant le jeu de données présenté à notre architecture ainsi que les métriques mises en place pour évaluer les représentations apprises de notre modèle auto-supervisé.

Jeu de données Comme dans le chapitre précédent, nous utilisons le jeu de données commun de reconnaissance de chiffres manuscrits MNIST [13] où l'objectif est de dis-

tinguer les chiffres manuscrits de 0 à 9 sur une image en niveaux de gris de 28x28 pixels. Le jeu de données est composé à l'origine de 60000 exemples d'entraînement et 10000 exemples de test, nous avons divisé le corpus d'entraînement en deux sous-ensembles (entraînement/validation) avec un ratio de 80%-20%.

Métrique Nous évaluons la justesse – *accuracy* – de notre architecture dans le cadre de deux protocoles différents en suivant toujours la méthode proposée par Wu *et al.* [25].

Le premier consiste à utiliser un classifieur kNN. Plus précisément, nous calculons la représentation z_A de l'image test x puis nous la comparons à l'ensemble des représentations des données d'entraînement en utilisant une similarité en cosinus. Les k plus proches voisins sont utilisés pour faire la prédiction. La valeur de k est fixée à 200.

Le second protocole consiste à entraîner un classifieur supervisé, plus précisément une régression logistique. Afin d'entrainer ce classifieur linéaire, nous utilisons la même stratégie d'optimisation déployée lors de l'entraînement de notre modèle non supervisé expliquée dans la partie optimisation de la section 3.2.

3.3.2 Résultats

Deux méthodes de référence – *baseline* – ont été mises en œuvre.

La première est une méthode naïve qui consiste à mesurer la justesse – *accuracy* – des résultats obtenus à l'issue de la mise en œuvre des 2 protocoles précédemment décrits directement sur notre jeu de données. C'est-à-dire que les 2 métriques sont calculées depuis les images qui sont considérées comme des vecteurs caractéristiques de 28×28 valeurs.

La seconde consiste à entraîner le modèle SNN de Diehl & Cook [6]. La représentation extraite pour l'évaluation est le potentiel de membrane de la couche de neurones excitateurs après transformation de ces neurones LIF en LI en augmentant le seuil des neurones à l'infini.

Évaluation de la représentation La justesse – *accuracy* – top-1 obtenue sur les données de test est reportée dans le tableau 3.2. Notre méthode obtient une précision maximale de 93% sans problème d'effondrement des représentations – *representational collapse* – [3]. Ce score est obtenu avec une taille de lots – *batch size* – égale à 128.

TABLE 3.2 – Justesse – *accuracy* – des résultats Top-1 sur le corpus MNIST en appliquant un kNN et une régression linéaire sur les vecteurs caractéristiques appris.

Method	Nearest Neighbor	Linear classifier
MNIST	39.36	92.2
Diehl&Cook	10.28	90.78
Ours	89.92	93.06

3. L'effondrement des représentations désigne le phénomène où un ANN a tendance à produire des représentations similaires pour des entrées différentes. Autrement dit, les couches plus profondes du réseau ne sont plus en mesure de capturer les caractéristiques distinctives des entrées et de les séparer en catégories uniques, ce qui peut entraîner une perte de performance de la classification.

Le mauvais score du modèle de Dielh&Cook peut s'expliquer par le fait que son architecture n'est pas conçue pour utiliser uniquement la couche des neurones excitateurs lors de l'extraction des représentations de notre jeu de données. Dans l'article original [6], la précision du modèle est calculée en associant d'abord aux neurones de sortie, une étiquette correspondant à la fréquence d'émission des impulsions, puis en calculant la précision sur les données de test, ce qui donne un score de 82,9% avec 100 neurones.

Simulation de période Dans un deuxième temps, nous avons voulu évaluer les performances du modèle en fonction de différentes longueurs de train d'impulsions. Contrairement aux résultats précédemment obtenus après un apprentissage d'une durée de 1000 époques – *epochs* –, l'apprentissage réalisé ici a été limité à 100 époques afin de réduire les temps de calculs pour chaque expérimentation sur les différentes longueurs de train d'impulsions. Les résultats sont présentés dans le tableau 3.3. Plus la longueur du train d'impulsions est importante, plus le temps d'apprentissage est long. La question est de trouver le bon rapport entre le temps de calcul et la précision du modèle obtenu. Dans notre cas, la durée de 150 ms semble être un bon compromis. Les temps de calcul indiqués sur la dernière ligne du tableau 3.3 sont relatifs au modèle utilisant des trains d'impulsions de 150 ms.

TABLE 3.3 – Performance du modèle en fonction de la longueur de la séquence d'impulsions.

Spike train length (in ms)	50	150	250	350
k-NN Accuracy (%)	90.12	89.37	90.28	89.84
Linear Accuracy (%)	91.54	92.09	92.07	91.72
Relative computing time	0.52	1	2.1	2.93

Influence de la taille du lot L'un des derniers facteurs étudiés dans cette étude est l'influence de la taille du lot – *batch size* – sur les performances du modèle présentées dans la figure 3.3. Encore une fois, il est intéressant de constater que plus la taille du lot est grande, plus les performances sont réduites, on peut perdre jusqu'à 10 points. Il est donc pertinent de travailler sur des *mini-batch*. En effet, il semble logique que l'augmentation de la taille du lot – *batch size* – se traduise par un effet moyenneur plus prononcé, induisant une augmentation de la variance du gradient. Cette dernière, bien qu'accélérant la convergence, risque toutefois de conduire à une convergence vers un minimum local plutôt qu'un minimum global.

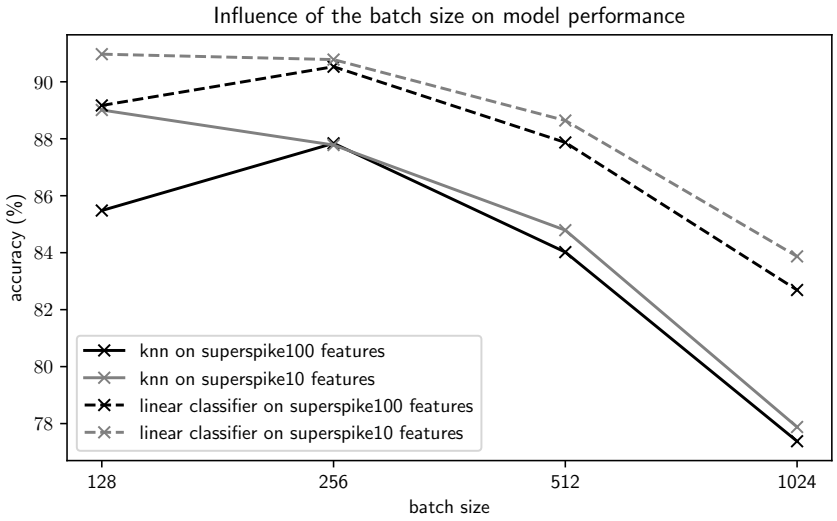


FIGURE 3.3 – Performance du modèle pendant l'entraînement avec différentes tailles de lots.

3.4 Conclusion

Dans cette étude, nous nous sommes inspirés de stratégies non supervisées issues de l'apprentissage profond et des approches SNN pour produire une représentation discriminante des images. Le vecteur de représentation exploite le potentiel de membrane de la dernière couche du modèle construit avec par des neurones LI. Nous avons montré que cette représentation améliore la capacité à discriminer ces images par des méthodes de classification standard, tout en étant facile à implémenter. L'approche stochastique établie par le processus de Poisson couplé à la fonction de perte *Barlow Twins* pousse le réseau à identifier des éléments caractéristiques communs, ouvrant ainsi sa capacité à la généralisation. Ce dernier point reste néanmoins à démontrer expérimentalement et pourront faire l'objet de travaux ultérieurs.

Ces travaux ont fait l'objet d'une publication d'un article à la conférence CBMI 2022 [2].

Références

[1] Yoshua Bengio, Nicholas Léonard et Aaron C. Courville. “Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation”. In : *CoRR* abs/1308.3432 (2013).

[2] B. Chamand et P. Joly. “Self-Supervised Spiking Neural Networks applied to Digit Classification”. In : *Proceedings of the 19th International Conference on Content-Based Multimedia Indexing – CBMI 2022*. Graz (Austria) : ACM, 2022.

[3] Ting Chen et al. “A Simple Framework for Contrastive Learning of Visual Representations”. In : *Proceedings of the 37th International Conference on Machine Learning*. Sous la direction d’Hal Daumé III et Aarti Singh. Tome 119. Proceedings of Machine Learning Research. PMLR, 2020, pages 1597-1607.

[4] Xinlei Chen et Kaiming He. “Exploring Simple Siamese Representation Learning”. In : *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. Juin 2021, pages 15750-15758.

[5] Xiang Cheng et al. “LISNN : Improving Spiking Neural Networks with Lateral Interactions for Robust Object Recognition”. In : *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*. Sous la direction de Christian Bessiere. Main track. International Joint Conferences on Artificial Intelligence Organization, juill. 2020, pages 1519-1525.

[6] Peter Diehl et Matthew Cook. “Unsupervised learning of digit recognition using spike-timing-dependent plasticity”. In : *Frontiers in Computational Neuroscience* 9 (2015).

[7] Peter U. Diehl et Matthew Cook. “Unsupervised learning of digit recognition using spike-timing-dependent plasticity”. In : *Frontiers in Computational Neuroscience* 9 (2015).

[8] Jean-Bastien Grill et al. “Bootstrap Your Own Latent - A New Approach to Self-Supervised Learning”. In : *Advances in Neural Information Processing Systems*. Sous la direction de H. Larochelle et al. Tome 33. Curran Associates, Inc., 2020, pages 21271-21284.

[9] Jesse Hagenaars, Federico Paredes Valles et Guido De Croon. “Self-Supervised Learning of Event-Based Optical Flow with Spiking Neural Networks”. In : *Advances in Neural Information Processing Systems*. Sous la direction d’A. Beygelzimer et al. 2021.

[10] Kaiming He et al. “Momentum Contrast for Unsupervised Visual Representation Learning”. In : *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020, pages 9726-9735.

[11] Saeed Reza Kheradpisheh et al. “STDP-based spiking deep convolutional neural networks for object recognition”. In : *Neural Networks* 99 (2018), pages 56-67.

[12] Diederik P. Kingma et Jimmy Ba. “Adam : A Method for Stochastic Optimization”. In : *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Sous la direction d’Yoshua Bengio et Yann LeCun. 2015.

[13] Yann LeCun et Corinna Cortes. “MNIST handwritten digit database”. In : (2010).

[14] Chankyu Lee et al. “Enabling Spike-Based Backpropagation for Training Deep Neural Network Architectures”. In : *Frontiers in Neuroscience* 14 (2020).

[15] Chankyu Lee et al. “Spike-FlowNet : Event-based Optical Flow Estimation with Energy-Efficient Hybrid Neural Networks”. In : *European Conference on Computer Vision*. Springer. 2020, pages 366-382.

[16] Yuhang Li et al. “Neuromorphic Data Augmentation for Training Spiking Neural Networks”. In : *Computer Vision – ECCV 2022*. Sous la direction de Shai Avidan et al. Cham : Springer Nature Switzerland, 2022, pages 631-649.

[17] Emre O. Neftci, Hesham Mostafa et Friedemann Zenke. “Surrogate Gradient Learning in Spiking Neural Networks : Bringing the Power of Gradient-Based Optimization to Spiking Neural Networks”. In : *IEEE Signal Processing Magazine* 36.6 (2019), pages 51-63.

[18] F. Paredes-Vallés, Kirk Y. W. Scheper et Guido de Croon. “Unsupervised Learning of a Hierarchical Spiking Neural Network for Optical Flow Estimation : From Events to Global Motion Perception”. In : *IEEE Transactions on Pattern Analysis and Machine Intelligence* 42 (2020), pages 2051-2064.

[19] Adam Paszke et al. “PyTorch : An Imperative Style, High-Performance Deep Learning Library”. In : *Advances in Neural Information Processing Systems 32*. Sous la direction de H. Wallach et al. Curran Associates, Inc., 2019, pages 8024-8035.

[20] Christian Pehle et Jens Egholm Pedersen. *Norse - A deep learning library for spiking neural networks*. Version 0.0.7. Documentation : https ://norse.ai/docs/. Jan. 2021.

[21] Michael Pfeiffer et Thomas Pfeil. *Deep Learning With Spiking Neurons : Opportunities and Challenges*. 2018.

[22] Kaushik Roy, Akhilesh Jaiswal et Priyadarshini Panda. “Towards spike-based machine intelligence with neuromorphic computing”. In : *Nature* 575.7784 (2019).

[23] Sumit Bam Shrestha et Garrick Orchard. “SLAYER : Spike Layer Error Reassignment in Time”. In : *Advances in Neural Information Processing Systems*. Sous la direction de S. Bengio et al. Tome 31. Curran Associates, Inc., 2018.

[24] Yujie Wu et al. “Spatio-Temporal Backpropagation for Training High-Performance Spiking Neural Networks”. In : *Frontiers in Neuroscience* 12 (2018).

[25] Zhirong Wu et al. “Unsupervised Feature Learning via Non-Parametric Instance Discrimination”. In : *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018.

[26] Jure Zbontar et al. “Barlow Twins : Self-Supervised Learning via Redundancy Reduction”. In : *Proceedings of the 38th International Conference on Machine Learning*. Sous la direction de Marina Meila et Tong Zhang. Tome 139. Proceedings of Machine Learning Research. PMLR, juill. 2021, pages 12310-12320.

[27] Friedemann Zenke et Surya Ganguli. *Superspike : Supervised learning in multilayer spiking neural networks*. 2018.

[28] Friedemann Zenke et Tim P. Vogels. “The Remarkable Robustness of Surrogate Gradient Learning for Instilling Complex Function in Spiking Neural Networks”. In : *Neural Computation* 33.4 (mars 2021), pages 899-925.

Deuxième partie

La connaissance guidée par l'IA

Chapitre 4

Heuristique pour la température

La multitude d'hyperparamètres présents dans les méthodes d'apprentissage automatique peut rendre la tâche d'entraînement du modèle complexe. Afin de simplifier la tâche d'initialisation des hyperparamètres du modèle, il pourrait être intéressant que des heuristiques établissent des choix automatiques permettant d'accélérer les résultats, et ainsi d'adapter plus rapidement le modèle à une tâche donnée. C'est ce que nous proposons à travers l'étude de l'influence d'un hyperparamètre : la température, sur la performance d'un modèle.

La température est un hyperparamètre largement utilisé dans diverses tâches impliquant des réseaux de neurones profonds comme la tâche de classification ou l'apprentissage de distance – *metric learning* –, dont le choix peut avoir un impact significatif sur les performances du modèle. La plupart des travaux existants sélectionnent sa valeur à l'aide de méthodes d'optimisation d'hyperparamètres qui nécessitent plusieurs entraînements pour trouver la valeur optimale, en utilisant des méthodes basées sur la validation croisée par exemple. Nous proposons d'analyser l'impact de la température sur les tâches de classification en décrivant un ensemble de données comme un ensemble de statistiques calculées sur des représentations pré-extraites. Dans l'idéal, une telle méthode aiderait à découvrir une heuristique fondée sur les statistiques les plus corrélées et l'hyperparamètre étudié.

Nous proposons ici d'étudier la corrélation entre ces statistiques extraites et les températures optimales observées sur plus d'une centaine de combinaisons de différents ensembles de données et extracteurs de caractéristiques afin de construire une heuristique générale pour la température.

Sommaire

4.1 Définition et travaux principaux

4.1.1 Introduction

Comme je l'ai montré dans les chapitres précédents, les réseau de neurones à impulsions (SNN) disposent d'une multitude d'hyperparamètres (paramètres de la méthode d'apprentissage, du modèle de neurones, de l'encodage, etc.) et dont leurs choix peuvent grandement affecter les performances du réseau. Ces paramètres sont difficiles à trouver du premier coup, il faut lancer plusieurs itérations de l'algorithme pour converger vers un bon jeu d'hyperparamètres initials permettant d'obtenir un comportement dériré durant l'apprentissage. Cela s'applique également à toutes les autres disciplines en apprentissage automatique. Afin de simplifier la tâche, nous proposons d'utiliser non plus des modèles impulsionnels, mais des modèles fréquentielles comme les ANNs dû à sa communauté très forte comme en témoigne le nombre de modèles et de datasets existants. Afin de simplifier la tâche, nous nous focalisons à l'étude d'un hyperparamètre dans le domaine des ANNs et plus particulièrement présent dans le domaine très actif de la vision par ordinateur : la température.

La température est un paramètre souvent utilisé dans un réseau de neurones lié à la couche de la fonction exponentielle normalisée, appelé *softmax* et jouant le rôle de facteur d'échelle à la sortie du modèle. Généralement, la couche *softmax* est suivie d'une fonction de coût de type entropie croisée (CE) pour optimiser le modèle. En allusion à la physique statistique, la température a été introduite pour choisir le niveau d'uniformité de la distribution en sortie de la couche *softmax*. Étant donné que la plupart des modèles de classification en apprentissage profond fait appel en même temps à la couche *softmax* et aux fonctions de perte de type CE (malgré leurs limites [27, 46]) pour leur optimisation, la détermination de manière automatique d'une température optimale pour une tâche particulière est un sujet important pour la communauté [1, 20].

Par exemple, ce paramètre est largement étudié dans diverses tâches telles que la dis-

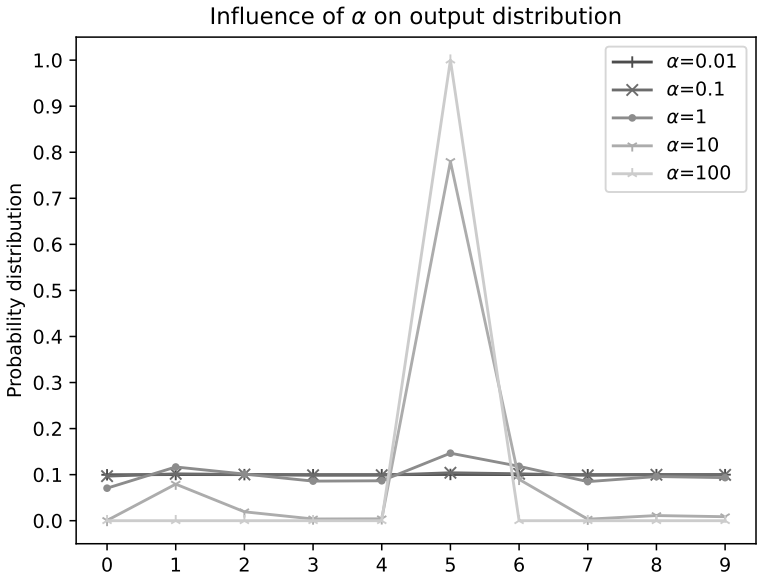


FIGURE 4.1 – Pour une dizaine de *logits* en entrée de la couche *softmax*, le choix de la température α fait tendre la distribution de probabilités différemment. En effet, lorsque $\alpha \to 0$, la distribution est plutôt uniforme, tandis qu'avec $\alpha \to \infty$, la distribution est piquée.

tillation de connaissance, la classification, la génération de textes, l'apprentissage auto-supervisé et de métrique [2, 6, 13, 18, 19, 20, 30, 35, 40, 43, 44]. Traditionnellement, dans la plupart de ces domaines et dans les applications sous-jacentes, la température est déterminée empiriquement, avec une valeur qui peut être constante (typiquement à partir d'une recherche sur grille) ou évoluer dynamiquement au fil des itérations, dans la même veine que l'hyperparamètre d'apprentissage du modèle (*learning rate*). Néanmoins, ces stratégies de détermination d'une « bonne » température peuvent être sous-optimales ou trop lourdes d'un point de vue computationnel. De manière surprenante, il existe très peu d'études proposant des stratégies pour déterminer une température optimale. Dans ce chapitre, nous nous concentrons sur le problème particulier suivant : étant donné une tâche de classification, nous souhaitons trouver une corrélation entre une valeur optimale pour la température et les statistiques décrivant le jeu de données telles que la complexité, la dimension, le nombre de classes, etc.

