Détection, anticipation, action face aux risques dans les bâtiments connectés

Adrien Legrand

Thèse de Doctorat

Mention Informatique

présentée à ***l'Ecole Doctorale en Sciences Technologie et Santé (ED 585)***

de l'Université de Picardie Jules Verne

par

Adrien LEGRAND

pour obtenir le grade de Docteur de l'Université de Picardie Jules Verne

Détection, anticipation, action face aux risques dans les bâtiments connectés

Soutenue le 29 Novembre 2019 après avis des rapporteurs, devant le jury d'examen :

M. Gilles Dequen, Professeur Université de Picardie Jules Verne, MIS	**[Président du jury]**
M. Philippe Preux, Professeur, Université de Lille, CRISTAL	**[Rapporteur]**
M. Christophe Rigotti, Maître de Conférences HDR, INSA Lyon, LIRIS	**[Rapporteur]**
M^me^ Catherine Pothier, Maître de Conférences, INSA Lyon, LIRIS	**[Examinatrice]**
M. Alain Cournier, Professeur, Université de Picardie Jules Verne, MIS	**[Directeur de thèse]**
M. Harold Trannois, Maître de Conférences, Université de Picardie Jules Verne, MIS	**[Co-encadrant]**

RÉSUMÉ

Ces dernières années, nous avons constaté une augmentation significative du nombre d'objets connectés et de smartphones. Cette augmentation, notamment liée à la baisse du coût de traitement et de connexion, devrait se poursuivre dans les prochaines années.

En effet, Gartner a estimé en 2018 que le nombre d'objets connectés passera de 14.2 milliards en 2019 à plus de 25 milliards en 2021 [Gartner, 2018]. Les conséquences de cette augmentation du nombre d'objets connectés soulèvent de nombreux challenges et opportunités dans des domaines variés, notamment en termes de traitement, de stockage et de bandes passantes.

Cette multitude d'objets connectés sera notamment source de données massives et hétérogènes. Ces données recèlent quantité de connaissances encore inexploitées dont l'identification et l'exploitation de ces connaissances augmenteront le potentiel d'intéraction de ces objets avec l'Homme et serviront à développer de nouveaux services. Ces services pourront aller du simple but récréatif à l'augmentation du confort et de la sécurité des utilisateurs.

Les bâtiments sont les principaux lieux de l'activité humaine, que ce soit dans le cadre professionnel ou personnel. Améliorer et optimiser les bâtiments en leur apportant une nouvelle intelligence est donc un enjeu conséquent. Au sein de ces bâtiments, de nombreux scénarios pouvant être source d'inconfort, de pertes financières, de pertes de temps... sont susceptibles de se dérouler, et ce de manière imprévisible. Doter un bâtiment de fonctionnalités permettant de détecter et de minimiser la durée de ces scénarios bénéficierait donc aux utilisateurs et au personnel ayant la charge dudit bâtiment. Cette thèse s'inscrit dans le cadre de l'exploration des nouvelles fonctionnalités qui pourront être développées à partir des données issues de bâtiments connectés.

Nous abordons, dans nos travaux, la thématique de la détection d'anomalies afin de contribuer au développement de ces fonctionnalités.

Après une phase exploratoire, nous avons sélectionné un modèle d'intelligence artificielle pouvant répondre au mieux aux problèmes spécifiques des données issues des bâtiments connectés : les réseaux de neurones artificiels auto-encodeurs [Goodfellow et al., 2016a]. Ce modèle, comparé aux méthodes de détection d'anomalies les plus sensibles utilisées actuellement (dispositifs détecteurs de fumée, systèmes d'alarme anti-intrusion...) a l'avantage de pouvoir détecter les relations existantes entre les données issues de divers objets connectés et permettent de détecter un panel de scénarios anormaux plus étendu.

Passé cette sélection, nous avons comparé des architectures d'auto-encodeurs permettant d'appréhender les données sous forme de séries temporelles. Nous proposons ensuite une nouvelle méthode qui permet d'améliorer les scores issus des auto-encodeurs. Cette méthode a donné des résultats empiriques probants sur les données issues de bâtiments connectés.

REMERCIEMENTS

La réalisation de cette thèse a été possible grâce au concours de plusieurs personnes à qui je souhaite témoigner toute ma reconnaissance.

Je voudrais tout d'abord adresser mes remerciements à mes encadrants académiques, Alain Cournier et Harold Trannois, pour avoir accepté de m'accompagner au cours de ces travaux. Je suis ravi d'avoir pu travailler en leur compagnie car, outre leur appui scientifique, j'ai toujours pu trouver conseils et soutien lors de l'élaboration de cette thèse.

Je remercie également mes encadrants professionnels, Arnaud Buisine et Gwennaël Buchet, qui ont rendu ce projet possible grâce à leur implication. Leur accueil au sein de l'agence de Zenika Lille m'a permis de trouver un cadre de travail exemplaire.

Je tiens à remercier Brad Niepceron pour avoir accepté de participer à ces travaux au travers de ses deux stages ainsi que pour toute l'aide et le soutien qu'il a pu me témoigner.

Je remercie Philippe Preux et Christophe Rigotti d'avoir accepté de participer à mon jury en tant que rapporteurs, ainsi que Gilles Dequen et Catherine Pothier en tant qu'examinateurs.

Je remercie toutes les personnes avec qui j'ai partagé mes études et notamment ces années de thèse. Mes derniers remerciements vont à Adeline Cayet qui a tout fait pour m'aider, qui m'a soutenu et surtout supporté dans tout ce que j'ai entrepris.

TABLE DES MATIÈRES

CHAPITRE 1

INTRODUCTION

Dans ce chapitre, nous présentons de manière générale le contexte et les éléments ayant motivé cette thèse. Nous détaillons les objectifs visés, les contributions apportées lors de ces travaux, ainsi que l'organisation de leur présentation.

1.1 Contexte

1.1.1 L'Internet des Objets

L'augmentation significative du nombre d'objets connectés a permis la popularisation du concept d'Internet des Objets, plus connu sous le nom de « Internet of Things » (IoT). En effet, divers cabinets d'expertises ont reconnu ce phénomène. Gartner, par exemple, a notamment estimé en 2017 le nombre d'objets connectés à 8.4 milliards et a prévu plus de 20 milliards d'objets d'ici 2020 [Gartner, 2017]. En 2018, Gartner a actualisé ses prévisions et a prévu un nombre de 25 milliards d'objets connecté en 2021 [Gartner, 2018]. De plus, comme le soulignait Gubbi [Gubbi et al., 2013] en 2013, l'IoT est encore au début de sa courbe d'adoption et la croissance du nombre d'objets connectés est donc susceptible d'évoluer dans l'avenir. Le concept de l'IoT n'est néanmoins pas une nouveauté et a été introduit par Kevin Ashton [Li et al., 2015] en 1999. Celui-ci le caractérise notamment par le fait de connecter des objets par radio-identification, plus souvent désignée par le sigle RFID (Radio Frequency IDentification).

Depuis l'introduction du concept, l'IoT et les objets connectés évoluent continuellement, tout comme leurs définitions. A l'heure actuelle, la définition formelle de l'IoT n'est pas encore stable et fluctue au rythme des évolutions technologiques. De nombreuses organisations travaillent encore sur sa standardisation[1]. La IEEE IoT Initiative [Minerva et al., 2015] a notamment établi un regroupement des caractéristiques récurrentes dans toutes les définitions que l'on peut trouver. Ainsi, l'IoT est défini de manière générique comme étant un réseau de réseaux,

[1]Parmi ces organisations se trouvent notamment l'European Telecommunications Standards Institute (ETSI), l'International Telecommunication Union (ITU), l'Institute of Electrical and Electronics Engineers (IEEE), l'Internet Engineering Task Force (IETF), la National Institute of Standards and Technology (NIST), l'Organization for the Advancement of Structured Information Standards (OASIS), ou encore le World Wide Web Consortium (W3C).

incluant les réseaux locaux d'objets physiques, permettant l'émergence de toutes sortes de nouveaux services. Pour offrir ces nouveaux services, l'IoT s'appuie sur quatre axes fondamentaux qui ont été développés dans [Chen, 2012] et [Khan et al., 2012] :

- l'observation, permettant une compréhension locale de l'environnement d'un objet ou de l'objet lui-même (usuellement cette tâche est réalisée par l'objet),
- le traitement, exploitant les connaissances issues de la phase de détection, effectué là où la puissance de calcul est disponible pour réaliser le service souhaité (typiquement un serveur distant, parfois sur l'objet lui-même),
- l'actionnement, permettant aux objets qui en ont la capacité de réagir aux ordres d'un service,
- la communication, permettant l'envoi de données détectées aux plateformes réalisant les traitements et l'envoi de données aux actionneurs.

Dans ce contexte, on peut catégoriser les nœuds de ces réseaux. Nous nous servons de diverses propositions d'architectures [Mrissa et al., 2015] [Sarkar et al., 2014] [Sarkar et al., 2014] afin de mettre en place les catégories suivantes :

- Les objets physiques connectés, tels que les capteurs ou les actionneurs.
- Les passerelles, permettant de faire le lien entre les réseaux physiques (reposant sur des technologies de transfert de données locales, peu coûteuses en énergie) et les réseaux typiques, utilisés pour les applications classiques (notamment Internet).
- Les objets virtuels, regroupant notamment des agrégats d'objets physiques ou d'autres objets virtuels en objets métiers (exemple : une maison peut être considérée comme un seul objet, mais est en réalité un regroupement de plusieurs pièces, qui elles-mêmes regroupent divers objets physiques).

Cette conceptualisation de l'IoT offre des capacités de dimensionnement, elle permet de modéliser des systèmes de petites tailles jusqu'à des systèmes très complexes ayant potentiellement une très grande taille (exemple notable : les villes connectées, aussi appelées « Smart Cities » [Naphade et al., 2011]).

De plus, les objets inclus dans cette idée d'IoT « scalable » auraient des capacités variables et dépendantes de leur cas d'utilisation initial, ceux-ci allant des simples interrupteurs connectés aux complexes systèmes de thermostats connectés contrôlant les chauffages d'un bâtiment. Ainsi, l'IoT soutient l'idée que le monde physique et le monde virtuel, représentés par les objets et par Internet, peuvent coexister pour améliorer la condition de vie des utilisateurs mais aussi la façon dont ils interagissent avec leurs environnements.

De nombreux travaux ont contribué à classifier ces objets, notamment les travaux de Pérez Hernández [Hernandez and Reiff-Marganiec, 2014]. Le dénominateur commun des définitions

montrées dans les travaux de Liu [Liu and Baiocchi, 2016] est que, pour être considérés comme « Thing » de l'IoT, les capteurs ou les actionneurs doivent avoir un moyen de communiquer sur un réseau, une capacité de stockage et une puissance de calcul minimale. Dans le cadre de cette thèse, nous prenons en compte les données venant de n'importe quel objet tant que celui-ci est capable de les fournir (soit directement par internet, soit via une passerelle) à une fréquence suffisante.

Nous nous sommes donc concentrés sur l'utilisation des données sans avoir eu à nous préoccuper de problèmes d'échantillonnages. De tels problèmes imposeraient en effet des études préalables ainsi que des pré-traitements avant de pouvoir utiliser les données [Ding et al., 2008].

Néanmoins, faire face au nombre considérable à venir d'objets connectés est source de divers défis [Khan et al., 2012] [Chen, 2012] [Stankovic, 2014]. Comme on peut le constater figure 1.1 ci-dessous, ces nouveaux défis appartiennent à plusieurs domaines très variés.

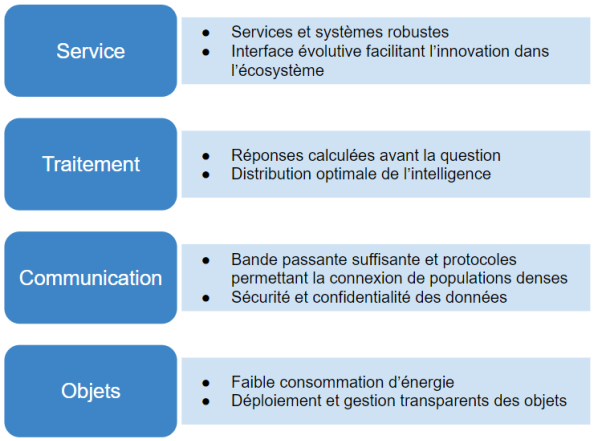

FIGURE 1.1 – Figure tirée de « Challenges and Opportunities of Internet of Things » dressant une liste des principaux domaines et leur challenges associés [Chen, 2012].

L'IoT trouve ses défis dans le développement de matériel sécurisé et efficace, mais aussi dans le développement des protocoles de communication associés, ainsi que dans le développement des applications qui les utilisent. De plus, ces protocoles doivent répondre à de multiples contraintes, notamment quant à la consommation d'énergie.

En effet, il est possible qu'un objet connecté ne soit pas destiné à être branché à une source d'alimentation externe, car tout le matériel inhérent à la connexion de cet objet doit usuellement être aussi léger et imperceptible que possible afin de ne pas dénaturer esthétiquement ou fonctionnellement l'objet initial. Ainsi, la « nouvelle connexion » de l'objet ne sera pas un obstacle à l'adoption de celui-ci. Le grand nombre d'objets connectés attendus est au cœur de tous les défis liés à l'ergonomie du logiciel ou à l'ergonomie du matériel. Ces défis impliquent le déploiement de capteurs lors de l'exécution du système IoT, la détection automatique et transparente d'objets intelligents au sein du réseau...

L'exploitation et la gestion de la masse de données à venir sont à la fois source de complications et d'opportunités donnant des chances d'améliorer l'expérience utilisateur et de créer

de nouveaux types d'applications. Ces applications peuvent impliquer la détection transparente de contexte physique, telle que la localisation de l'objet ou celle de ses voisins, et permettront également de doter les objets connectés de nouvelles fonctionnalités.

Dans nos travaux, nous allons chercher à tirer parti de la prolifération des appareils connectés et de la grande quantité de données qu'ils produisent pour détecter des situations inhabituelles dans le contexte des bâtiments connectés.

Nous supposons que les scénarios inhabituels que nous voulons détecter sont difficiles à prévoir. En effet, nous ne cherchons pas un moyen d'éviter ou de détecter les scénarios récurrents nécessitant une intervention humaine (ouverture d'une porte, augmentation ou diminution d'un thermostat ...), mais les scénarios inattendus impliquant des dommages potentiels. Ces scénarios peuvent impliquer des facteurs très différents tels qu'une activité humaine inhabituelle ou malveillante, catastrophe naturelle, défaillance matérielle... Par conséquent, sauf si un capteur ou un système de capteurs est dédié à chaque scénario possible (comme c'est le cas pour les incendies et les intrusions), ceux-ci ne peuvent pas être trivialement détectés.

Nous cherchons à détecter les anomalies avant que l'utilisateur ne ressente le besoin de s'y intéresser. Ainsi, notre travail de détection en « temps réel » s'inscrit dans le défi Traitement - La réponse précède la question (cf : figure 1.1).

1.1.2 Les bâtiments connectés

Comme nous l'avons évoqué précédemment, nous cherchons à détecter les scénarios anormaux au sein des bâtiments connectés. Nous devons donc nous intéresser avant tout aux caractéristiques que nous sommes susceptibles de trouver au sein des données que nous allons analyser.

Il existe, au sein d'une grande majorité de bâtiments connectés, un tronc commun concernant les types d'objets connectés. En effet, parmi tous les bâtiments connectés, une majorité de ceux-ci sont destinés à recevoir une activité humaine (nous considérons que les bâtiments qui ne sont pas couramment fréquentés sont des cas à part, nos travaux se s'appliquent donc pas à ce type de bâtiments). En ne considérant que les bâtiments sujets à l'activité humaine, on peut supposer que les types de capteurs et actionneurs destinés à la maintenance du confort et de la santé mais aussi les objets destinés au monitoring et à l'optimisation seront communs à ceux-ci. Parmi ces types d'objets connectés, on peut retrouver les capteurs de température (déjà utilisés de nos jours par exemple dans les thermostats standards) souvent associés à des radiateurs ou thermostats connectés, les capteurs d'humidité, les capteurs de luminosité pouvant par exemple être associés à des stores connectés, mais aussi des capteurs de consommation électrique (par exemple les prises connectées).

Une particularité relativement commune à tous ces objets est d'être statique. Les données remontées par ces capteurs deviennent par conséquent fortement liées à divers types de contextes. Ces contextes qu'ils soient spatiaux ou temporels, dépendent aussi parfois des valeurs d'autres capteurs (exemple : les valeurs de plusieurs capteurs de températures situés dans une même maison auront tendance à évoluer de manière similaire). Il nous est donc possible d'établir des

correspondances entre les scénarios et les différents types d'anomalies susceptibles de se produire au sein d'un jeu de données (nous développerons cet aspect partie 2.1).

1.2 Objectifs

De nos jours, il existe déjà au sein de bâtiments, pour les scénarios anormaux les plus critiques, des systèmes de détection et de réaction. Ceux-ci embarquent des capteurs et des composants leur permettant de réagir de manière spécifique au scénario concerné. Parmi ces systèmes dédiés, on trouve par exemple les systèmes de détection de fumées, dont le but est de pallier le scénario de l'incendie domestique. Ils intègrent des capteurs spécifiques et un comportement à adopter en cas d'alerte (sirène d'alerte, notification à l'utilisateur ou aux pompiers selon le matériel installé...). On peut aussi trouver les systèmes de détection d'intrusion, qui, eux aussi, s'utilisent pour un scénario critique particulier.

Il existe aussi au sein de la littérature divers travaux concernant des scénarios anormaux ou des cas d'utilisations spécifiques. Parmi les cas traités, on trouve notamment la détection de chutes humaines grâce aux objets dits « wearable » [Oniga and Sütő, 2014], des scénarios anormaux de plus grandes échelles comme les inondations [Mitra et al., 2016] ou encore les problèmes de trafic routier [Akbar et al., 2015]... La majorité de ces travaux portent sur des systèmes dédiés à un type de scénario spécifique ou nécessitant la mise en place plus ou moins complexe de règles, ainsi qu'une certaine expertise concernant le domaine touché.

Nous constatons donc que, de nos jours, chaque scénario anormal est traité de manière indépendante et nécessite un système qui lui est dédié. Chaque système dédié a émergé pour répondre à un besoin issu de la criticité du scénario auquel il s'adresse et a nécessité un temps d'étude et d'élaboration.

L'objectif de nos travaux est de contribuer à un système qui, tirant parti du nombre d'objets connectés en augmentation, saurait apprendre afin d'être capable de détecter n'importe quel scénario comportant un risque. Cette détection donnerait alors la possibilité d'anticiper et de réagir face à celui-ci.

Nous illustrons cette notion avec l'exemple suivant : prenons le cas d'un utilisateur (propriétaire, habitant, gestionnaire, etc.) souhaitant surveiller l'état (système de chauffage, isolation, consommation énergétique, etc.) d'un bâtiment connecté. Dans ce contexte, l'objectif serait que l'utilisateur puisse détecter des anomalies physiques à travers un système IoT. Ces anomalies peuvent être caractéristiques de problèmes structurels soudains dans le bâtiment, d'appareils défectueux (radiateurs défectueux, fuites d'électricité ou d'eau, etc.), de changements soudains dans les habitudes des utilisateurs (problèmes de comportement, intrusions, etc.). La détection de tels scénarios permettrait à l'utilisateur de réagir à ces anomalies et de limiter les dommages potentiels (financiers, humains, écologiques, etc) pouvant en découler.

Un des premiers enjeux de cette thèse fut donc de contribuer à un modèle générique capable de détecter un maximum d'anomalies ayant lieu dans un bâtiment.

Actuellement, les systèmes de détection d'anomalies découlent soit d'un besoin établi, correspondant à des normes de sécurité (détection d'incendies, détection d'intrusion...), soit d'une volonté d'investiguer un type de scénario anormal précis (volonté de diminuer les coûts énergétiques, sécurisation d'un bâtiment dans un contexte particulier...). Le second cas est généralement négligé, du fait du temps et du coût à y consacrer. En effet, en cas de volonté d'investigation, l'effort de listage, d'évaluation et de pondération de la criticité des différents scénarios anormaux aurait un coût et nécessiterait potentiellement un temps considérable. De plus, l'aboutissement d'une telle réflexion donnerait soit une liste incomplète, comportant uniquement les scénarios les plus sensibles, soit une liste se voulant exhaustive et comportant de potentiels oublis.

Un second enjeu de cette thèse était la forte diminution du temps à consacrer pour détecter chacune des anomalies sélectionnées et traitées indépendamment dans un cas classique. Effectivement, nous nous plaçons dans un contexte où le bâtiment est « fortement » connecté. Si un modèle fiable est développable à partir des données remontées par des capteurs déjà présents, il n'y aurait aucun coût lié au temps de développement de matériel et d'installation. De plus, l'aboutissement d'un modèle logiciel générique réduirait considérablement le temps de développement par rapport à celui requis par de multiple systèmes traitant chacun un scénario anormal.

Répondre efficacement à ces enjeux serait un premier pas vers un système économique, autonome et adaptatif, proactif pour la détection de risques. Comme nous allons le développer dans la suite de cette thèse, nous nous sommes dirigés vers des algorithmes d'apprentissage automatique afin d'investiguer leurs capacités à s'intégrer dans un tel système.

Ning et al. ont comparé les différents capteurs d'un système IoT aux terminaisons nerveuses d'un être humain [Ning and Wang, 2011], les points de ramifications à des nœuds de contrôles et suggère un système de décision central, ici comparé au cerveau. Cette analogie est intéressante étant donné que nous nous sommes dirigés vers une architecture centralisant les données et les traitements, ces derniers consistant essentiellement en des travaux d'apprentissage et d'inférence basés sur les réseaux de neurones artificiels.

1.3 Contributions

Cette thèse s'articule autour de trois contributions principales :

- Un **travail exploratoire** consistant à rechercher, parmi les algorithmes de détection d'anomalies existant, les modèles les plus pertinents pour traiter des données issues de bâtiments connectés. Une proposition de méthode d'évaluation de modèle de détection d'anomalie sur un jeu de données partiellement labélisé ainsi qu'une comparaison de types d'architectures de réseaux de neurones (auto-encodeur convolutionnel et auto-encodeur récurrent) ont été réalisées.
- Une **proposition d'optimisation des modèles auto-encodeurs** utilisés pour la détection d'anomalies. Une étude de plusieurs fonctions de score a été réalisée dans le but de com-

parer les fonctions classiques, des fonctions plus récentes basées sur l'approximation de réseaux de neurones bayésiens, ainsi que des fonctions que nous avons proposées, intégrant l'incertitude de prédiction. Cette première étude a été réalisée sur des données « simples » n'ayant pas de rapport avec l'IoT.

- Le **test et la validation de l'étude** de la méthode d'optimisation citée précédemment dans le cadre de données spécifiques à des bâtiments connectés.

1.4 Organisation de la thèse

Dans le chapitre 2, nous apporterons un contexte technique et théorique pour les notions fondamentales utilisées tout au long de ce mémoire ainsi que nos réflexions initiales. Les notions abordées engloberont l'apprentissage, la détection d'anomalies, et les réseaux de neurones artificiels ainsi que leurs diverses déclinaisons telles que les auto-encodeurs, les réseaux récurrents ou encore convolutionnels.

Le chapitre 3 sera consacré à une étude comparative où les résultats d'auto-encodeurs convolutionnels et d'auto-encodeurs récurrents seront analysés dans le contexte de la détection d'anomalies au sein de données issues de bâtiments connectés.

Le chapitre 4 présentera une nouvelle méthode d'optimisation des scores issus d'auto-encodeurs standards utilisés pour la détection d'anomalies. Ces méthodes sont basées sur la prise en compte de l'incertitude du modèle lors du calcul des scores.

Le chapitre 5 détaillera l'utilisation de la méthode précédemment citée avec des auto-encodeurs récurrents en considérant le cas d'utilisation de la détection de scénarios anormaux au sein de bâtiments connectés.

Enfin, le chapitre 6 conclura cette thèse et présentera les perspectives de travaux futurs.

CHAPITRE 2

ETAT DE L'ART

Comme nous allons le voir dans la suite de ce chapitre, les anomalies sont des phénomènes rares, sous-représentées au sein d'un jeu de données, et dues à des facteurs incontrôlés. Dans la plupart des cas, ces facteurs sont considérés comme indépendants par rapport au sujet traité. Dans ce contexte, les scénarios anormaux se déroulant au sein d'un système IoT sont considérés comme des données inattendues, et donc imprévisibles. Nous nous sommes donc, dans cette thèse, concentrés sur la détection de scénarios anormaux.

Ce chapitre vise à apporter un contexte pour les éléments de fonds utilisés tout au long de cette thèse ainsi qu'à expliquer les réflexions qui nous ont mené aux choix préalables effectués. Ces éléments incluent notamment l'apprentissage automatique, la détection d'anomalies, les réseaux de neurones artificiels ainsi que les auto-encodeurs. Les notions spécifiques, telles que les auto-encodeurs à convolution, les auto-encodeurs récurrents ou encore l'incertitude au sein des réseaux de neurones, seront abordées et développées au fur et à mesure dans les chapitres suivants, pour se concentrer ici sur les notions transverses.

2.1 Anomalies théoriques et analogies avec l'IoT

Les anomalies sont considérées comme des motifs de données qui ne se conforment pas à une notion prédéfinie de comportement normal [Pimentel et al., 2014]. Nous pouvons trouver dans la littérature une classification des anomalies selon différents critères. La prise en compte de cette classification et des travaux en amont de celle-ci nous a permis de nous orienter dans le choix du modèle à utiliser. Chandola et al. décrivent notamment trois catégories d'anomalies [Chandola et al., 2009] :

- **Anomalies de valeurs (Peak Anomaly)** - C'est le type le plus simple d'anomalie, ils font référence à des points isolés du reste du jeu de données. Les anomalies de valeurs (aussi appelées anomalies de pics) comportent, pour au moins une des caractéristiques représentant ce point, une valeur très différente par rapport aux autres points du jeu de données. Dans le contexte des valeurs de capteurs de bâtiments connectés, il est peu probable que de telles anomalies apparaissent sauf en cas de malfonction d'un capteur (comme illustré par la figure 2.1, exemple issu du jeu de données REFIT [Firth et al.,

2017] utilisé lors de nos expérimentations, Personalised Retrofit Decision Support Tools for UK Homes Using Smart Home Technology, décrit section 3.3.1). En effet, étant donné l'inertie présente dans la majorité des capteurs, il est rare qu'une anomalie de type pic soit représentée par une seule instance de donnée isolée. Une telle anomalie sera généralement reflétée par un nombre relativement réduit d'instances (l'exemple figure 2.2 représente une telle anomalie et est aussi tiré du jeu de données REFIT).

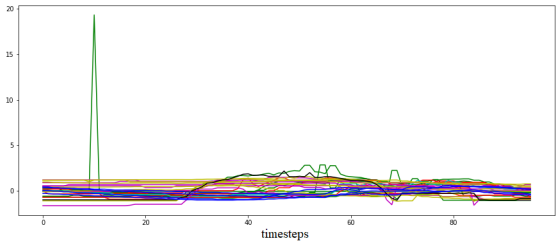

FIGURE 2.1 – Illustration d'une anomalie de valeur tirée du jeu de données REFIT. Chacune des courbes superposées correspond aux données de capteurs d'une maison pour une journée donnée. Afin de mieux visualiser l'anomalie, nous avons centré-réduit (cf : section 2.4.6) les données de tous les capteurs. Sans avoir besoin de s'intéresser aux capteurs, on constate un point unique ayant une valeur bien supérieure au reste ($\sim$20 pour un écart type à 1 et une moyenne de 0). Un point isolé indique une absence d'inertie et donc, probablement, la malfonction du capteur concerné ou une perturbation lors de la mesure.

- **Anomalies contextuelles** - Ici, les anomalies sont caractérisées comme des instances individuelles anormales dans un contexte spécifique. Cette notion de contexte doit être induite auparavant par la structure des données et doit faire partie de la formulation du problème [Chandola et al., 2009]. Un comportement contextuel anormal peut être identifié par le comportement d'une instance de données dans un contexte particulier. Une instance de données peut être une anomalie contextuelle pour un contexte donné, mais apparaître comme un comportement normal dans un contexte différent. En prenant un jeu de données représentant l'évolution de la température intérieure tout au long de l'année, une anomalie contextuelle peut par exemple être identifiée par une valeur anormale élevée en hiver qui pourrait être totalement normale en été. Une anomalie contextuelle est visible dans la figure 2.3, qui présente un exemple du jeu de données REFIT.

- **Anomalies collectives** - Ce type d'anomalie est susceptible d'apparaître pour les données séquentielles, dont les valeurs ne sont pas complètement indépendantes dans le temps. Si un groupe d'instances de données est anormal par rapport au reste de l'ensemble de données, le groupe est considéré comme une anomalie collective. Ce groupe d'instances, formant un « motif anormal », ne comporte pas nécessairement d'anomalies de valeurs et peut passer inaperçu si l'on ne prend pas en compte la dimension séquentielle des données. L'inverse est aussi possible : une anomalie de pic peut être révélée d'une part par les valeurs des capteurs mais aussi par les variations inhabituelles engendrées. Dans notre cas, une majorité des anomalies collectives sera souvent apparentée aux catégories d'anomalies de valeurs et contextuelles (comme c'est le cas figure 2.2).