4.1.2 Travaux connexes

L'hyperparamètre de température, généralement utilisé dans la couche *softmax*, permet de déterminer à quel point une distribution de probabilité en sortie est « piquée ». La figure 4.1 montre l'influence du paramètre de température sur une dizaine de *logits* [1] donnée en entrée de la couche *softmax*. Bien que l'utilisation d'une bonne température ait montré son impact dans de nombreuses tâches de vision par ordinateur, les stratégies existantes pour définir un tel paramètre de température sont assez limitées.

1. Dans le domaine de l'apprentissage profond, la couche *logits* désigne l'entrée de la couche *softmax* ou toute autre normalisation de ce type qui va générer une distribution de probabilités.

La température comme une constante La première façon de procéder est de considérer une température constante tout au long de l'apprentissage. Ce choix peut être fait de manière empirique [18, 43]. Il peut également être considéré comme un hyperparamètre fixe à optimiser via une recherche par grille dans un champ de valeurs possibles. Cependant, cela implique des besoins de calcul importants et conduit à des hyperparamètres différents pour chaque jeu de données et chaque architecture.

Une simple heuristique peut également permettre de fixer le paramètre. Par exemple dans les modèles « Transformers » [39] avec une température fixé à $\sqrt{d}$, d étant la dimension du vecteur requête – *queries* – et des vecteurs clés – *keys* . Ce choix est porté par l'idée de mettre à l'échelle le produit scalaire entre les deux vecteurs afin d'avoir une variance égale à un tout en évitant d'avoir le problème de la disparition du gradient lorsque ce produit scalaire prend une grande ampleur.

Ajustement dynamique D'autres stratégies repose sur un ajustement dynamique de la température pendant les itérations d'apprentissage. Dans ce cas, les éléments de la température peuvent évoluer à chaque époque grâce à un ordonnanceur [19], à la manière du pas d'apprentissage – *learning rate* – pour affiner l'optimisation du réseau. Dans [45], les auteurs ont également montré qu'une normalisation par lots – *batch-norm* – rééchelonnée par une température $\sqrt{d}$, avec d le nombre de dimensions des représentations – *embeddings* –, fonctionnait légèrement mieux qu'une simple normalisation ℓ_2, et permettait d'avoir plus de variances dans les vecteurs de représentations. L'ajustement dynamique de la température peut également être effectué en l'optimisant comme un paramètre standard du modèle [33, 34]. Cela nécessite généralement des étapes supplémentaires comme l'écrêtage – *clipping* – ou l'ajout d'une fonction exponentielle pour éviter les valeurs négatives. En outre, la température apprise dépend fortement de l'hyperparamètre du pas d'apprentissage.

Étude analytique Une autre approche consiste à déterminer la valeur de température de manière analytique. He *et al.* proposent à la fois une fonction d'évaluation conçue pour mesurer l'efficacité d'un paramètre de température et une règle de mise à jour itérative pour déterminer la valeur de température optimale [15]. Cependant, leur travail souffre de deux inconvénients : (1) les auteurs introduisent un nouvel hyperparamètre λ dans la formulation de la température, λ étant un facteur d'amélioration affectant le nombre d'itérations et la sélection de la température optimale ; (2) il a été conçu pour le problème du $\mathcal{D}$*-bandit* armé en apprentissage par renforcement et testé uniquement sur des données synthétiques.

Heuristique sur le seuil D'autres travaux proposent une limite inférieure théorique formulée comme une fonction de la valeur de la fonction de la coût et du nombre de classes [28]. Il est intéressant de noter que, contrairement à [39, 45], leur solution n'utilise aucune information issue des représentations comme la dimension.Cependant, comme la détermination de la température n'était pas la partie principale de leur contribution, aucune étude n'a été fait pour comparer la borne inférieure théorique proposée avec d'autres valeurs de température.

Alors que certaines des heuristiques mentionnées précédemment sont basées sur les dimensions des caractéristiques, d'autres utilisent le nombre de classes ou les mesures

de séparabilité des classes. Aucune d'entre elles n'a été conçue spécifiquement pour être utilisée dans une tâche de classification ou n'a été évaluée sur cet hyperparamètre particulier pour démontrer l'efficacité de l'heuristique proposée. D'autres heuristiques pourraient être dérivées d'autres critères reflétant des informations telles que la difficulté de l'ensemble de données à classer. Par exemple, [29] propose d'estimer la difficulté de classer des ensembles de données à partir de six classes de mesures basées sur des informations telles que les mesures de caractéristiques, de voisinages ou de dimensionnalités. Cependant, la plupart des mesures proposées ont une complexité au moins égale à $O(n^2)$, avec n le nombre de points dans le jeu de données, ce qui rend ces mesures difficiles à adapter à des jeux de données plus importants. De même, nous cherchons à décrire chaque ensemble de données par un ensemble de statistiques calculables en un temps raisonnable que nous appellerons *datasets statistics*. Nous proposons ensuite de déterminer quelles variables / statistiques sont réellement corrélées avec la meilleure température empirique afin de proposer une heuristique simple basée sur ces statistiques de l'ensemble des données.

4.2 Réflexions théoriques

Avant de continuer sur la présentation de notre chaîne d'apprentissage et les résultats qui en découlent, il faut s'assurer de bien définir et isoler le paramètre étudié, dans notre cas celui de la température.

4.2.1 Impact théorique de la température sur l'algorithme d'optimisation

Inspirés par la formulation de Zhang et al. [45], nous partons sur la même observation originelle qu'eux. Nous définissons un ensemble de N échantillons étiquetés $\{(x_1, y_1), \ldots, (x_N, y_N)\}$, où $x_i \in \mathbb{R}^d$ est la représentation – caractéristiques, *embeddings* – du i-ème exemple d'apprentissage, d étant la dimensionnalité de x_i, et $y_i \in \{1, \ldots, C\}$ est l'étiquette de la classe de l'échantillon x_i, C étant le nombre total de catégories. Considérons $W = [w_1, \ldots, w_C]$ où $w_j \in \mathbb{R}^d$ est le poids associé à la classe C_j, nous définissons $z_i = x_i W$ avec $i \in \{1, \ldots, N\}$. Dans notre cas, nous nous concentrons sur l'apprentissage des poids synaptiques W. Dans le même esprit que [20, 28, 43, 44], nous avons supprimé le terme de biais, et nous considérons les entrées x et les poids synaptiques W normalisés l_2 .

Nous optimisons la similarité cosinus puisque ce choix est à la fois populaire en classification et en apprentissage métrique.

$$\cos\theta = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|}. \tag{4.1}$$

La probabilité qu'un échantillon x appartienne à la catégorie $c \in \{1, \ldots, C\}$ peut être prédite par la fonction softmax :

$$p(c|x, \alpha) = \frac{\exp(\alpha z_c)}{\sum_{j=1}^{C} \exp(\alpha z_j)} \tag{4.2}$$

Pour simplifier la notation, nous notons $\alpha = 1/T$ comme l'inverse de la température T. Par abus de langage, nous désignerons dans la suite le terme α comme étant la « température ». Ce paramètre permet de produire une distribution resserrée autour du

centre de classe si T est grand ($alpha$ petit) ou étalée si T est petit ($alpha$ grand). Dans la définition classique de la *softmax*, T est implicitement présent en prenant la valeur de 1.

En supposant que la distribution de la vérité terrain de l'ensemble d'apprentissage est $q(c|x)$ généralement codée dans un vecteur binaire *one-hot* (qui vaut 1 si $c = y$ et 0 sinon), la fonction de coût CE par rapport à x est définie comme suit :

$$\mathcal{L}_{\text{CE}}(x, \alpha) = -\sum_{c=1}^{C} \log(p(c|x, \alpha))q(c|x) \tag{4.3}$$

et le gradient par rapport au poids w_c est :

$$\frac{\partial \mathcal{L}_{\text{CE}}}{\partial w_c} = \alpha(p(c|x, \alpha) - q(c|x))x^\top \tag{4.4}$$

D'après l'équation. 4.4, nous pouvons observer que la température intervient à deux niveaux dans le calcul du gradient. Le premier consiste, comme mentionné précédemment, dans l'ajustement de la distribution de probabilité $p(c|x, \alpha) \in [0, 1]$. Le second niveau consiste en l'intervention du paramètre α, la température, comme facteur d'amplification (multiplicatif) de l'écart entre les deux distributions dans le calcul du gradient. Comme α se situe souvent dans $[1, 250]$ de manière empirique, ce dernier effet peut être néfaste lors de l'apprentissage puisqu'il multiplie le pas d'apprentissage avec cette valeur ; cela peut conduire à une divergence. Pour annuler cet effet et étudier uniquement l'impact du choix de la distribution lors de l'apprentissage, nous proposons de normaliser la fonction de perte CE par une constante de valeur égale à α. Notons qu'il est important que cette valeur soit considérée comme une constante et non pas dérivable car cela changerait notablement les calculs des gradients.

L'équation 4.3 devient alors :

$$\mathcal{L}_{\text{CE}}(x, \alpha) = -\frac{1}{\alpha}\sum_{c=1}^{C} \log(p(c|x, \alpha))q(c|x) \tag{4.5}$$

Cela permet de corriger le gradient par rapport au poids w_c qui devient donc :

$$\frac{\partial \mathcal{L}_{\text{CE}}}{\partial w_c} = (p(c|x, \alpha) - q(c|x))x^\top \tag{4.6}$$

4.2.2 Validation empirique de la CE corrigée

Nous cherchons ici à observer empiriquement l'influence du choix de la température sur l'apprentissage tout en validant la proposition de correction de la CE précédemment introduite. Pour ce faire, nous utilisons le dataset *THINGS* [16] composé de 1854 classes et plus de 26000 images d'objets. Pour se concentrer uniquement sur la dernière couche de classification linéaire nous extrayons des représentations l_2 normalisées à partir d'un réseau pré-entraîné (ici CLIP [33]). Pour chaque exécution, nous découpons cette base en deux : *apprentissage, validation* et initialisons aléatoirement la matrice de poids W. Chaque apprentissage est fait durant 100 *epochs* et le meilleur score en validation est retenu. Chaque température α prise dans un ensemble prédéfini de valeur $\alpha \in [0.5, 1, 10, 20, 30, 40, 50, 70, 90, 100, 200, 1000]$ est testée 20 fois et les résultats sont agrégés dans la figure 4.2 afin d'être statistiquement plus significatifs et robustes au bruit

lié à la stochasticité d'un apprentissage. L'optimiseur choisi pour les expériences de la figure 4.2 est un algorithme du gradient stochastique (SGD) avec un pas d'apprentissage à 10.

Nous pouvons observer sur la figure 4.2 que les premières valeurs de température donnent des résultats similaires. Lorsque la CE n'est pas corrigée, la température sur des valeurs plus élevées provoque une divergence du réseau pendant l'apprentissage, menant à des scores dramatiques. Ce constat est cohérent avec l'idée qu'un pas d'apprentissage trop élevé peut faire diverger un réseau. En revanche, une fois la fonction de coût corrigée, la température permet d'améliorer les scores (cf. température à 40, 50). Notons que la température optimale n'est pas triviale ni sur les plus hautes ni les plus basses valeurs testées et varie selon l'ensemble de données utilisé.

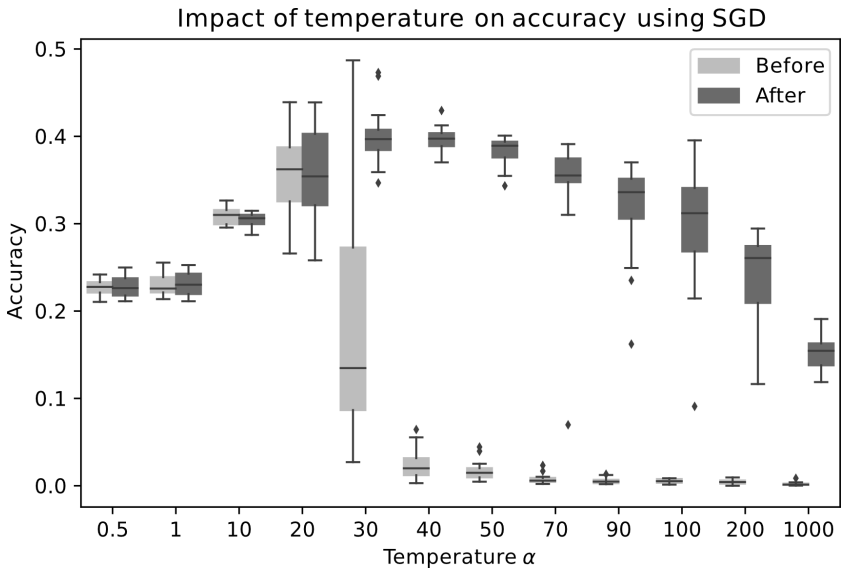


FIGURE 4.2 – Impact de la température α sur la justesse – *accuracy* – après apprentissage sur le jeu de données THINGS [16]

La même expérience a été réalisée en utilisant l'optimiseur adaptatif *Adam* [21]. Aucune différence significative dans les scores obtenus avant ou après la correction de la CE n'a pu être observée. Cela confirme, entre autres, la robustesse de l'optimiseur *Adam* sur le choix d'un pas d'apprentissage adapté tout au long de l'entrainement, alors même que la température affecte le gradient. La température optimale trouvée est également aux alentours de $[40, 50]$.

Il est donc intéressant de comprendre quels sont les facteurs affectant le choix de la température nous permettant d'obtenir une heuristique optimale sur cet hyperparamètre.

4.3 Découverte de corrélations

Dans cette partie, nous présentons l'approche proposée pour découvrir des corrélations entre les données d'apprentissage et le paramètre de température. Comme l'illustre la figure 4.3, notre méthode se compose de deux parties principales : (1) la création d'un

métacorpus $\mathcal{M}$ à partir de la combinaison de différents jeux de données et extracteurs de caractéristiques, de la représentation dans un même espace et la création de la vérité terrain ; (2) la découverte d'heuristiques par le moyen de calcul de corrélations avec le paramètre étudié. Nous détaillons chaque sous-composante dans les sections suivantes.

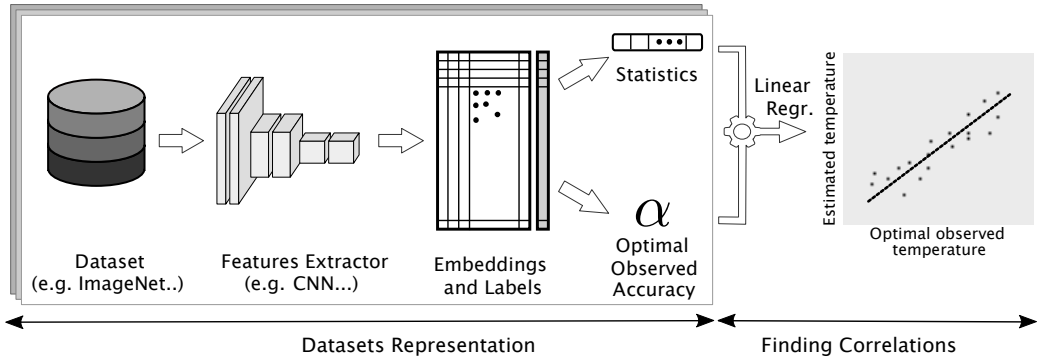


FIGURE 4.3 – Chaîne de traitements proposée pour trouver des relations entre la température optimale et des statistiques issues des données.

4.3.1 Bases de données, extracteurs de caractéristiques

Afin de trouver une heuristique généralisable couvrant un large éventail de cas pour une tâche de classification, nous avons sélectionné 12 jeux de données et 9 extracteurs de caractéristiques. Le nombre de classes des corpus choisi varie de 10 à 1854 tandis que la dimensionnalité des représentations extraites varie de 256 à 2048.

Ensemble de données Le tableau 4.1 présente les jeux de données sélectionnés pour l'expérimentation. Ces choix ont été réalisés en fonction de la facilité d'obtention du jeu de données ainsi que la taille associée afin de rester raisonnable sur la puissance de calculs à déployer pour l'étude. Même si l'ensemble des corpus sélectionnés présente majoritairement des images naturelles, celles-ci restent différentes par leur résolution, leurs classes, etc.

Extracteurs de caractéristiques On appelle extracteur de caractéristique, un réseau de neurones artificiels (ANN) pré-entrainé sur une tâche spécifique dans lequel la dernière couche, généralement appelée couche de classification (par exemple, la couche *softmax*), est tronquée permettant ainsi une réduction de dimensionnalité des données d'entrée. En alimentant nos jeux de données dans ce type de modèle, nous obtenons donc des représentations – caractéristiques – encodant de l'information pertinente.

Pour la création de notre métacorpus $\mathcal{M}$, plusieurs architectures ont été choisies et sont résumées dans le tableau 4.2. Ces modèles d'extracteur de caractéristiques ont été sélectionnées en raison de leurs différents pré-entraînements et leurs dimensions en sortie distinctes. Cela permet de décorréler ces paramètres des performances du réseau. Par exemple, on s'attend à ce qu'un réseau de type *FaceNet* soit peu performant sur le jeu de données *CIFAR* puisqu'il est appris sur une tâche de reconnaissance de visages, à l'inverse, un réseau de type *ResNet-18* pré-entraîné sur le jeu de données *ImageNet* est censé être plus performant sur une tâche de classification d'images naturelles que sur une tâche de reconnaissance de visages avec le jeu de données *105-PinterestFaces*.

TABLE 4.1 – Liste des jeux de données utilisés triés par leur nombre de classes associé.

Dataset	**Nb. classes**	**Description**
CIFAR10 [22]	10	Images naturelles
MNIST [25]	10	Images de chiffres
DTD [7]	47	Images de textures (tâcheté, plissé, etc.)
PhotoArt [42]	50	Images issues de photo ou d'œuvres d'art
CIFAR100 [22]	100	Images naturelles
105-PinterestFaces [32]	105	Images de visages de différentes célébrités
CUB200 [41]	200	Images d'oiseaux
ImageNet-R [17]	200	Images diverses (sculptures, broderies, etc.)
Caltech256 [12]	256	Images naturelles
FSS1000 [26]	1000	Images naturelles
ImageNetMini [24]	1000	Images naturelles
THINGS [16]	1854	Images naturelles

TABLE 4.2 – Listes des extracteurs de caractéristiques et leur type d'initialisation.

Architecture	**ImageNet** [9]	**VGGFaces2** [4]	**N/A**	**Aléatoire**
AlexNet [23]	✓			
CLIP-{RN50, ViT32b} [33]			✓	
FaceNet [37]		✓		
ResNet-{18, 50, 101} [14]	✓			
ResNet-{34, 152} [14]				✓

Concernant les modèles CLIP-RN50 et CLIP-ViT32b, les auteurs proposent les modèles pré-entraînés sur des millions de paires images/textes sans définir les sources des données, d'où la notation « N/A ».

Couverture des représentations extraites Nous avons fait en sorte de couvrir un large champ de dimensions aussi bien à l'entrée de notre chaîne de traitement que sur la granularité des représentations extraites. La figure 4.4 montre cette couverture entre toutes les combinaisons d'extracteurs de caractéristiques et de jeux de données, ainsi que la meilleure performance obtenue, représentée de manière proportionnelle à la taille des cercles.

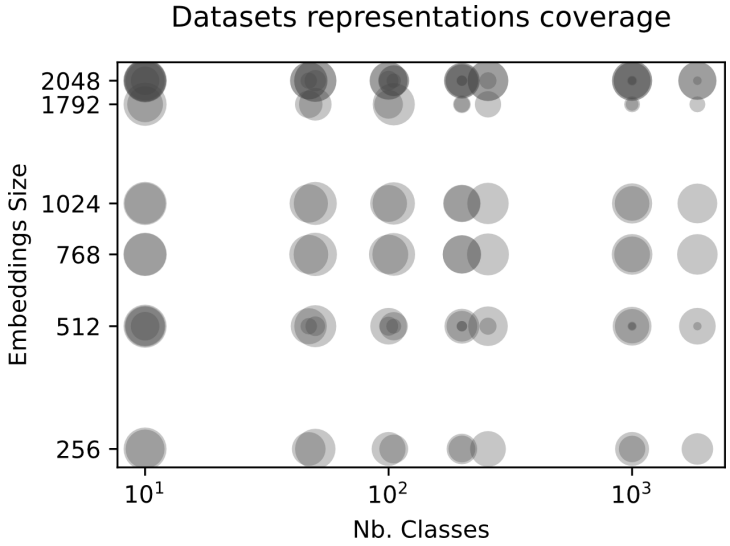


FIGURE 4.4 – Couverture de toutes les combinaisons d'extracteurs de caractéristiques avec tous les jeux de données utilisés. La taille des cercles représente la meilleure justesse – *accuracy* – obtenue pour chaque ensemble de données.

4.3.2 Création du métacorpus

Les représentations – *embeddings* – obtenues précédemment posent le souci qu'elles ne sont pas toutes de la même dimensionnalité en raison de la différence de sortie des extracteurs de caractéristiques. Pour un jeu de données et deux extracteurs différents, nous obtenons deux ensembles de représentations de tailles différentes. Afin de trouver la relation entre un jeu de données et sa température optimale associée, nous proposons de décrire chaque corpus par un vecteur de caractéristiques s dans un espace commun $\mathcal{S}$ de même dimensionnalité. Pour cela, les représentations – *embeddings* – de nos différents corpus issues des extracteurs de caractéristiques sont de nouveau transformées en un nouvel ensemble de caractéristiques pouvant être représentés comme un vecteur de taille fixe correspondant aux nombres de caractéristiques extraites, évitant ainsi les problèmes de dimension des extracteurs. S'en vient ensuite la création des annotations. Pour cela, nous allons chercher à trouver la température α optimale pour chaque vecteur de représentations de nos jeux de données.

Choix des caractéristiques Les caractéristiques choisies sont listées dans le tableau 4.3 et correspondent à des statistiques ou des scores que l'on peut directement extraire sur nos représentations – *embeddings* – ou nos corpus de départ. À noter que ces statistiques sont relativement élémentaires et qu'elles sont souvent combinées ou transformées de manière non linéaire pour en tirer un meilleur parti dans des indicateurs statistiques plus élaborés. Par exemple, le ratio de Fisher [11] utilisé dans l'analyse discriminante linéaire (LDA) est composé des deux traces des matrices de covariance intra et inter-classes (*sb_trace*, *sw_trace*), tandis que la dimensionnalité *dim* est passée à la racine carrée dans les « Transformers » [39].

TABLE 4.3 – Liste des statistiques extraites de nos représentations pour la construction du metacorpus $\mathcal{M}$.

	Notation	Description
Statistique	*dim*	Dimensionnalité des représentations
	n_classes	Nombre de classes de sortie
	mean	Moyenne de toutes les représentations
	var	Variance de toutes les représentations
	sb_trace	Trace de la matrice moyenne de toutes les matrices de covariance intra-classe
	sw_trace	Trace de la moyenne de toutes les matrices de covariance inter-classes
	feats_corr	Erreur quadratique moyenne (MSE) entre la matrice de corrélation des caractéristiques et la matrice identité
	feats_cos_sim	Similarité moyenne en cosinus entre chaque paire de dimensions
	n_samples	Nombre d'échantillons dans l'ensemble d'apprentissage
	avg_samp_class	Nombre moyen d'échantillons par classe
	pca_%	Pourcentage de dimensions à retenir pour une variance expliquée donnée (à l'aide de la PCA) de 50, 75, 80, 90, 95, 99
	train_mean	Moyenne de toutes les valeurs des représentations
	train_std	Écart type de toutes les valeurs des représentations
	avg_kurtosis	Valeur moyenne des Kurtosis calculés sur chaque dimension
	avg_normality	Valeur moyenne de Shapiro-Wilk calculés sur chaque dimension [38]
Score	*silhouette*	Score de Silhouette [36]
	calinski_harabasz	Score de Calinski Harabasz [3]
	david_bouldin	Score de David Bouldin [8]

Création des annotations Une fois que nous avons extrait l'ensemble des représentations – *embeddings* – de nos jeux de données par les différents extracteurs de caractéristiques, nous proposons de trouver la meilleure température pour chaque cas. Pour ce faire, nous divisons chaque ensemble de données de représentations en un ensemble de données d'apprentissage et de test (respectivement 75% et 25%) et puis nous entrainons le modèle pendant 1000 époques – *epochs* – avec une taille de lots – *batch size* – de 2048 pour un ensemble de températures α comprenant la valeur 1, puis les valeurs de 5 à 250 espacées d'un pas de 5 : $\alpha \in \{1, 5, \ldots, 245, 250\}$. En observant la justesse – *accuracy* – sur l'ensemble de test, nous sommes en mesure d'identifier la meilleure précision réalisable pour chaque température. Notre modèle utilise la fonction de coût CE normalisée par une valeur constante égale à α et présentée dans la section 4.2.2. Les expérimentations confirment bien que cette fonction de coût permet de fortes améliorations de la précision à haute température avec l'optimiseur SGD, mais nous n'avons constaté aucun impact pendant l'apprentissage lors de l'utilisation d'un optimiseur plus intelligent tel qu'Adam [21].

Création de notre métacorpus De ce fait, nous avons obtenu un métacorpus associant à chaque vecteur de représentation statistique s, une valeur empirique optimale de la température.

4.3.3 Étude expérimentale

Afin de trouver une heuristique pour définir une température adaptée à un jeu de données, nous avons cherché les fortes corrélations entre les paires de températures empiriques optimales et les statistiques extraites des ensembles de données. Pour cela, nous avons utilisé le coefficient de Pearson.

Coefficient de Pearson Le coefficient de Pearson [31], appelé également coefficient de corrélation linéaire de Pearson, est un indice indiquant la relation linéaire entre deux variables continues. Cette indice varie entre -1 (lorsque l'une des variables diminue que l'autre augmente et vice-versa) et $+1$ (les variables varient dans le même sens), 0 étant la relation nulle entre les deux variables. Ce coefficient s'exprime par le quotient entre la covariance Cov entre les deux variables et le produit de leurs écarts types σ :

$$r = \frac{\text{Cov}(X, Y)}{\sigma_X \sigma_Y} \tag{4.7}$$

où X et Y sont des variables aléatoires réelles.

Valeur-p La valeur-p – *p-value* – complète généralement le coefficient de Pearson en évaluant la fiabilité du résultat en rejetant l'hypothèse nulle [2]. Elle indique approximativement la probabilité qu'un système non corrélé produise des ensembles de données ayant une corrélation de Pearson au moins aussi extrême que celle calculée à partir de ces ensembles de données. Cette valeur correspond

$$p = 2 * \text{P}(X > |z|) \tag{4.8}$$

2. L'hypothèse nulle H_0 repose sur l'égalité entre des paramètres statistiques de deux échantillons équivalents. En d'autres termes, la moyenne et la variance des deux échantillons sont égales.

où X est la loi de la probabilité de la statistique et z est la valeur calculée de la statistique de test (z-score). De manière générale, les chercheurs suivent le conseil de Fisher [10] qui a proposé plusieurs seuils pour la valeur-p. Il exprime ainsi que la notion de « significativité »[3] de la relation entre deux variables doit être associée à une valeur-$p \leq 0.05$ (ce qui correspond à une "relation" 20 fois supérieure à celle d'une relation entre une des variables et le hasard) Afin de valider notre étude, il faut donc que la valeur-p soit très « petite ».

Analyse de la corrélation La figure 4.5 montre la corrélation absolue obtenue entre chaque valeur statistique et la température optimale empirique α associée. Nous avons constaté que la variable la plus intéressante était la mesure de la corrélation entre les caractéristiques des représentations *feats_corr* avec une corrélation de Pearson de 0.8184 et une valeur-p de $8.35\text{e}{-26}$.

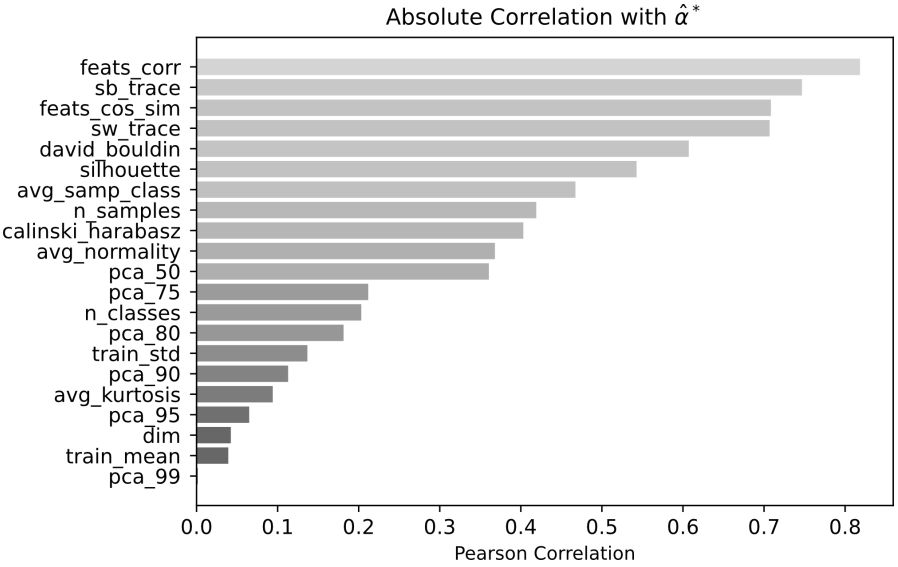
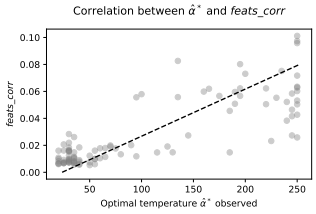
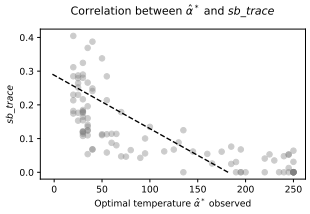
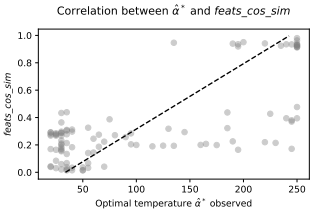


FIGURE 4.5 – Valeur absolue de la corrélation de Pearson entre chaque statistique de l'ensemble des données et la température empirique optimale.

Améliorer les résultats Pour améliorer le choix de la valeur de température α par rapport à celle qui pourrait être déterminée avec seulement la mesure $feats\_corr$, nous proposons d'apprendre une régression linéaire à partir de nos statistiques et des températures déterminées empiriquement, avec une stratégie de validation croisée. Cette dernière omet tous les ensembles des représentations statistiques $\mathcal{S}$ d'un jeu de données (par exemple MNIST) pendant la phase d'apprentissage afin de les utiliser pour la validation. Le tableau 4.4 présente une visualisation de la régression linéaire issue des trois meilleures statistiques trouvées (*feats_corr*, *sb_trace*, *feats_cos_sim*) avec la température optimale empirique α.

3. La significativité statistique exprime la fiabilité du résultat d'étude portant sur des échantillons de populations.

TABLE 4.4 – Corrélation entre les trois meilleures statistiques trouvées et la température optimale empirique α pour tous jeux de données.

feats_corr	*sb_trace*	*feats_cos_sim*
		

Ensuite, nous avons répété cette procédure sur un sous-ensemble des quatre statistiques les plus corrélées. Puis, nous avons ajusté une régression linéaire sur tous les points. La figure 4.6 présente la corrélation entre notre température prédite et la température optimale empirique.

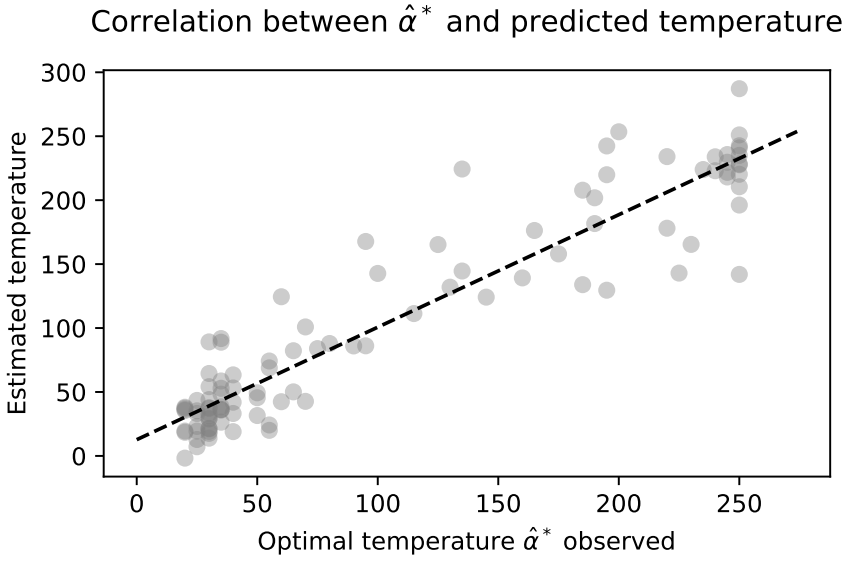


FIGURE 4.6 – Corrélation entre la température optimale observée et la température prédite. La température prédite correspond ici à une combinaison linéaire des 4 meilleures statistiques des ensembles de données. Nous avons mesuré une corrélation de Pearson de 0.9379.

Les scores de ces trois expérimentations sont présentés dans le tableau 4.5. Les résultats obtenus sont prometteurs et statistiquement significatifs avec une corrélation de Pearson de 0.9563 et une valeur-p de $0.014 < 0.05$.

TABLE 4.5 – Corrélations observées entre la variable la plus corrélée, une combinaison linéaire de 4 variables et une combinaison linéaire de toutes les variables avec la température empirique optimale. La médiane, la corrélation moyenne et la valeur p moyenne sont calculées en validation croisée.

Method	Median Pearson's r	Avg. Pearson's r ($\pm$ std)	p-value
Best Stat	0.9262	0.851 ($\pm$ 0.159)	0.031
4-Best Stats	0.9265	0.865 ($\pm$ 0.131)	0.018
All Stats	**0.9563**	0.884 ($\pm$ 0.124)	0.014

Bi-modalité Pour aller plus loin, après avoir proposé une heuristique basée sur une combinaison linéaire de plusieurs statistiques, nous avons cherché à comprendre le problème de bi-modalité présente dans les figures du tableau 4.4 et la figure 4.6, où deux groupements de points sont distincts à chaque extrémité de la droite linéaire. Nous formulons l'hypothèse que la température α doit dépendre de l'initialisation des extracteurs de caractéristiques de telle sorte que les statistiques extraites depuis les représentations issues d'un extracteur de caractéristiques pré-entrainé ne soit pas dépendante de la température dans la couche *softmax* pour obtenir une bonne performance ($\alpha = 1$). La figure 4.7 permet d'illustrer cette hypothèse dans laquelle une température élevée est nécessaire pour obtenir une bonne performance à partir de statistiques issues d'extracteurs de caractéristiques initialisés de manière aléatoire, tandis qu'à l'inverse, la température a un impact limité sur les statistiques issues sur des modèles pré-entrainés.

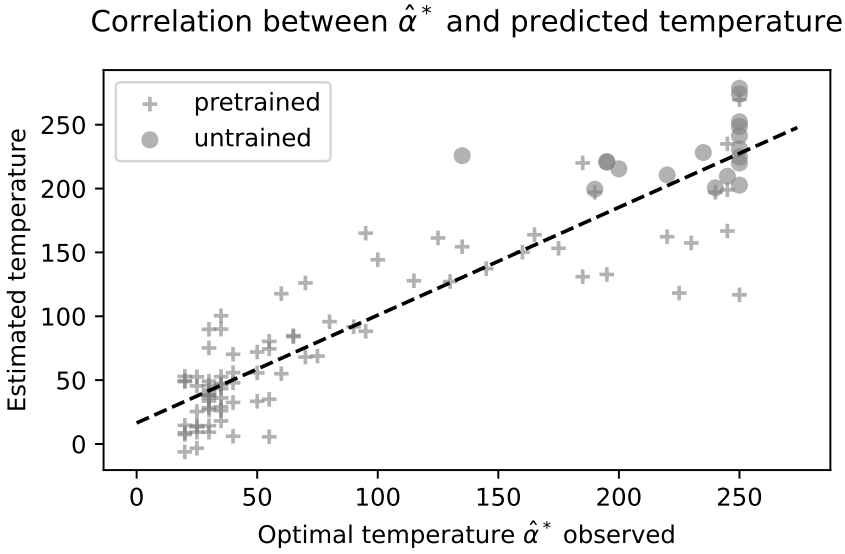


FIGURE 4.7 – Présence de deux blocs distincts malgré une forte corrélation de Pearson.

En analysant la corrélation de la température avec la variable binaire indiquant l'utilisation d'un extracteur de caractéristiques pré-entraîné, nous trouvons une valeur plus faible que notre prédiction de la température. Cela suggère que la corrélation des caractéristiques peut être affectée par le pré-entraînement de l'extracteur de caractéristiques,

mais que des informations plus fines que cette variable binaire sont nécessaires pour identifier la meilleure valeur de l'hyperparamètre à utiliser.

4.4 Discussion

Dans ce chapitre, nous avons montré l'importance de l'hyperparamètre de température permettant d'améliorer un classificateur linéaire. Nous avons montré que la fonction de coût de l'entropie croisée (CE) peut souffrir d'une température élevée si elle n'est pas correctement remise à l'échelle. Après avoir corrigé la CE, nous avons proposé d'étudier les corrélations entre la température empirique optimale observée sur de nombreux jeux de données, comprenant une large gamme de classes et de dimensions, et les statistiques calculées sur les représentations de ces données. Ainsi, nous avons révélé que certaines heuristiques (telles que la dimensionnalité des représentations) avaient peu de corrélation avec la température optimale, alors qu'une combinaison linéaire entre les caractéristiques montrait une forte corrélation.

L'ensemble des statistiques possibles n'est pas seulement limité à celles présentées ici, mais nous avons montré comment une simple chaîne de traitement permet de trouver des corrélations entre un paramètre du modèle et des représentations universelles basés ici sur des statistiques et des scores.

Plus généralement, nous n'avons pas réussi à dégager une heuristique générale permettant de maximiser le score d'apprentissage en raison de la présence de la bi-modalité présentée précédemment. Il serait possible d'envisager un système d'ordonnanceur modifiant l'hyperparamètre de température α durant l'apprentissage, mais cela ne répond plus à notre but initial.

La question que nous nous posons maintenant est : « Est-ce que le paramètre de température influe réellement sur l'*accuracy* ou inversement, est-ce que ce n'est pas le fait d'avoir une bonne *accuracy* dès le début grâce à un modèle pré-entrainé qui nous dicte le choix d'une température plus faible ». Pour répondre à cela, nous proposons d'utiliser cette étude comme socle pour le chapitre suivant afin de proposer une chaîne de traitement plus complexe permettant de guider nos choix sur une heuristique à proposer pour une tâche non plus basée sur un hyperparamètre mais sur la prédiction de la justesse – *accuracy* – du modèle.

Ces travaux ont fait l'objet d'une publication d'un article à la conférence IEEE ICIP 2022 [5].

Références

[1] Eric B. Baum et Frank Wilczek. “Supervised Learning of Probability Distributions by Neural Networks”. In : *NIPS, Procs.* Sous la direction de Dana Z. Anderson. 1987, pages 52-61.

[2] Massimo Caccia et al. “Language GANs Falling Short”. In : *ICLR, Procs.* 2020.

[3] T. Caliński et J Harabasz. “A dendrite method for cluster analysis”. In : *Communications in Statistics* 3.1 (1974), pages 1-27.

[4] Qiong Cao et al. “Vggface2 : A dataset for recognising faces across pose and age”. In : *FG*. 2018, pages 67-74.

[5] B. Chamand et al. “Fine-tune your classifier : Finding correlations with temperature”. In : *Proceedings of the IEEE International Conference on Image Processing – ICIP 2022*. Bordeaux (France) : IEEE Computer Society, 2022, pages xx-xx.

[6] Ting Chen et al. “A simple framework for contrastive learning of visual representations”. In : *ICML, Procs.* 2020, pages 1597-1607.