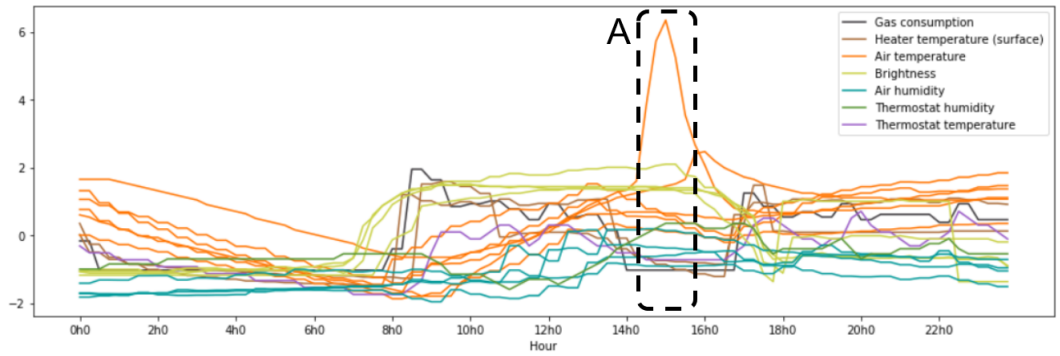

FIGURE 2.2 – Pic de température (les variables ont été centrées-réduites) : tous les capteurs de température de l'air sont corrélés, mais vers 15h, un capteur cesse d'être corrélé avec les autres (anomalie contextuelle) et affiche une grande valeur inhabituelle (anomalie de pic / collective) pendant 1h30. La fenêtre anormale est dans le bloc A.

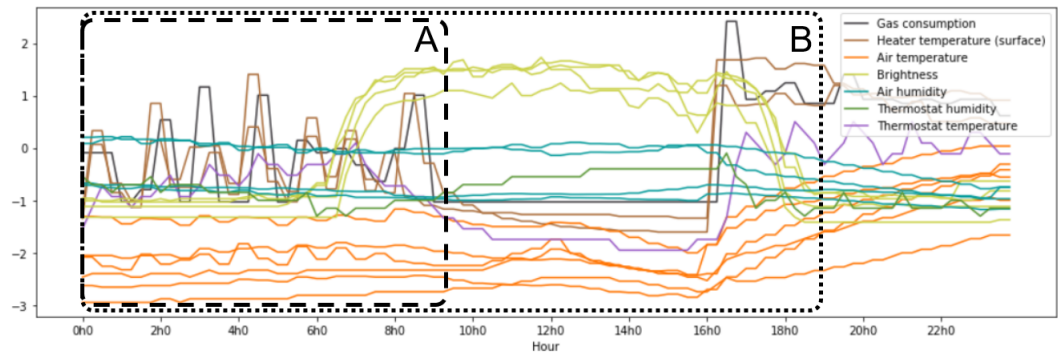

FIGURE 2.3 – Basse température malgré le fonctionnement des chauffages (les variables ont été centrées-réduites) : Cette fenêtre montre une activité des chauffages et de la consommation de gaz pendant la nuit et la température reste à son point le plus bas de l'ensemble de données (anomalies de valeurs/pic et anomalies contextuelles). Les anomalies de valeurs sont dans le bloc B et les anomalies contextuelles (basses températures malgré le fonctionnement des chauffages) dans le bloc A.

De manière générale, nous pouvons, à notre connaissance, classifier tous les scénarios anormaux susceptibles d'apparaître au sein des bâtiments connectés dans ces catégories d'anomalies. Nous nous réfèrerons donc à ceux-ci en les appelant simplement « anomalies » par la suite.

2.2 Apprentissage

Comme expliqué dans la partie précédente, les anomalies sont présentes en minorité au sein d'un jeu de données et peuvent être potentiellement très diverses. De plus, nous souhaitons contribuer à un système générique, pouvant être utilisé quelque soit le bâtiment. Nous nous sommes donc orientés vers une méthode capable de déterminer le comportement normal des données avant de se pencher sur les anomalies. La détermination de cette notion de normalité passe, pour certains algorithmes de détection d'anomalies, par une phase d'apprentissage. Ainsi,

nous nous concentrons sur les algorithmes de détection d'anomalies faisant partie d'une branche de l'intelligence artificielle appelée Apprentissage automatique (plus connue sous le nom de « machine learning », ou ML).

Mitchell apporte une définition succincte de l'apprentissage [Mitchell, 1997] : « On dit d'un programme qu'il apprend d'une expérience E si, admettant une ou plusieurs tâches T et indices de mesures de performances correspondantes P, ses performances face aux tâches T, mesurés par P, augmentent avec l'expérience E. ». C'est de cette capacité que bénéficient les algorithmes d'apprentissage automatique.

Cette partie vise, au travers de divers exemples et explications de tâches T, d'indices de performances P et d'expériences E, à donner un contexte au machine learning et à positionner les algorithmes de détection d'anomalies au sein de celui-ci.

2.2.1 La tâche, T

L'intérêt de l'apprentissage automatique réside en partie dans sa capacité à nous permettre d'adresser des problèmes qui seraient difficiles et/ou chronophages à résoudre avec des programmes « classiques » (on inclut dans ce terme tous les programmes nécessitant une formalisation explicite de tous les aspects de la tâche à traiter).

Dans le contexte de l'apprentissage automatique, l'apprentissage ne constitue pas en lui-même la tâche T. L'apprentissage est un moyen utilisé pour éventuellement réaliser la tâche. Par exemple, nous cherchons à détecter les anomalies au sein d'un bâtiment connecté.

De nombreux types de tâches peuvent être résolues grâce au machine learning. Les plus communes sont les suivantes :

- **La classification** : Ce type de tâche est l'un des plus fréquemment rencontré. Dans ce type de tâche, on demande au programme de spécifier à quelles catégories certaines entrées appartiennent [Krizhevsky et al., 2012a] [Taigman et al., 2014].

- **La régression** : Dans ce type de tâche, on demande au programme de prédire une ou plusieurs valeurs numériques en fonction de l'entrée qui lui est donnée [Smola and Schölkopf, 2004] [Huang et al., 2011]. Ce type de tâche peut être considérée comme similaire à la classification (association d'une sortie pour un individu en entrée), exception faite que l'on demande à l'algorithme une donnée d'un format différent : une ou plusieurs données discrètes (exemple : un numéro de catégorie) lors d'une classification et une ou plusieurs données continues (exemple : un score, un prix, etc.) lors d'une régression.

- **Le ranking** : Ce type de tâche correspond à l'attribution d'une priorité à un individus dans le but de déterminer un ordre (rank) parmi les données. Le système apprend une fonction de notation où les éléments mieux classés devraient avoir des scores plus élevés. Divers cas d'utilisation peuvent bénéficier d'une telle tâche (ex : ordonner les liens d'une page web, réaliser de la recommandation ou de la suggestion de produits...) [Haveliwala, 2003] [Liu and Shih, 2005].

- **Le clustering** : Le clustering (aussi appelée classification non supervisée) est une tâche visant à diviser un ensemble de données en plusieurs regroupements (clusters) selon des critères de proximités. Le but d'une telle tâche est d'obtenir des regroupements les plus homogènes possibles et que chacun de ces regroupements soient bien différenciés [Johnson, 1967] [Estivill-Castro, 2002].

- **La transcription** : Dans cette tâche, on demande au système d'apprentissage d'observer une représentation relativement déstructurée des données et de transcrire les informations en une forme discrète. Cette tâche a lieu notamment dans le cas d'un traitement de signal, par exemple lors de la transcription audio pure. On demande alors au programme de traiter un signal sonore. Le système doit alors apprendre à traduire ce signal en séquences de caractères ou en identificateur de mots permettant ainsi de l'interpréter [Hinton et al., 2012a].

- **La synthèse / échantillonnage** : Lors de ce type de tâche, on demande au programme de générer de nouveaux exemples (individus) similaires aux données déjà existantes [Luo et al., 2013]. La synthèse ou l'échantillonnage en apprentissage automatique peuvent être utiles lorsque la génération manuelle de données fidèles se révèlerait trop coûteuse ou demanderait trop de temps.

- **Le débruitage** : Dans ce type de tâche, on donne typiquement à l'algorithme des exemples d'individus corrompus (généralement issus d'un jeu de données standard, corrompus par un processus inconnu de l'algorithme). Le programme doit prédire la version « propre » de l'individu en fonction de sa version corrompue [Xie et al., 2012].

- **La déduction de valeurs manquantes** : Dans ce type de tâche, on donne au programme d'apprentissage un nouvel individu, mais dont une ou plusieurs entrées x_i sont manquantes. On attend donc du modèle entraîné une prédiction de x complète, où une estimation des entrées manquantes est fournie.

- **La détection d'anomalies** : Ce type de tâche nous concerne directement. Le but de cette tâche est de déterminer, au sein d'un jeu de données, les individus répondant aux critères d'une anomalie. La tâche consiste donc à attribuer, au sein d'un jeu de données existant, un score reflétant la probabilité qu'un individu du jeu de données soit une anomalie ou non, ou un label indiquant directement si, au regard des critères de l'algorithme, un individu est une anomalie ou non [Chandola et al., 2009]. Nous développons cette notion dans la partie 2.2.4.

Cette liste n'est pas exhaustive mais donne néanmoins une idée de la variété de tâches réalisables par apprentissage. Nous rappelons que tous les algorithmes de détection d'anomalies ne sont pas des algorithmes d'apprentissage automatique. Nous nous concentrons, dans ces travaux, sur les méthodes de détection d'anomalies nécessitant une phase d'apprentissage.

2.2.2 L'expérience, E

Les algorithmes peuvent être, de façon générale, catégorisés en tant que supervisés, non-supervisés, ou semi-supervisés, en fonction du type d'expérience qu'ils éprouveront pendant le processus d'apprentissage [Russell and Norvig, 2009].

L'apprentissage non-supervisé concerne les algorithmes qui, lors de leurs entraînements, sont exposés à des jeux de données constitués d'individus connus, représentés par des caractéristiques (aussi appelées « features », chaque caractéristique représente une dimension du jeu de données). Les individus ne sont pas associés à une sortie attendue du modèle. L'utilité de tels algorithmes est d'apprendre les propriétés et la structure des données.

Exemple : Dans le cas où on chercherait à attribuer des classes au sein d'un jeu de données, on peut utiliser un algorithme de clustering afin de déterminer des groupes d'individus relativement similaires (cf : exemple figure 2.4).

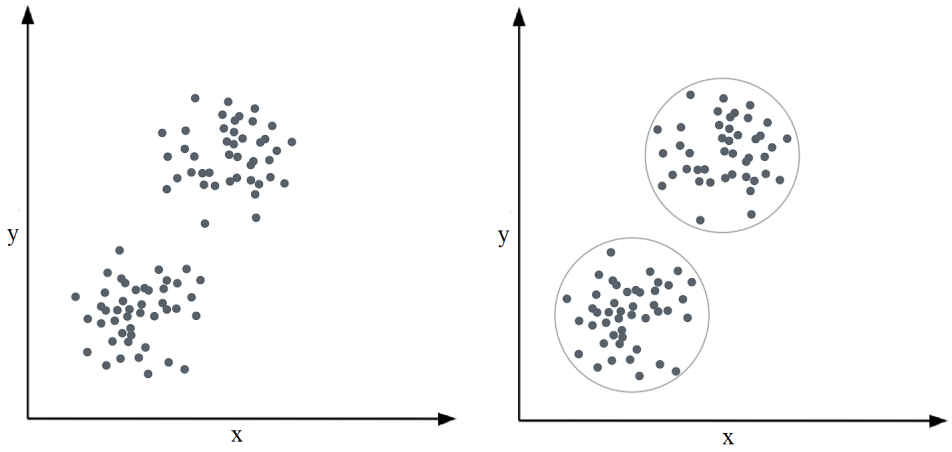

FIGURE 2.4 – Exemple graphique de deux groupes de données à deux dimensions constitués par clustering.

L'apprentissage supervisé concerne les algorithmes sont exposés à des jeux de données constitués, de la même manière que les jeux de données utilisés par les algorithmes d'apprentissage non-supervisé, d'individus. La différence principale est que chaque individu est associé à la sortie attendue du modèle. Cette sortie peut être soit un label, soit un score (plus communément appelé « target »). Exemple : Contrairement au clustering, les classes sont connues lors d'une tâche de classification. Le modèle apprend alors à estimer une fonction qui attribuera la classe des prochains individus (cf : figure 2.5).

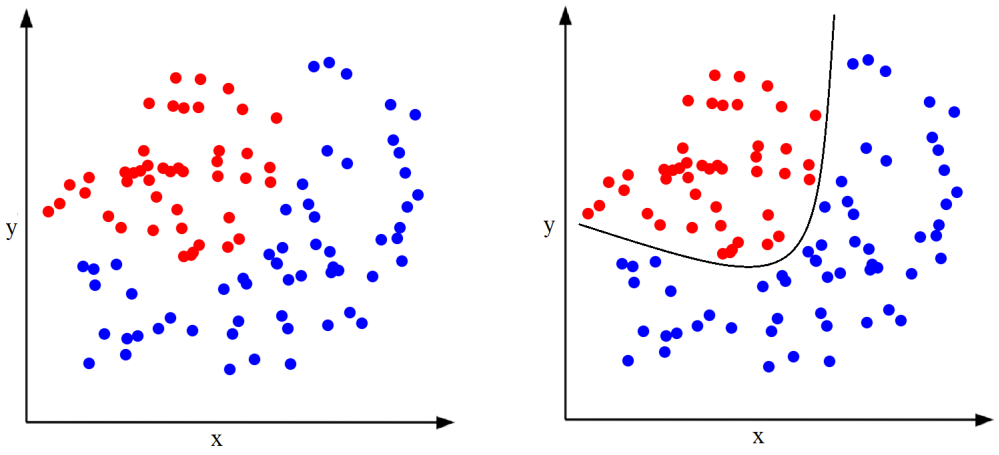

FIGURE 2.5 – Exemple de problème résolu de classification binaire.

De manière plus générale, l'apprentissage non-supervisé implique l'observation de plusieurs exemples d'un vecteur aléatoire x et la tentative d'apprentissage de la distribution de probabilités de $p(x)$ ainsi que les propriétés pertinentes de cette distribution [Bengio et al., 2013]. L'apprentissage supervisé quant à lui implique d'abord l'observation de vecteurs aléatoires x et de la valeur ou du vecteur associé y, et ensuite le fait d'apprendre à prédire y à partir de x, usuellement en estimant $p(y|x)$.

Le terme d'apprentissage supervisé vient du fait que, lors de l'apprentissage, la cible y est connue et portée par un instructeur à la connaissance du système qui, de ce fait, est guidé dans ce qu'il doit réaliser [Hush and Horne, 1993]. Dans l'apprentissage non-supervisé, il n'y a ni instructeur ni cible et le système doit apprendre à trouver du sens dans les données sans être guidé.

Comme nous allons le voir pour notre cas, il est parfois possible d'utiliser apprentissage supervisé ou non-supervisé pour répondre à un même problème.

2.2.3 La mesure de performance, P

Afin d'évaluer les capacités d'un algorithme de machine learning, il convient de mettre en place une mesure quantitative de sa performance. Cette mesure de performance P dépend de la tâche T que le système tente de réaliser.

Pour les tâches telles que la classification, la classification avec des valeurs manquantes, la transcription, ou plus généralement toute tâche nécessitant un modèle produisant un ou plusieurs résultats discrets (un nombre fini de classes ou de catégories, une combinaison de celles-ci. . .), c'est souvent la précision qui est mesurée. La précision est la proportion d'individus pour lesquels le modèle a réussi à produire la bonne sortie. Nous pouvons obtenir une information équivalente en mesurant le taux d'erreur, c'est-à-dire la proportion d'exemples pour lesquels le modèle a produit une mauvaise sortie. Il est aussi possible de produire plusieurs indices de pré-

cision afin de déterminer la propension de chaque catégorie à inclure des individus mal classifiés par le modèle (cf : sensibilité et spécificité présentés à la fin de la partie 2.3.2).

Pour les tâches telles que la régression, le débruitage, la déduction de valeurs manquantes, ou plus généralement toutes tâches nécessitant un modèle produisant un ou plusieurs résultats numériques, il convient de trouver un moyen de mesurer l'erreur entre la sortie du modèle et le résultat attendu. Afin de mesurer cette erreur (communément appelée « Loss »), de nombreuses fonctions adaptées à divers cas d'utilisation existent. Parmi les méthodes les plus communément utilisées figurent les fonctions d'erreur quadratique moyennes (Mean Squared Error - MSE), la racine de l'erreur quadratique (Root Mean Squared Error - RMSE), l'erreur absolue moyenne (Mean Absolute Error - MAE), ou encore l'erreur logarithmique moyenne (Mean Squared Logarithmic Error - MSLE) [Mood et al., 1974].

Ces différentes fonctions d'erreur correspondent aux formules suivantes :

Mean squared error	$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$
Root mean squared error	$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$
Mean absolute error	$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}\lvert(y_i - \hat{y}_i)\rvert$
Mean Squared Logarithmic Error	$\text{MSLE} = \frac{1}{n}\sum_{i=1}^{n}(log(y_i + 1) - log(\hat{y}_i + 1))^2$

Avec n le nombre d'individus, y_i la sortie attendue pour l'individu i, et $\hat{y}_i$ la sortie produite par le modèle pour l'individu i.

Nous notons, pour MSLE, que nous prenons en compte $log(y_i + 1)$ et $log(\hat{y}_i + 1)$ car y_i et $\hat{y}_i$ peuvent potentiellement être inférieurs à 1, voir même proches de 0 (étant donné que $\lim_{\substack{x \to 0 \\ x > 0}} log(x) = -\infty$, une mesure associée à de telles valeurs biaiserait le calcul de la distance).

Au cours de nos expérimentations, nous avons utilisé la plupart du temps la fonction d'erreur quadratique et, dans de plus rares cas, la fonction d'erreur logarithmique.

Les précédents paragraphes concernaient essentiellement les indices de performance utilisés pour évaluer des tâches résolubles par des algorithmes d'apprentissage supervisés. La mesure de la performance de système d'apprentissage non-supervisé est quant à elle plus délicate à mettre en place et requiert une connaissance approfondie de la ou des tâches adressées à la base. Effectivement, les données utilisées n'ayant ni label ni score associé, la mesure de la performance devient alors plus complexe que de calculer le taux d'erreur ou l'écart entre la sortie du modèle et la cible attendue. Afin d'illustrer cet aspect, nous imaginons une tâche où l'on souhaiterait réduire la taille de données trop volumineuses (nous rappelons que tous les agorithmes de réduction de dimension ne sont pas des algorithmes d'apprentissage automatique). Plusieurs variantes de mesure de performance sont possible :

- Dans le cas où les données doivent être reconstruites avec précision après la compression, l'évaluation serait à la fois portée sur la capacité d'un modèle à réduire la dimension de la donnée et mais plus d'importance serait accordée à la donnée post-reconstitution.

  La partie de l'évaluation prenant en compte le calcul de l'écart entre les données post-reconstitution et les données initiales pourrait s'apparenter à des méthodes d'apprentissage supervisé car le « but » est connu. Cet exemple souligne bien la limite floue existant entre ces catégories d'apprentissage.

- Dans le cas où le but de l'algorithme serait de compresser le plus possible les données, l'évaluation serait similaire à la précédente mais accorderait plus d'importance au taux de réduction de dimension.

- Dans le cas où nous cherchons à épurer le jeu de données initial de caractéristiques superflues, la méthode d'évaluation serait donc de déterminer à quel point les caractéristiques restantes seraient représentatives des individus.

Cette liste d'exemples n'est évidemment pas exhaustive mais souligne la complexité et la réflexion à accorder à la méthode d'évaluation d'une tâche résoluble par apprentissage non-supervisé.

Il est important de différencier la capacité d'un modèle à avoir de bonnes performances lors de l'entraînement et sa capacité à avoir de bonnes performances sur de nouvelles données. La mesure des performances permet d'évaluer un modèle au fur et à mesure de son entraînement et d'avoir une idée de la qualité de celui-ci (généralement, un phénomène de convergence vers une erreur minimum a lieu). Néanmoins, il convient, lors de la préparation des données, de garder un jeu de test afin d'évaluer les capacités du modèle face à de nouvelles données.

La prise en compte de la performance lors de l'entraînement et dans la phase de test permettra de faire face à des problèmes très connus de l'apprentissage qui sont le sur-apprentissage et le sous-apprentissage (plus communément appelés « overfitting » et « underfitting » respectivement [Lawrence and Giles, 2000]).

Le sous-apprentissage est un problème ayant lieu lorsque que le système d'apprentissage répond mal à la tâche à résoudre car il n'a pas réussi à approximer la fonction attendue. Les performances lors de l'entraînement et lors de la phase de test sont toutes les deux mauvaises. La cause typique est le le sous-dimensionnement du modèle.

Le sur-apprentissage est un problème ayant lieu lorsque le système apprend très bien à partir des données d'entraînement mais peine à généraliser ses résultats. La performance semble généralement excellente pendant la phase d'entraînement mais moins bonne voir mauvaise pendant la phase de test. La cause typique est l'entraînement d'un modèle surdimensionné.

Les figures données en exemple (2.6 à 2.8) illustrent ces phénomènes sur des jeux de données simples avec un problème visualisable facilement car en 2 dimensions.

Les phénomènes sont applicables à des données et des problèmes plus complexes. Nous avons donc pris ces risques en considération lors de toutes les étapes de nos expérimentations.

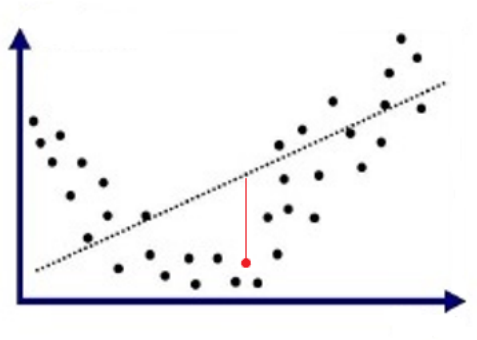

FIGURE 2.6 – Exemple de sous-apprentissage lors d'un problème de régression (illustration avec une donnée en phase de test et l'écart avec la donnée attendue correspondante en rouge). Cette figure et les deux figures ci-dessous sont tirées de [Liew, 2019]

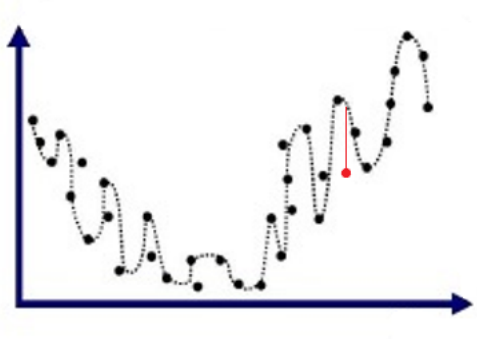

FIGURE 2.7 – Exemple de sur-apprentissage lors d'un problème de régression (illustration avec une donnée en phase de test et erreur correspondante en rouge).

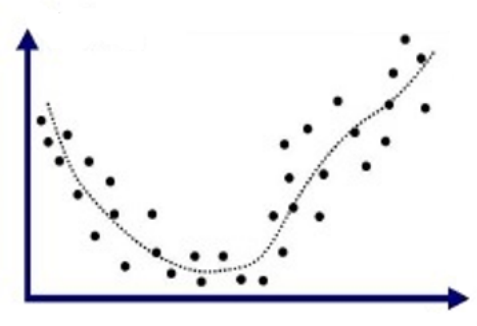

FIGURE 2.8 – Exemple de compromis acceptable entre sur-apprentissage et sous-apprentissage. La courbe en pointillé représente une approximation correcte de la fonction répondant au problème de régression.

2.2.4 La détection d'anomalies

Nous allons résumer dans cette sous-partie les enjeux de la détection d'anomalies, développer les notions inhérentes à cette tâche et énoncer dans ce contexte les spécificités des données provenant des bâtiments connectés.

La détection des anomalies peut être utilisée dans une grande variété d'applications telles que la détection des fraudes pour les cartes de crédit, les assurances ou les soins de santé, la détection des intrusions pour la cybersécurité, la détection des pannes, etc.

L'importance de la détection des anomalies est due au fait que les anomalies dans les données reflètent souvent des informations exploitables critiques. Par exemple, un motif anormal se trouvant dans des données de trafic réseau peut signifier qu'un serveur piraté envoie des données à un destinataire non autorisé [Kumar, 2005]. Dans le domaine médical, des images anormales (IRM, rayons X ...) peuvent correspondre à une maladie ou à une blessure [Spence et al., 2001]. Les anomalies dans les données de transaction par carte de crédit pourraient indiquer une usurpation d'identité ou un vol de carte de crédit [Aleskerov et al., 1997].

Ces exemples montrent que les enjeux de cette thèse diffèrent des sujets traités habituellement. En effet, comme énoncé précédemment, les scénarios les plus critiques ayant lieu dans les bâtiments sont généralement déjà traités avec des systèmes dédiés. Nous rappelons donc que notre enjeu est de s'adresser à toutes les anomalies, de la plus critique à la plus bénigne, pouvant avoir lieu dans un bâtiment connecté.

Plus exactement, notre enjeu est de détecter toutes les anomalies pouvant être reflétées dans les données issues des capteurs se trouvant dans le bâtiment. En effet, il serait, par exemple, bien plus délicat de détecter un dégât des eaux au sein d'un bâtiment ne disposant que de capteurs de température et de luminosité, qu'au sein d'un bâtiment équipé de capteur d'humidité et d'hygrométrie.

Les anomalies sont des données ou des portions de données qui ne sont pas conformes à une notion bien définie de comportement standard ou normal. La notion de normalité dépend uniquement des tendances générales présentes dans notre cadre d'observation. Ainsi, les anomalies sont purement relatives à l'ensemble de données dont elles proviennent.

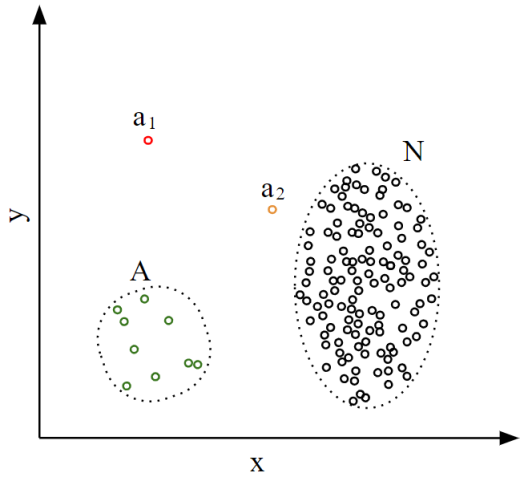

FIGURE 2.9 – Exemples de comportement normal, d'anomalies et de données ambiguës au sein d'un jeu de données bidimensionnel.

La figure 2.9 illustre divers cas dans un jeu de données à deux dimensions. La majorité d'individus sont positionnés dans la région établie comme normale N. Il est à peu près sûr de considérer les points suffisamment éloignés de cette région comme des anomalies ; le point a_1 est un exemple d'anomalie clairement identifiée. Il est toutefois possible que des données ambiguës soient présentes dans un jeu de données, ce qui rend la tâche de détection d'anomalies plus délicate. Sur la figure 2.9, le point a_2 peut, en fonction d'un seuil de tolérance que l'analyste doit déterminer, être considéré comme une anomalie ou pas.

De plus, nous pouvons également trouver un ensemble d'anomalies A proches les unes des autres. Si de nouvelles données devaient être positionnées à proximité de ces anomalies, notre notion du comportement normal inhérent au jeu de données de base pourrait s'en trouver modifié. La prise en compte de ces nouvelles données pour un modèle déjà entraîné s'inscrit la problématique de maintenir une vue cohérente de la normalité au gré des changements dans les données.

Un parallèle peut aussi être établi avec l'apprentissage par renforcement, où l'algorithme interagit avec un environnement dynamique via un système de feedback entre le système d'apprentissage et les expériences auxquelles on le soumet. En effet, Sutton et Barto décrivent le « reinforcement learning » comme « le fait d'apprendre à agir ou réagir de manière à maximiser une récompense délivrée a postériori. On ne dit pas au système apprenant quoi faire, il doit découvrir par lui même quelles actions le récompensent le plus » [Sutton and Barto, 2018]. On peut donc imaginer un modèle apprenant à altérer sa notion de normalité au fur et à mesure que de nouvelles données apparaîssent afin de maximiser sa récompense.

Nous notons que l'une des raisons pouvant entraîner l'apparition de « nouveautés » peut tout simplement être une confusion due au fait que le jeu de données d'entraînement est sous-dimensionné par rapport à la tâche prévue.

Les anomalies peuvent apparaître dans les données pour un grand nombre de raisons étroitement liées au domaine d'application d'où provient le jeu de données. Néanmoins, certaines anomalies

peuvent apparaître pour des raisons tout à fait inattendues à la base. En effet, bien qu'il soit dans la nature des anomalies d'être inattendues, il existe souvent des raisons plus ou moins évidentes à l'apparition de celles-ci en fonction du domaine d'application.

L'apparition de nouveautés [Pimentel et al., 2014] est un bon exemple mais n'est pas le seul. Par exemple, dans le cas de bâtiments connectés, le dysfonctionnement ou le mauvais placement de capteurs peut aussi entraîner l'apparition de valeurs aberrantes, et donc d'anomalies, alors qu'il ne se déroule aucun scénario anormal au sein du bâtiment. Enfin, il est à noter que certaines anomalies clairement identifiées comme telles peuvent n'avoir aucune incidence d'un point de vue métier. Il convient donc à ce moment d'étudier l'existence d'éventuelles variables cachées afin d'expliquer l'apparition de ces anomalies, le but étant de réduire la levée de fausses alertes.