[7] M. Cimpoi et al. “Describing Textures in the Wild”. In : *CVPR, Procs.* 2014.

[8] David L. Davies et Donald W. Bouldin. “A Cluster Separation Measure”. In : *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-1.2 (1979), pages 224-227.

[9] Jia Deng et al. “ImageNet : A large-scale hierarchical image database”. In : *CVPR, Procs.* 2009, pages 248-255.

[10] Ronald A. Fisher. *Statistical methods for research workers*. Edinburgh Oliver & Boyd, 1925.

[11] Ronald A. Fisher. “The Use of Multiple Measurements in Taxonomic Problems”. In : *Annals of Eugenics* 7.2 (1936), pages 179-188.

[12] Greg Griffin, Alex Holub et Pietro Perona. “Caltech256 Image Dataset”. In : (2006).

[13] Chuan Guo et al. “On calibration of modern neural networks”. In : *ICML, Procs.* 2017, pages 1321-1330.

[14] Kaiming He et al. “Deep residual learning for image recognition”. In : *CVPR, Procs.* 2016, pages 770-778.

[15] Yu-Lin He et al. “Determining the optimal temperature parameter for Softmax function in reinforcement learning”. In : *Applied Soft Computing* 70 (2018), pages 80-85.

[16] Martin N. Hebart et al. “THINGS : A database of 1, 854 object concepts and more than 26, 000 naturalistic object images”. In : *PLOS One* 14.10 (2019), e0223792.

[17] Dan Hendrycks et al. “The Many Faces of Robustness : A Critical Analysis of Out-of-Distribution Generalization”. In : *CoRR* abs/2006.16241 (2020).

[18] Geoffrey E. Hinton, Oriol Vinyals et Jeffrey Dean. “Distilling the Knowledge in a Neural Network”. In : *CoRR* abs/1503.02531 (2015).

[19] Zhiting Hu et al. “Toward Controlled Generation of Text”. In : *ICML, Procs.* 2017, 1587–1596.

[20] Prannay Khosla et al. “Supervised Contrastive Learning”. In : *NIPS, Procs.* 2020.

[21] Diederik P. Kingma et Jimmy Ba. “Adam : A Method for Stochastic Optimization”. In : *ICLR, Procs.* 2015.

[22] Alex Krizhevsky et Geoffrey Hinton. *Learning multiple layers of features from tiny images*. Rapport technique. University of Toronto, 2009.

[23] Alex Krizhevsky, Ilya Sutskever et Geoffrey E Hinton. “Imagenet classification with deep convolutional neural networks”. In : *NIPS, Procs.* 2012.

[24] Ya Le et Xuan S. Yang. “Tiny ImageNet Visual Recognition Challenge”. In : 2015.

[25] Yann LeCun, Corinna Cortes et Christopher J.C. Burges. *MNIST handwritten digit database*. `http://yann.lecun.com/exdb/mnist`. 2010.

[26] Xiang Li et al. “FSS-1000 : A 1000-Class Dataset for Few-Shot Segmentation”. In : *CVPR, Procs.* 2020, pages 2866-2875.

[27] Weiyang Liu et al. “Large-Margin Softmax Loss for Convolutional Neural Networks”. In : *ICML, Procs.* 2016, pages 507-516.

[28] Yu Liu, Hongyang Li et Xiaogang Wang. “Rethinking Feature Discrimination and Polymerization for Large-scale Recognition”. In : *CoRR* abs/1710.00870 (2017).

[29] Ana C Lorena et al. “How Complex is your classification problem ? A survey on measuring classification complexity”. In : *ACM Computing Surveys* 52.5 (2019), pages 1-34.

[30] Xuezhe Ma et al. “Softmax Q-Distribution Estimation for Structured Prediction : A Theoretical Interpretation for RAML”. In : *CoRR* abs/1705.07136 (2017).

[31] Karl Pearson. “VII. Note on regression and inheritance in the case of two parents”. In : *Proceedings of the Royal Society of London* 58 (1895), pages 240-242.

[32] *Pins Face Recognition — kaggle.com*. `https://www.kaggle.com/hereisburak/pins-face-recognition`. [Accessed 13-Dec-2021].

[33] Alec Radford et al. “Learning transferable visual models from natural language supervision”. In : *ICML, Procs.* 2021, pages 8748-8763.

[34] Rajeev Ranjan, Carlos Domingo Castillo et Rama Chellappa. “L2-constrained Softmax Loss for Discriminative Face Verification”. In : *CoRR* abs/1703.09507 (2017).

[35] Olivier Risser-Maroix, Camille Kurtz et Nicolas Loménie. “Learning an Adaptation Function to Assess Image Visual Similarities”. In : *ICIP, Procs.* 2021, pages 2498-2502.

[36] Peter J. Rousseeuw. “Silhouettes : A graphical aid to the interpretation and validation of cluster analysis”. In : *Journal of Computational and Applied Mathematics* 20 (1987), pages 53-65.

[37] Florian Schroff, Dmitry Kalenichenko et James Philbin. “Facenet : A unified embedding for face recognition and clustering”. In : *CVPR, Procs.* 2015, pages 815-823.

[38] S. S. Shapiro et M. B. Wilk. “An Analysis of Variance Test for Normality (Complete Samples)”. In : *Biometrika* 52.3/4 (déc. 1965), page 591.

[39] Ashish Vaswani et al. “Attention is All you Need”. In : *NIPS, Procs.* 2017, pages 5998-6008.

[40] Pei-Hsin Wang et al. “Contextual Temperature for Language Modeling”. In : *CoRR* abs/2012.13575 (2020).

[41] P. Welinder et al. *Caltech-UCSD Birds 200*. Rapport technique CNS-TR-2010-001. California Institute of Technology, 2010.

[42] Qi Wu, Hongping Cai et Peter Hall. “Learning graphs to model visual objects across different depictive styles”. In : *ECCV, Procs.* 2014, pages 313-328.

[43] Zhirong Wu, Alexei A. Efros et Stella X. Yu. “Improving Generalization via Scalable Neighborhood Component Analysis”. In : *ECCV, Procs.* 2018, pages 712-728.

[44] Andrew Zhai et Hao-Yu Wu. “Classification is a Strong Baseline for Deep Metric Learning”. In : *BMVC, Procs.* 2019, page 91.

[45] Xu Zhang et al. “Heated-Up Softmax Embedding”. In : *CoRR* abs/1809.04157 (2018).

[46] Zhilu Zhang et Mert R. Sabuncu. “Generalized Cross Entropy Loss for Training Deep Neural Networks with Noisy Labels”. In : *NIPS, Procs.* 2018, pages 8792-8802.

Chapitre 5

Qu'apprenons-nous en prédisant l'*accuracy* ?

Dans le chapitre précédent, nous avons recherché une méthode pouvant trouver une formule afin d'initialiser certains hyperparamètres permettant l'amélioration de la justesse – *accuracy* – d'une tâche de classification de représentations. Nous avons constaté que définir une heuristique pour un hyperparamètre suivant notre méthodologie pouvait initialement dépendre de la capacité du modèle à discriminer correctement les données d'entrée.

La tâche de classification constitue l'un des domaines les plus explorés en matière d'apprentissage automatique, à tel point que plusieurs fonctions de perte ont été conçues en vue de maximiser cet objectif. Ces fonctions de perte sont généralement conçues en s'appuyant sur des intuitions ou de travaux théoriques qui sont ensuite validés par l'expérimentation. En revanche, nous proposons ici d'aborder le problème de la création d'une fonction de perte en fonction des connaissances acquises à partir des expériences. Pour cela, nous proposons donc d'étudier directement la justesse – *accuracy* – d'un modèle afin d'en extraire des connaissances qui nous guideront dans le choix de meilleures heuristiques. Cette approche basée sur les données est semblable à celle utilisée en physique pour découvrir des lois générales à partir d'expériences.

Dans ce chapitre, nous cherchons donc à répondre directement à la question suivante : « *Quelles connaissances pouvons-nous acquérir à travers une tâche telle que la prédiction de la justesse – accuracy – d'un modèle ?* ».

Pour répondre à cette question, nous proposons de reprendre notre chaîne de traitement proposée dans le chapitre précédent en y ajoutant une méthode de régression symbolique afin de déterminer automatiquement une expression mathématique qui est fortement corrélée avec la justesse – *accuracy* – d'un classifieur basé sur une fonction *softmax* linéaire. La formule nouvellement découverte a été obtenue à partir de l'analyse de plus de 260 ensembles de données et présente une corrélation de Pearson de 0,96 et un r^2 de 0,93 avec la précision prédite. Cette formule est hautement explicable et elle confirme les conclusions de divers articles antérieurs sur la conception d'une fonction de perte.

Sommaire

5.1 Introduction

La plupart des travaux en apprentissage machine (ML) sont réalisés en construisant et en évaluant des systèmes à partir d'intuitions. Une approche différente consisterait à acquérir des connaissances à partir de l'expérimentation, de la même manière que les physiciens cherchent à modéliser les lois analytiques qui sous-tendent les phénomènes physiques dans la nature à partir d'observations. Avec les progrès de l'intelligence artificielle, une nouvelle tendance à l'automatisation et à l'aide à la décision à l'aide d'outils de ML se dessine. Certains chercheurs ont commencé à l'utiliser en mathématiques [10] et en physique [13, 50]. En apprentissage automatique, le cadre le plus similaire serait le méta-apprentissage, dans lequel un modèle acquiert de l'expérience au cours de plusieurs épisodes d'apprentissage afin d'améliorer ses performances d'apprentissage futures sans intervention humaine. Bien que ce paradigme ait fait l'objet d'applications réussies sur diverses tâches [22] telles que l'optimisation d'hyperparamètres, la recherche d'architecture neuronale (NAS), etc., les solutions trouvées ne sont généralement pas interprétables.

5.2 Travaux connexes

La tâche de prédiction de la justesse – *accuracy* – peut sembler étrange à première vue. Pourtant, sa résolution a de multiples applications telles que : l'accélération du NAS en étant capable de prédire la performance d'une architecture aléatoire sans avoir à l'entraîner [27, 58] ; l'évaluation de la précision d'un classifieur sur un ensemble de données non étiquetées [11] ; ou la mesure de la difficulté à traiter un ensemble de

données [9, 49]. Dans la littérature, la justesse – *accuracy* – a été estimée à partir des poids du réseau [60], de l'architecture du réseau [58] ou, comme dans notre cas, de statistiques calculées sur le jeu de données [9, 11].

À partir des statistiques La question de trouver de telles statistiques est toujours considérée comme une question ouverte [11]. Les travaux précédents s'appuient principalement sur des modèles de régression tels que les réseaux de neurones ou les forêts aléatoires, ce qui rend leurs solutions non explicables [11, 60]. Plus proche de notre travail, la tâche de classification de texte a déjà été étudiée avec des caractéristiques telles que les *n-grams* [9]. Les auteurs de ces travaux ont proposé de trouver une corrélation entre des statistiques et le score F1. Cependant, leur travail est limité par le choix des caractéristiques, qui ne le rend utilisable que pour des ensembles de données textuelles. La prédiction du score F1 ne peut prendre que la forme d'une somme d'un sous-ensemble des statistiques proposées. Dans un cadre plus général, les statistiques pour décrire les ensembles de données ont été explorées [21, 40, 42]. En outre, une analyse de ces mesures a également suggéré que la relation entre ces statistiques et la difficulté à traiter un ensemble de données est complexe et nécessiterait une combinaison non linéaire de ces variables [21]. Cependant, les auteurs n'ont étudié chaque variable que de manière indépendante sans tester la généralisation.

À partir d'un ensemble de modèles Dans un autre travail, [4] propose d'estimer la justesse – *accuracy* – de plusieurs classifieurs afin de sélectionner le plus adapté à un jeu de données particulier. Dans leur analyse, les auteurs n'ont étudié qu'un seul modèle linéaire : l'analyse discriminante linéaire (LDA). Cependant, l'état de l'art actuel utilise des modèles basés sur la *softmax* nécessitant des approches de descente de gradient. Afin d'extraire des connaissances d'un métacorpus de données tabulaires, ils ont utilisé Cubist, une bibliothèque produisant des modèles sous forme de jeux de règles. Cependant, en étant nombreuses et formulées avec des valeurs codées en dur, les règles ainsi générées sont complexes et difficiles à généraliser.

Nous nous concentrons sur la production de descripteurs issus de jeux de données extrêmement diversifiés. Ainsi, le nombre de classes différentes concernées dans nos expérimentations atteint 1824, alors que ce nombre atteint un maximum de 115 dans [9]. Dans ce travail, nous choisissons de décrire nos jeux de données avec 19 statistiques non spécifiques à un domaine particulier (ex. texte). De plus, nous comparons la solution trouvée par notre chaîne de traitement avec les solutions trouvées avec les précédentes approches proposées [4, 9].

Explicabilité de la solution trouvée Dans les sciences expérimentales, telles que la physique, sociologie ou comme dans notre cas : l'apprentissage automatique, nous cherchons à comprendre les relations entre les variables d'un système donné. Trouver une fonction qui explique les relations cachées dans les données sans avoir aucune connaissance préalable de la forme de la fonction est le but que la régression symbolique (SR) tente de résoudre. Comme la régression symbolique est un problème NP-complet [54], des méthodes évolutionnistes ont été développées pour obtenir des solutions approximatives [1, 30, 31, 41]. La tâche de régression symbolique a récemment connu un regain en popularité, et de nouvelles approches combinant la programmation génétique classique

et l'apprentissage par renforcement profond moderne ont vu le jour [35, 43, 44, 53, 55]. En raison de l'absence de normes d'évaluation uniformes, robustes et transparentes, il est difficile de choisir un cadre de RS plutôt qu'un autre. Récemment, une étude comparant 14 cadres de RS et 7 algorithmes ML classiques sur des ensembles de données synthétiques et des problèmes du monde réel a été proposée [34]. Les auteurs ont observé que les techniques basées sur l'apprentissage profond et les algorithmes génétiques ont des performances similaires dans la tâche de retrouver des équations connues.

Dans ce chapitre, nous présentons nos travaux qui aboutissent à l'expression d'une formule générale en étudiant plus de 260 jeux de données de représentations – *embeddings* – aux caractéristiques très différentes (taille, dimension, nombre de classes, etc.). Nous proposons de projeter ces jeux de données dans une représentation partagée commune en les décrivant par le moyen d'un ensemble de statistiques. À partir de ces représentations statistiques, nous avons trouvé une formule capable de prédire la performance de classification future d'un classifieur linéaire avec une forte corrélation de Pearson et un score r^2 élevé.

5.3 Méthodologie

Dans cette partie, nous présentons l'approche proposée pour trouver une formule qui soit à la fois corrélée avec la justesse – *accuracy* – d'un modèle, et à la fois explicable. Comme l'illustre la figure 5.1, l'approche réutilise notre méthode générale présentée dans le chapitre 4 composée de deux parties principales : (1) la création d'un méta-corpus $\mathcal{M}$ à partir de la combinaison de différents jeux de données et extracteurs de caractéristiques, de la représentation dans un même espace et la création de la vérité terrain ; (2) la découverte d'une heuristique explicable obtenue par régression symbolique. Nous détaillons chaque sous-composante dans les paragraphes suivants.

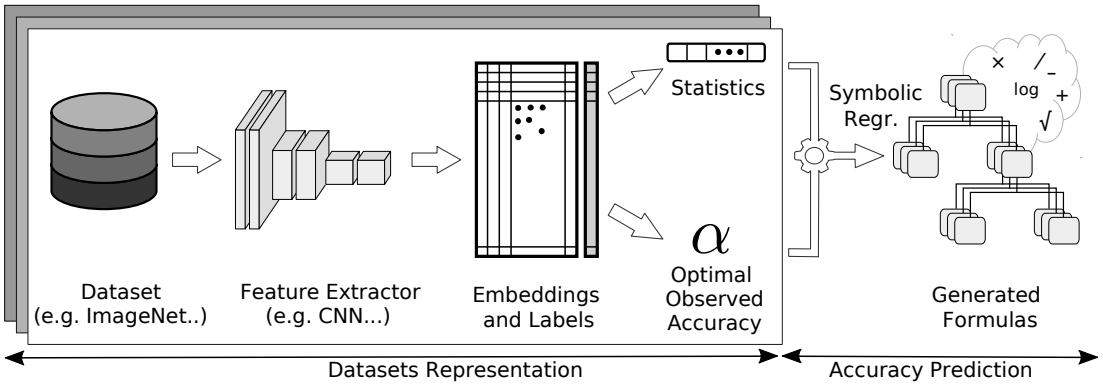


FIGURE 5.1 – Chaine de traitements proposée pour trouver la justesse – *accuracy* – optimale à partir des statistiques issues des données.

5.3.1 Jeux de données, extracteurs de caractéristiques

Afin de trouver une loi générale couvrant un large éventail de cas pour une tâche de classification, nous avons sélectionné, de la même manière que dans le chapitre précédent, 12 jeux de données ainsi que 22 extracteurs de caractéristiques.

Ensemble de données Le tableau 5.1 présente les jeux de données sélectionnés pour l'expérimentation. Le choix de ces ensembles de données reste identique à l'expérimentation du chapitre précédent présenté dans la table 4.1.

TABLE 5.1 – Liste des jeux de données utilisés triés par leur nombre de classes associées.

Dataset	Nb. classes	Description
CIFAR10 [32]	10	Images naturelles
MNIST [37]	10	Images de chiffres
DTD [8]	47	Images de textures (tâcheté, plissé, etc.)
PhotoArt [59]	50	Images issues de photo ou d'œuvres d'art
CIFAR100 [32]	100	Images naturelles
105-PinterestFaces [45]	105	Images de visages de différentes célébrités
CUB200 [57]	200	Images d'oiseaux
ImageNet-R [20]	200	Images diverses (sculptures, broderies, etc.)
Caltech256 [17]	256	Images naturelles
FSS1000 [39]	1000	Images naturelles
ImageNetMini [36]	1000	Images naturelles
THINGS [19]	1854	Images naturelles

Extracteurs de caractéristiques Concernant les extracteurs de caractéristiques, nous avons ajouté de nouveaux modèles avec différents pré-entraînements pour augmenter la diversité des représentations, passant ainsi de 9 extracteurs de caractéristiques à 22. Les modèles sélectionnés avec leur type d'initialisation sont présentés dans le tableau 5.2.

TABLE 5.2 – Listes des extracteurs de caractéristiques et leur type d'initialisation.

Architecture	ImageNet	VGGFaces2	N/A	Aléatoire
AlexNet [33]	✓			
DenseNet-{169, 201} [25]	✓			✓
CLIP-{RN50, ViT16b, ViT32b} [46]			✓	
FaceNet [51]		✓		
MobileNetV2 [48]	✓			✓
MobileNetV3-{Small, Large} [23]	✓			✓
ResNet-{18, 50, 101} [18]	✓			
ResNet-{34, 152} [18]				✓
SqueezeNet [26]	✓			✓

Couverture des représentations extraites L'utilisation de ces multiples jeux de données et extracteurs de caractéristiques va nous permettre de couvrir un grand nombre de dimensions et de niveaux de difficulté. La figure 5.2 montre cette couverture en termes

de nombres de dimensions et de nombres de classes, ainsi que la meilleure justesse – *accuracy* – obtenue exprimée proportionnellement à la taille des cercles.

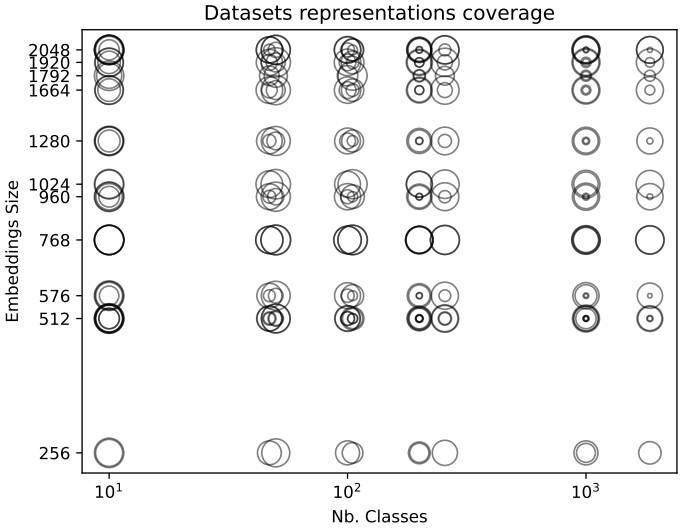


FIGURE 5.2 – Couverture de toutes les combinaisons d'extracteurs de caractéristiques avec tous les jeux de données utilisés. La taille des cercles représente la meilleure justesse – *accuracy* – obtenue pour chaque ensemble de données.