Il existe de nombreux algorithmes de détection d'anomalies qui ne nécessitent pas d'apprentissage (Local Outlier Factor [Breunig et al., 2000], Isolation Forest [Liu et al., 2008]...). Ces algorithmes ne sont généralement applicables que sur des données « standard » (i.e. des vecteurs à n dimensions) et ne fonctionnent qu'avec des dimensions faibles à moyennes.

Exemple : Local Outlier Factor calcule la variance de la densité d'un échantillon donné par rapport à ses k-voisins les plus proches. En comparant la densité locale d'un échantillon aux densités locales de ses voisins, on peut identifier des échantillons ayant une densité nettement inférieure à celle de leurs voisins.

Ces algorithmes ne sont donc pas directement applicables à notre cas. En effet, comme expliqué dans la partie 1, les données issues de l'IoT peuvent potentiellement présenter des dimensions élevées. De plus, ces données sont sensibles aux anomalies collectives qui ne peuvent être détectables que lorsque l'on analyse les données sous forme de séries temporelles. Nous nous sommes donc orientés vers des systèmes basés sur la réduction de dimension, comme expliqué dans la partie 2.3.

Dans ce contexte :

- La tâche T que l'on cherche à réaliser avec un système d'apprentissage utilisé pour la détection d'anomalie consiste à discriminer les données normales et les anomalies à travers un score ou l'attribution d'un label.

- L'expérience E qui permettra au système de s'améliorer est une exposition aux données au travers d'un entraînement afin que celui-ci apprenne le comportement normal de celles-ci.

- L'indice de performance P est, dans le cas d'un système attribuant un label aux données, typiquement la mesure de la sensibilité et de la spécificité. Pour un système attribuant un score, la mesure consistera à quantifier la qualité de tous les seuils discriminatoires possibles. Pour la quantification de cette qualité, on peut utiliser la précision (cf : section 2.2.3), mais on utilise généralement les notions de sensibilité et spécificité afin d'avoir des informations supplémentaires.

TABLE 2.1 – Tableau explicatif concernant les terminologies des données bien et mal catégorisées.

Données	Anomalies	Standards
Catégorisées anomalies	Vrai positifs (VP)	Faux positifs (FP)
Catégorisées standards	Faux négatifs (FN)	Vrai négatifs (VN)

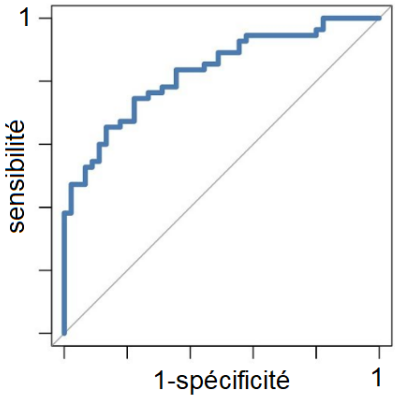

FIGURE 2.10 – Exemple de courbe ROC

La sensibilité correspond au ratio $\frac{\text{VP}}{\text{VP}+\text{FN}}$ et la spécificité correspond au ratio $\frac{\text{VN}}{\text{VN}+\text{FP}}$ (les termes sont introduits par le tableau 2.1) [Hill and Lewicki, 2007].

Comme énoncé précédemment dans le cas d'un système attribuant un score, l'indice de performance portera sur la possibilité de discriminer correctement les données normales des anomalies en déterminant un seuil sur ce score. Une mesure typiquement utilisée est l'aire sous la courbe ROC (Receiver Operating Characteristic) [Hanley and McNeil, 1982]. Cette courbe représente tous les seuils possibles avec leurs sensibilités et spécificités correspondantes (exemple figure 2.10). Ainsi, on peut associer une courbe ROC aux résultats d'un modèle : plus l'aire sous la courbe se rapproche de 1, plus le modèle a fourni des scores permettant une discrimination totale.

2.3 Choix du modèle initial

Nous présentons dans cette section notre réflexion initiale quant aux algorithmes usuels de détection d'anomalies. Nous présentons des algorithmes utilisés dans le cadre de bâtiments connectés et notre positionnement par rapport à ceux-ci.

Comme cité précédemment, de nombreux algorithmes de détection d'anomalies existent. Parmi les travaux à propos de détection d'anomalies au sein de bâtiments connectés, on peut citer

[Fan et al., 2018] ou encore [Araya, 2016]. Il est néanmoins à noter que ces travaux visent spécifiquement les anomalies de consommation d'énergie. Les méthodes que l'on retrouve sont les suivantes : ACP (cf : section 2.3.1), les auto-encodeurs (cf : section 2.3.1), les auto-encodeurs à convolution (cf : section 3.1.2), les Forêts d'arbres décisionnels (développées ci-après), SVM (développé ci-après).

L'algorithme des Forêt d'arbres décisionnels (Random Forests), proposé par Breiman [Breiman, 2001], est un algorithme très utilisé de classification et de régression. Cet algorithme opère en entraînant un ensemble de B arbres de décisions produisant chacun une sortie. L'algorithme va ensuite calculer la moyenne des sorties afin de donner un résultat. « L'aléatoire » (random) ayant lieu lors du déroulement de cet algorithme vient du fait que chacun des arbres n'utilise pas toutes les caractéristiques des données pour produire une décision, mais seulement une partie. Chaque arbre utilise un sous-ensemble des caractéristiques déterminé aléatoirement.

SVM (Support Vector Machines), proposé par Vapnik [Cortes and Vapnik, 1995] est aussi un algorithme supervisé de classification et de régression. Utilisé dans le cadre de la classification, l'algorithme cherchera à déterminer un hyperplan séparant les données des catégories différentes, tout en maximisant la distance entre lui et des données les plus proches. On trouve, depuis 1999 [Schölkopf et al., 1999], une extension de cet algorithme spécifiquement dédiée à la détection d'anomalies appelée SVM à une classe (« 1 class SVM »). Cette variante consiste à chercher à inclure les données d'entraînement dans une seule classe et de considérer toute nouvelle valeur ne rentrant pas dans cette classe comme une anomalie.

Néanmoins, la plupart de ces algorithmes ne sont pas conçus pour traiter des données de grandes dimensionnalités. La détection d'anomalies n'est pas le seul domaine concerné par les problèmes liés aux données de grandes dimensions. L'apprentissage automatique en général, la fouille de données, l'analyse numérique en général, etc. sont des domaines concernés par divers phénomènes ayant lieu lorsque l'on cherche à traiter des données évoluant dans des espaces de grandes dimensions. Le « Fléau de la dimension » (originellement « Curse of dimensionality » [Bellman, 1961]) est un terme qui a été introduit par Richard Bellman et désigne l'idée suivante : Lorsque le nombre de dimensions augmente, le volume de l'espace dans lequel évolue les données grandit rapidement de telle façon que les données se retrouvent de plus en plus isolées les unes par rapport aux autres.

Le fait de traiter des données trop éparses pose des problèmes dans tous procédés liés à la notion de généralisation. Dans le cadre de l'apprentissage automatique, et en particulier des réseaux de neurones, le fait de ne pas pouvoir généraliser rend impossible le fait d'inférer efficacement des résultats. Dans le cadre de la détection d'anomalies, le fait de ne pas pouvoir généraliser rend impossible la compréhension globale de la normalité au sein d'un jeu de données. Ainsi, toutes les données (ou aucune donnée ne) seraient considérées comme des anomalies.

Nous étudions, lors de nos expérimentations, des fenêtres constituées de n vecteurs consécutifs, eux mêmes composés de m données de capteurs. En effet, nous ne nous restreignons pas aux anomalies liées à la consommation d'énergie et nous souhaitons utiliser un modèle sensible aux anomalies collectives (cf : partie 2.1). La dimension des données que nous traitons est donc susceptible d'être bien supérieure à la capacité de traitement des algorithmes « classiques » de

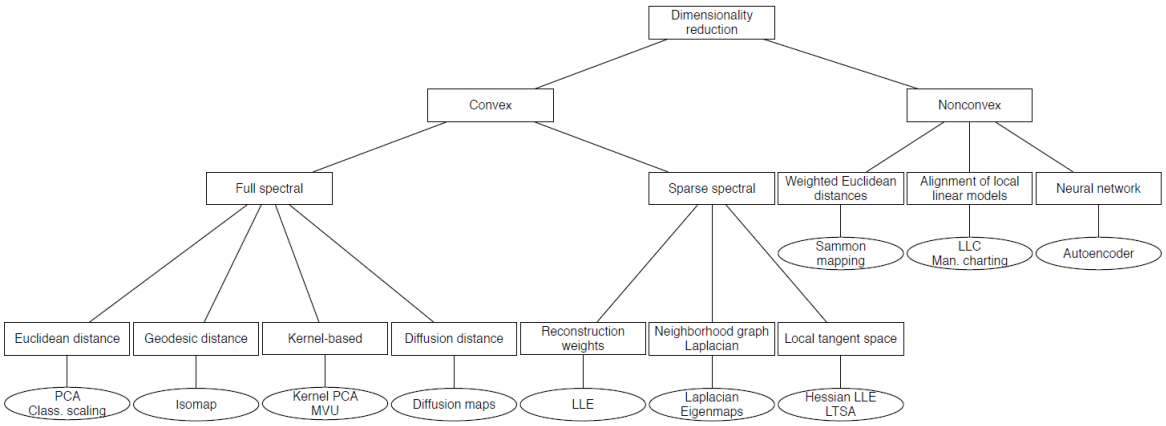

FIGURE 2.11 – Taxonomie non exhaustive de techniques de réduction de dimension introduite par Laurens van der Maaten [Van Der Maaten et al., 2009].

détection d'anomalies. Nous nous sommes donc décidés à adopter un modèle pouvant réduire cette dimension, et délaissons les techniques comme SVM, random forests, isolation forest...

2.3.1 Réduction de dimensions

Comme nous l'avons énoncé précédemment, nous avons choisi de nous orienter vers des méthodes de détection d'anomalies utilisant l'apprentissage et basés sur la réduction de dimension. Pour la grande majorité des algorithmes de réduction de dimension, lorsque ceux-ci sont utilisés pour de la détection d'anomalies, on cherche typiquement à n'apprendre que le comportement normal décrit dans la partie 2.2.4. Ainsi, en effectuant l'inverse du procédé de réduction de dimension, il devrait exister une différence conséquente entre la donnée originale, contenant une anomalie, et la donnée recréée.

Les techniques de réduction de dimensions s'appuient sur l'existence supposée d'une dimension intrinsèque propre au jeu de données de taille D traité. Cette dimension L correspond à un nombre de variables indépendantes pouvant « expliquer » de manière satisfaisante le jeu de données initial, avec $L << D$ [Roweis and Saul, 2000]. La taille L dépend donc d'un facteur subjectif (« expliquer de manière satisfaisante »), et donc du cas d'utilisation. Cette dimension intrinsèque, déterminée par l'utilisateur dans la majorité des algorithmes, est utilisée par tous les algorithmes de réduction en tant que paramètre.

Afin de déterminer les variables appartenant à cette dimension intrinsèque, de nombreuses techniques existent [Van Der Maaten et al., 2009]. La figure 2.11 présente une taxonomie des techniques de réduction de dimension les plus populaires. Celle-ci divise dans un premier lieu les techniques de réduction de dimension adaptées aux données évoluant dans des espaces convexes et celles adaptées aux données évoluant dans des espaces non-convexes.

Parmis toutes les techniques de réduction de dimension non-supervisées, la plus populaire est sans doute la méthode ACP.

ACP

L'analyse en composantes principales ACP, ou encore la méthode PCA (Principal Components Analysis), est une technique de réduction de dimension linéaire publiée par Pearson en 1901 [Pearson, 1901]. Cette technique cherche donc à inclure les données dans un sous-espace linéaire de dimension inférieure à l'espace initial.

Cette méthode de réduction de dimension étant linéaire, elle n'est pas adaptée pour extraire les relations complexes ayant lieu entre les différents capteurs de bâtiments connectés.

La méthode ACP construit une représentation décrivant autant que possible la variance des données. Ainsi, la méthode ACP cherche à déterminer la matrice $\boldsymbol{M}$ qui maximise la fonction de coup $trace(\boldsymbol{M}^T cov(\boldsymbol{X})\boldsymbol{M})$, où $cov(\boldsymbol{X})$ est la matrice de covariance des données $\boldsymbol{X}$. Ce procédé repose sur les d vecteurs propres de la matrice de covariance. La méthode ACP résout donc le problème suivant :

$$cov(\boldsymbol{X})\boldsymbol{M} = \lambda \boldsymbol{M}$$

pour les principales valeur propres λ. La représentation $\boldsymbol{y}_i$ du point $\boldsymbol{x}_i$ est donc tout simplement calculée par $\boldsymbol{Y} = \boldsymbol{X}\boldsymbol{M}$ [Almotiri et al., 2017].

Bien que cette technique ne soit pas adaptée pour répondre directement à notre cas d'utilisation, nous avons pu néanmoins utiliser celle-ci lors du pré-traitement de certaines données (nous détaillons notre utilisation de la méthode ACP section 5).

Approches à base de réseaux de neurones

L'une des architectures de réseaux de neurones les plus notables est l'auto-encodeur. Cette idée, bien plus récente que celle de ACP, a pour la première fois été introduite par Yann LeCun [LeCun, 1987] puis par Bourlard et Kamp [Bourlard and Kamp, 1988] avant d'être popularisé par Kramer [Kramer, 1991].

Ces premiers travaux traitent de réseaux de neurones standards (MLP, Multilayer Perceptron), paramétrés en mode « auto-association ». Un tel réseau de neurones est, du fait de ses fonctions d'activations, capable d'appréhender des données de manière non-linéaire. Cette capacité est à souligner, elle s'accorde bien avec le type de données que nous cherchons à analyser.

Les premiers travaux effectués dans le but d'étudier la répartition des données avec des réseaux de neurones ont néanmoins été réalisés par Kohonen [Kohonen, 1982]. Le principe derrière les réseaux de Kohonen est de discrétiser l'espace dans lequel évoluent les données en plusieurs « zones » et à attribuer une zone à chaque individu.

Parmi tous les algorithmes de réduction de dimension non-linéaires, l'auto-encodeur se démarque des autres, notamment car il repose sur une utilisation spécifique des réseaux de neurones artificiels (nous expliquons le fonctionnement des réseaux de neurones dans la partie 2.4 et le fonctionnement des auto-encodeurs dans la partie 2.5). Cet aspect rend cet algorithme particulièrement souple et versatile, notamment grâce à tous les travaux qui ont été menés sur les réseaux de neurones.

Parmi ces travaux, nous pouvons citer les travaux concernant les réseaux de neurones à convolutions [LeCun et al., 1989] ou encore les réseaux de neurones récurrents [Rumelhart et al., 1986] (nous expliquons leurs fonctionnements dans les parties 3.1 et 3.2). Ces travaux sont notamment intéressants pour notre cas car ils constituent deux avancées majeures dans le domaine du traitement des images et des données séquentielles avec des réseaux de neurones.

Les auto-encodeurs constituent un candidat intéressant pour :

- réduire la dimension des données,
- effectuer cette réduction de dimension de manière non-linéaire,
- considérer les données de manière séquentielle ou les considérer comme des ensembles de motifs.

2.3.2 Détection d'anomalies avec un auto-encodeur

Afin d'éprouver notre choix, nous avons effectué une première manipulation. Nous avons entraîné et utilisé un auto-encodeur de manière classique, en évaluant l'écart entre les données d'origine et les données recréées post-compression (cf : partie 2.5). Cet écart est le score utilisé pour la détection d'anomalies : un auto-encodeur idéal produirait un ensemble de scores au sein duquel il serait possible de fixer un seuil séparant toutes les anomalies des données saines.

Nous avons placé dans une pièce un capteur de luminosité, de température et d'humidité pendant une semaine. Ces capteurs ont remonté des données toutes les 5 secondes. Pour compléter ces données de capteurs, nous avons ajouté des informations concernant la date d'acquisition des données afin de permettre à l'auto-encodeur d'appréhender la saisonnalité des données (motif d'une journée). Ce procédé a été ré-utilisé lors d'une expérimentation plus conséquente et est expliqué dans la partie 3.3.1.

Ainsi, nous avons pu constituer un jeu de données à 5 dimensions. Nous rappelons que les données correspondent à un cadre très spécifique (très peu de capteurs, temps de captation relativement court...) et que cette manipulation est un test afin de valider notre choix.

L'auto-encodeur utilisé est un auto-encodeur « simple ». Il ne comporte pas de couches récurrentes ni de couches à convolutions. Nous avons constaté, en accord avec notre hypothèse, que les données obtenues en sortie étaient similaire aux données d'entrée. L'auto-encodeur a été paramétré pour utiliser un taux de compression à $\sim$40%, encodant les données dans un vecteur latent d'une taille de 3.

Nous avons induit manuellement 2 anomalies :

- Une anomalie de valeur, en exposant le capteur de température à un matériau réfrigéré. L'auto-encodeur n'ayant pas été entraîné avec des valeurs en dessous de 18°C, nous constatons, comme le montre la courbe de la figure 2.12, qu'il est incapable de recréer des valeurs en dessous de ce seuil.

- Une anomalie contextuelle, en décalant d'une demi journée les valeurs de temporalité ajoutée lors de l'enrichissement des données. On constate sur la figure 2.13 que l'auto-encodeur a recréé, pour les individus concernés par cette anomalie, des données de luminosité bien plus basses qu'à l'initial. L'auto-encodeur semble donc bien avoir associé les données remontées la nuit à des valeurs de luminosité basses.

Les individus composant ces deux scénarios anormaux font partie de deux jeux de données de test respectifs. Ces jeux de données incluent aussi les individus qui précèdent et succèdent aux scénarios anormaux (période de 30 minutes avant et après).

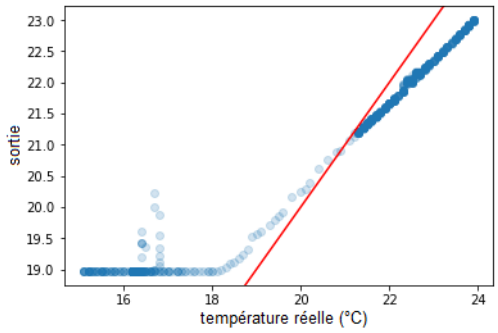

FIGURE 2.12 – Evolution de la température recréée par l'auto-encodeur par rapport à la température réelle lors d'une anomalie de valeur induite (jeu de donnée de test associé). L'idéal est $x = y$, représenté par la droite rouge.

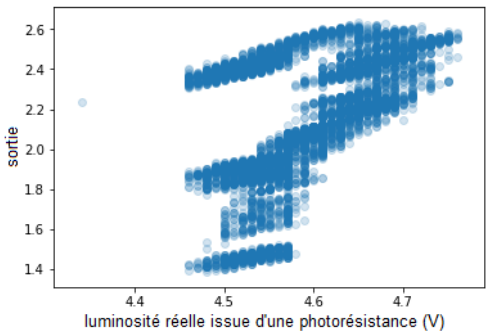

FIGURE 2.13 – Données de luminosité recréée par rapport à la luminosité réelle lors d'une anomalie contextuelle induite (la photo-résistance a une valeur max de 5V).

Ces résultats nous ont conforté à utiliser des auto-encodeurs pour la suite des travaux. De plus, nous pouvons trouver, au sein de la littérature, diverses utilisations d'auto-encodeurs traitant de données de capteurs [Malhotra et al., 2016] [Fan et al., 2018].

2.4 Réseaux de neurones artificiels

Dans cette partie nous allons expliquer le fonctionnement de l'algorithme sur lequel se base le modèle de détection d'anomalie choisi ainsi que le cheminement qui a mené à son aboutissement.

Nous commençons par expliquer brièvement le fonctionnement d'un neurone biologique, duquel s'est directement inspiré le modèle fondateur de McCulloch-Pitts [McCulloch and Pitts, 1943], modèle posant les bases des réseaux de neurones artificiels contemporains. Nous aboutissons ensuite au fonctionnement des réseaux de neurones utilisés aujourd'hui et listons de manière non-exhaustive les différentes optimisations dont nous avons pu bénéficier pendant nos expérimentations.

2.4.1 Biomimétisme

Comme de nombreuses autres approches biomimétiques, les réseaux de neurones artificiels reposent sur l'imitation d'un système ayant émergé naturellement et a pour but de répondre à une ou plusieurs problématiques complexes.

Comme son nom l'indique, l'objectif ayant motivé le développement d'un tel algorithme est l'imitation du fonctionnement et de la structure du cerveau, humain ou animal. Bien qu'il existe des zones d'ombres dans nos connaissances relatives au cerveau, de nombreux travaux nous ont permis d'acquérir une certaine compréhension de la structure de celui-ci ainsi que du fonctionnement des cellules le constituant [Sporns et al., 2005] [Alberini et al., 2014].

Les cellules nerveuses, ou neurones, constituent l'intégralité de notre système nerveux. Le cerveau contient environ 10^{11} neurones et 10^{14} à 10^{15} connexions. Il existe de nombreux types de neurones, ayant un nombre de connexions et une taille très variable, mais tous sont basés sur la même structure. Le neurone est doté de plusieurs propriétés fonctionnelles :

- Il peut recevoir des signaux venant de l'environnement ou d'autres neurones.
- C'est un système de traitement du signal.
- Il est capable d'envoyer le résultat de ces traitements à d'autres neurones.

Le neurone dispose de dendrites, lui permettant de recevoir des informations. Une fois les informations traitées, le résultat sera acheminé via un influx nerveux au sein de l'axone (cf : illustration figure 2.14). La propagation de l'information s'effectuera dans toutes les terminaisons de celui-ci afin de la transmettre à tous les neurones connectés (ces connexions sont appelées « synapses »).

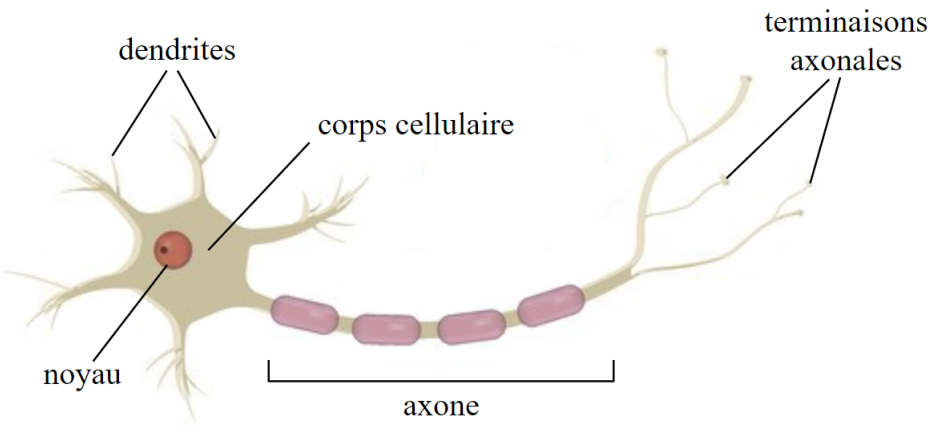

FIGURE 2.14 – Schéma simplifié de l'anatomie d'un neurone biologique. Tiré de [Abbas, 2019].

Sa capacité de traitement du signal fonctionne comme suit : L'information, transmise au neurone sous forme de signaux via les synapses connectées aux dendrites, va avoir une action sur le neurone. Il existe deux catégories de synapses, les synapses inhibitrices et les synapses excitatrices. L'information transmise par les synapses excitatrices va contribuer à stimuler le neurone tandis que l'information transmise par les synapses inhibitrices va contribuer à baisser le taux d'excitation du neurone. La capacité d'une synapse à plus ou moins exciter le neurone est appelée « poids synaptique ».

Ces poids synaptiques sont susceptibles d'être modifiés au cours de l'activité du réseau de neurones à travers l'habituation (baisse du poids) ou la sensibilisation (augmentation du poids).

Ces mécanismes font partie de la plasticité cérébrale, phénomène directement lié à l'apprentissage. Pour un être vivant, en cas de stimulations répétées, la réaction de base tend à disparaître (habituation). En revanche, si la stimulation est associée à une expérience douloureuse, on constatera une intensification progressive de la réaction de défense (sensibilisation). Lorsque l'excitation du neurone dépasse un seuil, un influx nerveux appelé potentiel d'action est propagé dans l'axone. Ce potentiel, contrairement aux informations reçues par le neurone, fonctionne en « tout ou rien », il n'a pas de poids particulier.

2.4.2 Le modèle fondateur

Considérant son fonctionnement, on peut interpréter le neurone biologique comme un automate simple à seuil. Si l'on considère un ensemble de n signaux d'entrée $s_1, s_2, \ldots, s_n$ auxquels sont associés les poids synaptiques respectifs $w_1, w_2, \ldots, w_n$, l'influx nerveux reçu I est de

$$I = \sum_{i=0}^{n} (w_i s_i)$$

Le signal de sortie s est alors donné par :

$$s = \begin{cases} 0, & \text{si } I \leq \theta \\ 1, & \text{si } I > \theta \end{cases}$$

avec θ seuil d'activation du neurone (cf : schéma figure 2.15).

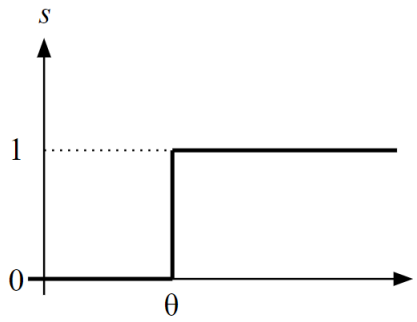

FIGURE 2.15 – Représentation graphique de la fonction d'activation pour un seuil θ.

C'est en se basant sur cette approche que le modèle fondateur est apparu. Le neurophysiologiste Warren McCulloch (1898-1972) s'est associé au mathématicien Walter Pitts (1923-1970) pour publier en 1943 ce qui est aujourd'hui considéré comme la base ayant mené au développement des théories contemporaines portant sur les réseaux de neurones artificiels : A Logical Calculus of Ideas Immanent in Nervous Actifivity [McCulloch and Pitts, 1943].

Les neurones présentés dans le modèle de McCulloch-Pitts satisfont cinq propriétés, dont certaines diffèrent des neurones biologiques :

- 1- L'activité des neurones est un processus « tout ou rien ».
- 2- Un certain nombre, fixe, de synapses doivent être excitées pour exciter un neurone. Ce nombre est indépendant de l'activité précédente ou de la position du neurone.
- 3- Le seul retard significatif au sein du réseau est le retard synaptique.
- 4- L'activité de toute synapse inhibitrice bloque totalement l'activation du neurone.
- 5- La structure du réseau est stable dans le temps.

Nous constatons que la première règle est une transposition directe de ce que l'on sait du modèle biologique.

La seconde règle est une transposition approximative. En effet, une sortie dépendant d'un nombre fixe de synapses excitées implique un ancrage des valeurs des poids synaptiques, occultant ainsi la faculté d'apprentissage des neurones biologiques. Le modèle fondateur de McCulloch et Pitts n'avait pas de faculté d'apprentissage et les poids synaptiques devaient être fixés manuellement.

La troisième règle, quant à elle, a pour objectif de permettre une discrétisation du processus. En effet, il n'y a pas de cycle qui régit l'envoi et la réception de signaux au sein des cellules nerveuses. Les communications au sein du cerveau se font de manière continue : l'excitation d'un neurone augmente et diminue au fil du temps en fonction des poids synaptiques mais aussi en fonction de la proximité de réception de ces signaux dans le temps. Cette règle nous permet donc d'utiliser des outils logiques traitant d'états discrets.

Les deux dernières règles ne sont pas conformes au modèle biologique mais McCulloch et Pitts ont montré une équivalence fonctionnelle entre les réseaux suivant ces règles et ceux conforment au modèle biologique.

Formalisation et implémentation du modèle de McCulloch-Pitts

On peut formaliser la construction de McCulloch-Pitts de manière simple. Pour un neurone, soit :

- un ensemble d'entrées excitatrices : $x_1, x_2, \ldots, x_n$ appartenant à 0,1 ;
- un ensemble d'entrées inhibitrices : $h_1, h_2, \ldots, h_m$ appartenant à 0,1 ;
- un seuil θ ;
- une fonction de transfert, ou fonction d'activation f ;
- un signal de sortie : $s = f(x, h)$;

La fonction de Heavyside a été la fonction généralement utilisée dans la fonction de transfert. Cette fonction a la forme :

$$H(x) = \begin{cases} 0, & \text{si } x < 0 \\ 1, & \text{si } x \geq 0 \end{cases}$$

Par extension, la fonction de transfert peut donc être formulée comme suit :

$$f(x, h) = \begin{cases} 0, & \text{si } H(\sum_{i=0}^{n} x_i - \theta) \leq 0 \text{ ou } \sum_{j=0}^{m}(h_j) \neq 0 \\ 1, & \text{si } H(\sum_{i=0}^{n} x_i - \theta) > 0 \text{ et } \sum_{j=0}^{m}(h_j) = 0 \end{cases}$$

Les neurones de McCulloch-Pitts sont qualifiés d'unités linéaires logiques à seuil (« Threshold Linear Units »). Il est, en effet, facile de réaliser des « neurones booléens » permettant d'implémenter les trois portes élémentaires, ET, OU et NON, comme montré sur les figures 2.16 à 2.18. McCulloch et Pitts ont montré, par extension, qu'il était possible, en combinant ces neurones booléens, de réaliser toutes les expressions logiques.

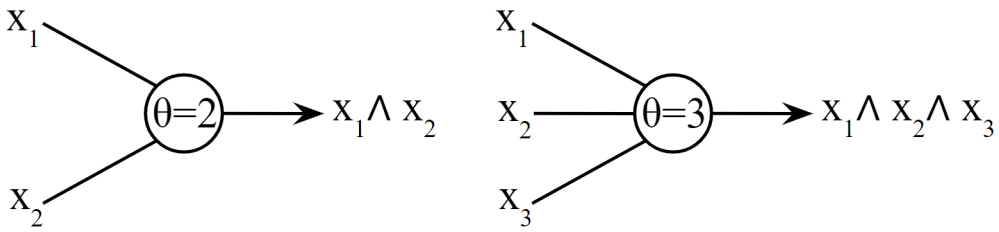

FIGURE 2.16 – Schématisation d'une porte AND à deux entrées (gauche) et à trois entrées (droite) basée sur les neurones de McCulloch-Pitts.

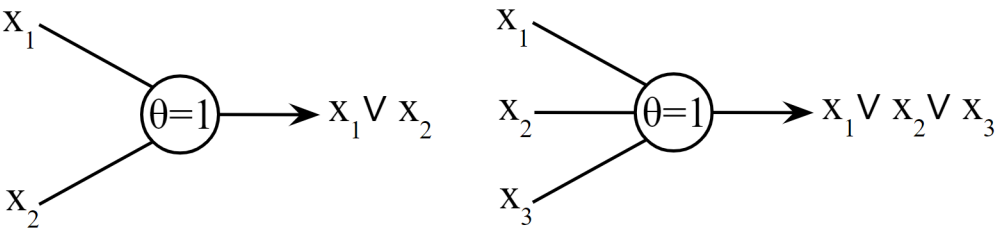

FIGURE 2.17 – Schématisation d'une porte OR à deux entrées (gauche) et à trois entrées (droite) basée sur les neurones de McCulloch-Pitts.

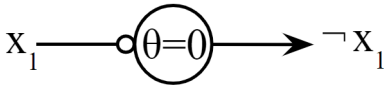

FIGURE 2.18 – Schématisation d'un neurone NOT de McCulloch-Pitts. Le symbole à gauche du neurone représente une entrée inhibitrice.

Réseaux récurrents de McCulloch-Pitts

Le « délai synaptique » évoqué plus haut signifie que, si l'entrée arrive à l'instant t, alors la sortie sera activée en t+1. Cette notion de délai est, dans certains cas, sans importance, par exemple dans le cas d'un réseau de neurones implémentant une expression logique. Néanmoins, ce délai rend possible la prise en compte de la temporalité et de l'ordre dans lequel les entrées sont envoyées au réseau. De plus, le modèle McCulloch-Pitts permet d'établir des boucles au sein d'un réseau de neurones. Ce type de réseau, contrairement aux réseaux où l'information ne va que dans un seul sens (dits « feedforward »), rend possible l'implémentation d'automates. Il a d'ailleurs été démontré par Kleene que la puissance d'expression des réseaux de neurones de McCulloch-Pitts étaient équivalente à celle d'une machine de Turing [Kleene, 1956].

Similitude et différences avec le modèle contemporain

Bien que les notions de délai synaptique et de récurrence ne soient plus réellement d'actualité, il reste aujourd'hui des similitudes rappelant le modèle fondateur. Les réseaux d'aujourd'hui

sont encore constitués d'un ensemble de neurones interconnectés, les données d'entrée d'un neurone restent, dans la plupart des cas, une combinaison des sorties des neurones de la couche précédente, les neurones ont toujours un nombre fixe d'entrées et de sorties et il existe encore des fonctions d'activation « tout ou rien ». Néanmoins, la différence fondamentale entre les réseaux de neurones contemporains et le modèle de McCulloch-Pitts est le fait d'être capable d'apprendre.

Il est à noter qu'il existe aujourd'hui des modèles de réseaux de neurones récurrents mais ceux-ci diffèrent des réseaux récurrents de McCulloch-Pitts (cf : partie 3.2). Contrairement au modèle fondateur, les boucles ne sont pas formées entre des neurones potentiellement éloignés mais ont lieu au sein d'une même couche. Cette notion, constituant un des socles d'un modèle que nous étudions, est détaillée partie 3.2.

2.4.3 Apprentissage : de la règle de Hebb à la rétropropagation du gradient

Le premier modèle opérationnel de réseau apprenant est le Perceptron [Rosenblatt, 1958]. Le Perceptron est constitué des éléments suivants :

- des unités sensitives (S), caractérisant les stimulations physiques envoyant 0 en cas d'inactivité et 1 en cas de stimulation ;
- des unités d'association (A) disposant de connexions entrantes et sortantes et renvoyant un signal de sortie à 1 en cas de stimulation franchissant un seuil théta, et 0 sinon ;
- des unités de réponse (R) disposant de connexions entrantes et générant un signal à la sortie du réseau à 1 en cas de stimulation suffisante, -1 en cas de stimulation insuffisante, et 0 en cas d'indétermination ;
- une matrice d'interactions qui, de la même façon que les poids synaptiques, attribue un poids à chaque connexion.

Ce modèle est organisé en couche : les seules connexions possibles vont de S vers A et de A vers R. Le modèle est très proche de celui de McCulloch-Pitts, la différence essentielle étant le fait d'attribuer des poids aux connexions du réseau afin de les pondérer. Les poids des connexions de S vers A restent fixes, tandis que celles de A vers R sont susceptibles de varier. Ce sont d'ailleurs les variations de ces poids qui permettent l'apprentissage au sein du Perceptron.

La règle d'apprentissage du Perceptron est une variante de la règle d'apprentissage de Hebb. La règle de Hebb, basée sur une hypothèse scientifique en neurosciences, décrit un mécanisme de plasticité synaptique dans laquelle l'efficacité synaptique augmente lors d'une stimulation présynaptique répétée et persistante de la cellule postsynaptique (« neurons wire together if they fire together »). Cette règle ne peut que renforcer les poids synaptiques et ne tient pas compte de l'écart entre la sortie réelle et la sortie souhaitée. La règle du perceptron en est une version différente que l'on peut formuler de la manière suivante :

$$w_{ij}(t+1) = w_{ij}(t) + \eta(x_j - x'_j)s_i$$

avec :

- $w_{ij}(t)$, poids de la connexion entre les neurones i et j à l'instant t
- η, pas d'apprentissage compris entre 0 et 1 (usuellement élevé et réduit de plus en plus à l'approche de la solution)
- x_j, la sortie désirée pour l'entrée courante
- x'_j, la sortie calculée et obtenue pour l'entrée courante
- s_i, le signal transmis par le neurone i.

Rosenblatt a démontré le « théorème de convergence du Perceptron » qui pose que, si une solution existe (i.e. si les données sont linéairement séparables), alors l'algorithme du Perceptron convergera vers elle en un nombre fini d'itérations.

Un modèle très similaire à celui du Perceptron est l'Adaline (adaptive linear element) [Widrow and Hoff, 1960]. Ce modèle utilise comme fonction d'activation la fonction d'identité $f(x) = x$ plutôt que la fonction Heavyside step.

2.4.4 La règle du delta

La méthode d'apprentissage de l'Adaline [Widrow and Hoff, 1960] repose sur la minimisation de l'erreur quadratique entre la sortie réelle et la sortie désirée. Cette méthode implique de modifier tous les paramètres de la fonction à minimiser.

Dans le cas présent, les paramètres de la fonction à minimiser sont les poids des connexions liant la couche unique à la couche de sortie.

Nous notons $e_k(n)$ l'erreur observée pour le neurone de sortie k et l'individu n issu du jeu de données d'entraînement :

$$e_k(n) = \frac{1}{2}(d_k(n) - y_k(n))^2 \tag{2.1}$$

avec $d_k(n)$ sortie attendue et $y_k(n)$ sa sortie observée.

Nous notons $E(n)$ la somme des erreurs quadratiques définie par :

$$E(n) = \sum_{k=1}^{m} e_k(n)$$

avec m nombre total de neurones dans la couche de sortie.

La méthode du delta vise à réduire E en corrigeant les poids dans le sens opposé du gradient. Il convient donc de déterminer les dérivées partielles de la fonction d'erreur en fonction de chaque poids, noté ici $\frac{\partial e_k(n)}{\partial w_i(n)}$.

On notera la variation du poids à appliquer Δw_i de la manière suivante :

$$\Delta w_i = -\eta \frac{\partial e_k(n)}{\partial w_i(n)}$$

avec η le pas d'apprentissage (usuellement compris entre 0 et 1) et avec

$$\frac{\partial e_k(n)}{\partial w_i(n)} = -(d_k - y_k)$$

étant donné $e_k(n)$ (cf formule 2.1).

Cela permet d'aboutir à la règle d'apprentissage suivante :

$$w_i(t+1) = w_i(t) + \eta(d_k - y_k) * s_i$$

Cette règle, très proche de la règle du perceptron, a été obtenu en dérivant la fonction d'erreur quadratique.

L'intérêt de cette méthode est donc qu'elle est capable d'approximer la descente du gradient en utilisant uniquement les informations locales (l'entrée et la sortie courante). Elle se révèlera indispensable pour l'algorithme de rétro-propagation.

L'algorithme peut ne pas trouver le minimum global quand plusieurs minimums locaux sont présents (cf : figure 2.19). Dans le contexte des réseaux de neurones, les solutions correspondant à des minimums locaux sont généralement acceptées, tant que le minimum correspondant est suffisamment bas.

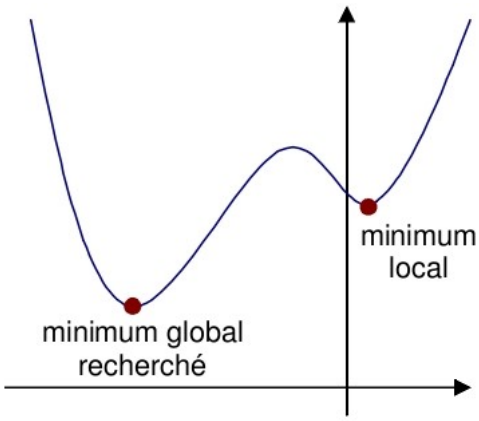

FIGURE 2.19 – Illustration de minimum local pouvant potentiellement être le point de convergence d'un réseau de neurones avant l'atteinte de l'optimum.

2.4.5 Réseaux multicouches et rétropropagation

Les modèles cités, bien qu'ayant énormément influencé les réseaux d'aujourd'hui, sont soumis à la limite de n'avoir qu'une seule couche séparant l'entrée de la sortie. Un exemple illustrant particulièrement cette limitation est l'impossibilité de réaliser un XOR [Minsky, 1967] (cf : figure 2.20). En effet, celui-ci n'étant pas linéairement séparable, il n'existe pas de solution pour les modèles Adaline et Perceptron.

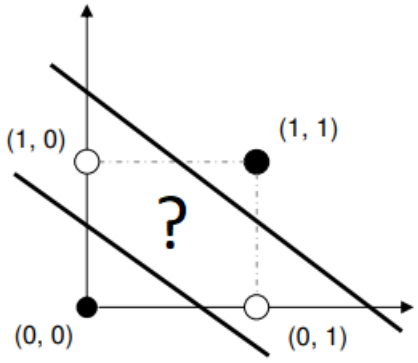

FIGURE 2.20 – Schématisation du XOR. Chaque ligne de la table de vérité de cette opération est représentée sur un repère (x et y sont les entrées, et la couleur du point représente la sortie associée). On ne peut pas séparer linéairement les points noirs des points blancs, ce problème nécessite donc l'ajout de couches supplémentaires. Figure adaptée de [Swingler, 2012]

La solution est donc de rajouter des couches aux réseaux afin de leur permettre d'apprendre à répondre à des problèmes plus complexes. De nombreux réseaux de neurones aujourd'hui sont issus de cette idée (la catégorie la plus répandue de réseaux de neurones est appelée « MLP » pour « Multi-Layer Perceptron »).

Ajouter des couches implique néanmoins de retravailler sur les règles d'apprentissage des modèles. En effet, il est délicat de déterminer la sortie attendue d'un neurone si celui-ci n'est pas directement lié à la couche de sortie mais à une couche intermédiaire.

La solution qui a été adoptée pour pallier la difficulté évoquée précédemment est l'algorithme de rétro-propagation, introduit par Werbos en 1974 [Werbos, 1974].

Nous présentons ici cet algorithme tel qu'il est utilisé aujourd'hui.

De la même manière que présentée dans la partie 2.4.4, cet algorithme vise à assigner aux neurones des couches cachées une erreur en fonction des dérivées partielles de tous les poids.

Pour le poids w_{ij} avec i et j désignant respectivement un neurone sur la couche précédente et un neurone sur la couche courante, on note la dérivée partielle $\frac{\partial E(n)}{\partial w_{ij}(n)}$.

On peut donc noter la variation à effectuer sur le poids w_{ij} :

$$\Delta w_{ij}(n) = -\eta \frac{\partial E(n)}{\partial w_{ij}(n)} = \eta \delta_j(n) y_i(n)$$

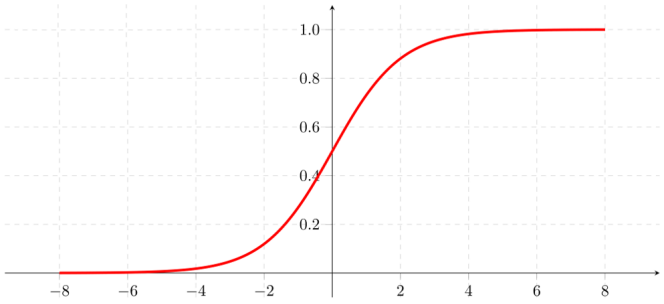

FIGURE 2.21 – Représentation graphique de la fonction logistique « sigmoïde ».

Avec :

$$\delta_j(n) = y_i(n)[1 - y_j(n)] \sum_{k=0}^{m} \delta_k(n) w_{kj}(n)$$

L'algorithme peut donc se résumer aux étapes suivantes :

- 1- Initialisation aléatoire des poids (usuellement entre -0.5 et 0.5).
- 2- Mélange des données d'entraînement.
- 3- Pour chaque donnée d'entraînement n :
  - a. Calcul des sorties en propageant les données vers l'avant du réseau.
  - b. Rétropropagation de l'erreur et ajustement des poids :

$$w_{ij}(n) = w_{ij}(n-1) + \eta \delta_j(n) y_i(n)$$

- 4- Répétition des étapes 2 et 3 jusqu'a obtenir une erreur acceptable.

Il est à noter qu'à ce stade, la fonction d'activation de Heavyside n'est plus utilisée. En effet, celle-ci n'est pas dérivable car discontinue en 0 (dans le cas de la fonction de transfert déjà présentée, discontinue en θ) et a une dérivée nulle lorsqu'elle est continue. On ne peut donc pas utiliser d'algorithme utilisant le gradient avec cette fonction.

Une des fonctions d'activation les plus couramment utilisées encore aujourd'hui est la fonction logistique (fonction sigmoïde typique), imitant l'aspect « tout ou rien » de Heavyside tout en étant dérivable (cf : figure 2.21) [Hinton, 1990].

La formalisation de cette fonction est la suivante :

$$f(x) = \frac{1}{1 + e^{-\lambda x}}$$

avec λ correspondant au quart de la pente de la courbe au point d'inflexion.

2.4.6 Les réseaux de neurones aujourd'hui

Tout le cheminement vu précédemment a dressé les lignes directrices des réseaux de neurones utilisés aujourd'hui.

En effet, un réseau de neurones établi aujourd'hui consistera en :

- Une couche d'entrée
- Une ou plusieurs couches cachées
- Une couche de sortie

Les couches de sorties et couches cachées étant composées de neurones :

- Recevant la totalité des sorties de la couche précédente
- Pondérant ses entrées par leurs poids respectifs
- Traitant la somme de celles-ci via une fonction d'activation
- Transmettant le résultat du traitement à la couche suivante (ou à l'utilisateur dans le cas de la couche de sortie)

Avec un processus d'apprentissage suivi par le réseau de neurones consistant en la répétition de nombreuses itérations suivantes. Pour chaque donnée d'entraînement :

- Propager vers l'avant les informations venant de la couche d'entrée
- Comparer la sortie réelle avec la sortie attendue
- Assigner une erreur particulière à chaque sortie de neurone grâce à la rétropropagation
- Faire évoluer les poids des entrées dans le bon sens grâce à la descente du gradient

Ce sont via ces lignes directrices que, ces dernières années, les avancées dans le domaine ont pu être guidées. Les principes listés précédemment sont aussi applicables à tous les modèles de réseaux de neurones utilisés lors de nos expérimentations.

Méthodes d'optimisation

Le fait que l'algorithme de rétropropagation soit toujours utilisé aujourd'hui témoigne de son succès et de son efficacité. La version originelle de l'algorithme souffre néanmoins d'importantes limitations :

- Sa complexité algorithmique : l'algorithme de rétropropagation est très lourd en termes de calculs et peut rapidement se révéler très lent si le problème est assez conséquent. Ainsi, Hinton a montré empiriquement que le temps d'apprentissage, traité séquentiellement, est à $\mathcal{O}(N^3)$ avec N le nombre de connexions dans le réseau [Hinton, 1990].

- Sa difficulté d'utilisation : il n'existe pas d'outil efficace de dimensionnement de réseau de neurones, en particulier en ce qui concerne le nombre et la taille de couches cachées à utiliser pour adresser un problème particulier.

Ces limitations ont suscité de nombreux travaux. Nous allons lister les principaux axes d'amélioration de vitesse du temps de traitement et d'utilisation des réseaux de neurones.

Dimensionnement d'un réseau de neurones : L'algorithme de rétropropagation fonctionne de manière sûre pour trouver la solution idéale si la fonction d'erreur ne contient pas de minima locaux. Néanmoins, de nombreux problèmes produisent des fonctions d'erreurs non-convexes, sujettes aux minima locaux. Comme cité précédemment, il est acceptable de se contenter d'un résultat issu d'un minimum local si celui-ci est assez probant (ceci reste à l'appréciation de l'utilisateur).

Il arrive toutefois que l'algorithme se retrouve coincé dans un minimum local non satisfaisant. On peut dans ce cas utiliser des méthodes stochastiques pour essayer de sortir du minimum local (ajout de bruit, modification de la procédure de changement de poids...). La méthode la plus classique reste l'ajout de neurones cachées, afin d'élargir la dimension du domaine. Cette méthode a ses limites car, comme expliqué partie 2.2.3, un système surdimensionné sera sujet au sur-apprentissage.

Il n'existe malheureusement toujours aucun outil pour déterminer a priori la dimension d'un réseau de neurones. Bien que la règle empirique de Baum-Haussler [Baum and Haussler, 1989] permettent d'avoir une idée sur le nombre d'unités cachés, elle ne permet pas de déterminer le nombre de couches cachées ou encore leurs nombres respectifs de neurones ; en outre, elle n'est pas toujours fiable.

Règle de Baum-Haussler :

$$n_c \leq \frac{n_i * \epsilon}{n_p + n_s}$$

Avec n_c le nombre d'unités cachées, n_i le nombre d'individus dans le jeu de données d'apprentissage, ϵ l'erreur relative acceptable, n_p le nombre de caractéristique d'un individu et n_s le nombre de neurones de sortie.

Le dimensionnement d'un réseau de neurones reste, en pratique, un procédé réalisé au travers de plusieurs essais et corrections.

La topologie des réseaux est très liée au problème traité. Etablir des topologies de références en fonction du domaine est une thématique de recherche très active. Ainsi, de nombreux travaux existent afin d'établir des réseaux de neurones de références dans des domaines précis (VGG [Chatfield et al., 2014], AlexNet [Krizhevsky et al., 2012b], NETtalk [Sejnowski and

Rosenberg, 1988]...). Malheureusement, nous n'avons pas pu, lors de nos expérimentations, bénéficier de tels réseaux de références.

Améliorations de l'algorithme et bonnes pratiques : Bien qu'il n'existe pas de moyen sûr de dimensionner un réseau de neurones a priori, il existe divers moyens d'accélérer l'apprentissage.

Une bonne gestion des paramètres d'un réseau de neurones est essentielle afin de produire un entraînement rapide et efficace. Le pas d'apprentissage fait partie de ces paramètres critiques. En effet, un pas d'apprentissage trop faible ralentira considérablement la convergence du réseau tandis qu'un pas trop élevé produira des oscillations et, dans le pire des cas, empêchera d'approcher du minimum, comme illustré par les graphiques figure 2.22.

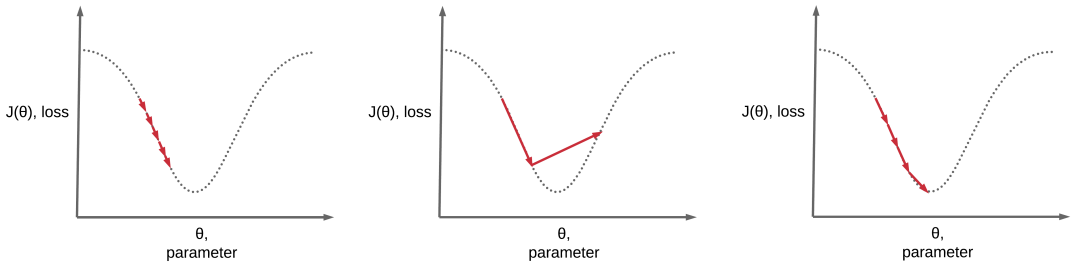

FIGURE 2.22 – Exemples schématisant les effets de différents pas sur le processus d'apprentissage via descente de gradient et rétropropagation. La courbe représente la fonction d'erreur et l'axe des abscisses tous les paramètres du réseau de neurones. Les effets d'un pas trop petit sont schématisés à gauche, ceux d'un pas trop grand au milieu, et ceux d'un pas acceptable à droite. Cette figure est tirée de [Ritchie Ng, 2019].

Néanmoins, même le choix d'un bon pas d'apprentissage peut entraîner un apprentissage très long ou des oscillations en cas de fonction d'erreur peu pentue ou d'un minimum situé au fond d'une courbe resserrée.

Une solution à ce problème est d'utiliser un facteur inertiel modifiant la règle d'apprentissage de la manière suivante :

$$\Delta(w_{ij}(t+1)) = -\eta * err(w_{ij}) + \alpha\Delta(w_{ij}(t))$$

avec α correspondant au moment de l'évolution de la vitesse de convergence, et $err(w_{ij})$ l'erreur associées au poids w_{ij}. La figure 2.23 illustre l'optimisation offerte par l'intégration d'un terme intertiel lors de la descente de gradient.

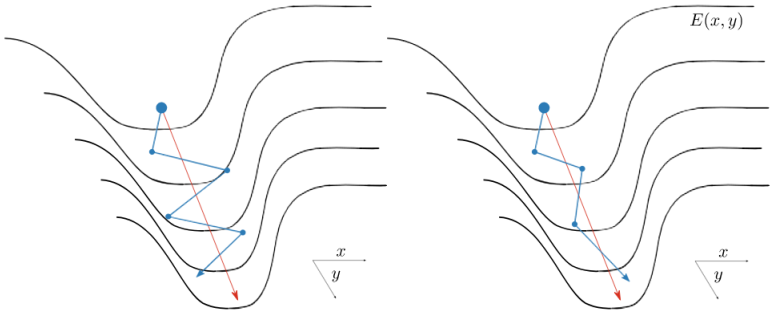

FIGURE 2.23 – Exemple de schématisation de l'utilisation d'un terme inertiel. En rouge la direction du minimum, en bleu à gauche, les oscillations d'un algorithme de rétropropagation avec un pas élevé, et en bleu à gauche, les oscillations d'un algorithme de rétropropagation avec moment. Un entraînement basé sur la rétropropagation avec moment a besoin de moins d'itérations pour converger vers le minimum. Cette figure a été tirée de [Ioannou, 2018] et inspirée par [Martens, 2010].

La « backpropagation with momentum » [Silva and Almeida, 1990] a été largement démocratisée peu après l'adoption de la rétropropagation standard. Depuis, de nombreuses optimisations ont été apportées à l'algorithme. Parmi les plus utilisées de nos jours, la plupart reposent sur l'intégration d'inertie. Ainsi, nous pouvons citer la méthode de Nesterov [Nesterov, 1983], Adagrad [Duchi et al., 2011], RMSProp [Hinton et al., 2012b]...

Comme nous l'avons vu précédemment, le choix d'un bon pas d'apprentissage est primordial. Lors de la conception d'un réseau de neurones, de nombreux facteurs peuvent être réfléchis et choisis de manière à optimiser le temps d'entraînement et la capacité d'un modèle à produire de bons résultats.

Parmi ces facteurs, l'initialisation des poids est un des aspects les plus importants à prendre en compte. L'initialisation des poids, constituant le point d'entrée de la tâche d'approximation de fonction lors de l'apprentissage, peut considérablement réduire le nombre d'itérations nécessaire et permettrait aussi d'être moins sensible aux phénomènes d'oscillations lors de convergence du gradient vers un minimum.

De nombreuses méthodes d'initialisation de poids existent et rendent un réseau sensiblement plus efficace qu'avec une initialisation « standard » (par exemple, une initialisation de tous les poids à 0, 1, ou tout autre constante).

Parmi elles, on trouve :

- une initialisation aléatoire (traditionnelle) suivant une distribution uniforme $\mathcal{U}(a, b)$ centrée autour de 0 avec a et b typiquement -1 et 1,
- une initialisation aléatoire suivant une distribution normale $\mathcal{N}(\mu, \sigma^2)$ avec une moyenne μ de 0 et un écart type σ généralement inférieur à 0.1,
- des initialisations aléatoires dépendant du nombre d'unité, tel que la méthode Xavier Glorot [Glorot and Bengio, 2010] où l'initialisation s'effectue de manière normale autour

de 0 avec un écart type égal à $\sqrt{\frac{2}{x+o}}$ avec x le nombre d'entrées d'une couche et o le nombre de sorties,

- ...

Cette liste n'est pas exhaustive mais inclut les méthodes d'initialisation les plus répandues. Ce sont notamment ces méthodes d'initialisation que nous avons utilisé lors de nos expérimentations

Implémentation sur GPU : Traditionnellement, les réseaux de neurones étaient entraînés sur des ordinateurs embarquant uniquement un CPU. Comme cité en début de section 2.4.6, il a été montré que de telles machines ne pouvaient effectuer les traitements lourds demandés par les réseaux de neurones de manière efficace. La plupart des implémentations modernes de réseaux de neurones sont basées sur des processeurs graphiques (graphics processing units, ou GPU).

Consacrés de base au rendu de jeux vidéo, les GPUs sont conçus pour réaliser rapidement de nombreuses opérations en parallèle. En effet, environnements, objets et personnages sont renseignés et caractérisés par des listes de coordonnées 3-D (vertex). Le GPU traitant l'information doit donc, au travers de diverses multiplications et divisions matricielles, convertir de nombreux vertex en coordonnées 2-D à afficher à l'écran. Une fois les coordonnées 2-D connues, le GPU calcule parallèlement la couleur de chaque pixel du rendu.

Dans les deux cas, les traitements à effectuer restent simples et ne sont pas comparables aux traitements réalisés par un CPU en termes de complexité, les CPUs pouvant effectuer des instructions bien plus diverses. Néanmoins les calculs sont très nombreux et doivent être effectués rapidement et sur de nombreuses données impliquant parfois des mémoires caches conséquentes (notamment au niveau des textures) afin d'avoir un rafraîchissement suffisant à l'écran. Les processeurs graphiques embarquent donc de très nombreuses unités de calcul parallèles, une bande passante mémoire élevée afin de délivrer les résultats des très nombreux calculs.

Les réseaux de neurones requièrent des performances ayant les mêmes caractéristiques que les algorithmes de rendus graphiques. En effet, les réseaux de neurones impliquent généralement de nombreuses données (paramètres, valeurs d'activations, valeurs de gradients...) qui doivent être mises à jour à chaque itération de l'entraînement. La mémoire cache d'un CPU traditionnel étant généralement insuffisante, la bande passante mémoire de celui-ci deviendrait un facteur limitant les performances. De plus, de nombreux calculs au sein d'un réseau de neurones peuvent être exécutés en parallèle. Tous les neurones au sein d'une couche donnée peuvent être traités indépendamment et de nombreux calculs sont réalisés sous forme d'opérations matricielles.

Ainsi, l'utilisation d'une implémentation de réseaux de neurones fonctionnant sur GPU permet un gain de temps conséquent face à l'approche traditionnelle. Steinkrau et al. [Steinkraus et al., 2005] ont implémenté sur GPU un réseau de neurones à deux couches et ont constaté une vitesse de traitement trois fois supérieure à la vitesse d'entraînement sur leur implémentation basée sur CPU. Peu après, Chellapilla et al. [Chellapilla et al., 2006] ont démontré que la même technique pouvait être utilisée pour accélérer les réseaux de neurones convolutionnels (dont nous expliquons le fonctionnement dans la partie 3.1).

Le succès de l'utilisation de processeurs graphiques pour les réseaux de neurones a explosé avec l'apparition des GP-GPUs (general purpose GPUs). Le langage CUDA [Cook, 2013] développé par NVIDIA permet d'exécuter du code massivement parallèle et d'exploiter la bande passante de ces processeurs en utilisant un modèle de programmation pratique et dédié à cette tâche. Cette plateforme a rapidement été adoptée dans le domaine du machine learning [Raina et al., 2009] [Ciresan et al., 2011] et est au cœur de divers framework de plus haut niveau permettant le développement de réseaux de neurones exécutables sur GPU et CPU sans avoir à changer le code (exemple : Theano [Theano Development Team, 2016], Torch [Collobert et al., 2002] ou encore TensorFlow [Abadi et al., 2016]).

Au sein de nos expérimentation, nous avons pu tirer parti du framework TensorFlow. Nous avons écrit notre code de manière classique et avons entraîné nos modèles sur GPU (Nvidia Tesla V100[1]).