5.3.2 Création du métacorpus

De la même manière que pour le chapitre précédent, en raison de dimensionnalités différentes entre les représentations – *embeddings* – extraites à partir de la combinaison de tous les ensembles de jeux de données par tous les extracteurs de caractéristiques, nous proposons de décrire chacune de ces représentations sous la forme d'un vecteur de caractéristiques s dans un espace de représentation commun $\mathcal{S}$. Ce vecteur de caractéristiques s est de taille fixe. Il est constitué des caractéristiques détaillées dans la suite. Vient ensuite la création des annotations. Celles-ci seront la valeur de justesse – *accuracy* – optimale de chaque représentation – *embeddings* . Nous obtenons ainsi un métacorpus $\mathcal{M}$ à partir de plus de 260 vecteurs de représentations S.

Choix des caractéristiques Inspirés par [5, 21, 40, 42], nous avons sélectionné diverses caractéristiques s_i qui sont présentées dans le tableau 5.3. Conformément au protocole expérimental du chapitre précédent, nous avons supprimé le calcul des scores sur le vecteur sortant de l'extracteur de caractéristiques qui n'apportait aucune information sur notre recherche d'une heuristique de la température. De plus, nous avons ajouté de nouvelles statistiques comme la corrélation entre les représentations moyennes de chaque classe et la similarité cosinus moyenne entre les moyennes des représentations de chaque classe.

Création des annotations Après avoir extrait l'ensemble des représentations – *embeddings* – de nos jeux de données avec les différents extracteurs de caractéristiques et après une normalisation ℓ_2, nous proposons de trouver la meilleure justesse – *accuracy* –

TABLE 5.3 – Listes des statistiques extraites de nos représentations pour la construction du métacorpus $\mathcal{M}$.

Notation	**Description**
dim	Dimensionnalité des représentations
n_classes	Nombre de classes de sortie
n_train	Nombre d'exemples dans l'ensemble d'entrainement
n_test	Nombre d'exemples dans l'ensemble de test
sb_trace	Trace de la matrice moyenne de toutes les matrices de covariance intra-classe
sw_trace	Trace de la matrice moyenne de toutes les matrices de covariance inter-classes
st_trace	Somme de *sb_trace* et *sw_trace*
feats_corr	Erreur quadratique moyenne (MSE) entre la matrice de corrélation des caractéristiques et la matrice identité
feats_cos_sim	Similarité moyenne en cosinus entre chaque paire de dimensions
pca_%	Pourcentage de dimensions à retenir pour une variance expliquée donnée (de manière analogue à la PCA) de 50, 75, 99
train_mean	Moyenne de toutes les valeurs des représentations
train_std	Écart type de toutes les valeurs des représentations
avg_kurtosis	Moyenne du kurtosis calculé sur chaque dimension
std_kurtosis	Variance du kurtosis calculé sur chaque dimension
avg_normality	Valeur moyenne de Shapiro-Wilk [52]
prototypes_corr	Corrélation entre les moyennes des représentations de chaque classe
prototypes_cos_sim	Similarité moyenne en cosinus entre les moyennes des représentations de chaque classe

atteignable par un classifieur multiclasses linéaire avec une couche de *softmax* en sortie pour chaque cas. La procédure est analogue à celle employée dans le chapitre précédent pour trouver la température optimale empiriquement sur les représentations – *embeddings* . Pour ce faire, nous divisons chaque ensemble de données de représentations en un ensemble de données d'apprentissage et d'évaluation, respectivement $3/4$ et $1/4$, excepté pour les jeux de données dont le nombre d'images par classe est faible tels que THINGS où une proportion de $2/3$ et $1/3$ a été utilisée afin de garantir un ensemble de données d'apprentissage d'au moins 10 images par classe. Nous avons ensuite entrainé le modèle pendant 1000 époques – *epochs* – avec une taille de lot – *batch size* – arbitraire de 2048. En monitorant l'évolution de la justesse – *accuracy* – sur l'ensemble des données de test, nous sommes en mesure d'observer la meilleure justesse atteinte α que nous considérerons comme une bonne approximation de la meilleure justesse atteignable α^* avec l'optimiseur Adam [29].

Création de notre métacorpus En couplant chaque représentation statistique $s_i \in \mathcal{S}$ avec sa vérité terrain trouvée de manière empirique $\alpha_i \in \mathcal{A}$, nous obtenons un métacorpus $\mathcal{M} = \{(s_i, \alpha_i)\}$ de plus de 260 données. Nous avons ensuite divisé ce nouveau jeu de données créé en deux sous corpus, un corpus d'entrainement et un de test avec un ratio de 75% et 25%.

Étude de la corrélation Avant d'aller plus loin, intéressons-nous à la corrélation des statistiques extraites avec la justesse – *accuracy* – de la même manière que dans le chapitre précédent avec la température. Nous pouvons ainsi observer dans la figure 5.3 que certaines statistiques ont peu d'influence sur le score de la justesse cherché comme le nombre de classes *n_classes* ou encore la dimension des représentations *dim*. Toutefois, notre formule issue de la programmation génétique (GP) permet d'augmenter considérablement cette corrélation grâce à une combinaison non-linéaire de cinq de ces statistiques que nous allons détailler dans la section suivante.

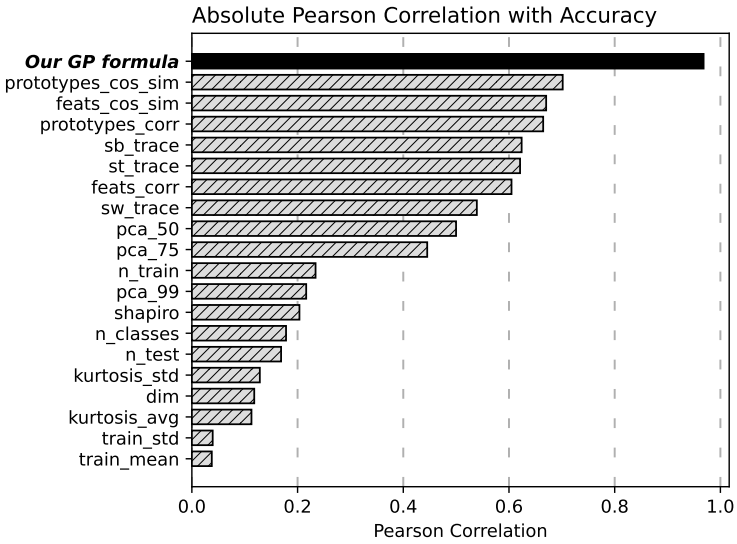


FIGURE 5.3 – Corrélation de Pearson absolue entre chaque statistique et la justesse – *accuracy* – observée.

5.3.3 Régression symbolique

Le but de la régression symbolique (SR) est de trouver des relations intrinsèques, généralement non linéaires, entre des variables afin de décrire un phénomène donné. Pour cela, nous recherchons une fonction de prédiction $p : \mathbb{R}^S \rightarrow \mathbb{R}$ à partir de notre métacorpus $\mathcal{M}$, avec S le nombre de caractéristiques statistiques. Comme mentionné au début du chapitre, différentes approches ont été développées pour la SR. La Cava *et al.* [34] ont proposé une étude de comparaison des méthodes existantes de SR. Il en ressort que DSO (Deep Symbolic Optimization) [44], une approche basée sur l'apprentissage profond, et *gplearn*, un environnement de développement en programmation génétique (GP) dédié à l'apprentissage, font partie des cinq meilleures méthodes proposant un système clé en main de type boite noire pouvant réaliser cette tâche de SR.

Deep Symbolic Optimization (DSO) Le DSO est un environnement de développement – *framework* – dédié d'apprentissage profond spécialisé dans les tâches d'optimisation symbolique. L'utilisation de DSO sur notre tâche n'a pas permis d'obtenir des résultats significatifs. Lors de notre essai, les solutions trouvées sont sensiblement inférieures à celles obtenues avec *gplearn* et génèrent des formules plus complexes et un temps d'apprentissage plus long.

Programmation génétique (GP) Contrairement à la méthode précédente, l'utilisation de *gplearn* a permis de trouver des solutions plus compactes avec une meilleure efficacité en termes de rapidité d'exécution. Nous allons donc nous concentrer sur cette méthode pour trouver notre heuristique de la justesse – *accuracy* .

Dans cette méthode de régression symbolique (SR), la programmation génétique (GP) initialise une population $\mathcal{P}$ aléatoire d'expressions mathématiques qu'elle va modifier dans le but de trouver un individu p qui respecte davantage les critères définis dans la fonction d'évaluation – *fitness* – $\mathcal{F}$. Pour cela, elle utilise le principe de l'« évolution » grâce à des opérations telles que la « sélection », l'« enjambement » et la « mutation » qui va modifier la composition hiérarchique de fonctions primitives τ et de nœuds terminaux des individus p. Dans notre cas, l'ensemble des nœuds terminaux correspond aux statistiques s_i extraites et présentées précédemment, tandis que τ représente l'ensemble des fonctions primitives sélectionnées pour la recherche avec $\tau = \{\log, e, \sqrt{}, +, -, \times, \div\}$. Pour l'expérimentation, nous avons généré une population de 5000 individus pendant 20 étapes et testé les 3 fonctions d'évaluation – *fitness* – différentes présentées ci-après.

Fonction d'évaluation n°1 La première fonction d'évaluation correspond au coefficient de détermination, noté r^2 et défini dans la section 5.4, entre la formule prédite et le résultat attendu. Cette fonction n'a pas fourni de résultat concluant autant sur les ensembles d'entraînement que de test, car elle n'était jamais satisfaite pas l'algorithme de GP.

Fonction d'évaluation n°2 La seconde fonction d'évaluation correspond à la corrélation de Pearson r entre les justesses – *accuracy* attendues et celles prédites. Contrairement à la première fonction, la GP arrive plus facilement à optimiser celle-ci, mais pose un nouveau problème. L'heuristique trouvée sépare en deux blocs distincts les représentations statistiques issues de réseaux pré-entrainées d'un côté et des réseaux non

entraînées. Nous pouvons constater ce même problème en calculant la corrélation de Pearson entre certaines statistiques telles que *feat_cos_sim* et la justesse – *accuracy* – attendue, présenté sur la figure 5.4. En effet, la corrélation de Pearson entre la justesse – *accuracy* – attendue d'un modèle et la variable booléenne spécifiant si ce modèle est pré-entraîné ou non est déjà de 77.59%. Nous nous retrouvons donc dans la même situation que pour le chapitre précédent où nous avions identifié deux groupes distincts dû à l'initialisation des extracteurs de caractéristiques.

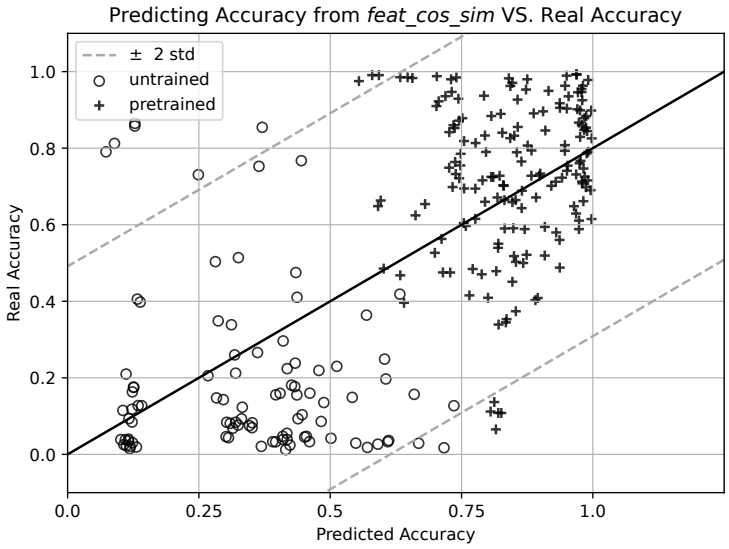


FIGURE 5.4 – Présence de deux blocs distincts malgré une forte corrélation de Pearson.

Fonction d'évaluation n°3 Pour corriger ce problème, nous avons défini une nouvelle fonction d'évaluation afin que les représentations statistiques extraites, issues d'extracteurs de caractéristiques pré-entrainés ou non, présentent une corrélation linéaire avec la justesse – *accuracy* –, et ce, de manière indépendante du mode d'initialisation. Pour un individu donné $p(\cdot)$, c'est-à-dire pour une formule de prédiction de GP, nous évaluons son score d'évaluation $\mathcal{F}$ comme suit :

$$\begin{aligned}\mathcal{F} = \min \Big[&\big|\text{pearsonr}\Big(p(\mathcal{S}_{pretrained}), \mathcal{A}_{pretrained}\Big)\big|, \\ &\big|\text{pearsonr}\Big(p(\mathcal{S}_{untrained}), \mathcal{A}_{untrained}\Big)\big| \Big]\end{aligned} \tag{5.1}$$

avec $\mathcal{S}_{subset}, \mathcal{A}_{subset}$ correspondant respectivement aux ensembles de représentations statistiques et aux justesses – *accuracy* – cibles du $subset \in \{pretrained, untrained\}$. La fonction d'évaluation – *fitness* – est donc évaluée sur les deux sous-ensembles pris séparément. La figure 5.5 montre que cette fonction d'évaluation tend à regrouper les valeurs prédites de la justesse – *accuracy* – issues d'extracteurs de caractéristiques pré-entrainés ou non et à resserrer leur distribution autour de la première diagonale par rapport à ce qui est observable dans la figure 5.4.

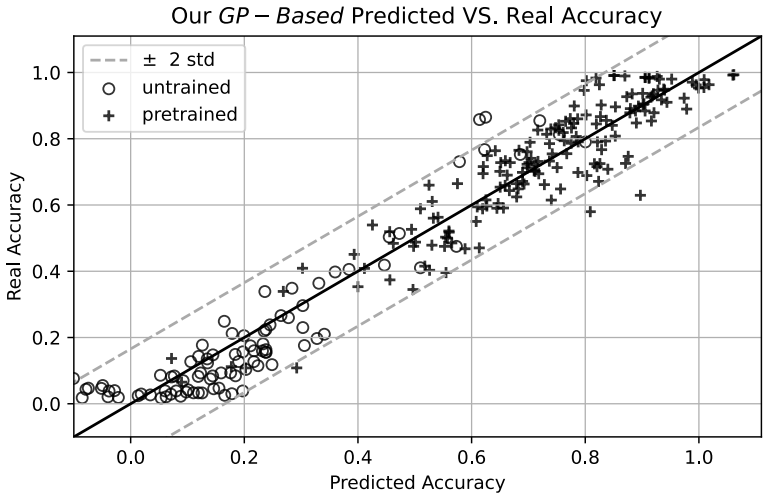


FIGURE 5.5 – Notre fonction d'évaluation permet de trouver une heuristique avec une forte relation linéaire entre notre justesse – *accuracy* – prédite et la justesse – *accuracy* – réelle.

Afin de trouver une formule générale qui ressort du lot, nous répétons l'expérience de GP avec cette dernière fonction d'évaluation 1000 fois. Comme $\mathcal{F}$ ne recherche que la corrélation entre des variables, une transformation linéaire de la valeur de sortie est apprise sur l'ensemble des données d'entraînement afin de prédire la justesse – *accuracy* – $\hat{\alpha} = a \cdot p(\cdot) + b$.

5.4 Étude expérimentale

Afin d'évaluer la corrélation entre la justesse – *accuracy* – prédite et la justesse – *accuracy* – empirique, nous utilisons le coefficient de Pearson et la valeur-p comme vu dans le chapitre précédent. En revanche, nous introduisons en plus le coefficient de détermination r^2.

Coefficient de détermination Appelé aussi score r^2, est une métrique qui va quantifier la performance d'un modèle de régression linéaire. Contrairement au coefficient de Pearson qui quantifie la relation entre deux variables, le score r^2 explique dans quelle mesure la variance d'une variable explique la variance de l'autre. En d'autres termes, c'est une mesure de la qualité de l'ajustement d'un modèle de régression aux données observées. Elle peut s'exprimer comme le carré de la corrélation de Pearson COR entre les valeurs prédites $\hat{X}$ et les valeurs mesurées X :

$$r^2 = \text{COR}(\hat{X} - X)^2 = 1 - \frac{SS_{res}}{SS_{tot}} \tag{5.2}$$

où SS_{res} est la somme des distances au carré entre les valeurs mesurées X et les valeurs prédites $\hat{X}$, et où SS_{tot} est la somme des distances au carré entre chaque observation X et la moyenne des valeurs prédites $\hat{X}$.

5.4.1 Base de référence

Comme base de comparaison pour évaluer la performance de notre modèle, nous utilisons d'autres méthodes de régression populaires, notamment : une régression linéaire – *linear regression* –, une régression par arbre de décision – *decision tree* – et une régression par forêt aléatoire – *random forest* . L'implémentation de ces méthodes est celle proposée dans *sklearn*, une bibliothèque Python destinée à l'apprentissage automatique, avec le même partitionnement des données d'entraînement et d'évaluation utilisé pour la découverte de notre formule issue de la GP. Pour initialiser ces méthodes, nous utilisons les hyperparamètres définis par défaut dans *sklearn*. Toutes les statistiques extraites sur les représentations – *embeddings* – sont utilisées simultanément et les performances sur l'ensemble de test sont rapportées dans le tableau 5.4. Notre formule présente une meilleure corrélation avec seulement 5 variables tandis que les autres modèles utilisent les 19 variables (toutes les valeurs-$p < 0.01$). Pour gagner en clarté, nous notons notre formule découverte par la GP comme *Genetic Programming Formula* (GPF).

TABLE 5.4 – Comparaison de notre formule avec d'autres méthodes de régression.

Method	Pearson's r	r^2
Linear Regression	0.9042	0.8011
Decision Tree Regressor	0.9472	0.8868
Random Forest Regressor (10 trees)	0.9643	0.9246
Our GP formula (GPF)	**0.9671**	**0.9319**

Avec un écart important du score r^2 entre la régression linéaire et notre formule GPF, nous pouvons conclure que la tâche de prédiction de la justesse – *accuracy* – nécessite une combinaison non linéaire de certaines variables. Les méthodes de régression non linéaires, telles que les arbres de décision et les forêts aléatoires, ont été sélectionnées en raison de leurs performances et de leurs réputations d'être facilement interprétables, contrairement à d'autres méthodes comme le perceptron multicouche (MLP). Toutefois, notre formule surpasse ces méthodes tout en étant plus explicable.

Le tableau 5.5 donne une comparaison entre la formule de GP présentant le meilleur score sur les données d'entraînement GPF_{train} (équation 5.3) et celle présentant le meilleur score sur les données d'évaluation GPF_{test} (équation 5.4). GPF_{train} correspond à la formule obtenue sur le corpus d'apprentissage et évaluée sur le corpus d'apprentissage tandis que GPF_{test} correspond à la formule obtenue sur le corpus d'apprentissage et évaluée sur le corpus de test. Ces formules sont définies comme suit :

$$GPF_{train} = \log\left(\frac{sb\_trace/feats\_cor}{sw\_trace \cdot \sqrt{n\_classes}}\right) \tag{5.3}$$

$$GPF_{test} = \log\left(\frac{sb\_trace/st\_trace}{\sqrt{n\_classes \cdot feats\_corr \cdot prototypes\_cos\_sim}}\right) \tag{5.4}$$

La faible différence de scores entre ces deux formules indique que la solution GPF_{test} ne présente pas de problème de sur-apprentissage.