Gestion des données d'apprentissage : Constituer un jeu de données d'entraînement de taille suffisante est un enjeu primordial lors de l'utilisation de réseaux de neurones. De la même manière que pour le dimensionnement d'un réseau de neurones, il n'existe pas de règle donnant un nombre d'individus minimum requis. Néanmoins, certains facteurs sont à considérer.

La taille du jeu de données à utiliser dépend de la taille du réseau et, de fait, si celui-ci a été correctement dimensionné, de la complexité de la fonction à approximer. De plus, la représentativité des données est un facteur qui sera déterminant lors de l'entraînement de celui-ci. Par exemple, si l'on prend un problème de classification, il est essentiel que toutes les classes soient équitablement représentées. En effet, nous cherchons un modèle ayant été influencé de la même manière pour toutes les sorties possibles. Ainsi, si dans les données brutes une classe est sous-représentée, il conviendra de chercher une solution à ce problème a priori (augmentation de données, diminution de la représentativité des autres classes...).

Dans notre cas, et notamment comme nous le développons chapitre 4, nous avons parfois eu besoin de contrôler le nombre d'anomalies présentes dans le jeu de données d'entraînement afin de rendre nos modèles plus ou moins sensibles à celles-ci. Plus un jeu de données d'entraînement inclut d'anomalies, moins le modèle entraîné aura tendance à être sensible à celles-ci.

La taille du jeu de données a un impact conséquent, tout comme la sélection des caractéristiques que nous utilisons (ou que nous sommes contraints d'utiliser).

Ainsi, comme nous le développons chapitre 3, enrichir notre jeu de données est une étape pouvant augmenter les performances d'un modèle. Tout comme en statistiques classiques, l'utilisateur a tout intérêt à chercher et à exploiter les potentielles variables cachées, pouvant exprimer de manière bien plus efficace les caractéristiques des individus. Dans le cas de la détection d'anomalies, l'exploitation de données contextuelles ou de capteurs supplémentaires permettrait notamment d'infirmer ou de confirmer la catégorisation d'une donnée. Par exemple, il est plus facile d'expliquer une température basse ayant lieu pendant la nuit que pendant la journée (information contextuelle sur la saisonnalité, cf : 3).

[1] https ://www.nvidia.com/fr-fr/data-center/tesla-v100/

Enfin, il est possible de faciliter l'apprentissage du modèle en effectuant divers pré-traitements sur les données [Pyle, 1999]. En effet, comme l'illustre la figure 2.24, normaliser les données permet de simplifier un problème qui aurait dû nécessiter un apprentissage plus poussé à la base. La normalisation permet notamment de remettre les données dans le même ordre de grandeur, décomplexifiant ainsi une partie de la tâche à réaliser. Ainsi, il est établi qu'un réseau de neurones entraîné avec des données normalisées convergera vers un optimum plus rapidement qu'un réseau de neurones entraîné avec les mêmes données non normalisées [Sola and Sevilla, 1997].

De nombreuses techniques de normalisation existent. Parmi les plus courantes, on trouve, si les données évoluent dans un domaine bien défini, la normalisation entre 0 et 1 (min-max scaling) :

$$x_{norm} = \frac{x - min(x)}{max(x) - min(x)}$$

ou encore la normalisation entre -1 et 1 :

$$x_{norm} = 2\frac{x - min(x)}{max(x) - min(x)} - 1.$$

La technique de normalisation que nous avons utilisée sur nos données est applicable aux données dont on ne connait potentiellement pas les extremums. Cette technique consiste à remplacer la donnée initiale par son z-score, calculé comme suit :

$$x_{norm} = \frac{x - \bar{x}}{\sigma}$$

avec $\bar{x}$ la valeur moyenne de x et σ son écart type. Cette technique a l'avantage de redistribuer les données autour de 0 et avec un écart type de 1.

Fonctions d'activation : Parmi les fonctions d'activation utilisées aujourd'hui figurent largement des fonctions imitant encore l'aspect tout ou rien de la fonction « Heavystep » tout en étant dérivables. La fonction logistique décrite dans la partie 2.4.4 en est un exemple, ou encore la fonction tangente hyperbolique.

$$f(x) = \frac{2}{1 + e^{-2x}} - 1$$

Celle-ci présente l'intérêt d'être une fonction sigmoïde centrée sur 0 et évoluant entre -1 et 1.

Une des fonctions les plus utilisées de nos jours, introduite en 2000 par Hahnloser et al. [Hahnloser et al., 2000], est la fonction ReLU (Rectified Linear Unit). Très simple, celle-ci correspond à la formule suivante :

$$f(x) = max(0, x)$$

Celle-ci présente divers avantages :

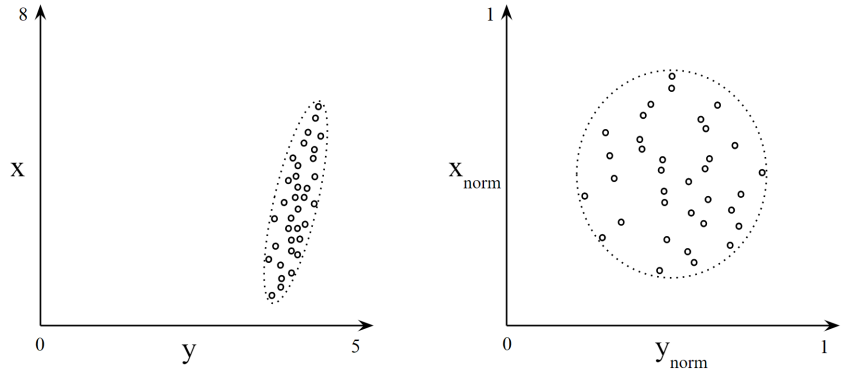

FIGURE 2.24 – Illustration de l'effet d'une technique de normalisation sur un jeu de données à deux dimensions avec un problème de clustering simple. On constate que les données brutes (gauche) nécessitent un apprentissage plus poussé que les données normalisées (droite). La normalisation permet de travailler sur des valeurs ayant le même ordre de grandeur et permet d'avoir un problème plus facile à généraliser.

- La possibilité d'envoyer une sortie nulle
- Meilleure dans les réseaux très profonds (composés de nombreuses couches)
- Efficiente en termes de calcul

Ces deux fonctions d'activation sont illustrées figure 2.25.

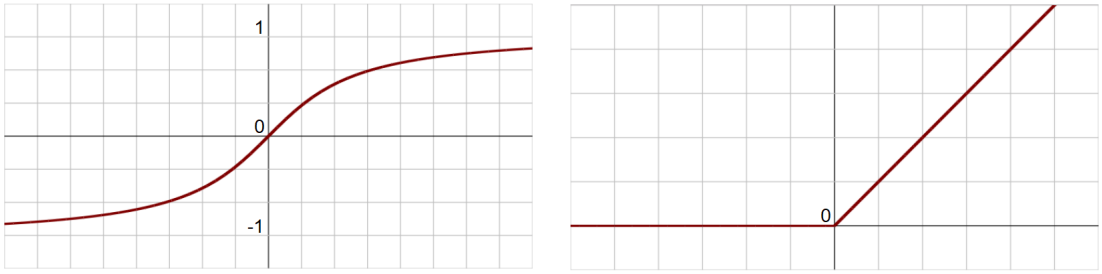

FIGURE 2.25 – Représentation graphique des fonctions d'activation tangente hyperbolique (gauche) et ReLU (droite).

Toutes ces fonctions d'activation peuvent être plus ou moins pertinentes en fonction du cas d'utilisation et leur choix est généralement effectué après avoir testé plusieurs d'entre elles lors du développement d'un réseau de neurones artificiels. Au sein de nos expérimentations, nous nous sommes servi essentiellement de la fonction tangente hyperbolique ReLU ainsi que la fonction identité (notamment dans les couches de sortie).

Dropout : Le dropout est une méthode de régularisation des poids lors de l'apprentissage [Srivastava et al., 2014]. Pour un coût pratiquement nul, cette méthode permet d'augmenter les performances d'un réseau de neurones et de le rendre sensiblement résistant au sur-apprentissage.

On peut voir le dropout comme une méthode permettant d'entraîner l'ensemble des sous-réseaux appartenant au réseau de base. Historiquement, entraîner plusieurs réseaux de neurones et obtenir ensuite un consensus (usuellement la moyenne arithmétique des résultats) lors de la phase de test s'est révélé efficace. Cette méthode (aussi appelée « bagging », détaillée dans un paragraphe qui suit) est néanmoins difficile à mettre en place, notamment lors de l'utilisation de modèles conséquents.

Le dropout permet de fournir une approximation fidèle de bagging [Dietterich, 2000] de manière très optimisée. En effet, cette méthode consiste, lors d'une itération de la phase d'entraînement, à désactiver des neurones (ne faisant pas partie de la couche de sortie) en fonction d'une probabilité p attribuée a priori (cf : exemple figure 2.26).

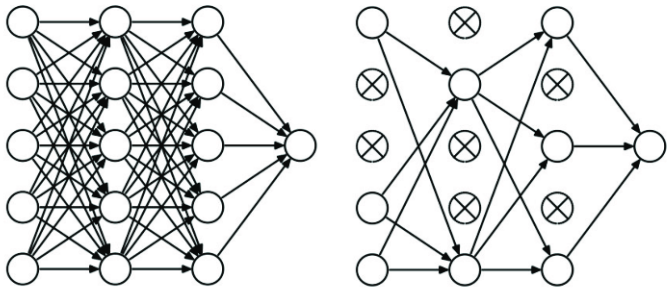

FIGURE 2.26 – Exemple tiré de [Srivastava et al., 2014] de schéma de réseau de neurones standard (gauche) et schéma du même réseau de neurones sur lequel un masque de dropout a été appliqué (droite).

Le bagging requiert l'entraînement de k modèles différents nécessitant réflexion et construction de k différents jeux d'entraînement. Le dropout permet de fournir une estimation issue de ce procédé, mais en utilisant seulement le jeu de données d'origine et un nombre plus élevé de modèles (selon la taille du modèle de base). En effet, ce nombre plus élevé de modèles correspond à la totalité des combinaisons possibles de neurones que l'on peut inactiver.

A chaque itération, le masque de dropout $\boldsymbol{m}$ est différent et désigne quelle unité inclure. Ainsi, chaque itération visera à réduire l'erreur $E(\boldsymbol{\theta}, \boldsymbol{m})$, $\boldsymbol{\theta}$ étant l'ensemble des paramètres du sous-réseau dépendant du masque $\boldsymbol{m}$. Pour chaque entrée $\boldsymbol{x}$, sa sortie associée $\boldsymbol{y}$ suit donc, pour chaque sous-réseau, la distribution de probabilité $p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{m})$. L'estimation issue de la totalité des itérations correspond donc à :

$$\sum_{\boldsymbol{m}} p(\boldsymbol{m}) p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{m})$$

avec $p(\boldsymbol{m})$ la probabilité que le masque $\boldsymbol{m}$ ait été utilisé.

Usuellement, même un nombre réduit de masques, et donc un nombre réduit de sous-réseaux, permet d'augmenter les performances d'un réseau de neurones. De plus, le principe même d'utilisation de masques de dropout rend le réseau beaucoup moins prompt au sur-apprentissage. En effet, tous les sous-réseaux utilisés lors de l'entraînement ont des poids en commun. Les entraînements des sous-réseaux ne sont donc pas entièrement indépendant (l'entraînement d'un sous-

réseaux impactera potentiellement les poids d'un autre sous-réseaux). Ainsi, plus le nombre de masques $\boldsymbol{m}$ utilisés est élevé, plus la possibilité de sur-apprentissage va décroître.

A ce stade, nous avons décrit la majorité des améliorations et optimisations dont nous avons pu nous servir lors de nos expérimentations. L'optimisation des réseaux de neurones reste un domaine actuellement très actif et de nouvelles techniques apparaîssent encore souvent de nos jours.

2.5 Auto-encodeurs

Comme nous l'avons expliqué section 2.3.1, nous nous sommes orienté vers les auto-encodeurs. Nous détaillons dans cette partie le modèle à la base de nos expérimentations.

Le modèle de base

Un réseau de neurones encodeur-décodeur (auto-encodeur) est un réseau de neurones entraîné à produire une sortie identique à son entrée [Goodfellow et al., 2016a]. Au sein du réseau, une couche cachée h représente un code utilisé pour recréer l'entrée. Ainsi, ce type de réseau de neurones peut être décomposé en deux parties : une fonction qui encode les données en entrée $h = f(x)$ et une fonction qui décode h de sorte à produire une reconstruction $r = g(h)$.

Un auto-encodeur apprenant parfaitement $g(f(x)) = x$ serait donc en mesure de produire un code h utilisable pour recréer l'entrée à chaque utilisation. Ce type d'application, à première vue très utile, présente en réalité peu d'intérêts. Généralement, les auto-encodeurs sont conçus pour être incapables d'apprendre à copier parfaitement. En effet, lors de la conception de l'auto-encodeur, celui-ci est volontairement restreint afin qu'il ne soit capable de reproduire que des données ressemblant à celles utilisées lors de l'entraînement. Comme illustré figure 2.27, la couche h sera de taille inférieure à celle de la donnée à reproduire. Ainsi, le modèle sera forcé de prioriser les aspects les plus représentatifs des données et apprendra les propriétés les plus utiles.

Comme indiqué dans la partie 2.3, les auto-encodeurs étaient traditionnellement utilisés pour la réduction de dimension.

Denoising auto-encodeur

Comme nous venons de le décrire, on peut considérer les auto-encodeurs comme des réseaux de neurones feed-forward standards. Les fonctions de pertes sont les mêmes.

Ainsi, un réseau de ce type cherchera à minimiser la perte L (cf : « Loss » section 2.2.3) :

$$L(\boldsymbol{x}, g(f(\boldsymbol{x}))),$$

avec f la fonction d'encodage du réseau, g la fonction de décodage et L la fonction de perte.

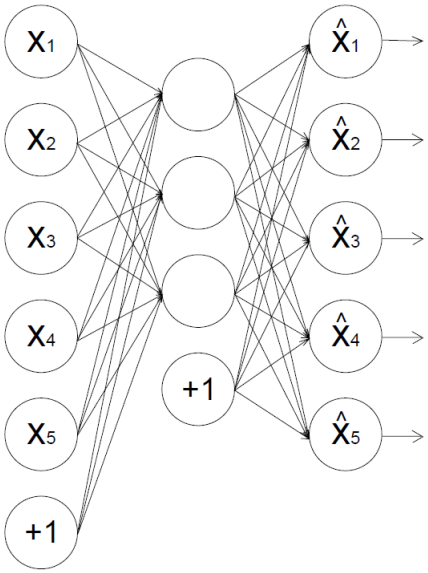

FIGURE 2.27 – Schéma tiré de [Sakurada and Yairi, 2014] d'un auto-encodeur simple présentant une couche d'entrée, une unique couche cachée plus petite, et une couche de sortie de la même taille que la couche d'entrée. Ici, les neurones étiquetés "+1" sont des neurones biais (ces neurones, utilisés dans la plupart des réseaux de neurones, permettent de décaler les fonctions d'activation de chaque neurone de la couche suivante).

Comme évoqué dans la partie précédente, ce procédé encourage le réseau à approximer la fonction identité correspondant aux données $\boldsymbol{x}$. Un denoising auto-encodeur (DAE) cherchera à minimiser

$$L(\boldsymbol{x}, g(f(\tilde{\boldsymbol{x}}))),$$

où $\tilde{\boldsymbol{x}}$ est une copie corrompue des données initiales $\boldsymbol{x}$. Les DAEs doivent donc apprendre à enlever cette corruption plutôt que de simplement copier les données [Vincent et al., 2008].

Cette étape force implicitement f et g à apprendre la structure de $p(\boldsymbol{x})$, distribution du jeu de donnée d'entraînement $\boldsymbol{x}$, ainsi que les relations globales entre les caractéristiques de chaque individu.

Il est donc nécessaire d'introduire un procédé de corruption c indépendant du DAEs des données tel que $p(\tilde{\boldsymbol{x}}) = c(p(\boldsymbol{x}))$. Ainsi, le DAE cherchera donc à minimiser :

$$L(\boldsymbol{x}, g(f(c(\boldsymbol{x})))),$$

en gardant l'approche standard basée sur la minimisation du gradient.

Au sein de nos expérimentations, nous avons utilisé un procédé de corruption des données basé sur l'utilisation d'une étape de dropout sur la couche d'entrée, tel que décrit dans [Vincent et al., 2010]. Ainsi, la corruption des données a lieu en ignorant certaines caractéristiques à chaque itération de l'apprentissage et le procédé est directement inclu dans le réseau.

Utilisé dans le cadre de la détection d'anomalies, ce procédé permet au réseau de déceler les relations entre les caractéristiques de manière plus efficace [Xu et al., 2015] [Marchi et al., 2015]. En effet, en cas de relations redondantes (par exemple, plusieurs capteurs de température répartis dans différentes pièces) ou prépondérantes, cela force l'auto-encodeur à « s'intéresser » aux autres relations plus discrètes mais néanmoins présentes, afin de minimiser l'erreur.

Ce procédé aide à la différenciation des données présentant une légère altération due à un bruit inhérent au cas d'utilisation d'une donnée reflétant une anomalie réelle. Dans notre cas, un bruit peut provenir de la précision du capteur en jeu, d'altérations directement issues de l'activité humaine (par exemple, se placer devant un capteur ou toucher involontairement celui-ci), etc.

CHAPITRE 3

TRAVAIL EXPLORATOIRE – ETUDE COMPARATIVE

Comme nous l'avons présenté dans la partie 2.1, les données issues de l'IoT sont sensibles à divers types d'anomalies. Les auto-encodeurs classiques, tout comme la majorité des autres algorithmes de détection d'anomalies, considèrent chaque individu du dataset de manière indépendante.

De tels algorithmes ne sont donc pas sensibles aux anomalies de troisième catégorie, celles-ci étant caractérisées par des variations anormales dans les données. Afin de tirer parti de la dimension séquentielle des données, nous proposons d'utiliser deux types d'architectures prenant en compte la temporalité et de les appliquer aux auto-encodeurs.

Nous traitons dans ce chapitre une étude comparative de ces deux types d'architectures appliquées à des modèles auto-encodeurs traitant de données de bâtiments connectés. En effet, comme nous l'avons indiqué partie 2.3.2, des travaux impliquant des auto-encodeurs convolutionnels et récurrents existent. Néanmoins, aucune comparaison n'a encore été effectuée, à notre connaissance. Nous présentons tout d'abord les réseaux de neurones récurrents ainsi que les réseaux de neurones à convolution. Nous détaillerons ensuite l'expérimentation réalisée dans le but de comparer les performances de ces deux types de modèles appliqués à notre cas d'utilisation.

3.1 Réseaux de neurones à convolution

Les réseaux de neurones à convolution, ou réseaux de neurones convolutionnels (CNN, Convolutional Neural Networks) sont spécialisés dans le traitement de données ayant une topologie sous forme de grille.

Parmis les données de ce type, on trouve notamment les images, présentées sous forme de grilles à 2 dimensions composées de pixels, ainsi que les séries temporelles, présentées sous forme de grilles à 1 dimension composées de valeurs échantillonnées à intervalle régulier.

Les CNNs, introduits par LeCun en 1989 [LeCun et al., 1989], ont eu un très grand succès dans de nombreuses applications pratiques.

Comme son nom l'indique, les CNNs opèrent en employant une opération mathématique appelée convolution, empruntée au domaine du traitement de matrices et plus particulièrement au traitement d'image.

Nous détaillons dans cette partie le fonctionnement des CNNs, expliquant dans un premier temps la convolution, comment celle-ci est utilisée au sein des CNNs, et expliquant ensuite le fonctionnement d'autres opérations telles que le pooling.

3.1.1 Modèle théorique

La convolution

Dans sa forme la plus générale, la convolution est une opération sur deux fonctions. Nous illustrons cette notion par l'exemple suivant.

On suppose que l'on dispose d'un capteur de décibel physique constamment positionné à côté d'un arbitre lors d'un évènement sportif et que nous cherchions à déterminer un moyen de stopper automatiquement le chrono en cas de coup de sifflet.

Notre capteur envoie à intervalle régulier (par exemple, 500 millisecondes) sa valeur courante $x(t)$ au temps t. x et t ont toutes les deux des valeurs réelles, et la valeur de $x(t)$ ne dépend pas de $x(t-1)$.

On suppose maintenant que le lieu où est positionné le capteur soit bruyant de base (présence de foule, parole de l'arbitre, etc.) et que le relevé des valeurs par le capteur soit lui même soumis à un certain bruit.

Afin d'obtenir une estimation moins bruitée du moment où il y a un coup de sifflet, on cherche à faire la moyenne de plusieurs mesures. De toute évidence, les mesures récentes sont plus importantes que les anciennes, donc nous voulons une moyenne pondérée donnant un poids plus élevé aux mesures les plus récentes.

Nous devons donc créer une fonction de pondération $w(a)$, où a est l'âge de la mesure. En appliquant cette fonction de pondération à la fonction $x(t)$, on obtient une nouvelle fonction s qui nous permet d'avoir une estimation lissée des moments sifflés :

$$s(t) = \int x(a)w(t-a)da. \tag{3.1}$$

Ce calcul est celui effectué lors d'une convolution. La convolution est typiquement exprimée de cette façon :

$$s(t) = (x * w)(t).$$

Dans notre exemple, w est soumis à quelques limitations. Par exemple, nous avons besoin que w suive une fonction de densité valide, et que $w(a)$ nous donne 0 en cas d'argument négatif.

De manière générale, la convolution est définie pour n'importe quelle fonction dont l'intégrale (comme celle présentée dans la formule 3.1) est définie. On peut donc imaginer de nombreuses autres fonctions qu'une moyenne pondérée.

Au sein de la terminologie des CNNs, le premier argument (dans notre exemple, la fonction x) est défini comme l'entrée, et le second argument est défini comme le kernel [Haussler, 1999] [Bloomenthal and Shoemake, 1991], aussi appelé masque, de la convolution. Celui-ci correspond donc à une matrice de poids appliquée aux données sur lesquelles la convolution est utilisée.

Dans notre exemple, nous imaginons un cas se déroulant en temps réel avec des données dépendant d'un réel t. Dans les cas pratiques où la convolution est utilisée, les données sont bien plus souvent indexées par un entier et toutes les données sont déjà connues.

En supposant que x et w dépendent d'un entier t non borné, on peut définir la convolution discrète :

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a).$$

Il est courant, dans les applications au machine learning, que l'entrée soit un tableau à plusieurs dimensions et que le kernel soit un tableau à plusieurs dimensions contenant des paramètres ajustés par l'algorithme.

Si l'on développe la formule précédente pour un tableau à deux dimensions représentant par exemple une image en noir et blanc, on peut imaginer une convolution sur plusieurs axes comme suit :

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n)$$

pour l'entrée I, et un kernel à deux dimensions K (le traitement effectué par une convolution est illustré par la figure 3.1).

Grâce à la commutativité de la convolution, on peut reformuler la formule ci-dessus. De nombreuses implémentations dans les bibliothèques de machine learning s'appuient sur une légère variante basée sur la commutativité de la convolution :

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m+i, n+j)K(m,n)$$

appelée « cross-correlation » [Knox, 1974]. Cette variante, utilisée majoritairement de nos jours, est aussi celle que nous avons sélectionné afin de réaliser nos expérimentations.

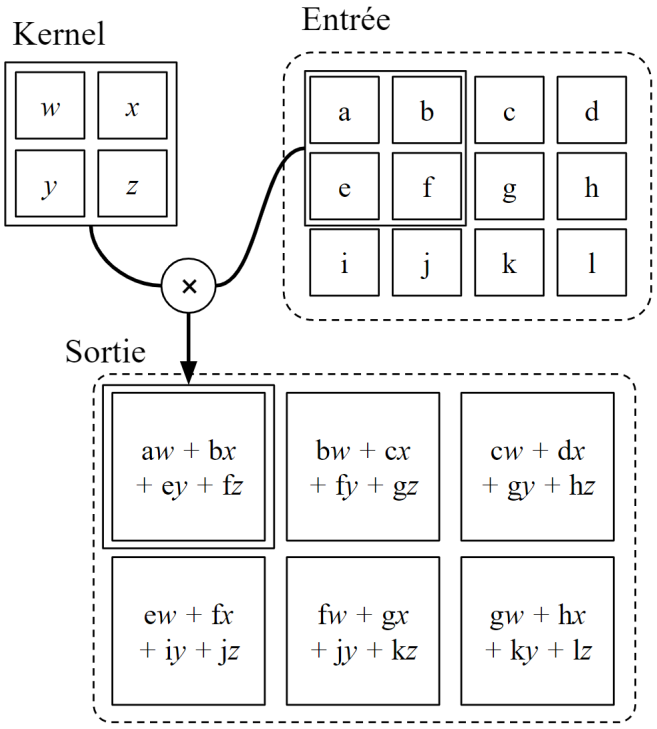

FIGURE 3.1 – Exemple d'une convolution à 2 dimensions. L'opération représentée par les flèches constitue la première itération de la convolution. Dans la sortie finale, on voit que seules les opérations où le kernel est totalement pris en compte sont effectuées. Exemple inspiré par [Goodfellow et al., 2016b].

Le principe de convolution permet, au sein des réseaux de neurones, des intéractions locales (qu'on appellera « sparse »).

En effet, au sein des réseaux neurones « classiques » présentés dans la partie 2.4.6, les couches sont connectées de manière dite « dense ». Chacun des neurones d'une couche sera connecté à tous les neurones de la couche suivante.

Dans un CNN, utiliser un kernel d'une taille plus faible que celle de l'entrée permet de dégager des informations locales pertinentes qui auraient été noyées dans la quantité d'informations traitée par une couche dense standard.

Par exemple, une image peut être constituée de milliers voir de millions de pixels. Une couche dense standard serait sûrement inefficace dans le traitement de celle-ci, ou extrêmement longue et compliquée à entraîner. Une couche opérant une convolution permettrait, quant à elle, d'effectuer un premier filtre ne gardant que les caractéristiques locales les plus importantes (par exemple, les bords présents dans l'image) avec un kernel qui n'occuperait que quelques dizaines de pixels.

L'utilisation de la convolution réduit donc aussi le nombre de paramètres à stocker et à modifier. Un CNN est donc a priori beaucoup plus efficient qu'un réseau standard, étant donné que le calcul d'une sortie compte beaucoup moins d'opérations qu'avec une approche standard.

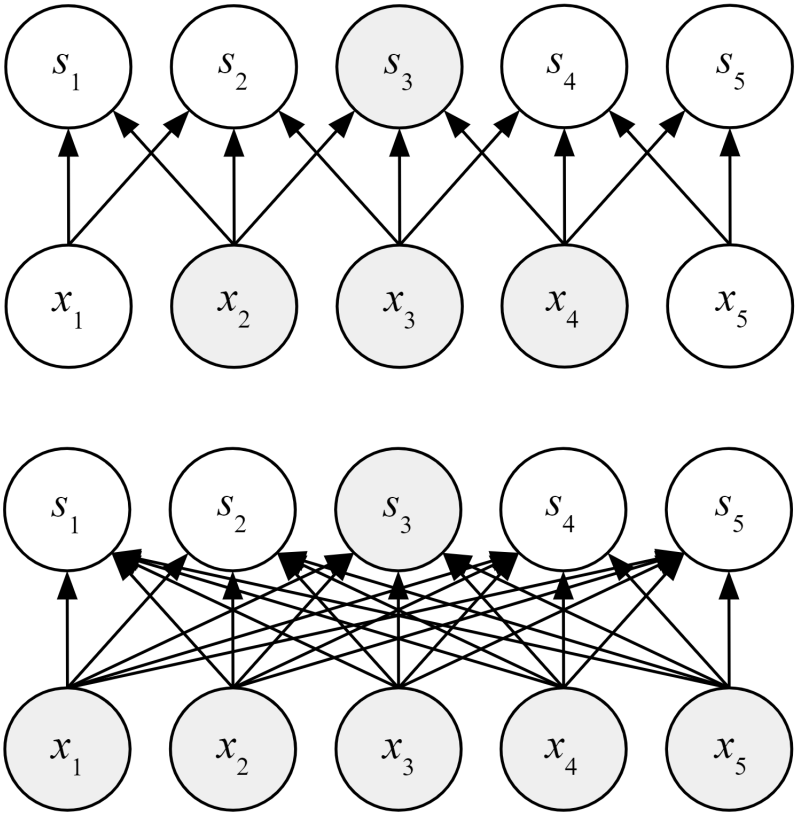

FIGURE 3.2 – Illustration d'une couche de convolution utilisant un kernel de 3 (en haut) et d'une couche dense standard (en bas). On constate que le nombre d'informations pour une unité de la couche de convolution est bien plus réduit que pour une unité de la couche dense (connectivité sparse). En effet, selon notre exemple, la cellule s_3 est affectée par 3 unités d'entrée dans la couche de convolution tandis que la même cellule est affectée par toutes les unités de $\boldsymbol{x}$ dans une couche dense.

En effet, pour une couche, si on a m entrées et n sorties, le temps de calcul sera de $O(m * n)$ avec l'approche standard. Si on limite le nombre de connexions qu'une sortie doit à avoir à un nombre k, alors la couche connectée de manière « sparse » nécessitera $k*n$ paramètres, limitant ainsi le temps de calcul à $O(k * n)$.

Dans de nombreux cas pratiques, il est possible d'obtenir de très bons résultats avec une limite k ayant un ordre de grandeur largement inférieur à m. Nous montrons une illustration de couches de réseaux de neurones effectuant une convolution à une dimension sur la figure 3.2 et à deux dimensions sur la figure 3.3.