TABLE 5.5 – Meilleure formule trouvée après évaluation sur le jeu de données d'entraînement et de test ; toutes les valeurs-$p < 0.01$

	Train		Test	
GP-Formula	Pearson's r	r^2	Pearson's r	r^2
GPF_{train}	**0.9684**	**0.9379**	0.9484	0.8946
GPF_{test}	0.9636	0.9285	**0.9659**	**0.9319**

5.4.2 Formule découverte par régression symbolique

Comme dit un peu plus haut dans le manuscrit, nous avons exécuté notre chaine de traitement de GP 1000 fois sur le même ensemble de données d'apprentissage et de test. Nous avons ensuite analysé toutes les formules proposées ainsi que leurs scores respectifs afin d'en extraire les meilleures classées en fonction du nombre de nœuds de calcul. Ces résultats sont affichés dans le tableau 5.6. Il est intéressant de noter que la solution ayant le meilleur score r^2 sur les données de test a été trouvée 6 fois et est composée de 6 nœuds, c'est-à-dire que l'on trouve un total de 6 opérations dans la formule.

TABLE 5.6 – Meilleures formules en fonction du nombre de nœuds ; avec toutes les p-values < 0.01.

Best test GP-formulas found	Nodes	Pearson's r	r^2
$\sqrt{sb\_trace/feats\_corr} - \log(n\_classes)$	4	0.9406	0.8812
$\log(prototypes\_cos\_sim \cdot n\_classes \cdot feats\_corr \cdot sw\_trace/sb\_trace)$	5	-0.9572	0.9157
$GPF_{test} = SEP_{test} - COR_{test}$ (Équation 5.4)	6	**0.9671**	**0.9319**

Complexité des formules trouvées Sur la figure 5.6, nous comparons les performances des formules trouvées en fonction de leur complexité. La complexité est définie ici comme étant le nombre de nœuds présents dans la formule générée par la GP. Les formules générées ont majoritairement entre 4 à 6 nœuds.

Réécriture de la meilleure formule Reprenons notre modèle GPF_{test} (équation 5.4) produisant le meilleur score de Pearson ainsi que le meilleur score r^2. Pour des raisons de lisibilité, nous posons $GPF = GPF_{test}$. Grâce aux propriétés du $\log$, nous pouvons réécrire GPF d'une manière plus lisible comme étant $GPF = SEP - COR$ avec :

$$SEP = \log\left(\frac{sb\_trace}{st\_trace}\right) \tag{5.5}$$

$$COR = \frac{1}{2}\log\left(n\_classes \cdot feats\_corr \cdot prototypes\_cos\_sim\right) \tag{5.6}$$

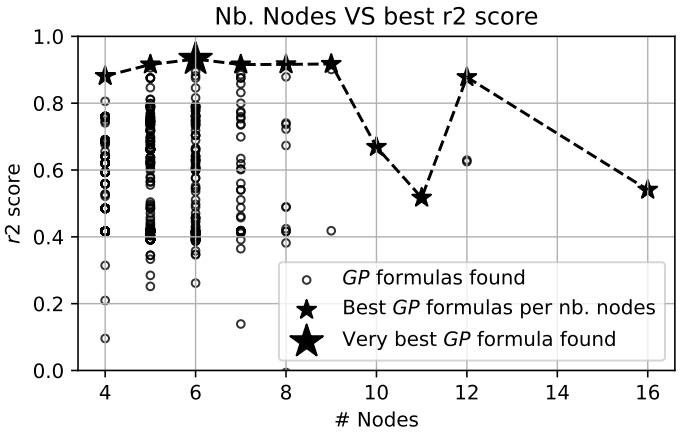


FIGURE 5.6 – Comparaison des performances des formules trouvées par notre GP en fonction de la complexité associée.

De cette manière, nous pouvons interpréter la formule GPS comme la composition d'un critère de séparabilité SEP d'une part et d'une information de corrélation COR d'autre part. Nous discuterons de cette interprétation plus en détail dans la section 5.5. De plus, ces deux parties de notre formule GPF sont complémentaires, car lorsque nous calculons la corrélation de Pearson sur chaque partie, nous obtenons un score de $0,65$ pour SEP et 0.87 pour COR, alors que la combinaison des deux atteint $0,96$.

Analyse des caractéristiques statistiques utilisées Nous avons aussi remarqué que toutes les meilleures formules issues de la GP ont une structure et des variables similaires. La figure 5.7 présente le nombre de fois qu'une des variables de notre vecteur de représentation statistique a été utilisée au cours des 1000 exécutions.

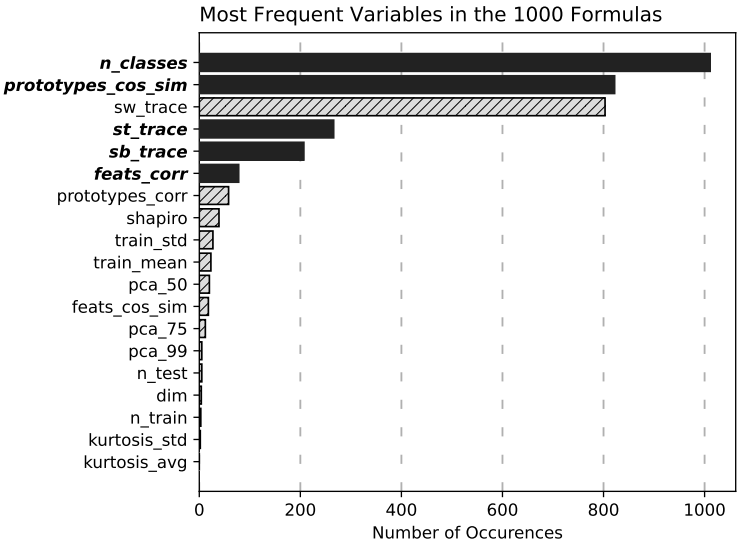


FIGURE 5.7 – Fréquence d'utilisation des variables dans les 1000 formules générées. Les barres noires indiquent les variables présentes dans la meilleure formule issue de la GP.

5.4.3 Étude ablative

En s'inspirant de plusieurs travaux en apprentissage profond, nous proposons ici d'étudier l'ablation de certaines composantes de notre formule GPF. L'étude ablative vise à déterminer l'importance relative des différentes composantes d'un modèle en les retirant successivement et en mesurant l'impact sur les performances du modèle afin de détecter les caractéristiques pertinentes.

Importance des 5 variables de notre formule Nous avons vu que notre formule GPF découverte par l'algorithme de SR utilise une combinaison de 5 variables statistiques. Dans un premier temps, nous proposons de mesurer l'influence de ces variables sur les résultats des 3 méthodes de régression de notre référence de base présentées dans le tableau 5.4. Le tableau 5.7 montre une légère différence des performances lorsque ces 5 variables statistiques sont utilisées. La méthode de régression à base de forêt aléatoire – *decision tree* – est la seule méthode qui gagne légèrement en performance en passant de 0.9472 à 0.9538. Toutes les valeurs-p de ces méthodes sont inférieures à 0.01, suggérant ainsi que ces 5 variables sélectionnées sont suffisantes pour établir une corrélation avec la justesse – *accuracy* . En revanche, la faible performance du modèle de régression linéaire qui passe de 0.9042 à 0.8796, nous laisse conclure qu'une transformation non linéaire de ces variables est nécessaire.

TABLE 5.7 – Influence du choix des 5 variables statistiques de notre formule GPF sur les résultats de nos méthodes de références (toutes les valeurs-$p < 0.01$).

Method	Pearson's r	r^2
Linear Regression	0.8796	0.7689
Decision Tree Regressor	0.9538	0.8937
Random Forest Regressor (10 trees)	0.9532	0.9057
Our GP formula (GPF)	**0.9671**	**0.9319**

Quelle est la composante la plus importante de notre formule ? Pour répondre à cette question, nous cherchons à trouver l'influence de chaque variable statistique sur le score de corrélation finale. Pour cela, nous fixons pour chaque expérience la variable étudiée par sa valeur moyenne. Ainsi, plus la performance décroit, plus l'importance de la variable est grande. Le tableau 5.8 montre que l'ablation de chacune des variables statistiques de notre formule GPF, diminue aussi bien le score de corrélation de Pearson r que le score r^2. Les 5 variables sont ainsi importantes, en particulier Sb_trace qui, pour rappel, représente la trace de la matrice moyenne de toutes les matrices de covariance intra-classe calculées sur les représentations – *embeddings* – issues d'un couple corpus de données et extracteur de caractéristiques. À noter que toutes les valeurs-p sont significatives à l'exception de la valeur-p associée à *Sb_trace* qui vaut 0.6882. Cela s'explique par le fait que toute corrélation entre notre formule GPF et la justesse est supprimée dû à la valeur fixée de la variable statistique.

TABLE 5.8 – Évaluation de la performance de notre formule pour une des variables statistiques fixée à sa valeur moyenne. Plus le score est mauvais, plus la variable statistique est importante pour notre formule.

Ablated Variable	Pearson's r	r^2
Sb_trace	$\mathbf{-0.0503}$	$\mathbf{-2.1806}$
n_classes	0.7918	0.4341
St_trace	0.8028	-1.2761
feats_corr	0.8530	0.5818
prototypes_cos_sim	0.9420	0.8764
No variable ablated	0.9671	0.9319

5.4.4 Reformulation de la combinaison linéaire

Nous avons montré la faible performance de la régression linéaire en utilisant nos 5 variables. Pourtant, en réappliquant les propriétés du $\log$ sur notre formule GPF, nous pouvons réduire ainsi celle-ci à une combinaison linéaire de nos 5 variables au format logarithmique avec, pour coefficients respectifs, $[1, -1, -\frac{1}{2}, -\frac{1}{2}, -\frac{1}{2}]$. Notre formule GPF de l'équation 5.4 peut être reformulée comme suit :

$$\begin{aligned} GPF = & \, log(Sb\_trace) \\ & - log(St\_trace) \\ & - 0.5 \, log(n\_classes) \\ & - 0.5 \, log(feats\_corr) \\ & - 0.5 \, log(prototypes\_corr) \end{aligned} \tag{5.7}$$

Cette découverte est particulièrement intéressante sachant qu'aucune contrainte structurelle de la formule n'a été imposée durant la phase de découverte de la SR. Cela soulève maintenant la question de la performance de cette même régression linéaire avec les 5 variables au format logarithmique et par extension avec la transformation logarithmique de toutes les variables statistiques.

Régression linéaire avec les 5 variables au format logarithmique En entrainant un modèle de régression linéaire sur ces 5 statistiques transformées au format logarithmique, nous observons à travers le tableau 5.9 que la corrélation de Pearson a augmenté de 6 points passant de 0.9042 à 0.9607 sur nos données d'apprentissage, mais présente des performances légèrement plus faible que notre formule GPF sur les données de test.

Cependant, en comparant la pondération apprise, le signe ainsi que la magnitude des poids associés à chacune des variables statistiques étaient comparables à ceux de notre formule GPF avec une similarité cosinus de 0.9923. Nous en concluons que notre formule de l'équation 5.4 reste intéressante, car elle ne nécessite aucune pondération supplémentaire.

Régression linéaire avec toutes les variables au format logarithmique En suivant la même procédure que précédemment avec toutes les statistiques transformées au format logarithmique, les scores présentés dans le tableau 5.9 montre que la méthode linéaire surpasse notre formule GPF de plus d'un point. Notons que deux des variables générant des valeurs négatives, il était impossible de les convertir en échelle $\log$. Par le fait, l'apprentissage a été réalisé sur 17 variables statistiques au lieu des 19 originales.

TABLE 5.9 – Comparaison des résultats de la régression linéaire transformées au format log.

Method	Nb. variables	$\log$-format	Pearson's r	r^2
Linear Regression	5		0.8796	0.7689
Linear Regression	19		0.9042	0.8011
Our GP formula	5		*0.9671*	*0.9319*
Linear Regression	5	✓	0.9607	0.9206
Linear Regression	17	✓	**0.9795**	**0.9586**

5.4.5 Comparaison

Dans cette section, nous nous intéressons à comparer notre méthode avec des travaux connexes présentés dans l'état de l'art, en particulier les travaux de Collins *et al.* utilisant un algorithme génétique (GA) [9] ainsi que ceux de Bensusan *et al.* utilisant des règles [4]. La principale difficulté pour comparer nos travaux avec eux repose sur le fait qu'aucun des deux auteurs n'utilise un ensemble de données ou de statistiques similaires pour décrire les ensembles de données sélectionnés. Cependant, nous proposons de comparer les solutions trouvées par leur chaîne de traitement sur notre propre métacorpus. Les résultats sont présentés dans le tableau 5.10.

TABLE 5.10 – Comparaison de nos meilleurs résultats avec les méthodes décrites dans [9] et [4].

Method	Nb. variables	$\log$-format	Pearson's r	r^2
GA unweighted sum	19		0.7763	0.5744
GA unweighted sum	17	✓	*0.9621*	*0.9254*
Cubist Rules	19		0.9666	0.9343
Cubist Rules	17	✓	*0.9772*	*0.9525*
Cubist Rules	5	✓	0.9642	0.9276
Our Linear Regression	5	✓	0.9607	0.9206
Our Linear Regression	17	✓	**0.9795**	**0.9586**
Our GP formula	5		*0.9671*	*0.9319*

Algorithme génétique (GA) Dans un premier temps, nous comparons nos résultats avec ceux obtenus en utilisant la méthode décrite par Collins *et al.* [9]. Celle-ci consiste à trouver une somme pondérée de quelques variables grâce à un algorithme génétique (GA) avec 3 valeurs de pondération possibles : $\{-1, 0, 1\}$. La fonction d'évaluation – *fitness* – utilisée est la corrélation de Pearson entre la prédiction et la justesse – *accuracy* – empirique. L'expérience est réalisée avec le même partitionnement des données d'apprentissage et de test que pour notre étude, et une simulation comportant une population initiale de 5000 individus pendant 300 itérations. Nous avons remarqué que le choix de ces paramètres de simulation conduisaient à des résultats stables.

Comme prévu, en réalisant cette expérience, nous nous attendions à avoir un résultat significativement plus mauvais que la somme pondérée de notre modèle de régression linéaire utilisé comme référence (tableau 5.4) car les 19 variables sont soumises à une pondération fortement quantifiée. L'expérience confirme cela.

Nous avons alors réitéré la même expérience avec une transformation logarithmique sur les données d'entrée, c'est-à-dire une transformation de nos 17 variables statistiques positives. Conformément aux résultats rapportés dans la section 5.4.4, les scores sont relativement meilleurs, mais ne dépasse toujours pas notre solution. L'équation 5.8 présente la formule associée au meilleur résultat trouvé, définie comme :

```
out=  log(Sb_trace) + log(shapiro)
    + log(dim) - log(feats_corr)
    - log(Sw_trace) - log(kurtosis_avg)
    - log(prototypes_corr)
```
(5.8)

Cette solution utilise un total de 7 variables dont 3 figurent dans notre formule GPF composé, elle, de seulement 5 variables. Il nous est cependant difficile de saisir la manière dont les variables choisies par cette naïf interagissent entre elles.

Règles Cubist Nous comparons nos résultats avec ceux issus de la procédure proposée par Bensusan *et al.* [4] qui utilise Cubist, un logiciel capable de générer un ensemble de règles interprétables. Dans notre cas, nous utilisons une surcouche de Cubist implémentée par un paquet pour le langage R sur lequel nous avons expérimenté les trois ensembles de variables statistiques en entrée de la chaîne de traitement comme présenté dans le tableau 5.10.

Le premier des ensembles de données correspond à l'utilisation des 19 variables statistiques originales de notre métacorpus sans aucune transformation. Cubist génère sur ces variables 10 règles à l'interprétation relativement ardue. La figure 5.8 montre l'une de ces règles. En termes de performance, les scores sont comparables à notre formule GPF.

Dans une deuxième expérience, nous avons utilisé le prétraitement logarithmique sur nos 17 variables statistiques positives. Nous constatons que ce prétraitement a un effet bénéfique sur les règles produites par Cubist qui nous a proposé une solution basé sur 6 règles tout étant plus efficace que la version précédente.

Dans une dernière expérience, nous testons l'utilisation des 5 mêmes variables que pour notre formule GPF avec toujours la transformation logarithmique. Cubist a généré 2 règles qui restent moins performantes que notre formule pour la tâche de prédiction de la justesse – *accuracy* .

De manière générale, nous constatons que le prétraitement du $\log$ sur les variables d'entrée a un effet bénéfique sur les règles produites par Cubist, comme pour la régression linéaire vue précédemment. Mais, surtout, notre formule est plus claire et explicable que l'ensemble des règles générées par Cubist pour des performances similaires.

```
if
 percentage_dims_exp_var_99 > 0.9316406
 feats_corr <= 0.05039461
 kurtosis_std <= 9.284021
 n_test > 1880
 n_test <= 4384
then
 outcome = 5.4992777
    - 0.031372 kurtosis_std
    - 5.02 percentage_dims_exp_var_99
    + 1.31 feats_corr
    + 0.83 Sb_trace
    - 0.19 prototypes_cos_sim
    + 0.085 prototypes_corr
    + 8e-06 n_test + 1.6 train_mean
    - 3.2e-05 n_classes
    - 1.8 train_std
```

FIGURE 5.8 – Exemple d'une des 10 règles générées par *Cubist* lors de l'apprentissage sur les 19 variables statistiques.

Bilan Suivant le principe d'Ockham [1], ces résultats tendent à montrer que notre formule GPF est un meilleur choix. En outre, notre formule GPF propose le meilleur compromis entre précision et simplicité d'écriture sans avoir d'a priori sur la structure de la formule générée, et qui plus est, elle est facilement explicable comme nous le verrons dans la section 5.5.

5.4.6 Généralisation

Dans cette section, nous voulons montrer que notre méthode présentée tout au long de ce chapitre peut être appliquée à tout corpus de données pouvant être décrit comme un ensemble de statistiques. Notre formule GPF étant obtenue par l'utilisation de jeux de données et d'extracteurs de caractéristiques issus du domaine de la vision par ordinateur, il est intéressant d'analyser la généralisation de notre formule à d'autres domaines.

Application sur des données de type textuel Ainsi, nous proposons de tester la possibilité d'appliquer notre formule sur une nouvelle modalité telle que des données de types texte. Pour cela, nous avons réuni un ensemble de 7 jeux de données textuelles ainsi que 4 extracteurs de caractéristiques provenant du paquet Python *sentence-transformers* [2]. En appliquant notre méthodologie d'extraction de représentations statistiques et de recherche de justesse – *accuracy* – empirique sur toutes les possibilités de

1. La rasoir d'Ockham, aussi connu sous le nom de principe de parcimonie, est un principe de la logique et de la science qui stipule qu'entre deux hypothèses équivalentes, celle qui implique le moins d'éléments supplémentaires est généralement préférable.
2. `https://www.sbert.net`

combinaison entre les ensembles de données et les extracteurs de caractéristiques, nous obtenons un nouveau métacorpus contenant 28 points d'analyse. Le tableau 5.11 présente la différence des résultats entre ceux obtenus sur les données vision et ceux issus des données textuelles pour les différentes méthodes de régression de références ainsi que pour notre formule GPF. Notre formule GPF adaptée consiste à une adaptation des coefficients de la transformation linéaire pour notre métacorpus textuel (Toutes les valeurs-$p < 0.01$). Toutes les corrélations rapportées ont une valeur-p statistiquement significative.

TABLE 5.11 – Test de l'application de notre formule GPF sur des données textuelles.

	Vision		Texte	
Method	Pearson's r	r^2	Pearson's r	r^2
Linear Regression	0.9042	0.8011	0.6191	0.3052
Decision Tree	0.9472	0.8868	0.7944	0.1928
Random Forest (10 trees)	0.9643	0.9246	0.7231	-0.0722
Our GP formula (GPF)	**0.9671**	**0.9319**	**0.8618**	0.4565
Adapted GP formula (GPF)	-	-	**0.8618**	**0.7428**

Une baisse significative de la corrélation de Pearson ainsi que du score r^2 est remarquée pour toutes les méthodes. Cependant, notre solution surpasse toujours les autres méthodes avec une forte corrélation de Pearson de 0.8618. Dans un second temps, nous avons recalibré les coefficients a et b de la transformation linéaire sur les données textes. Pour rappel, cette transformation linéaire est nécessaire pour passer de la sortie de notre algorithme de GP qui maximise la corrélation avec la justesse – *accuracy* – à une valeur de sortie correspondant à l'estimation de la justesse. Cette transformation nous permet d'améliorer grandement le score r^2.

Bilan Avec, pour coefficient de la transformation linéaire, $(a, b) = (0.2417, 1.0327)$ sur notre métacorpus vision et $(a, b) = (0.2508, 0.9121)$ sur notre métacorpus texte, nous pouvons voir que les paramètres a et b sont similaires pour les deux modalités. Cependant, le score $r2$ semble extrêmement sensible, peut-être en raison du faible nombre de points. Alors que les ensembles de données avec plus de 200 classes sont courants dans le domaine de la vision par ordinateur, les tâches de classification de texte ont un nombre de classes beaucoup plus faible dans les corpus que nous avons utilisés, comme 2 pour l'analyse des sentiments ou 20 pour la modélisation de thèmes – *topic modeling* .

Bien que nos résultats ne soient pas parfaits, ils semblent néanmoins prometteurs. Toutefois, ils gagneraient à intégrer davantage de métacorpus provenant d'autres domaines, comme des ensembles de données de type audio, vidéo, à base de graphes ou encore tabulaires.

5.5 Discussion

Revenons sur notre formule GPF présentée dans l'équation 5.4 et composée des 5 variables. Notre formule peut s'écrire comme la somme de deux composantes : SEP et

COR.

Séparabilité des classes En examinant de plus près le premier terme SEP, nous pouvons constater qu'il est comparable au critère de Fisher utilisé dans l'analyse discriminante linéaire (LDA) [15], où l'objectif est de trouver une projection linéaire qui maximise le rapport entre la variance interclasse et la variance intraclasse. Ainsi, ce terme correspondrait à une mesure de séparabilité des classes. Il est intéressant de noter que ce critère a été utilisé avec succès comme fonction de perte en apprentissage profond [12, 16] mais reste peu courant au détriment de l'entropie croisée (CE). Cependant, les fortes similitudes entre la LDA et la entropie croisée (CE) nous permettent de remplacer cette première mesure de séparabilité par la seconde. En effet, Wan *et al.* ont constaté que l'une des options les plus étudiées pour surmonter certaines limitations de la fonction*softmax* dans la perte basée sur la CE est d'encourager une compacité intraclasse plus forte et une séparabilité interclasse plus grande de manière comparable au critère de Fisher [56].

Décorrélation des variables Le second terme COR est quant-à-lui négativement corrélée à la justesse – *accuracy* – et reste facilement compréhensible en examinant chaque élément qui le compose.

Le premier élément correspond au nombre de classes $n\_classes$. En effet, lorsqu'un modèle d'apprentissage automatique est entraîné sur un jeu de données, il est naturel de s'attendre à une diminution des scores lorsque le nombre de classes augmente. Cette intuition peut être vérifiée empiriquement sur des ensembles de données présentant des granularités de classe différentes. Par exemple, dans les travaux de Chang *et al.* [7], les auteurs ont observé une baisse de la précision de $0,97$ à $0,82$ sur le jeu de données CUB200 [57] en faisant passer le nombre de classes d'un niveau grossier (13 classes) à un niveau plus fin (200 classes).

Concernant les deux autres variables $feats\_corr$ et $prototype\_cos\_sim$, elles correspondent à des informations d'orthogonalité et de décorrélation. En se référant à la littérature, il nous est possible d'expliquer l'importance de ces deux termes de décorrélation. Pour justifier le terme $prototypes\_cos\_sim$ correspondant à la décorrélation des poids, Bansal *et al.* ont constaté sur plusieurs CNNs de l'état de l'art qu'ils pouvaient obtenir une meilleure justesse – *accuracy* –, un apprentissage plus stable et une convergence plus douce grâce à une régularisation orthogonale des poids [2]. Sur le terme $feats\_corr$ correspondant à la décorrélation des caractéristiques, plusieurs travaux justifient la présence d'une telle décorrélation [3, 14, 24, 28, 38, 56, 61, 62]. En effet, LeCun *et al.* ont remarqué que les variables d'entrée corrélées conduisent généralement les vecteurs propres de la matrice hessienne à être orientés loin des axes de coordonnées, ce qui entraîne une convergence plus lente de l'apprentissage [38]. Ainsi, plusieurs propositions ont été développées afin de mieux décorréler les variables telles que l'analyse en composantes principales (PCA) ou l'analyse des composants nuls (ZCA) [3] [28]. Plus récemment, la décorrélation a joué un rôle important dans l'augmentation des performances des méthodes auto-supervisées [3, 14, 24, 61, 62]. Par exemple, Bardes *et al.* [3] ont inclus une partie de décorrélations dans leur fonction de perte. Ils affirment

3. Pour information, la ZCA est une technique de réduction de dimensionnalité qui utilise une transformation linéaire pour projeter les données sur un sous-espace de dimensions plus faible. Elle est semblable à la technique PCA, mais utilise une matrice de *whitening* pour égaliser l'échelle des composantes. Cela permet de conserver autant d'informations que possible tout en réduisant le nombre de dimensions.

que ce terme décorrèle les variables et empêche l'effondrement des représentations – *embeddings collapsing* .

Bilan Pour résumer, afin de répondre à cette tâche de prédiction de la justesse – *accuracy* –, les représentations – *embeddings* – extraites doivent répondre à deux contraintes : pouvoir séparer les classes facilement et présenter des variables décorrélées entre elles. Ce résultat nous est donné par l'intelligence artificielle (AI) et nous permet de comprendre qu'une fonction de perte pertinente doit nécessairement répondre à ces deux contraintes.

5.6 Conclusion

Dans ce chapitre, nous avons montré que notre chaîne de traitement peut aider à extraire et à donner un sens concret à des intuitions sur la capacité à classer correctement des données. Pour ce faire, nous avons mené une étude sur un métacorpus de plus de 260 jeux de données de représentations – *embeddings* – extraits de la combinaison d'un large éventail de corpus et d'extracteurs de caractéristiques. Ces représentations ont ensuite été projetées dans un nouvel espace commun à l'aide d'un ensemble de statistiques générales. Cette méthode permet de caractériser n'importe quel ensemble de données. Par conséquent, notre travail est applicable, non seulement à la vision par ordinateur, mais également à tous les autres domaines de l'apprentissage automatique. Enfin, une heuristique capable de prédire la justesse – *accuracy* – d'un classifieur linéaire, produite de manière automatique, présente une corrélation de Pearson de $0,96$ et un r^2 de $0,93$. De plus, cette formule est hautement explicable et est cohérente avec des décennies de recherche.

Ces travaux ont fait l'objet de deux publications d'articles : l'une à AIMLAI 2022, un workshop de CIKM [6] et la seconde à la conférence IEEE/CVF WACV 2023 [47].

Références

[1] Douglas Adriano Augusto et Helio J. C. Barbosa. “Symbolic Regression via Genetic Programming”. In : *SBRN, Procs.* 2000, pages 173-178.

[2] Nitin Bansal, Xiaohan Chen et Zhangyang Wang. “Can We Gain More from Orthogonality Regularizations in Training Deep Networks ?” In : *NIPS, Procs.* Tome 31. 2018.

[3] Adrien Bardes, Jean Ponce et Yann LeCun. “VICReg : Variance-Invariance-Covariance Regularization for Self-Supervised Learning”. In : *International Conference on Learning Representations*. 2022.

[4] Hilan Bensusan et Alexandros Kalousis. “Estimating the predictive accuracy of a classifier”. In : *European Conference on Machine Learning*. Springer. 2001, pages 25-36.

[5] B. Chamand et al. “Fine-tune your classifier : Finding correlations with temperature”. In : *Proceedings of the IEEE International Conference on Image Processing – ICIP 2022*. Bordeaux (France) : IEEE Computer Society, 2022, pages xx-xx.

[6] Benjamin Chamand et Olivier Risser-Maroix. “Finding Components of a Good Accuracy with XAI !” In : *Advances in Interpretable Machine Learning and Artificial Intelligence (AIMLAI@CIKM)*. 2022.

[7] Dongliang Chang et al. “Your" Flamingo" is My" Bird" : Fine-Grained, or Not”. In : *CVPR, Procs.* 2021, pages 11476-11485.

[8] M. Cimpoi et al. “Describing Textures in the Wild”. In : *CVPR, Procs.* 2014.

[9] Edward Collins, Nikolai Rozanov et Bingbing Zhang. “Evolutionary Data Measures : Understanding the Difficulty of Text Classification Tasks”. In : *CoNLL*. 2018.

[10] Alex Davies et al. “Advancing mathematics by guiding human intuition with AI”. In : *Nature* 600.7887 (2021), pages 70-74.

[11] Weijian Deng et Liang Zheng. “Are labels always necessary for classifier accuracy evaluation ?” In : *CVPR, Procs.* 2021, pages 15069-15078.

[12] Matthias Dorfer, Rainer Kelz et Gerhard Widmer. “Deep Linear Discriminant Analysis”. In : *ICLR, Procs.* 2016.

[13] Michael R Douglas. “Machine learning as a tool in theoretical science”. In : *Nature Reviews Physics* (2022), pages 1-2.

[14] Aleksandr Ermolov et al. “Whitening for self-supervised representation learning”. In : *ICML, Procs.* PMLR. 2021, pages 3015-3024.

[15] Ronald A. Fisher. “The Use of Multiple Measurements in Taxonomic Problems”. In : *Annals of Eugenics* 7.2 (1936), pages 179-188.

[16] Benyamin Ghojogh et al. “Fisher discriminant triplet and contrastive losses for training siamese networks”. In : *2020 international joint conference on neural networks (IJCNN)*. IEEE. 2020, pages 1-7.

[17] Greg Griffin, Alex Holub et Pietro Perona. “Caltech256 Image Dataset”. In : (2006).

[18] Kaiming He et al. "Deep residual learning for image recognition". In : *CVPR, Procs.* 2016, pages 770-778.

[19] Martin N. Hebart et al. "THINGS : A database of 1, 854 object concepts and more than 26, 000 naturalistic object images". In : *PLOS One* 14.10 (2019), e0223792.

[20] Dan Hendrycks et al. "The Many Faces of Robustness : A Critical Analysis of Out-of-Distribution Generalization". In : *CoRR* abs/2006.16241 (2020).

[21] Tin Kam Ho et Mitra Basu. "Complexity measures of supervised classification problems". In : *IEEE transactions on pattern analysis and machine intelligence* 24.3 (2002), pages 289-300.

[22] Timothy Hospedales et al. "Meta-learning in neural networks : A survey". In : *arXiv preprint arXiv :2004.05439* (2020).

[23] Andrew Howard et al. "Searching for mobilenetv3". In : *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pages 1314-1324.

[24] Tianyu Hua et al. "On feature decorrelation in self-supervised learning". In : *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pages 9598-9608.

[25] Gao Huang et al. "Densely connected convolutional networks". In : *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pages 4700-4708.

[26] Forrest N Iandola et al. "SqueezeNet : AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size". In : *arXiv preprint arXiv :1602.07360* (2016).

[27] Roxana Istrate et al. "Tapas : Train-less accuracy predictor for architecture search". In : *Proceedings of the AAAI Conference on Artificial Intelligence*. Tome 33. 01. 2019, pages 3927-3934.

[28] Agnan Kessy, Alex Lewin et Korbinian Strimmer. "Optimal whitening and decorrelation". In : *The American Statistician* 72.4 (2018), pages 309-314.

[29] Diederik P. Kingma et Jimmy Ba. "Adam : A Method for Stochastic Optimization". In : *ICLR, Procs.* 2015.

[30] John R. Koza. *Genetic programming - on the programming of computers by means of natural selection*. Complex adaptive systems. MIT Press, 1993.

[31] John R. Koza. "Genetic programming as a means for programming computers by natural selection". In : *Statistics and Computing* 4.2 (1994).

[32] Alex Krizhevsky et Geoffrey Hinton. *Learning multiple layers of features from tiny images*. Rapport technique. University of Toronto, 2009.

[33] Alex Krizhevsky, Ilya Sutskever et Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In : *NIPS, Procs.* 2012.

[34] William La Cava et al. "Contemporary symbolic regression methods and their relative performance". In : *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*. Sous la direction de J. Vanschoren et S. Yeung. Tome 1. 2021.

[35] Mikel Landajuela et al. “Discovering symbolic policies with deep reinforcement learning”. In : *ICML, Procs.* 2021, pages 5979-5989.

[36] Ya Le et Xuan S. Yang. “Tiny ImageNet Visual Recognition Challenge”. In : 2015.

[37] Yann LeCun, Corinna Cortes et Christopher J.C. Burges. *MNIST handwritten digit database*. http://yann.lecun.com/exdb/mnist. 2010.

[38] Yann A. LeCun et al. “Efficient BackProp”. In : *Neural Networks : Tricks of the Trade : Second Edition*. Sous la direction de Grégoire Montavon, Geneviève B. Orr et Klaus-Robert Müller. Berlin, Heidelberg : Springer Berlin Heidelberg, 2012, pages 9-48.

[39] Xiang Li et al. “FSS-1000 : A 1000-Class Dataset for Few-Shot Segmentation”. In : *CVPR, Procs.* 2020, pages 2866-2875.

[40] Ana C Lorena et al. “How Complex is your classification problem ? A survey on measuring classification complexity”. In : *ACM Computing Surveys* 52.5 (2019), pages 1-34.

[41] Qiang Lu, Jun Ren et Zhiguang Wang. “Using genetic programming with prior formula knowledge to solve symbolic regression problem”. In : *Computational Intelligence and Neuroscience* 2016 (2016), 1021378 :1-1021378 :17.

[42] Ester Bernado Mansilla et Tin Kam Ho. “On classifier domains of competence”. In : *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.* Tome 1. IEEE. 2004, pages 136-139.

[43] T. Nathan Mundhenk et al. “Symbolic Regression via Neural-Guided Genetic Programming Population Seeding”. In : *NIPS, Procs.* 2021.

[44] Brenden K Petersen et al. “Deep symbolic regression : Recovering mathematical expressions from data via risk-seeking policy gradients”. In : *ICLR, Procs.* 2021.

[45] *Pins Face Recognition — kaggle.com*. https : / / www . kaggle . com / hereisburak/pins-face-recognition. [Accessed 13-Dec-2021].

[46] Alec Radford et al. “Learning transferable visual models from natural language supervision”. In : *ICML, Procs.* 2021, pages 8748-8763.

[47] Olivier Risser-Maroix et Benjamin Chamand. “What Can We Learn by Predicting Accuracy ?” In : *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*. 2023, pages 2390-2399.

[48] Mark Sandler et al. “Mobilenetv2 : Inverted residuals and linear bottlenecks”. In : *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pages 4510-4520.

[49] Florian Scheidegger et al. “Efficient image dataset classification difficulty estimation for predicting deep-learning accuracy”. In : *The Visual Computer* 37.6 (2021), pages 1593-1610.

[50] Michael Schmidt et Hod Lipson. “Distilling free-form natural laws from experimental data”. In : *Science* 324.5923 (2009), pages 81-85.

[51] Florian Schroff, Dmitry Kalenichenko et James Philbin. “Facenet : A unified embedding for face recognition and clustering”. In : *CVPR, Procs.* 2015, pages 815-823.

[52] S. S. Shapiro et M. B. Wilk. "An Analysis of Variance Test for Normality (Complete Samples)". In : *Biometrika* 52.3/4 (déc. 1965), page 591.

[53] Silviu-Marian Udrescu et Max Tegmark. "AI Feynman : A physics-inspired method for symbolic regression". In : *Science Advances* 6.16 (2020), page 2631.

[54] Marco Virgolin et Solon P Pissis. "Symbolic Regression is NP-hard". In : *arXiv preprint arXiv :2207.01018* (2022).

[55] Marco Virgolin et al. "Improving model-based genetic programming for symbolic regression of small expressions". In : *Evolutionary Computation* 29.2 (2021), pages 211-237.

[56] Weitao Wan et al. "Rethinking feature distribution for loss functions in image classification". In : *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pages 9117-9126.

[57] P. Welinder et al. *Caltech-UCSD Birds 200*. Rapport technique CNS-TR-2010-001. California Institute of Technology, 2010.

[58] Wei Wen et al. "Neural predictor for neural architecture search". In : *European Conference on Computer Vision*. Springer. 2020, pages 660-676.

[59] Qi Wu, Hongping Cai et Peter Hall. "Learning graphs to model visual objects across different depictive styles". In : *ECCV, Procs.* 2014, pages 313-328.

[60] Yasunori Yamada et Tetsuro Morimura. "Weight Features for Predicting Future Model Performance of Deep Neural Networks." In : *IJCAI*. 2016, pages 2231-2237.

[61] Jure Zbontar et al. "Barlow Twins : Self-Supervised Learning via Redundancy Reduction". In : *Proceedings of the 38th International Conference on Machine Learning*. Sous la direction de Marina Meila et Tong Zhang. Tome 139. Proceedings of Machine Learning Research. PMLR, juill. 2021, pages 12310-12320.

[62] Shaofeng Zhang et al. "Zero-CL : Instance and Feature decorrelation for negative-free symmetric contrastive learning". In : *ICLR, Procs.* 2022.

Conclusion, perspectives et publications

Conclusion

Synthèse des travaux

Ce manuscrit a été divisé en deux parties principales. Dans la première partie, nous nous sommes intéressés à l'utilisation du connexionnisme impulsionnel dans des tâches d'extraction de représentations de données images et audios. Puis, dans une seconde partie, nous nous sommes placés dans une démarche d'exploration du paradigme de l'intelligence augmentée dans le but d'améliorer les capacités de classification des représentations extraites d'un modèle.

Première partie En vue de contextualiser la relation entre le connexionnisme impulsionnel et les méthodes classiques couramment utilisées, j'ai proposé un état de l'art sur le fonctionnement d'un réseau de neurones à impulsions (SNN) en le comparant à celui d'un réseau de neurones artificiels (ANN). Cette étude inclut le fonctionnement du neurone biologique jusqu'à ses différentes modélisations mathématiques en passant en revue les règles d'apprentissage ainsi que le codage de l'information qui les régissent. Grâce à cette étude, j'ai pu développer une bibliothèque, sous le nom de « PyTorch SNN », qui conserve l'architecture et la simplicité d'utilisation de PyTorch. Dans le but de préserver l'objectif initial de générer des représentations non supervisées, cette bibliothèque utilise des règles d'apprentissage conformes au postulat de Hebb, telles que la plasticité fonction du temps d'occurrence des impulsions (STDP). La création de cette bibliothèque a servi de base à diverses expérimentations sur des données d'images et de sons en lien avec les travaux existants, et dont les résultats sont conformes à ceux de ces derniers. Afin d'accélérer significativement les calculs, une proposition d'utiliser des lots de données lors de l'apprentissage – également appelés *batch* – a été présentée, au détriment de certaines performances. Cette fonctionnalité permet une amélioration de la vitesse de calcul grâce à l'utilisation des GPU. Comparée à d'autres bibliothèques présentant les mêmes caractéristiques, la nôtre conduit à des résultats similaires mais se distingue surtout par une prise en main plus simple.

Durant ces recherches, nous avons constaté qu'il était complexe d'entraîner un SNN de manière non supervisée, soit en raison de la grande sensibilité des hyperparamètres choisis (en particulier le choix du taux d'apprentissage α_- et α_+ de la STDP), soit en raison de la nécessité d'entraîner chaque couche séparément lors de l'apprentissage, limitant ainsi la recherche de motifs complexes. Par ailleurs, il convient de noter que le processus d'apprentissage d'un SNN demeure considérablement plus long que celui d'une méthode d'apprentissage machine (ML) classique pour une même tâche donnée. En partant de ce constat, nous avons décidé d'adopter une approche hybride qui consiste à entraîner un SNN par une méthode auto-supervisée reposant sur la descente de gradient. Cette dernière est rendue possible grâce à l'utilisation d'un gradient alternatif –

surrogate gradient – et l'exploitation de la propriété stochastique du processus de Poisson qui permet de générer de multiples représentations d'une même donnée d'entrée. Cette approche hybride a permis de démontrer qu'il était possible d'entraîner aisément un modèle de manière non supervisée afin d'obtenir des représentations équivalentes à celles présentées dans l'état de l'art.