Ces connexions sparses permettent aussi d'avoir un partage des paramètres au travers des couches successives au sein d'un CNN. Le partage de paramètres au sein d'un CNN est illustré par la figure 3.4.

Ce partage de paramètres permet l'extraction et le traitement d'informations pouvant être utiles, notamment lorsque l'on positionne une couche dense après des couches de convolutions suc-

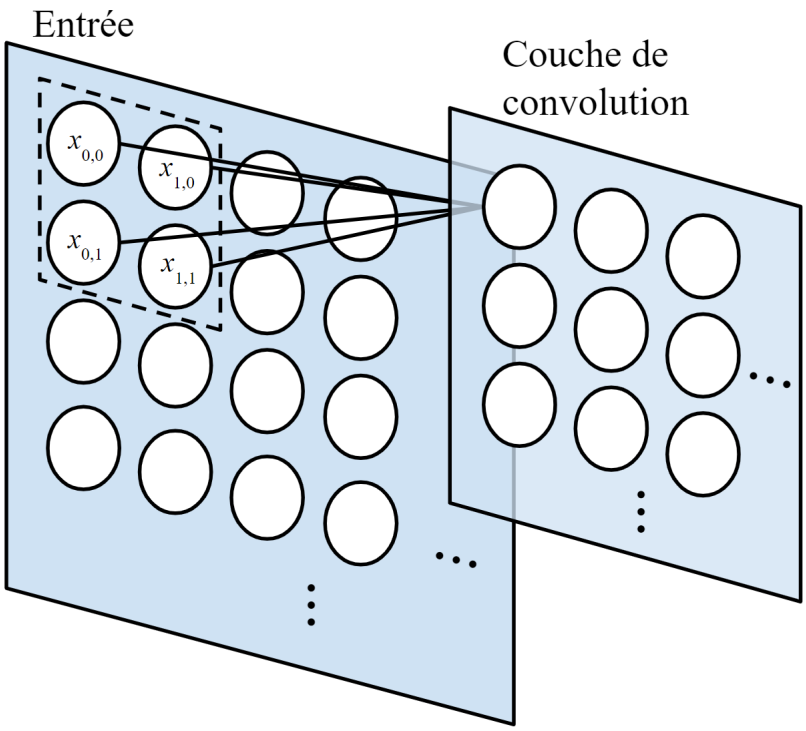

FIGURE 3.3 – Illustration d'opération effectuée par une couche de convolution sur une couche d'entrée à deux dimensions. Ici la première itération de la convolution est représentée par les liens liant les neurones présents dans la zone en pointillés et le premier neurone de la couche de convolution pour un kernel de 2 sur 2. De la même manière que le fonctionnement illustré par la figure 3.1, seul le premier neurone de la couche de convolution bénéficie des informations issues des neurones mis en évidence. Ici, les coefficients du kernel ne seront pas tout le temps fixes (contrairement aux coefficients w,x,y et z de la figure 3.1, qui restent les mêmes pour toute la convolution) car chaque coefficient sera déterminé lors de la phase d'apprentissage.

cessives. Une couche dense permettrait en effet de mettre en commun toutes les informations extraites par les convolutions successives, permettant ainsi la production d'une sortie pertinente.

De plus, il est également possible de positionner une couche dense juste après une couche unique de convolution, permettant ainsi de produire un résultat prenant directement en compte les informations locales. Nous développons cet aspect dans la partie 3.1.1.

Cette forme de partage de paramètres apporte aux couches de convolutions la propriété d'équivariance à la translation. Dans le cas de traitements de séries temporelles, on aurait donc une convolution produisant une chronologie montrant à quelles périodes apparaîssent des caractéristiques au sein de l'entrée. Si on décale un évènement dans l'entrée, alors une représentation identique sera présente dans la sortie, mais décalée de la même manière.

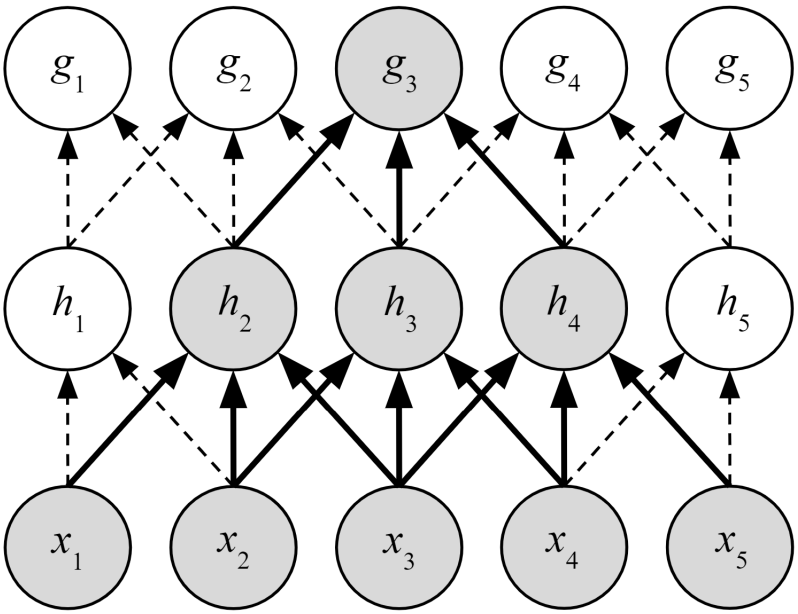

FIGURE 3.4 – Partage de paramètres au sein de couches successives de convolution. Pour une unité concernée, nous avons mis en valeur les paramètres concernés (en gras). On constate qu'une unité de la couche réceptive finale g, bien que n'utilisant que les n paramètres de la convolution, est affectée par la totalité des entrées $\boldsymbol{x}$. En effet, ces n paramètres ne concernent qu'une partie des sorties de la couche intermédiaire $\boldsymbol{h}$, mais les sorties associées sont elles mêmes des combinaisons d'informations locales concernant l'ensemble de la couche $\boldsymbol{x}$. Au sein d'une couche dense, l'intégralité des informations seraient utilisées par chacune des unités de la couche intermédiaire et on perdrait cette notion de partage de paramètres à travers les couches. Schéma inspiré de [Goodfellow et al., 2016b].

Ainsi, dans notre cas, les motifs récurrents (augmentation et baisse de luminosité en fonction du lever du jour, augmentation de la température d'une pièce lors de l'allumage d'un radiateur...) seront représentés de la même manière peu importe l'endroit de leur apparition dans l'entrée.

Le pooling

Comme précédemment évoqué, il est possible de positionner une couche dense directement après une couche de convolution, permettant ainsi d'utiliser une combinaison non linéaire de $n * m$ paramètres, n étant la taille de la convolution initiale et m étant la taille de la couche dense.

En effet, l'utilisation typique d'une couche de convolution consiste en trois étapes :

- Convolution initiale : extraction d'informations locales,
- étape de « détection » : couche dense avec activation non-linéaire,
- phase de pooling : filtre des résultats.

Appliqué à la sortie d'une couche, le pooling permet d'effectuer un traitement offrant un résumé statistique des sorties [Zeiler and Fergus, 2013]. La fonction de pooling la plus utilisée (aussi celle que nous avons utilisé dans nos modèles) est le max pooling.

La fonction de max pooling consiste simplement à signaler les valeurs les plus hautes dans un voisinage. D'autres fonctions de pooling consiste à calculer et à fournir en sortie la moyenne des valeurs au sein d'un voisinage.

Dans tous les cas, le pooling aide à déterminer une représentation approximativement invariante aux translations limitées effectuées sur l'entrée. Dans le cas du pooling, cette invariance approximative aux translations se traduit par une sortie quasi-identique si on translate peu l'entrée.

L'intérêt du pooling, malgré cet aspect approximatif, se trouve dans le fait que nous nous soucions plus de savoir si un motif est bien présent plutôt que où est ce que celui-ci se situe. Par exemple, dans le cas de reconnaissance faciale au sein d'image, il est plus important de savoir si un visage est bien présent et où il est généralement situé plutôt que de connaître exactement la position de toutes ses caractéristiques (yeux, bouche, etc.).

Nous pouvons bénéficier de cet aspect du pooling dans notre cas, où il est plus important de savoir si un évènement a bien lieu dans les proportions générales plutôt que de connaître le moment exact de sa venue. En effet, nous aurons tendance à fournir à nos modèles des fenêtres de données consécutives et à effectuer une convolution qui se « déplacera » uniquement latéralement en englobant tous les capteurs.

Si l'on prend le cas d'un capteur de température, il aura tendance à toujours augmenter de la même façon en début de journée (modulo la température extérieure, la saison courante, etc.). Il est donc plus important de savoir si cette augmentation a bien lieu dans les bonnes proportions plutôt que la date exacte de sa venue. Le pooling et son invariance approximative à la translation sont illustrés figure 3.5.

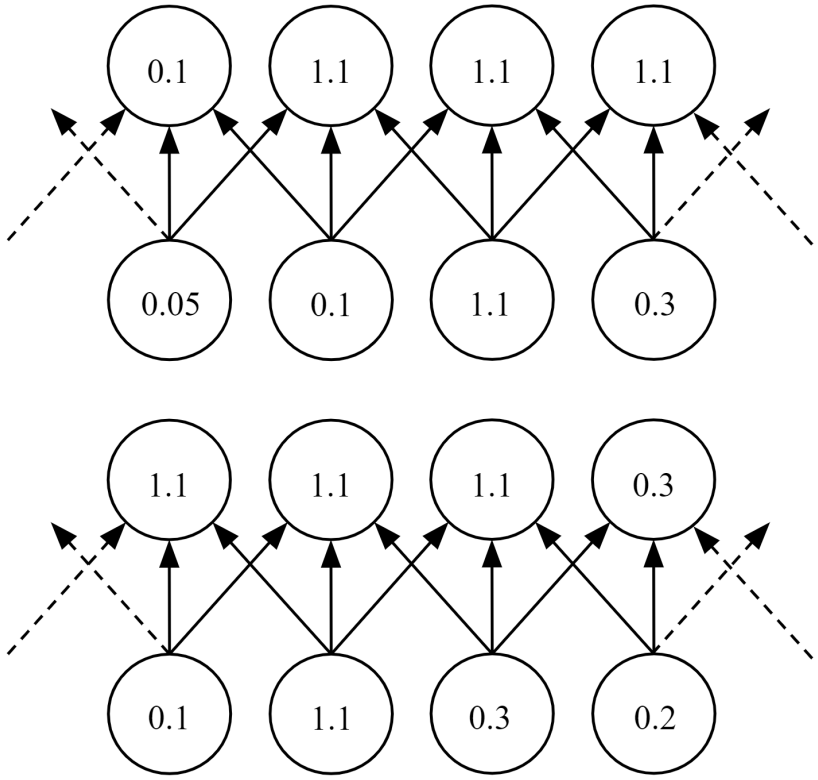

FIGURE 3.5 – Dans les deux cas illustrés, la couche inférieure correspond à la sortie de l'étape de détection et la couche supérieure correspond à la sortie du max pooling. Ici, il y a autant d'unités de pooling que dans la couche précédente. Ainsi, l'opération de pooling a lieu toutes les unes unités d'entrée, on dit qu'elle est effectuée avec un décalage, ou une foulée (plus communément appelée « stride »), de 1. En bas, une représentation du même réseau qu'en haut, avec un décalage vers la gauche des unités d'entrée. On constate que ce décalage ne change que la moitié des valeurs dans la couche supérieure. Cet exemple illustre l'invariance partielle à la translation du max pooling.

Comme le pooling résume les sorties dans un voisinage donné, il est possible d'utiliser moins d'unités que la couche précédente. En utilisant une couche de pooling synthétisant les informations de la couche précédente avec des unités espacées de k timesteps (dans notre cas, ou pixels lors du traitement d'une image), les couches suivantes auront environ k fois moins d'informations à traiter (cf : figure 3.6).

En plus de baisser la quantité de mémoire et de traitements requis, cette réduction permet un traitement efficace des images (et des fenêtres de séries temporelles), notamment lorsqu'une couche de pooling est située avant une couche dense ou une autre couche de convolution [Tolias et al., 2015].

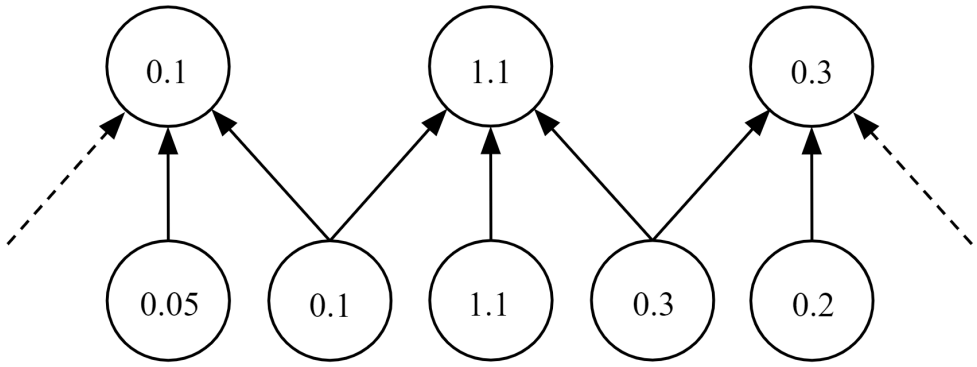

FIGURE 3.6 – Reprise du cas présenté dans la figure 3.5 pour illustrer le max pooling avec sous-échantillonnage. Ici, le max pooling est utilisé avec une stride de deux entre les unités de pooling. On constate que, pour une taille deux fois réduite (si l'on considère l'entièreté de la couche et pas seulement les unités représentées), les sorties de la couche précédente sont représentées équitablement.

3.1.2 Auto-encodeurs à convolution

Comme nous l'avons vu dans la partie précédente, une succession de couches de convolutions et de couches de pooling effectuant un sous-échantillonnage réduira la taille d'une entrée. Au fur et à mesure des traitements effectués par ces couches, ce sont les caractéristiques de plus en plus essentielles de l'entrée qui seront dégagées.

Une des applications est de réduire la dimension de l'entrée afin de résumer celle-ci dans un vecteur latent afin d'aboutir à une fonction d'encodage d'auto-encodeur.

Pour décoder ce vecteur, le cheminement inverse est suivi. En effet, la convolution est aussi utilisée afin d'extraire les données du vecteur qui seront utiles à sa reconstruction au travers de couches denses.

Afin de restaurer la dimension initiale, diverses méthodes existent. L'une des méthodes les plus répandues et les plus efficaces est le simple ré-échantillonnage de l'information, effectuant ainsi la tâche inverse d'une couche couplant max pooling et sous-échantillonnage [Holden et al., 2015] [Volpi and Tuia, 2016]. Lors de cette phase, chaque valeur issue de la convolution précédente sera répétée n fois.

La fonction de décodage d'un auto-encodeur à convolution consistera donc en une succession de couches de convolution et de couches de ré-échantillonnage. Suivant cette succession de couches, la fonction de décodage se terminera typiquement par une couche dense afin de recréer les données initiales.

Ainsi, dans notre cas, on a :

- Une entrée consistant en une série temporelle multivariée présentée sous forme de grille $\boldsymbol{x}$ de taille $n*t$, n étant le nombre de caractéristiques (ou de séries temporelles univariées) et t timesteps.

- Un CNN encodeur, constitué essentiellement d'une succession de combinaisons convolution / max pooling. Les convolutions sont effectuées de manière indépendante sur chaque série univariée et sont donc à une seule dimension.

- Un vecteur latent de taille c systématiquement inférieure à $n * t$, étant donné le sous-échantillonnage ayant lieu lors des phases de max pooling.

- Un CNN décodeur, constitué d'une succession de combinaisons convolution / ré-échantillonnage, puis généralement d'une couche dense recréant la donnée initiale.

De la même manière qu'avec un auto-encodeur standard (cf : partie 2.5), on peut comparer la sortie avec l'entrée afin de déterminer si celle-ci contient une anomalie. La figure 3.7 montre un exemple d'architecture d'auto-encodeur à convolution.

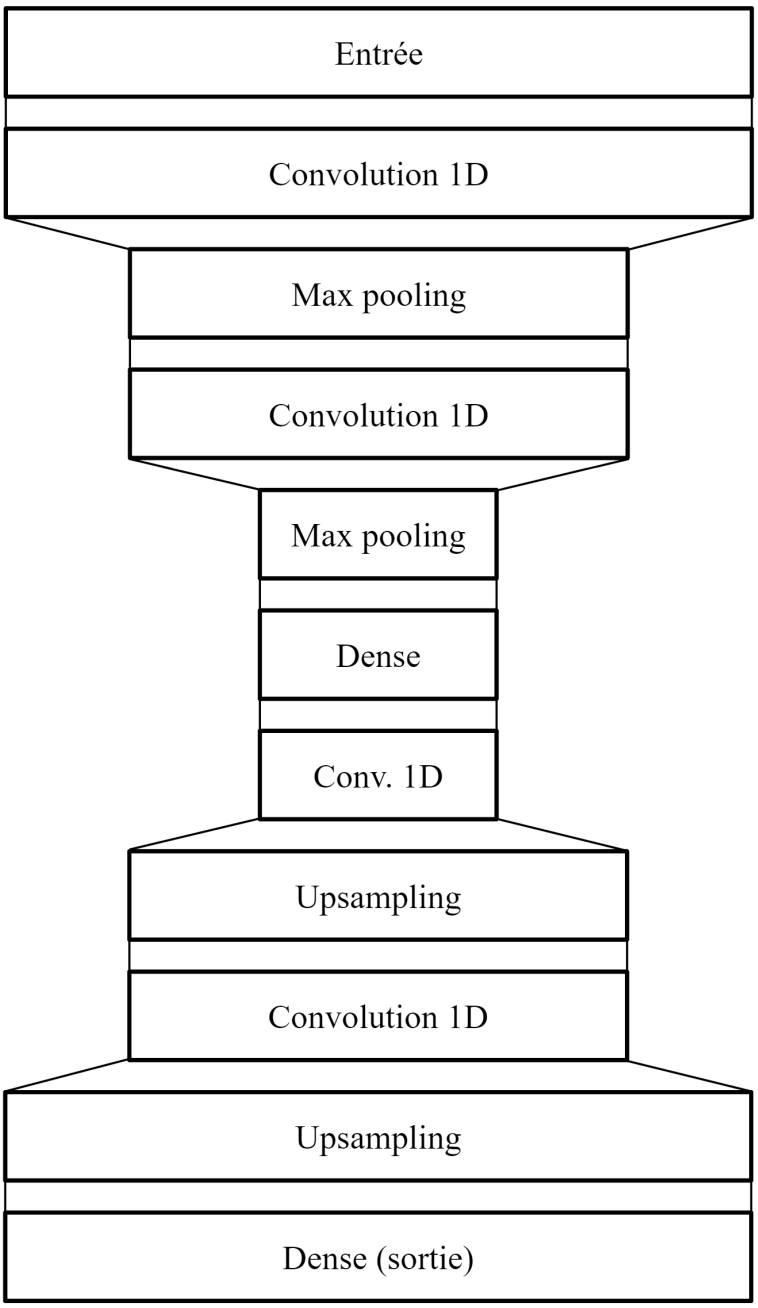

FIGURE 3.7 – Schéma illustrant un auto-encodeur à convolution typique, utilisant 2 phases de convolution lors de l'encodage et 2 phases de convolution lors du décodage. A chaque étape de max pooling, le sous-échantillonnage réduit la taille de l'information. Ainsi, après la deuxième couche de max pooling dans cet exemple, l'information est compressée en un vecteur latent qui sera ensuite exploité par une couche dense avant d'être soumis aux couches combinant convolution et ré-échantillonnage (Upsampling). Les fonctions d'activation ne sont pas précisées et dépendent du cas d'utilisation.

3.2 Réseaux de neurones récurrents

3.2.1 Le modèle théorique

Les réseaux de neurones récurrents (ou RNNs, Recurrent Neural Networks) sont une famille de réseaux de neurones destinés à traiter des données séquentielles [Rumelhart et al., 1986]. Si les réseaux de neurones à convolutions sont spécialisés dans les données sous forme de grille telle que les images, les réseaux de neurones récurrents sont spécialisés dans le traitement de séquences de valeurs $\boldsymbol{x}^{(1)}$,...,$\boldsymbol{x}^{(\tau)}$ [Graves, 2012].

L'une des idées derrière les réseaux de neurones récurrents vient des modèles d'apprentissage automatique et des modèles statistiques des années 80 et est aussi présente dans les CNNs : le partage de paramètres au sein de différentes parties du modèle.

Cette notion est néanmoins plus développée avec les RNNs.

En effet, avec des paramètres séparés pour chaque itération dans le temps (on parlera de timesteps), on ne pourrait pas généraliser ou partager d'informations statistiques pertinentes concernant les relations des différentes valeurs à travers la séquence.

Le partage de ces paramètres repose sur le principe de déploiement de graphes de calcul.

Un graphe de calcul est une manière de formaliser la structure d'un cheminement de traitements, tels que ceux que l'on peut trouver dans la modification de poids au sein d'un réseau en fonction de sa sortie et de la fonction d'erreur.

Si l'on applique ce principe à un système ayant une dimension récurrente ayant pour paramètres à partager θ, on peut considérer :

$$\boldsymbol{s}^{(t)} = f(\boldsymbol{s}^{(t-1)}; \theta)$$

avec $\boldsymbol{s}^{(t)}$ l'état courant du système. Ici, la récurrence est exprimée par la définition de $\boldsymbol{s}^{(t)}$ en fonction de $\boldsymbol{s}^{(t-1)}$.

Pour un nombre fini de timesteps n, on peut étendre cette formulation à une expression n'incluant plus de récurrence :

$$\boldsymbol{s}^{(n)} = \underbrace{f(...f(f(\boldsymbol{s}^{(1)}; \theta); \theta); \theta...)}_{\text{effectué n fois}}$$

On peut donc représenter cette expression sous forme d'un graphe de calcul acyclique classique.

Le cas d'un réseau de neurones récurrent est basé sur le même principe, excepté que l'on inclut un signal $\boldsymbol{x}^{(t)}$ extérieur au système (on aura tendance à utiliser la variable h pour représenter l'état du système afin d'indiquer que celui-ci concerne une couche cachée) :

$$\boldsymbol{h}^{(t)} = f(\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}; \theta)$$

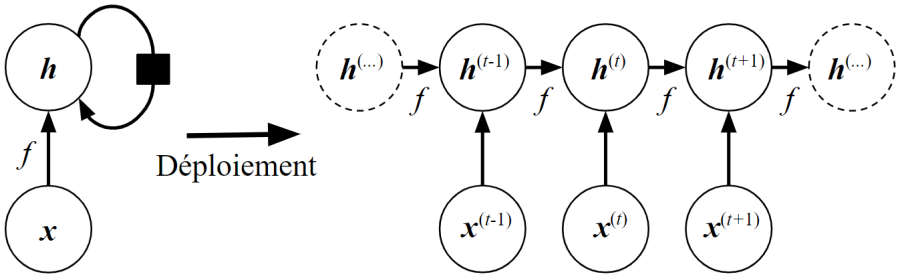

FIGURE 3.8 – Représentation d'un réseau récurrent (les sorties sont pour l'instant ignorées). Ce réseau traite les données $\boldsymbol{x}$ en l'incluant dans l'état $\boldsymbol{h}$ et en partageant celui-ci à chaque itération. A gauche, on trouve le diagramme standardisé (le bloc noir indique un délai d'une timestep). A droite, on trouve l'équivalent du même réseau déployé, où chaque étape est maintenant associée à une timestep. Figure tirée de [Goodfellow et al., 2016c].

comme illustré par la figure 3.8.

Quand le réseau récurrent est entraîné afin d'accomplir une tâche de prédiction du futur en fonction du passé, le réseau aura typiquement tendance à inclure dans l'état $\boldsymbol{h}^{(t)}$ un résumé des informations pertinentes de la séquence passée jusqu'à t.

On parle ici de résumé car convertir une séquence ($\boldsymbol{x}^{(t)}$, $\boldsymbol{x}^{(t-1)}$, ..., $\boldsymbol{x}^{(2)}$, $\boldsymbol{x}^{(1)}$) en un vecteur de taille fixe $\boldsymbol{h}^{(t)}$ implique, dans la grande majorité des cas, une perte d'informations (ce principe n'est d'ailleurs pas sans rappeler le principe des auto-encodeurs).

La quantité d'informations à résumer ainsi que la taille de $\boldsymbol{h}^{(t)}$ dépend du critère d'entraînement, et donc de la tâche à réaliser par le réseau.

3.2.2 Les différents types de réseaux de neurones récurrents

En se basant sur l'idée de déploiement de graphes et de partage de paramètres présentée dans la partie précédente, il est possible de concevoir divers types de réseaux de neurones récurrents.

Les types de RNNs les plus importants incluent les suivants [Goodfellow et al., 2016c] :

- Les réseaux récurrents produisant une valeur de sortie pour chaque timestep et ayant des connexions entre chaque unité cachée.

- Les réseaux récurrents produisant une valeur de sortie pour chaque timestep et ayant des connexions entre les sorties des unités cachées de la timestep précédente et les unités cachées courantes.

- Les réseaux récurrents produisant une valeur de sortie unique après avoir traité toute une séquence et ayant des connexions entre chaque unité cachée.

Les RNNs séquence à séquence

Le réseau récurrent illustré par la figure 3.9 dont il est question dans la formule 3.2 est un des réseaux de neurones le plus utilisé du domaine. Il permet d'approximer diverses fonctions et de répondre à un certain nombre de tâches, telles que la prédiction de séries temporelles, la prédiction de séries discrètes, la traduction automatique...

Nous développons la formule de propagation vers l'avant d'un RNN appartenant à cette famille. La figure 3.9 ne spécifie pas de choix de fonction d'activation pour les unités de sortie, ni de fonction de perte.

Nous définissons de manière générique, pour chaque timestep allant de $t = 1$ à $t = \tau$ et un état initial $\boldsymbol{h}^{(0)}$:

$$\boldsymbol{a}^{(t)} = \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)} \tag{3.2}$$

$$\boldsymbol{h}^{(t)} = \text{activation}(\boldsymbol{a}^{(t)}) \tag{3.3}$$

$$\boldsymbol{o}^{(t)} = \boldsymbol{o} + \boldsymbol{V}\boldsymbol{h}^{(t)} \tag{3.4}$$

où nous incluons néanmoins les vecteurs de biais $\boldsymbol{b}$ et $\boldsymbol{c}$ ainsi que les matrices de poids $\boldsymbol{U}$, $\boldsymbol{V}$ et $\boldsymbol{W}$ décrites dans la figure 3.9. Bien que nous ne l'ayons pas précisé ici, la fonction traditionnelle utilisée dans ce cadre est la fonction tangente hyperbolique (tanh, cf : partie 2.4.6) [Chung et al., 2014].

Afin de développer $\boldsymbol{L}$, nous prenons l'exemple d'un RNN prédisant une séquence de taille égale à celle en entrée et contenant des données continues.

Une manière très répandue de calculer $\boldsymbol{L}$ dans un tel cas serait de calculer la somme des distances entre $\boldsymbol{o}$ et $\boldsymbol{y}$ pour chaque timestep.

Ainsi, la perte totale pourrait s'exprimer par

$$\begin{aligned}\boldsymbol{L}\Big(\{\boldsymbol{o}^{(1)}, ..., \boldsymbol{o}^{(\tau)}\}, \{\boldsymbol{y}^{(1)}, ..., \boldsymbol{y}^{(\tau)}\}\Big) \\ = \sum_t L^{(t)} \\ = \sum_t MSE(\boldsymbol{o}^{(t)}, \boldsymbol{y}^{(t)})\end{aligned}$$

en cas d'utilisation de MSE (cf : partie 2.2.3).

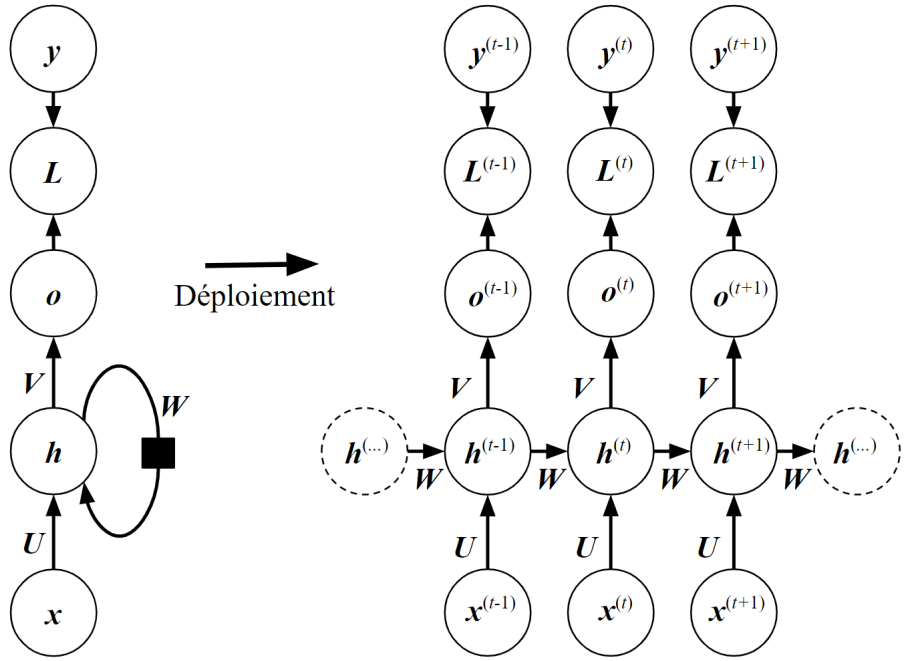

FIGURE 3.9 – Schéma de graphe de traitement standardisé (à gauche) et étendu (à droite) correspondant au calcul de la perte (ou « Loss ») pour une séquence d'entrée $\boldsymbol{x}$ et une séquence de sortie $\boldsymbol{o}$. L'évaluation de la perte $\boldsymbol{L}$ (cf : section 2.2.3) mesure à quel point chaque valeur de $\boldsymbol{o}$ est éloignée de chaque sortie attendue appartenant à $\boldsymbol{y}$. La matrice $\boldsymbol{U}$ correspond à la matrice des poids entre l'entrée et la première couche cachée. Les connexions récurrentes entre unités cachées sont paramétrées par la matrice de poids $\boldsymbol{W}$. Enfin, la matrice de poids $\boldsymbol{V}$ paramètre les connexions entre la couche cachée et la couche de sortie. La formule 3.2 définit la propagation vers l'avant de ce modèle. Cette figure est inspirée de [Goodfellow et al., 2016c].