Ces travaux ont fait l'objet d'un rapport technique dans le cadre d'une collaboration avec le ministère de l'Intérieur sous le projet RAVI (Reconnaissance Automatique de Véhicules dans les Images) ainsi que d'une publication dans un congrès international [1].

Seconde partie Comme mentionné précédemment, l'utilisation de modèles de SNNs dans l'extraction de caractéristiques de nos données est une tâche complexe en raison de la sensibilité des divers hyperparamètres. De plus, s'il est courant d'utiliser un classifieur linéaire pour catégoriser les représentations extraites, qu'elles proviennent aussi bien d'ANNs classiques que de SNNs, celle-ci peut nécessiter de nouveau hyperparamètres, tels que la température qui ajuste l'entropie de la distribution de probabilité de sorties du réseau. Il convient donc de trouver les hyperparamètres les plus adaptés à la tâche de classification pour maximiser les performances du modèle. Cela peut être réalisé par une recherche systématique des hyperparamètres ou en utilisant des techniques d'optimisation telles que la recherche bayésienne ou l'optimisation par essais et erreurs.

La contribution principale de cette partie est d'améliorer les éléments constitutifs d'un système de classification à l'aide de l'intelligence artificielle (AI). Pour cela, nous avons initialement examiné l'influence d'un hyperparamètre connu sous le nom de « température » présent dans la dernière couche d'un modèle de classification. Nous avons extrait des représentations issues de différents modèles et corpus, dans un espace commun sous forme de vecteurs de statistiques de tailles identiques. Chaque représentation statistique est ensuite liée à une température optimale empirique obtenue par une exploration systématique des valeurs possibles via une méthode de recherche en grille – *grid-search* . En utilisant des corrélations entre ces couples de représentations statistiques et de température empirique, nous avons mis en évidence une heuristique pour déterminer une valeur de température idéale dès l'initialisation du modèle. Cette heuristique est basée sur une combinaison linéaire des 4 variables statistiques les plus corrélées à la température optimale empirique.

Dans un second temps, nous avons élargi notre chaîne de traitement pour l'appliquer à une tâche plus complexe, à savoir la prédiction de la justesse – *accuracy* – d'un modèle. Notre objectif était double : d'une part, comprendre comment nous pourrions améliorer nos représentations afin d'obtenir une meilleure performance, et d'autre part, améliorer notre méthode afin d'obtenir une solution explicable grâce à une tâche prétexte – *proxy task* . Pour cela, nous avons introduit une différence notable par rapport à notre chaîne de traitement précédente, à savoir l'ajout d'un modèle de régression symbolique à la place d'une simple combinaison linéaire pour améliorer notre méthode. En reprenant la même méthodologie que précédemment adaptée à ce problème de prédiction de la justesse – *accuracy* –, nous avons obtenu une formule capable de réaliser des prédictions avec une corrélation de Pearson très forte, tout en ayant un faible coefficient de détermination r^2. Cette formule s'est avérée performante, avec des résultats comparables à ceux obtenus avec d'autres méthodes existantes, mais reposant seulement sur 5 variables, ce qui la rend « plus explicable » et nous permet d'établir sa conformité avec les méthodes scientifiques classiques de l'apprentissage automatique. Nous avons

également pu démontrer l'extensibilité de cette méthode à des données textuelles.

Cette dernière partie a mené à deux publications dans des congrès internationaux [2, 6].

Perspectives

Bien évidemment, une multitude de perspectives peut être envisagée pour la suite de mes travaux. Cependant, je vais présenter ici celles que je trouve importante sur les deux axes de recherches que j'ai évoqué dans mon manuscrit : l'approche connexionniste et l'explicabilité d'une tâche, pour les étendre ensuite sur un domaine qui m'intéresse fortement depuis quelques années maintenant : les sciences cognitives.

Sur l'approche connexionniste

Connexionnisme impulsionnel J'ai plusieurs propositions de nouvelles expériences pouvant être réalisées à partir des travaux exposés dans la première partie de ce manuscrit. Il serait envisageable d'adapter la fonction de coût employée dans le chapitre 3 pour inclure une distance de van Rossum entre les trains d'impulsions, classiquement utilisé pour cet usage [7]. Une autre piste serait de renouveler l'expérience en utilisant des données vidéos en entrée, avec différentes méthodes de conversion de la vidéo en trains d'impulsions [3]. Enfin, il serait envisageable d'entraîner un SNN à générer des représentations audiovisuelles à partir de la concaténation des trains d'impulsions des deux modalités. Cette possibilité découle de la nature binaire et temporelle des représentations dans le domaine du connexionnisme impulsionnel, pour lesquelles des stratégies d'encodage devront être établies afin d'équilibrer les deux modalités en termes de fréquence d'impulsions. L'analyse individuelle de chaque neurone après l'apprentissage du modèle à l'aide de la STDP, pourrait révéler des correspondances entre des motifs étroitement liés sur le signal audio et sur le signal vidéo.

Toutefois, mon expérience dans le domaine des ANNs et des SNNs m'amène à me questionner sur l'existence d'une méthode d'apprentissage intermédiaire permettant de combiner les avantages de ces deux modèles connexionnistes. Le cerveau humain est capable d'assimiler de nouvelles représentations rapidement, mais il n'existe aucune preuve démontrant explicitement que le comportement de celui-ci peut être approximé par une dérivée d'erreur ou la conservation des activités neuronales pour une descente de gradient ultérieure. Bien que la STDP se rapproche le plus du fonctionnement biologique en extrayant des motifs répétitifs dans un flux temporel, elle reste longue à calculer et difficile à maîtriser, surtout pour les modèles profonds. Une piste de recherche intéressante serait ainsi de trouver une méthode d'apprentissage capable de modifier les connexions synaptiques localement d'un ANN à travers une seule passe en avant comme peut le faire la STDP dans un SNN. Une telle approche pourrait ouvrir une nouvelle voie pour l'apprentissage non supervisé dans le domaine du *deep learning*, avec des temps de calcul encore plus rapides.

Connexionnisme classique En parallèle de mes travaux de thèse, j'ai aussi voulu répondre à la problématique initiale de la thèse en s'inspirant d'une méthode d'apprentissage profond issue des travaux de Pascual *et al.* [4] sur la génération de représentations acoustiques à travers leur modèle PASE (Problem Agnostic Speech Encoder) et PASE+

qui sont des méthodes d'apprentissage multi-tâche autosupervisées. Ces modèles se présentent comme un auto-encodeur avec plusieurs décodeurs – appelés *workers* – ayant comme objectif de résoudre une tâche spécifique intermédiaire – tâche *proxy* – comme l'identité du locuteur, la reconstruction de paramètres comme les MFCCs ou encore la présence d'émotion, dont les informations sont extraites depuis le fichier audio brut en entrée. Ces ouvriers – *workers* –, comme les nomment les auteurs, sont constitués soit de petits ANNs à quelques couches, soit de réseaux très complexes permettant la regénération du fichier audio en entrée par exemple. Grâce à cette méthode, le vecteur de représentations, *embeddings* – générés contient donc une information variée composée de différents attributs que ces ouvriers – *workers* – vont chercher à les décoder.

En partant de ce principe, j'ai expérimenté un modèle similaire adapté à la multimodalité en remplaçant le principe de l'auto-encodeur (AE) du modèle PASE par un auto-encodeur variationnel (VAE) de telle sorte que les représentations soient modélisées par une distribution statistique (loi normale) dont le but est de minimiser la distance entre les représentations latentes des deux modalités. Ce modèle est présenté sur la figure 5.9 et reprend la base d'un système traditionnel de correspondance audiovisuelle (AVC) qui va générer ici deux représentations en parallèle de nos données (une par modalité) toutes les 200 ms avec un recouvrement de 100 ms. Les représentations sont générées de la même manière qu'avec le modèle PASE pour la partie audio, et un système plus classique à base de CNNs 3D pour la partie vidéo. Les représentations vont ensuite alimenter les ouvriers – *workers* – spécifiques à une tâche donnée qui vont devoir décoder l'information. L'originalité de cette approche est d'avoir un croisement entre les représentations audio et vidéo par l'intermédiaire de ces ouvriers. C'est-à-dire que l'information contenue dans la représentation de la modalité audio va être décodée aussi bien par les ouvriers dédiés à l'audio que ceux spécifiques à la vidéo. Ainsi, nous définissons la fonction de perte $\mathcal{L}_1$ représentant la somme de toutes les fonctions de pertes de reconstruction. En reprenant le même principe issu des travaux de Schonfeld *et al.* [8] sur la capacité à minimiser la distance entre les représentations latentes de l'image et du texte, nous définissons dans l'architecture une seconde fonction de perte $\mathcal{L}_2$ qui minimise la distance de Wasserstein entre les distributions gaussiennes multivariées latentes de l'audio et de la vidéo.

Le calcul de la mesure de consistance d'un concept audiovisuel se fait ensuite par une mesure euclidienne entre les deux représentations latentes. Cette approche présentée ici est l'une des approches qui m'a permis d'obtenir des scores très prometteurs dans la résolution de ma tâche initiale.

Dans cette première version, un ouvrier – *worker* – décodant une tâche spécifique à partir de la représentation audio – *audio embeddings* – est totalement différent de celui qui décode à partir de la représentation visuelle – *frames embeddings*. Une fonction de perte $\mathcal{L}_2$ se charge de forcer les encodeurs à générer des représentations audio et vidéo dans un même espace. En partant des travaux étudiés dans le chapitre 3 concernant les différentes approches d'apprentissage auto-supervisé (SSL), la question que je me pose maintenant est : « serait-il envisageable de modifier cette architecture pour localiser spécifiquement les attributs audio dans la séquence vidéo et réciproquement ? » En ayant des ouvriers – *workers* – en charge de la fusion des 2 représentations d'entrée, il est envisageable de relocaliser des attributs présents dans l'audio sur la partie visuelle, conformément aux pratiques courantes dans les recherches portant sur la correspondance audiovisuelle (AVC). Autrement dit, l'extraction de la représentation audiovisuelle permettrait d'incorporer les caractéristiques physiques intrinsèques à un

objet au sein d'une scène, de manière entièrement non supervisée. Par exemple, pour l'ensemble de pixels représentant un visage dans la séquence vidéo, la méthode pourrait extraire des caractéristiques de la parole propre au locuteur issus de l'audio associé comme la fréquence fondamentale (hauteur de la voix propre au locuteur), prosodie (variations mélodiques de la voix), formants (pics de fréquence correspondant aux résonances du conduit vocal), etc. Ces caractéristiques sont apprises grâce à nos différents ouvriers – *workers*.

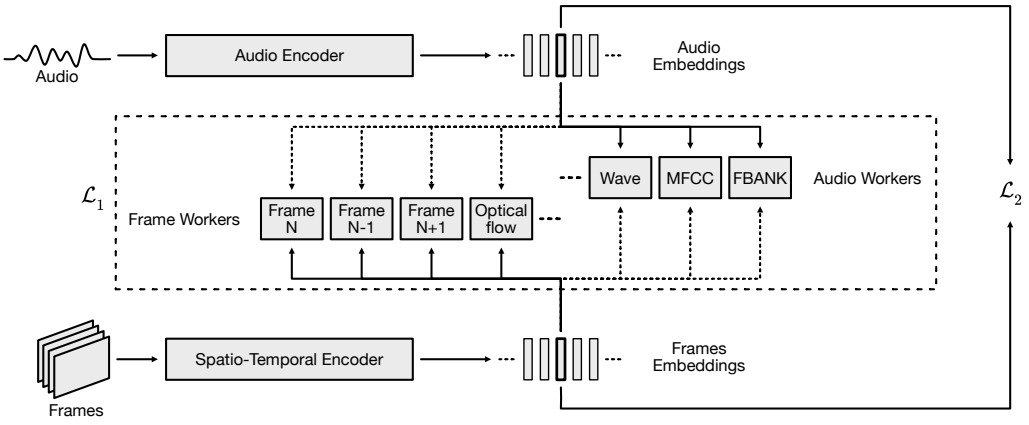


FIGURE 5.9 – Chaîne de traitements proposée comportant de multiples décodeurs pour la création de représentations audiovisuelles.

Sur l'explicabilité

Généralisation de notre chaîne de traitement vue dans le chapitre 5 En omettant les détails techniques, notre chaîne de traitement serait capable de produire une formule interprétable en résolvant une tâche prétexte – *proxy task*. Cela a été démontré en particulier dans le contexte de la prédiction de la justesse – *accuracy* –, où notre chaîne de traitement a montré sa capacité à produire une formule facilement compréhensible. Cela doit être maintenant appliqué à de nouvelles tâches afin d'en montrer son généralisabilité.

Cependant, il convient de souligner que l'interprétation de la formule générée par la méthode de programmation génétique (GP) a suscité notre capacité à interpréter le résultat en regard de la connaissance scientifique du domaine. Afin de rendre les éléments de la formule plus conformes aux formules mathématiques existantes dans la littérature, il pourrait être judicieux de comparer la régression symbolique basé sur la GP avec celle issue de l'apprentissage profond, lui-même entraîné sur des formules mathématiques existantes. Il serait même possible d'utiliser des techniques permettant de trouver des correspondances avec des équations de la littérature [5].

La causalité Nous avons vu dans les chapitres 4 et 5 que les heuristiques trouvées sont potentiellement liées à notre chaîne de traitement, et plus particulièrement aux statistiques extraites des représentations issues d'extracteurs de caractéristiques. Nous

avons pu constater que les variables qui ont été automatiquement intégrées dans notre formule ont un fort degré de pertinence pour l'estimation de la justesse – *accuracy* – obtenue. Mais, que pouvons-nous dire sur des tâches autres que la prédiction de la justesse ? De plus, il serait utile de s'assurer que la formule produite ne comprend que les éléments à l'origine du phénomène étudié, et non pas les conséquences de celui-ci. Cela garantirait que les résultats obtenus sont précis et fiables, et que la formule générée est appropriée pour l'étude en question.

Le concept qui se cache derrière ce problème est la causalité. Par définition, la causalité est le lien de cause à effet entre deux événements ou deux phénomènes. Elle consiste en l'influence que l'un peut avoir sur l'autre.

Autre constat intéressant, si nous extrayons des représentations d'images à l'aide d'un modèle tel que CLIP [4] et que nous entrainons un modèle de classification linéaire sur ces représentations, il est compliqué de prédire un attribut caractéristique concret tel que la couleur, contrairement à un attribut de contraste (basé sur des règles) tel que « est plus grand que ». Une hypothèse serait que ces attributs concrets ne seraient pas prédits, mais perçus indirectement dans la connaissance des classes détectées. Le vecteur de représentation n'engloberait donc pas directement les attributs caractéristiques. Pour illustrer les propos, prenons l'exemple d'un vecteur caractéristique représentant l'image d'un zèbre, notre modèle linéaire serait capable d'exprimer la différence avec un cheval, mais pas de décrire les caractéristiques de l'animal en lui-même tel que la présence de rayures noires et blanches. Or, il serait préférable que la classe soit prédite en fonction de ses attributs caractéristiques. De ce constat, nous vient la question suivante :

> Prédit-on un attribut en fonction de la classe détectée ou la classe en fonction des attributs observés ?

Plus exactement, nous souhaitons savoir s'il est possible de mesurer la qualité des représentations apprises, quant-à leur capacité à inférer des attributs directement ou non. À notre connaissance, peu d'articles de recherches se sont emparés de ce problème.

Vers les sciences cognitives et sociales

Un dernier domaine sur lequel je souhaiterais travailler est celui des sciences cognitives et sociales. Quand la science cognitive se concentre sur les processus qui sous-tendent l'apprentissage, la mémoire, la perception et le raisonnement, les sciences sociales, quant à elles, se concentrent sur l'étude des comportements et des relations sociales des individus et des groupes. Ces deux domaines se recoupent, car les processus cognitifs et sociaux sont étroitement liés dans les comportements humains. En particulier, cette corrélation se confirme notamment lors des premiers pas d'un enfant, où l'apprentissage repose sur une synergie entre ces deux domaines. Si nous reprenons le sujet initial de la thèse concernant la fusion multimodale, il est intéressant d'observer que durant les premières phases d'acquisition de connaissance chez l'enfant, celui-ci est incapable de créer une relation sémantique entre une perception auditive et visuelle.

4. Pour rappel, CLIP (Contrastive Language-Image Pre-Training) est un réseau de neurones entrainé sur une variété de paires (image, texte), permettant entre autres d'encoder des données, de prédire du texte à partir d'une image, etc. Ce modèle a été utilisé comme extracteurs de caractéristiques sur les corpus d'images dans les chapitres 4 et 5.

De mes observations, chaque modalité est traitée indépendamment. La fusion de celles-ci arrive que bien plus tard, sûrement par un mécanisme que l'on pourrait qualifier « d'apprentissage par renforcement ».

Il convient donc de ne pas écarter les modèles inspirés de la neuroscience, tels que les SNNs, sous prétexte qu'ils n'atteignent pas les performances obtenues avec les méthodes d'apprentissage profond – *deep learning*. En effet, compte tenu des caractéristiques intrinsèques du cerveau humain et de ses modèles, ces approches se révèlent parfaitement adaptées aux enjeux de demain.

Références

[1] B. Chamand et P. Joly. "Self-Supervised Spiking Neural Networks applied to Digit Classification". In : *Proceedings of the 19th International Conference on Content-Based Multimedia Indexing – CBMI 2022*. Graz (Austria) : ACM, 2022.

[2] B. Chamand et al. "Fine-tune your classifier : Finding correlations with temperature". In : *Proceedings of the IEEE International Conference on Image Processing – ICIP 2022*. Bordeaux (France) : IEEE Computer Society, 2022, pages xx-xx.

[3] Yuhuang Hu, Shih-Chii Liu et Tobi Delbruck. "v2e : From video frames to realistic DVS events". In : *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pages 1312-1321.

[4] Santiago Pascual et al. "Learning Problem-Agnostic Speech Representations from Multiple Self-Supervised Tasks". In : *Proc. of the Conf. of the Int. Speech Communication Association (INTERSPEECH)*. 2019, pages 161-165.

[5] Lukas Pfahler, Jonathan Schill et Katharina Morik. "The Search for Equations – Learning to Identify Similarities Between Mathematical Expressions". In : *Machine Learning and Knowledge Discovery in Databases : European Conference, ECML PKDD 2019, Würzburg, Germany, September 16–20, 2019, Proceedings, Part III*. Berlin, Heidelberg : Springer-Verlag, 2019, 704–718.

[6] Olivier Risser-Maroix et Benjamin Chamand. "What Can We Learn by Predicting Accuracy ?" In : *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*. 2023, pages 2390-2399.

[7] M C van Rossum. "A novel spike distance". en. In : *Neural Comput.* 13.4 (avr. 2001), pages 751-763.

[8] Edgar Schonfeld et al. "Generalized zero-and few-shot learning via aligned variational autoencoders". In : *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pages 8247-8255.

Publications

Conférences internationales

- Benjamin Chamand et Olivier Risser-Maroix. **What can we Learn by Predicting Accuracy ?**. *IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, 2023.
- Benjamin Chamand et Philippe Joly. **Self-Supervised Spiking Neural Networks applied to Digit Classification**. *International Conference on Content-Based Multimedia Indexing (CBMI)*, 2022.
- Benjamin Chamand, Olivier Risser-Maroix, Camille Kurtz, Philippe Joly et Nicolas Lomenie. **Fine-tune your classifier : Finding correlations with temperature** *IEEE International Conference on Image Processing (ICIP)*, 2022.

Workshops internationaux

- Benjamin Chamand et Olivier Risser-Maroix. **Finding Components of a Good Accuracy with XAI !**. *Advances in Interpretable Machine Learning and Artificial Intelligence (AIMLAI@CIKM)*, 2022.

Journées d'étude

- Benjamin Chamand et Philippe Joly. **Repeating Sound Detection with one Unsupervised Spiking Neuron**. *Journées NeuroSTIC*, 2019.

Rapports techniques

- Benjamin Chamand et Philippe Joly. **PyTorch SNN, a simulation tool for SNNs with STDP**. *Projet RAVI*, 2019.