Bien que les précédentes formules correspondent à la famille des RNNs illustrée par la figure 3.9, elles correspondent aussi fortement à d'autres familles. Par exemple, elles correspondent toutes à la famille de RNNs (proche mais moins puissante) illustrée par la figure 3.10 excepté pour la formule 3.2 qui est remplacée par :

$$\boldsymbol{a}^{(t)} = \boldsymbol{b} + \boldsymbol{W}\boldsymbol{o}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)} \tag{3.5}$$

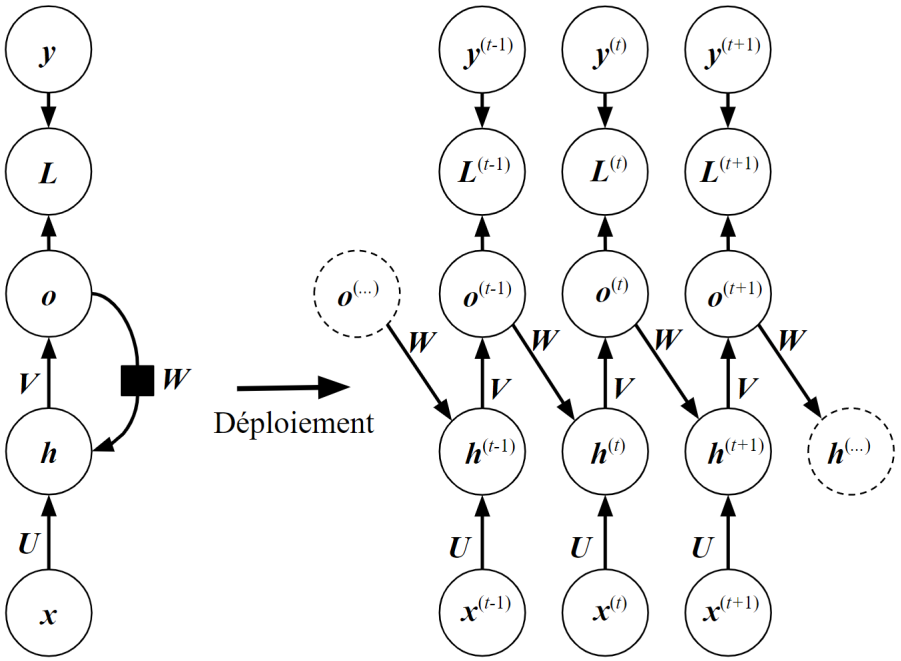

FIGURE 3.10 – Schéma de réseau récurrent où la seule récurrence réside dans le feedback de la sortie d'une unité cachée correspondant à une timestep à la suivante. A chaque timestep t, l'entrée est $\boldsymbol{x}^{(t)}$, la sortie est $\boldsymbol{o}^{(t)}$, la valeur cible est $\boldsymbol{y}^{(t)}$, et la perte est $\boldsymbol{L}^{(t)}$. Ainsi, la propagation vers l'avant de l'information s'effectue telle que définie par la formule 3.5. Un tel réseau peut être moins efficace [Goodfellow et al., 2016c] que ceux appartenant à la famille illustrée par la figure 3.9 car il n'est pas capable d'approximer autant de fonctions que ce dernier. En effet, un RNN appartenant à la famille de celui illustré par la figure 3.9 est libre de choisir les informations les plus pertinentes à incorporer dans $\boldsymbol{h}$ et de transmettre $\boldsymbol{h}$ lors du passage à la prochaine timestep. Le RNN illustré ici ne transmet pas d'informations pratiques au traitement de la prochaine timestep mais uniquement la sortie courante. Ainsi, à moins que la sortie $\boldsymbol{o}$ elle-même ne comporte énormément d'informations, il y aura un manque d'information du passé par rapport à une transmission de $\boldsymbol{h}$, celui-ci n'étant présent qu'indirectement. Néanmoins, bien qu'étant moins puissant, un tel réseau est plus facile à entraîner car chaque timestep peut être traitée indépendamment, augmentant ainsi le taux de parallélisation possible pendant l'entraînement. Cette figure est inspirée de [Goodfellow et al., 2016c].

Les RNNs séquence à sortie unique

De la même manière, les RNNs à sortie unique sont très proches de la première famille (cf : figure 3.11), excepté qu'ils ne produisent qu'un résultat. Ainsi, le calcul de la sortie est le même que précédemment, mais ne s'applique qu'à la timestep finale τ :

$$\boldsymbol{o}^{(\tau)} = \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(\tau)} \tag{3.6}$$

On peut ajouter que la fonction de perte $\boldsymbol{L}$ correspond donc à la simple mesure de la distance entre la sortie unique $\boldsymbol{o}^{(\tau)}$ et la sortie attendue $\boldsymbol{y}^{(\tau)}$.

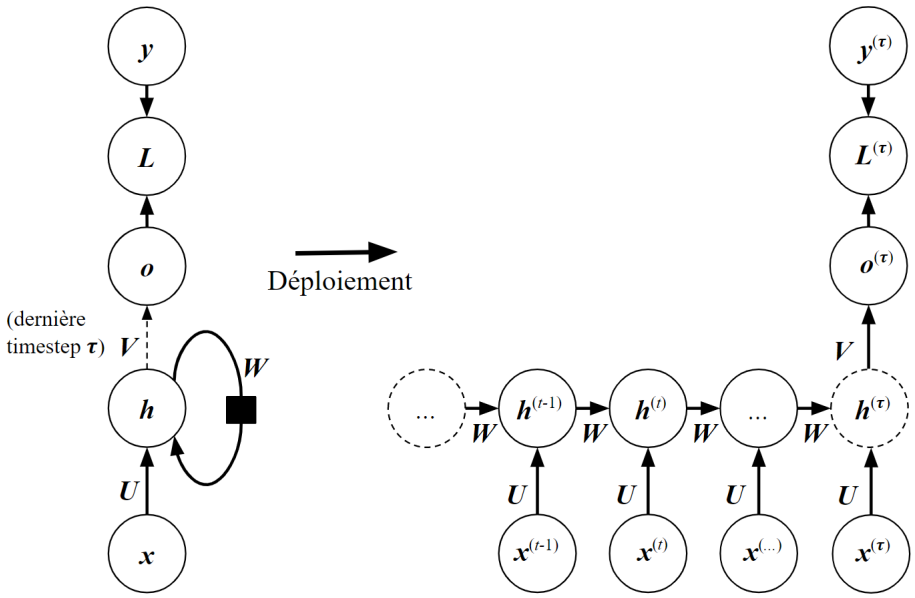

FIGURE 3.11 – Schéma d'un réseau de neurones récurrent ne produisant qu'une seule sortie à la fin du traitement de la séquence. Un tel réseau peut être utilisé pour résumer une séquence et pour produire une sortie de taille fixe pouvant être utilisée comme entrée lors de traitements plus avancés. Ainsi, la propagation s'effectue de la même manière que pour la figure 3.9 par la formule 3.2 mais avec une sortie finale définie par la formule de sortie différente 3.6. Cette figure est inspirée de [Goodfellow et al., 2016c].

Les types de RNNs présentés précédemment ne sont évidemment pas les seuls existants et représentent des catégories globales. De plus, la souplesse des réseaux de neurones permet l'utilisation de réseaux hybrides, mêlant différents types, voir différentes architectures.

Ainsi, il est possible de réaliser des réseaux récurrents utilisant des notions propres aux réseaux à convolution (notamment dans le domaine du traitement de vidéos), ou encore des auto-encodeurs récurrents, comme nous allons le voir dans la partie 3.2.5.

3.2.3 Calculer le gradient dans un RNN

L'algorithme de rétro-propagation utilisé lors de l'entraînement d'un RNN est le même que dans le cas d'un réseau de neurones standard (cf : partie 2.4.5), aucun algorithme spécifique n'est requis. En effet, il suffit d'appliquer l'algorithme de rétro-propagation au graphe de traitement déployé (ce procédé est appelé back-propagation through time, BPTT) [Werbos et al., 1990].

Nous montrons ici un exemple de calcul de gradient appliqué a un RNN (correspondant à la formule 3.2 et à la figure 3.9) afin d'illustrer le fonctionnement de BPTT.

Les noeuds de notre graphe de traitement incluent les paramètres $\boldsymbol{U}$, $\boldsymbol{V}$, $\boldsymbol{W}$, $\boldsymbol{b}$ et $\boldsymbol{c}$ ainsi que les séquences de noeuds par t dénotés $\boldsymbol{x}^{(t)}$, $\boldsymbol{h}^{(t)}$, $\boldsymbol{o}^{(t)}$ et $L^{(t)}$. Pour chaque noeud N, nous devons

calculer le gradient $\Delta_N L$ récursivement, basé sur le gradient du noeud suivant dans le graphe. On commence donc par la récursivité avec le gradient précédent la perte L finale :

$$\frac{\partial L}{\partial L^{(t)}} = 1.$$

Ainsi, le gradient $\Delta_{\boldsymbol{o}^{(t)}} L$ des noeuds de sortie pour la timestep t et pour chaque unité i s'exprime comme suit :

$$(\Delta_{\boldsymbol{o}^{(t)}} L)_i = \frac{\partial L}{\partial o_i^{(t)}} = \frac{\partial L}{\partial L^{(t)}} \frac{\partial L^{(t)}}{\partial o_i^{(t)}} = (\hat{y}_i^{(t)} - y_i^{(t)})^2$$

avec $\hat{\boldsymbol{y}}$ séquence de vecteurs de sorties continues et L la distance entre $\hat{\boldsymbol{y}}$ et $\boldsymbol{y}$ calculée par MSE.

Nous exprimons maintenant les gradients des autres noeuds en partant de la fin de la séquence. A la timestep finale τ, $\boldsymbol{h}^{(\tau)}$ n'a que $\boldsymbol{o}^{(\tau)}$ comme descendant dans le graphe, son gradient est donc simplement :

$$\Delta_{\boldsymbol{h}^{(\tau)}} L = \boldsymbol{V}^\intercal \Delta_{\boldsymbol{o}^{(\tau)}} L. \tag{3.7}$$

$\boldsymbol{h}^{(t)}$ a pour descendant $\boldsymbol{o}^{(t)}$ et $\boldsymbol{h}^{(t+1)}$. Nous pouvons donc itérer vers l'arrière pour propager le gradient à travers la séquence, de $t = \tau - 1$ jusqu'à $t = 1$ à partir de la formule 3.7 pour exprimer le gradient de $\boldsymbol{h}^{(t)}$:

$$\Delta_{\boldsymbol{h}^{(t)}} L = \Big(\frac{\partial \boldsymbol{h}^{(t+1)}}{\partial \boldsymbol{h}^{(t)}}\Big)^\intercal (\Delta_{\boldsymbol{h}^{(t+1)}} L) + \Big(\frac{\partial \boldsymbol{o}^{(t)}}{\partial \boldsymbol{h}^{(t)}}\Big)^\intercal (\Delta_{\boldsymbol{o}^{(t)}} L)$$

pouvant être simplifié en :

$$\Delta_{\boldsymbol{h}^{(t)}} L = \boldsymbol{W}^\intercal (\Delta_{\boldsymbol{h}^{(t+1)}} L) \operatorname{diag}\Big(1 - (\boldsymbol{h}^{(t+1)})^2\Big) + \boldsymbol{V}^\intercal (\Delta_{\boldsymbol{o}^{(t)}} L)$$

où $\operatorname{diag}\Big(1 - (\boldsymbol{h}^{(t+1)})^2\Big)$ correspond à la matrice diagonale contenant les éléments $1 - (\boldsymbol{h}^{(t+1)})^2$.

Une fois les gradients des unités cachées obtenus, nous pouvons obtenir les gradients de chacun des paramètres $\boldsymbol{U}$, $\boldsymbol{V}$, $\boldsymbol{W}$, $\boldsymbol{b}$ et $\boldsymbol{c}$ précédemment présentés.

$$\Delta_{\boldsymbol{c}} L = \sum_t \Big(\frac{\partial \boldsymbol{o}^{(t)}}{\partial \boldsymbol{c}}\Big)^\intercal \Delta_{\boldsymbol{o}^{(t)}} L = \sum_t \Delta_{\boldsymbol{o}^{(t)}} L \tag{3.8}$$

$$\Delta_{\boldsymbol{b}} L = \sum_t \Big(\frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{b}^{(t)}}\Big)^\intercal \Delta_{\boldsymbol{h}^{(t)}} L = \sum_t \operatorname{diag}\Big(1 - (\boldsymbol{h}^{(t)})^2\Big) \Delta_{\boldsymbol{h}^{(t)}} L \tag{3.9}$$

$$\Delta_{\boldsymbol{V}} L = \sum_t \sum_i \Big(\frac{\partial L}{\partial o_i^{(t)}}\Big)^\intercal \Delta_{\boldsymbol{V} o_i^{(t)}} = \sum_t (\Delta_{\boldsymbol{o}^{(t)}} L) \boldsymbol{h}^{(t)\intercal} \tag{3.10}$$

$$\Delta_{\boldsymbol{W}} L = \sum_t \sum_i \Big(\frac{\partial L}{\partial h_i^{(t)}}\Big) \Delta_{\boldsymbol{W}^{(t)}} h_i^{(t)} = \sum_t \text{diag}\Big(1 - (\boldsymbol{h}^{(t)})^2\Big)(\Delta_{\boldsymbol{h}^{(t)}} L)\boldsymbol{h}^{(t-1)\intercal} \tag{3.11}$$

$$\Delta_{\boldsymbol{U}} L = \sum_t \sum_i \Big(\frac{\partial L}{\partial h_i^{(t)}}\Big) \Delta_{\boldsymbol{U}^{(t)}} h_i^{(t)} = \sum_t \text{diag}\Big(1 - (\boldsymbol{h}^{(t)})^2\Big)(\Delta_{\boldsymbol{h}^{(t)}} L)\boldsymbol{x}^{(t-1)\intercal} \tag{3.12}$$

Il est évident qu'un RNN à sortie unique est plus simple à traiter en termes de calcul de gradient. La formule de calcul de gradient des couches cachées 3.2.3 est valable pour la première itération de BPTT, mais ensuite le terme $\boldsymbol{V}^\intercal(\Delta_{\boldsymbol{o}^{(t)}} L)$ disparaît, on aurait donc :

$$\Delta_{\boldsymbol{h}^{(t)}} L = \boldsymbol{W}^\intercal(\Delta_{\boldsymbol{h}^{(t+1)}} L)\ \text{diag}\Big(1 - (\boldsymbol{h}^{(t+1)})^2\Big).$$

De la même manière, les formules de calcul de gradient pour tous les paramètres (formules 3.8 à 3.12) s'appliquent pour la première itération, puis les termes incluant $\boldsymbol{V}$ ou $\boldsymbol{o}$ disparaissent pour la suite de l'algorithme (la formule calculant $\Delta_{\boldsymbol{V}} L$ n'est utilisée qu'une seule fois dans ce cas).

3.2.4 Optimisations

Nous développons dans cette section les optimisations qui ont suivi le développement des RNNs et dont nous avons pu bénéficier lors de nos expérimentations.

Les réseaux de neurones récurrents (et les grands réseaux de neurones en général) font généralement face à diverses difficultés.

En effet, le fait de répéter des opérations utilisant les mêmes paramètres sont source de problèmes bien spécifiques aux réseaux de neurones utilisant quantité de couches cachées, ou des opérations de récurrences, en particulier lorsque la séquence à traiter est longue.

Par exemple, on suppose que l'on ait un graphe de traitements contenant un chemin où un argument doit être multiplié plusieurs fois par une matrice $\boldsymbol{W}$. Après t étapes, on a alors multiplié l'argument par l'équivalent de $\boldsymbol{W}^t$.

On suppose que que $\boldsymbol{W}$ a une décomposition en éléments propres $\boldsymbol{W} = \boldsymbol{V}\ \text{diag}(\boldsymbol{\lambda})\boldsymbol{V}^{-1}$.

Dans notre cas, on voit donc que :

$$\boldsymbol{W}^t = (\boldsymbol{V}\ \text{diag}(\boldsymbol{\lambda})\boldsymbol{V}^{-1})^t = \boldsymbol{V}\ \text{diag}(\boldsymbol{\lambda})^t\boldsymbol{V}^{-1}. \tag{3.13}$$

En cas de t élevé, on constate que les valeurs λ_i exploseront si l'ordre de grandeur de leur valeur absolue est plus élevée que 1 ou tendront vers 0 si l'ordre de grandeur est plus bas que 1.

Ainsi, les problèmes d'explosion du gradient ou de disparition du gradient sont liés au fait que le calcul de gradient à travers les graphes de traitements évoqués dépend aussi de $\text{diag}(\boldsymbol{\lambda})^t$.

Au sein des RNNs, même en cas de stabilité ($\boldsymbol{h}$ stocke bien les paramètres en mémoire et le gradient n'explose pas), la difficulté, inhérente aux valeurs de plus en plus petites des poids, augmente exponentiellement plus les intéractions se font sur un long terme au sein de la séquence à traiter.

En effet, si on prend une relation de récurrence simple :

$$\boldsymbol{h}^t = \boldsymbol{W}^\intercal \boldsymbol{h}^{(t-1)}$$

(on ignore ici les entrées $\boldsymbol{x}$ ainsi que les autres paramètres évoqués dans la partie précédente), on peut la reformuler de la façon suivante :

$$\boldsymbol{h}^t = (\boldsymbol{W}^t)^\intercal \boldsymbol{h}^{(0)}.$$

Si l'on admet que $\boldsymbol{W}$ peut être décomposé en éléments simples $\boldsymbol{W} = \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^\intercal$ (avec $\boldsymbol{Q}$ orthogonale), on peut simplifier la récurrence par :

$$\boldsymbol{h}^t = \boldsymbol{Q}^\intercal \boldsymbol{\Lambda}^t \boldsymbol{Q} \boldsymbol{h}^{(0)}.$$

De la même manière que pour la formule 3.13, les valeurs de $\boldsymbol{\Lambda}$ exploseront ou tendront vers 0. Ainsi, les éléments de $\boldsymbol{h}^{(0)}$ qui ne sont pas alignés avec les plus grandes valeurs disparaîtront.

Un tel problème a motivé de nombreux travaux dans le domaine. A l'heure actuelle, les modèles de traitement de séquences les plus efficaces et les plus utilisés sont les réseaux de neurones récurrents à portes (« gated RNNs »).

LSTM et autres RNNs à portes

Les RNNs à portes ont pour but de créer des séquences de traitements où les dérivées ne vont ni disparaître ni exploser.

Ils sont basés sur l'idée qu'une fois utilisée, une information peut être oubliée par le réseau. Ainsi, les RNNs à portes implémentent un algorithme qui leur permet de décider lorsqu'il est utile d'oublier une information.

Permettre à une cellule de créer des chemins où le gradient peut évoluer normalement via une récurrence a été la contribution de base des modèles long short-term memory (LSTM) [Hochreiter and Schmidhuber, 1997].

Une évolution majeure a été ensuite de rendre le poids sur ce lien récurrent dépendant du contexte, plutôt que fixe [Gers et al., 2000].

Depuis, le modèle LSTM a eu beaucoup de succès dans de nombreux domaines d'application.

Une cellule LSTM ne se limite néanmoins pas à un lien récurrent. En effet, celle-ci bénéficie d'un système de portes contrôlant le passage de l'information. Le composant le plus important est l'état $s_i^{(t)}$ de l'unité, sur lequel est positionné le lien récurrent.

Le poids de ce lien est contrôlé par une porte d'oubli $f_i^{(t)}$ (forget gate). Celle-ci positionne le poids entre 0 et 1 via une fonction sigmoïde comme suit :

$$f_i^{(t)} = \sigma\Big(b_i^f + \sum_j U_{i,j}^f x_j^{(t)} + \sum_j W_{i,j}^f h_j^{(t-1)}\Big)$$

avec $\boldsymbol{x}^{(t)}$ l'entrée courante et $\boldsymbol{h}^{(t)}$ l'état courant contenant les sorties de toutes les portes, et $\boldsymbol{b}^f$, $\boldsymbol{U}^f$, $\boldsymbol{W}^f$ respectivement les biais, les poids en entrée et les poids des liens récurrents pour les forget gates.

Ainsi, l'état interne $s_i^{(t)}$ de l'unité est mis à jour par :

$$s_i^{(t)} = f_i^{(t)} s_i^{(t-1)} + g_i^{(t)} \sigma\Big(b_i + \sum_j U_{i,j} x_j^{(t)} + \sum_j W_{i,j} h_j^{(t-1)}\Big)$$

avec $\boldsymbol{b}$, $\boldsymbol{U}$, $\boldsymbol{W}$ respectivement les biais, les poids de l'entrée et les poids récurrents de la cellule LSTM. La porte d'entrée externe $g_i^{(t)}$ (external input gate) fonctionne de la même manière que la forget gate, avec ses propres paramètres :

$$g_i^{(t)} = \sigma\Big(b_i^g + \sum_j U_{i,j}^g x_j^{(t)} + \sum_j W_{i,j}^g h_j^{(t-1)}\Big)$$

La sortie $h_i^{(t)}$ de la cellule LSTM peut aussi être désactivée par la porte consacrée à la sortie $q_i^{(t)}$ (output gate), qui utilise elle aussi, une fonction sigmoïde. Ainsi, la sortie $h_i^{(t)}$ s'exprime en fonction de $q_i^{(t)}$ comme suit :

$$h_i^{(t)} = \text{activation}(s_i^{(t)}) q_i^{(t)}$$

avec

$$q_i^{(t)} = \sigma\Big(b_i^q + \sum_j U_{i,j}^q x_j^{(t)} + \sum_j W_{i,j}^q h_j^{(t-1)}\Big).$$

Le diagramme d'une cellule LSTM est détaillé dans la figure 3.12.

Des variantes de ce modèle existent. Par exemple, il est possible d'utiliser $s_i^{(t)}$ comme entrée supplémentaire des trois portes pour l'unité i.

Dans notre cas, nous nous sommes contentés de l'approche standard détaillée plus haut.

Les réseaux LSTM se sont montrés capables d'apprendre des dépendances temporelles à long terme plus facilement que les architectures classiques.

Il est à noter que d'autres cellules à portes existent. Les cellules GRUs (Gated Recurrent Units) [Cho et al., 2014] proposent une version épurée des cellules LSTM, où une seule unité interne contrôle le facteur d'oubli et le facteur de mise à jour de l'état de la cellule.

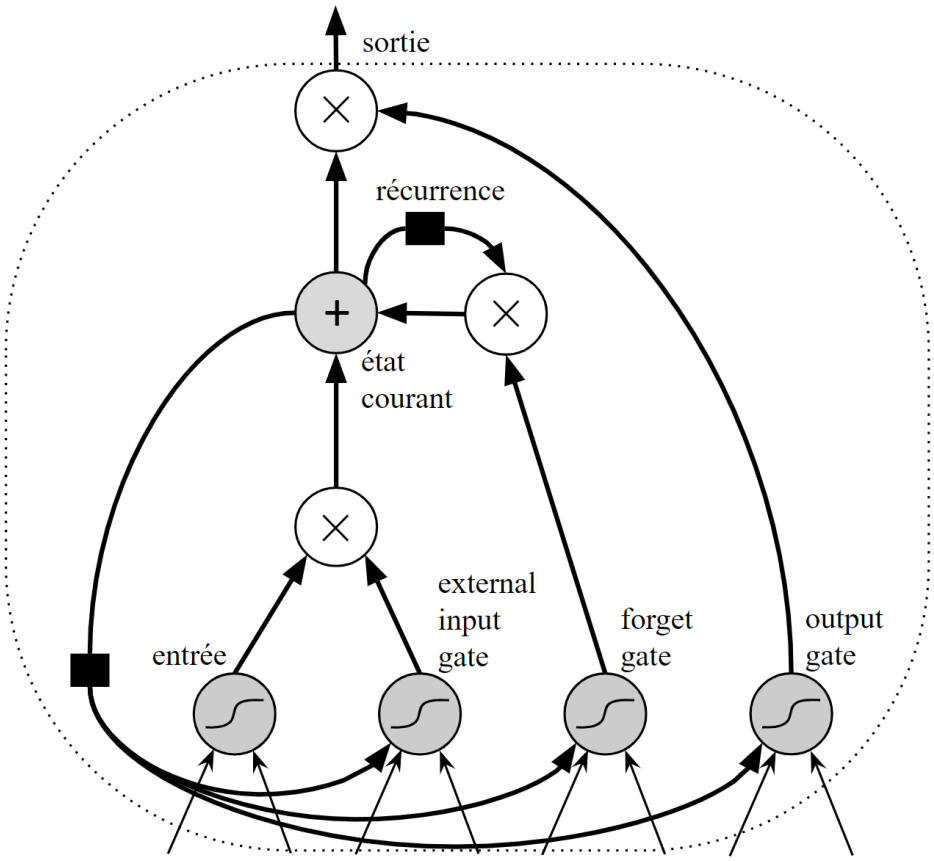

FIGURE 3.12 – Diagramme d'une cellule récurrente LSTM. Les cellules sont connectées de manière récurrente (un bloc noir représente encore une fois un délai d'une timestep lors du traitement de la séquence), remplaçant les unités cachées standards. Une entrée est d'abord traitée de la même manière que dans une cellule standard. La valeur sera ensuite stockée dans l'état courant si l'input gate le permet. L'unité stockant l'état courant est contrôlée par la forget gate, qui lui permettra ou non de se transmettre à la timestep suivante. Enfin, la sortie peut être désactivée ou non par l'output gate. Toutes les unités à portes contrôlent un aspect de la propagation de l'information de manière non-linéaire grâce à l'utilisation de la fonction sigmoïde.

La formule de la mise à jour est la suivante :

$$h_i^{(t)} = u_i^{(t-1)} h_i^{(t-1)} + (1 - u_i^{(t-1)}) \sigma\Big(b_i + \sum_j U_{i,j} x_j^{(t-1)} + \sum_j W_{i,j} r_j^{(t-1)} h_j^{(t-1)}\Big)$$

où $\boldsymbol{u}$ correspond à la porte de mise à jour (update gate) et $\boldsymbol{r}$ correspond à la porte de réinitialisation (reset gate).

Leur valeur est définie de manière standard par une sigmoïde :

$$u_i^{(t)} = \sigma\Big(b_i^u + \sum_j U_{i,j}^u x_j^{(t)} + \sum_j W_{i,j}^u h_j^{(t-1)}\Big)$$

et

$$r_i^{(t)} = \sigma\Big(b_i^r + \sum_j U_{i,j}^r x_j^{(t)} + \sum_j W_{i,j}^r h_j^{(t-1)}\Big).$$

LSTM et GRU sont les deux implémentations de modèles à portes les plus notables, bien qu'il en existe d'autres.

Chung et al. [Chung et al., 2014] ont évalué les performances de réseaux récurrents classiques, LSTM et GRU. Les réseaux à portes se sont révélés plus efficaces que les réseaux récurrents classiques.

Quand aux performances de GRU face à LSTM, elles dépendent du cas d'utilisation et demandent généralement un essai des deux types d'architectures afin d'effectuer une sélection.

RNNs bidirectionnels

Tous les RNNs que nous avons considéré jusque maintenant suivent un aspect « causal » du traitement de la séquence. On entend par là qu'à une timestep t, l'état $\boldsymbol{h}$ n'a engrangé que des informations du passé, présentes dans l'entrée courante $\boldsymbol{x}^{(1)}, ..., \boldsymbol{x}^{(t-1)}$.

Cependant, dans de nombreux cas d'utilisations, nous souhaitons une prédiction $\hat{\boldsymbol{y}}^{(t)}$ prenant en compte l'entièreté de la séquence. C'est par exemple le cas dans la reconnaissance de discours (on entend par là le traitement d'une séquence discrète de mots et non pas la traduction orale directe).

En effet, le bon traitement de phrases dépend de nombreux facteurs. De nombreuses phrases formulées différemment peuvent avoir rigoureusement le même sens, un mot peut avoir des significations complètement différentes selon le contexte, etc.

Examiner toute la séquence en partant à la fois du commencement vers la fin et de la fin vers le commencement permettrait de dégager des informations contextuelles pouvant orienter vers une meilleure compréhension d'une phrase ou permettant de lever une éventuelle ambiguïté sur le sens d'un mot.

Les réseaux de neurones récurrents bidirectionnels ont été inventés pour adresser ce besoin [Schuster and Paliwal, 1997]. Ils se sont révélés très efficaces et ont eu un grand succès dans de nombreux domaines [Baldi et al., 1999] [Graves and Schmidhuber, 2005] [Graves et al., 2008].

Comme le nom l'indique, les RNNs bidirectionnels combinent un RNN classique, traitant une séquence du passé vers la fin, et un RNN traitant la séquence du futur vers le début (cf : schéma figure 3.13).

Le principe de cette architecture, comme la majorité des architectures de réseaux de neurones, est associable avec d'autres. On peut donc imaginer des RNNs bidirectionnels à convolution, ou encore des auto-encodeurs récurrents bidirectionnels. Nous nous sommes orientés, dans nos expérimentations, vers l'utilisation d'auto-encodeurs récurrents LSTM bidirectionnels.

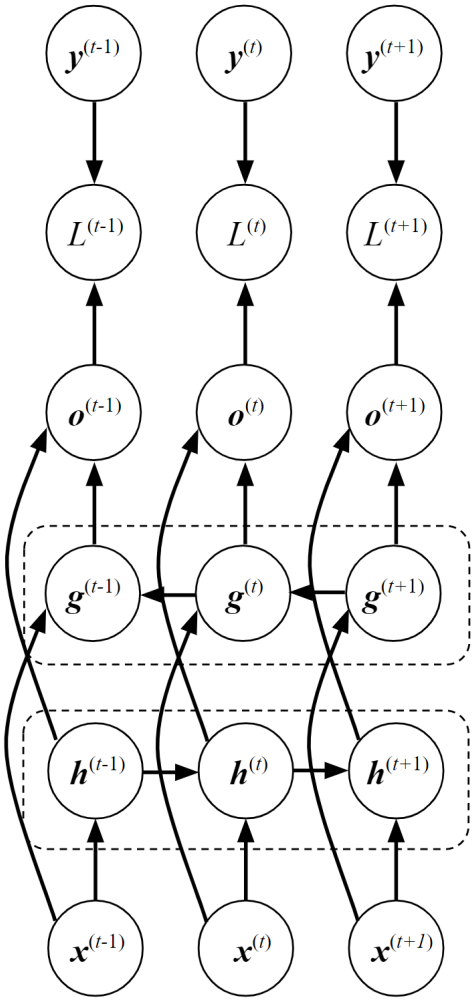

FIGURE 3.13 – Schéma de traitement d'un RNN bidirectionnel typique. La récurrence effectuée via $\boldsymbol{h}$ s'effectue du passé vers le futur et inversement pour la récurrence effectuée via $\boldsymbol{g}$. Ainsi, les noeuds de sortie $\boldsymbol{o}$ bénéficient à tout moment des résumés des informations venant du passé via son entrée $\boldsymbol{h}^{(t)}$ ou du futur grâce à son entrée $\boldsymbol{g}^{(t)}$. Cette figure est inspirée de [Goodfellow et al., 2016c].

3.2.5 Auto-encodeurs récurrents

Nous avons vu dans la partie 3.2.2 que les RNNs pouvaient produire une sortie unique pour une séquence passée en entrée (cf : figure 3.11).

Il est aussi possible pour un RNN de produire une séquence pour une entrée unique. Diverses méthodes existent afin de créer un tel RNN, et une des méthodes les plus répandues consiste à répéter l'entrée t pour une séquence à produire de taille t.

Ainsi, il est possible de créer un auto-encodeur récurrent constitué d'une partie encodeur basée sur un RNN « séquence vers sortie unique » (cf : partie 3.2.2) et d'une partie décodeur fonctionnant comme décrit précédemment.

Une telle architecture peut être utilisée dans de nombreux domaines, et notamment dans le domaine de la traduction de texte dans une autre langue. L'avantage d'une telle architecture est notamment sa capacité à produire une séquence en sortie ayant une taille potentiellement différente de celle de la séquence d'entrée.

Dans notre cas, nous l'utilisons dans le contexte de la détection d'anomalies. La séquence à produire est la même que celle passée en entrée.

Ainsi, on a :

- Une séquence d'entrée $\boldsymbol{x}$ composée de vecteurs de n caractéristiques fois t timesteps.
- Un RNN encodeur, transformant la séquence d'entrée en un vecteur latent $\boldsymbol{c}$ de taille fixe, usuellement bien inférieure à $n * t$. Une réduction de dimension a donc lieu, de la même manière que dans un auto-encodeur classique (cf : partie 2.5).
- Une répétition de t fois du vecteur latent $\boldsymbol{c}$, envoyé en entrée du RNN décodeur.
- Une approximation de la séquence de base $\hat{\boldsymbol{x}}$ par le RNN décodeur.

Le fonctionnement d'auto-encodeur récurrent utilisé pour la détection d'anomalies est illustré par la figure 3.14.

On peut donc traiter la sortie produite de la même manière qu'avec un auto-encodeur classique et évaluer la distance avec l'entrée, c'est à dire la sortie attendue.

Une distance anormalement élevée signifiera donc la présence d'anomalie au sein de la séquence traitée.

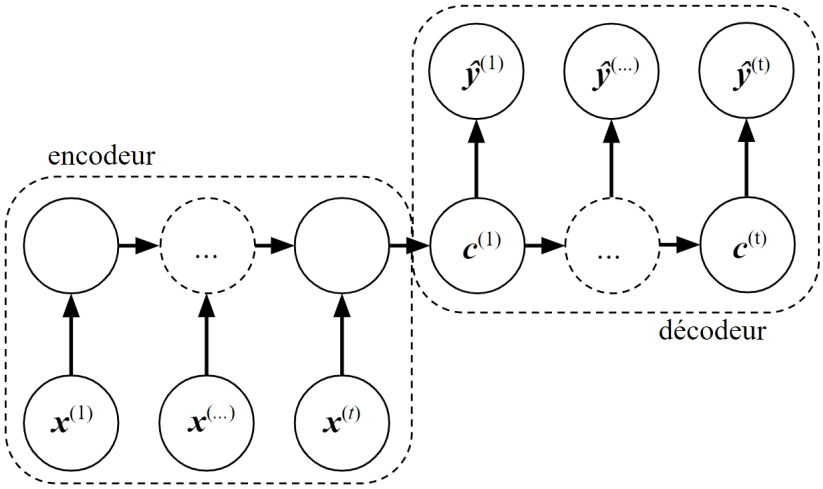

FIGURE 3.14 – Illustration du type d'auto-encodeur récurrent que nous avons utilisé pour les expérimentations présentées dans ce chapitre. Dans le cadre de la détection d'anomalies, la sortie $\hat{\boldsymbol{y}}$ correspond en fait à l'approximation de la séquence d'entrée $\hat{\boldsymbol{x}}$ de même taille t. Le vecteur $\boldsymbol{c}$ correspond à la sortie du RNN encodeur et est répété t fois en entrée du RNN décodeur. Bien que nous l'utilisions dans nos modèles, nous avons négligé ici la bidirection des couches récurrentes. De plus, il est à noter que nous avons utilisé des cellules LSTM, telles que présentées dans la partie 3.2.4.

3.3 Etude comparative

Comme nous l'avons montré, divers travaux, appliquant les deux architectures présentées précédemment à des données issues d'un contexte relativement similaire au notre, existent au sein de la littérature.

Dans cette partie, nous détaillons l'étude comparative des performances de ces deux architectures appliquées à notre cas d'utilisation. De plus, nous présentons un moyen de comparer différents modèles dans un contexte d'anomalies partiellement labélisées.

Les résultats présentés et méthodes proposées ont fait l'objet d'une publication [Legrand et al., 2018].

3.3.1 Données

Jeu de données REFIT

Afin d'effectuer notre expérimentation sur des données réelles, nous avons choisi d'utiliser le jeu de données REFIT (Personalised Retrofit Decision Support Tools for UK Homes Using Smart Home Technology) [Firth et al., 2017]. Le jeu de données REFIT Smart Home comprend des données provenant de divers capteurs placés dans 20 maisons, ainsi qu'une description détaillée desdits capteurs et maisons. Les données ont été recueillies sur une période allant de

2013 à 2015. Néanmoins, tous les capteurs n'ont pas été déployés simultanément et n'ont pas tous émis jusque fin 2015.

Nous avons donc sélectionné, parmi les 20 maisons, la fenêtre temporelle contenant le plus de capteurs émettant simultanément tout en étant suffisamment étendue.

Nous avons formalisé ces contraintes afin d'avoir un jeu de données uniforme, ayant la plus grande dimension possible, et permettant la réalisation d'une tâche d'apprentissage. Conformément à ces contraintes, nous avons sélectionné une période correspondant du 04/11/2014 au 01/03/2015 pour la maison identifiée comme 0.

En effet, celle-ci offre la fenêtre de 4 mois comportant, après tri de ceux-ci, les capteurs actifs les plus équitablement répartis à travers toutes les catégories disponibles :

- La consommation de gaz
- La température de surface des radiateurs
- La température de différentes pièces
- La luminosité dans différentes pièces
- Des informations relatives au thermostat
- ...

Pré-traitements et nettoyage des données

Comme nous venons de le mentionner, nous avons procédé à la sélection des capteurs les plus pertinents. En effet, nous avons mis de côté les capteurs remontant des données discrètes, très liés à un aspect métier qui n'aurait pas eu beaucoup de sens aux yeux de notre modèle d'apprentissage. Par exemple, nous avons privilégié les données telles que la consommation de gaz et la température en surface des radiateurs plutôt que les « modes » de chauffe de ceux-ci. De plus, nous avons aussi écarté les données des capteurs présentant très peu ou pas de variations, les données de capteurs présentant une tendance trop marquée, ou encore les données ne présentant visuellement aucune trace de saisonnalité.

Ainsi, nous avons travaillé avec 19 capteurs de cette maison qui étaient actifs pendant la période sélectionnée.

Ceux-ci ont émis une donnée toutes les 15 minutes. Comme présenté dans la section 2.4.6, les données de chaque capteur ont été centrées-réduites afin de faciliter le processus d'apprentissage.

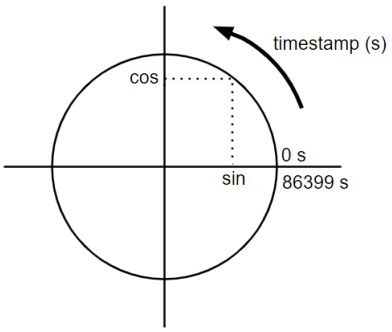

FIGURE 3.15 – Représentation visuelle de la transformation effectuée sur les timestamps de chacun des individus.

Enrichissement des données

Une étude préalable sur les données a été réalisée en établissant, pour chaque capteur, la densité spectrale de puissance. Cette méthode, basée sur la transformée de Fourier, nous a permis d'établir un périodogramme [Press and Rybicki, 1989] pour chaque capteur.

Cette étude a révélé que de nombreux capteurs présentaient de fortes saisonnalités, les données suivant pour la plupart des cycles d'une journée. Ce phénomène n'est pas surprenant. En effet, cette composante saisonnière s'explique facilement en prenant en compte la luminosité de chaque pièce augmentant et descendant au gré du soleil, la température augmentant le jour et baissant la nuit...

Nous avons fait le choix, basé sur cette étude préalable, d'enrichir les données. En effet, nous avons souhaité donner au modèle des informations explicites concernant la situation temporelle des données au sein de leur cycle préalablement établi. De cette manière, on apporte au modèle des informations contextuelles fortes, facilitant ainsi l'apprentissage des propriétés des données.

Chaque capteur émet une donnée toutes les 15 minutes ; bien que les capteurs n'émettent pas simultanément, nous pouvons établir qu'il existe un vecteur contenant toutes les données de chaque capteur pour une plage de 15 minutes donnée. Afin de prendre en compte la dimension cyclique des données, nous avons utilisé le timestamp associé à chacune de ces plages. Nous retirons du timestamp les valeurs du mois et de l'année afin que, pour chaque vecteur et son jour courant, il ne reste que le nombre de seconde écoulé depuis minuit.

Les réseaux de neurones accordent de l'importance à la valeur numérique des entrées. Prendre en compte ce timestamp restant n'est donc pas suffisant. En effet, le dernier timestamp d'une journée est 86399, la seconde suivante aura donc un timestamp de 0. Le contexte est très proche, mais l'écart entre les valeurs est important.

Nous avons donc choisi de traiter la valeur de ce timestamp comme les points d'un cercle, comme illustré par la figure 3.15.

Ainsi, nous pouvons créer une combinaison unique de deux nouvelles caractéristiques pour chaque instant de la journée basée sur le sinus et le cosinus.

$$sinpart = \sin \frac{2 * \pi * timestamp}{24 * 3600}$$

$$cospart = \cos \frac{2 * \pi * timestamp}{24 * 3600}$$

De cette manière, 23h59 :59 et 00h00 correspondront à deux combinaisons sinus/cosinus très proches, caractérisant mieux la similitude contextuelle.

3.3.2 Méthodologie

Une des difficultés dans le domaine de la détection d'anomalie est l'évaluation des performances des modèles. En effet, lorsque des labels (anomalie ou non) sont disponibles, la qualité d'une fonction de score peut être évaluée avec les courbes ROC ou PR. Malheureusement, la plupart du temps, aucun label n'est disponible, comme c'est le cas pour REFIT. Ce problème récurrent au sein du domaine a motivé des travaux visant à déterminer des moyens d'évaluer des modèles de détection d'anomalies non supervisés.

Parmi les méthodes issues de ces travaux, certaines ont obtenu des résultats prometteurs. En effet, les critères basés sur EM (Excess-Mass) et MV (Mass-Volume) [Goix, 2016] sont, dans environ 80% des cas, capables de déterminer quel algorithme est meilleur que l'autre sur des jeux de données ne comprenant pas de label. Malheureusement, ces deux méthodes reposent, entre autres choses, sur une estimation de Monté-Carlo, visant à établir un échantillon représentatif des données en générant des données aléatoirement [Goix et al., 2015].

Il est donc impossible pour nous d'utiliser ces méthodes en l'état. Effectivement, dans notre cas, afin de prendre en compte les anomalies collectives (cf : section 2.1), nous utilisons les données que nous avons à disposition sous forme de fenêtre d'une demie journée. Nous avons donc des entrées ayant une dimension de $19 * n$ (19 capteurs fois n timesteps). Les tailles de fenêtres que nous avons sélectionné sont d'une journée (96 timesteps), d'une demi-journée (48 timesteps) et d'un quart de journée (24 timesteps).

Les auteurs des travaux concernant EM et MV [Goix et al., 2015] recommandent de n'utiliser cette méthode que pour des données ayant une dimension relativement faible (inférieure à 8, d'après [Goix, 2016]). En effet, dans notre cas, si l'on prend en compte le domaine de définition de chaque capteur en plus de la dimension initiale, le nombre de fenêtres possibles devient largement trop grand. Si l'on imagine que nos 19 capteurs peuvent prendre une valeur se situant entre 0 et 20 avec une mesure précise jusqu'à la première décimale, le nombre de fenêtres, et donc de combinaisons de valeurs explose. En effet, on aurait dans ce cas, pour une donnée unique de capteur, 200 valeurs différentes possibles. Une fenêtre de 48 timesteps incluant 19 capteurs est constituée de 912 données. On a donc un espace comportant $(200)^{912}$ combinaisons différentes possibles.

Dans cet ensemble, la plupart des valeurs n'auraient aucun lien les unes avec les autres (par exemple d'une timestep à l'autre). De ce fait, la très grande majorité de ces fenêtres seraient du bruit qui ne serait pas représentatif des entrées possibles issues de la réalité (une fenêtre,

prise au hasard dans les $(200)^{912}$ combinaisons possibles, n'aurait que très peu de chance d'être représentative d'un scénario réel standard).

On pourrait donc chercher à réduire cet espace de taille $(200)^{912}$ en ne gardant que les scénarios plausibles. Néanmoins, le fait de générer des fenêtres simulées représentatives de la réalité et des anomalies que l'on peut rencontrer naturellement constituerait à lui seul un défi qui impliquerait d'avoir une connaissance très étendue du sujet et du bâtiment concerné, ce qui ne serait pas en adéquation avec le but recherché de travailler sur la mise au point d'un système de détection générique. En effet, un de nos enjeux, comme expliqué dans la section 1.2, est de réduire autant que possible le temps à passer sur un bâtiment afin de mettre en place un système détectant les scénarios anormaux pouvant s'y dérouler.

Nous nous sommes donc orientés vers une labélisation manuelle partielle des anomalies et avons déterminé deux moyens possibles de comparer des fonctions de scores.

Labélisation des fenêtres

Les fenêtres que nous avons labélisées comme anomalies sont des fenêtres contenant des anomalies correspondant à des scénarios anormaux dans la vraie vie (comme les deux fenêtres figurant dans la partie 2.1). Nous n'avons pas essayé de labéliser toutes les anomalies car nous avons fait face à un problème inhérent à la détection d'anomalies au sein de séries temporelles multivariées.

Imaginons qu'une anomalie se produise sur une période de plusieurs timesteps. Il n'est pas possible de déterminer au préalable quelle sera sa disposition optimale par rapport à la fenêtre prise en compte par le modèle. Par exemple, il n'est pas possible de savoir sans expérimentation si la prise en compte du « début » de l'anomalie (comme illustré par la figure 3.16) est suffisante ou pas pour produire un haut score. De plus, il n'est pas possible non plus de déterminer si, lors de la prise totale de l'anomalie, la position de celle-ci au sein de la fenêtre aura un grand impact ou non sur le score. Ces difficultés rendent délicate la labélisation manuelle de toutes les anomalies de troisième catégorie au sein d'un dataset.

Nous avons donc adopté l'approche suivante : pour chaque collection issue de chaque fenêtre glissante (journée, demi-journée, quart de journée), nous avons examiné une première fenêtre, puis la fenêtre démarrant là où la précédente s'achève, et ainsi de suite. Nous avons noté toutes celles qui contenaient une anomalie (pic de température, d'humidité…). Ainsi, nous n'avons pas examiné la totalité des fenêtres mais seulement un échantillon représentatif de celles-ci. Nous ignorons donc le problème précédent pour nous concentrer sur un échantillon d'anomalies représentatif. Ainsi, il n'est plus possible d'évaluer efficacement les performances d'un modèle, mais il est toujours possible de comparer des modèles différents traitant des mêmes données et des mêmes échantillons d'anomalies.

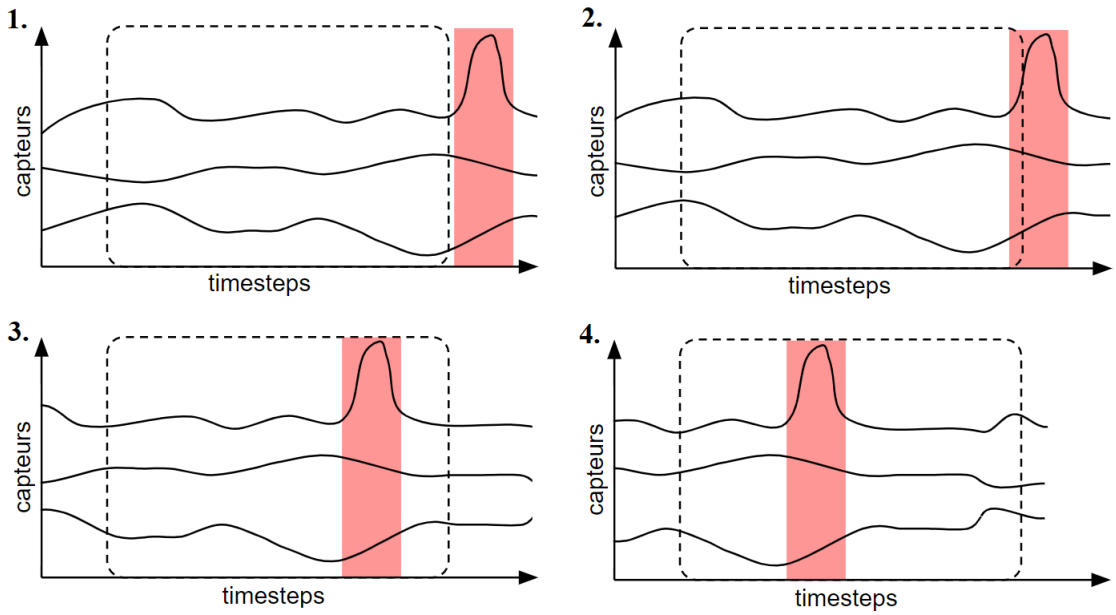

FIGURE 3.16 – Schéma illustrant le problème de labélisation rencontré. La plage contenant l'anomalie (en rouge) arrive progressivement dans la fenêtre examinée en continu par le modèle (en pointillés). Le problème réside dans le fait qu'il n'est pas possible de savoir a priori quelle configuration est optimale pour détecter l'anomalie (3 ou 4), ni même si la prise en compte partielle (2) suffit à détecter l'anomalie.

Méthode de comparaison des modèles

Le fait de disposer d'un ensemble de données (pour chaque collection de fenêtre on a $\sim$ 11500 individus) comportant un échantillon d'anomalies étiquetées représentatif (dans notre cas, 16 anomalies) nous permet d'établir une estimation générale de la performance de nos modèles. Nous avons donc développé deux index qui reflètent les critères visuels généralement utilisés pour évaluer un modèle. Habituellement, dans le cas d'un auto-encodeur correctement entraîné, la distribution de l'erreur entre les données originales et les données recréées a une forme similaire à une distribution normale [Fan et al., 2018] [Goix, 2016].

Les indices (appelés ici «indice de fiabilité» et «indice de fiabilité absolue») que nous avons établis évaluent la capacité d'un modèle à placer des anomalies du côté droit de la distribution des erreurs. Pour évaluer un modèle a, on considère ici un sous-ensemble d'erreurs d'anomalies de taille n (différent de la taille n de fenêtre prise en compte présentée en début de section) représentatif de l'ensemble correspondant à la totalité des erreurs de toutes anomalies. L'ensemble des erreurs, correspondant aux anomalies et aux données saines, est de taille m.

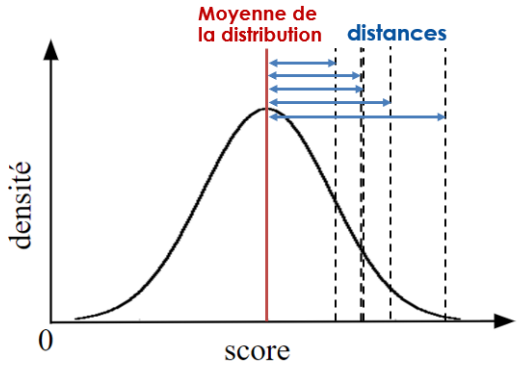

FIGURE 3.17 – Schéma illustrant les distances prises en compte dans le calcul de d. Ici, la distribution des scores est formalisée par une courbe gaussienne et les scores des anomalies sont formalisés par les lignes pointillées verticales.

L'un des premiers indicateur de la capacité des modèles à correctement placer les anomalies sur la distributions des scores est la distance entre les scores d'anomalies et la moyennes totale des scores. Nous cherchons, en effet, un modèle étant capable de placer les scores d'anomalies le plus à droite possible sur sa distribution de scores (cf : figure 3.17). Ainsi, lors de la comparaison de modèles, nous calculons la moyenne de ces distances dans un premier terme d comme suit :

$$d = \frac{\sum_{i=0}^{n} (anomaly_i - mean(scores))}{n} reg \tag{3.14}$$

Le terme reg correspond à un facteur de régulation destiné à ne pas laisser la performance des modèles, en termes de pure reconstruction, biaiser l'évaluation de leur capacité à détecter les instances de données anormales. En effet, nous cherchons à comparer les placements des scores d'anomalies relatifs à la distribution des scores. Au sein de nos expérimentations, nous avons utilisé un terme de régulation égal à $1/mean(scores)$ afin de normaliser les distances par rapport à l'ordre de grandeur général de la distribution.

d n'est néanmoins pas une information suffisante pour comparer des modèles. En effet, la forme de la distribution influence aussi notre jugement sur l'efficacité des modèles. Nous avons donc mis en place des indices prenant en compte d ainsi que l'utilisation de seuils hypothétiques (comme nous le décrivons figure 3.18).

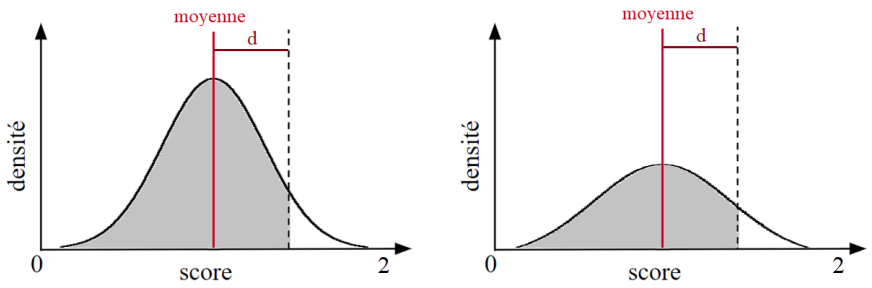

FIGURE 3.18 – Illustration de l'information prise en compte par l'évaluation de seuils hypothétiques. En pointillé, un score d'anomalie et en rouge la moyenne de la distribution. Dans les deux cas, d est identique. Néanmoins, dans le cas illustré à droite, la forme de la distribution des scores (plus large et écrasée que la distribution de gauche) montre que le modèle est moins performant correspondant est moins efficace que celui correspondant à la distribution de gauche. L'évaluation du ratio entre les aires grisées (à gauche des pointillés) et les aires blanches (à droite des pointillés) sous les courbes des distributions nous permet de nous en rendre compte.

Nous aboutissons aux indices suivant :

$$Fiability_a = d\frac{\frac{u+v}{2}}{m} \tag{3.15}$$

$$AbsFiability_a = d\frac{w}{m} \tag{3.16}$$

Les termes u et v dans la formule 3.15 correspondent respectivement au nombre d'erreurs (i.e. de scores), associées aux données saines, étant inférieurs au seuils hypothétiques correspondant à la moyenne des scores d'anomalies (u) et à la médiane des scores d'anomalies (v). La prise en compte de ces deux seuils permet de minimiser un éventuel biais en cas de présence de scores d'anomalies particulièrement excentrés. Nous divisons ensuite la moyenne de ces deux termes par m, le nombre total de score, afin de pondérer d par un terme entre 0 et 1.

Le terme w dans la formule 3.16 correspond au nombre de scores inférieurs au plus petit score d'anomalie.

Les deux indices évaluent la capacité globale d'un modèle à placer les anomalies autant que possible à droite de la distribution des erreurs. Cependant, l'indice de fiabilité (I.F.) permet de comparer les capacités globales de modèles, tandis que l'indice de fiabilité absolue (I.F.A.) donne plus d'importance au fait que tous les scores d'anomalies soient situés à droite de la distribution.

3.3.3 Résultats

Une instance hébergée par Google Compute Engine et équipée d'un GPU NVidia Tesla V100 a été utilisée afin de produire nos résultats. Les modèles ont été réalisés avec Tensorflow (cf : section 2.4.6) via le wrapper Keras.

TABLE 3.1 – Tableau résumant les données des meilleurs auto-encodeurs des deux types.

Type	Nombre de timesteps	Taille du goulot d'étranglement	% réduction	I.F.	I.F.A.	Temps d'entraînement (s)	Loss
Récurrent	48	200	80.15	1.339	**1.146**	1309	0.140
	48	150	85.12	1.282	1.112	1159	0.147
	48	250	75.20	1.279	0.943	1433	0.137
Convolutionnel	96	384	80.95	**1.381**	0.897	977	0.052
	48	192	80.95	1.251	0.606	593	0.051
	24	192	61.90	1.143	0.475	**412**	0.039

Nous avons testé une multitude de modèles des deux types, prenant en compte des fenêtres de la taille d'une journée, d'une demi-journée et d'un quart de journée. Pour chacun de ces modèles, nous avons testé diverses tailles de goulots d'étranglement afin de faire varier la réduction de dimensions. Pour chacun des modèles, nous avons utilisé du dropout sur toutes les couches (couche d'entrée afin de bénéficier de l'aspect « denoising » présenté section 2.5, et couches cachées afin d'améliorer les performances globales, cf : section 2.4.6). La probabilité de dropout a été fixée, lors d'une étude préalable (entraînements de divers auto-encodeurs des deux type sans comparaison des performances), à 0,2 (pour tous nos modèles, chacun des neurones présents avait 20% de chance d'être inactivé lors d'une itération lors de l'entraînement).

C'est essentiellement l'indice de fiabilité absolue qui nous permet de déterminer le modèle le plus performant en termes de détection des anomalies. En effet, comme le montre le tableau I, l'indice de fiabilité des auto-encodeurs récurrents et convolutionnels est du même ordre de grandeur. Cependant, les auto-encodeurs récurrents ont un meilleur indice de fiabilité absolue global. Nous pouvons confirmer, en regardant les distributions des meilleurs modèles sur la figure 3.19 et sur la figure 3.20, que l'auto-encodeur récurrent est plus efficace si nous considérons toutes les anomalies.

Outre le fait que notre expérience fournit des informations sur le type d'auto-encodeur le plus performant en matière de détection d'anomalies dans le contexte de bâtiments connectés, les informations rassemblées dans le tableau 3.1 nous donnent des indications sur les paramètres les plus importants à prendre en compte dans les modèles.

Comme nous l'avons mentionné précédemment, une Loss moindre ne signifie pas nécessairement une meilleure détection d'anomalies. Le paramètre le plus important est le taux de réduction de dimension ($\sim$80% pour les deux meilleurs auto-encodeurs à convolution et $\sim$80-85% pour les deux meilleurs auto-encodeurs récurrents). Ce taux de réduction de dimension est calculé en faisant le rapport du nombre de données présente dans la fenêtre (nombre de capteur * taille de la fenêtre, soit le nombre de timesteps) sur la taille du goulot d'étranglement. Nous pouvons voir sur la figure 3.21 que les résultats sont bien meilleurs quand une réduction de dimension suffisante est faite. Globalement, les résultats sont meilleurs entre 75% et 95%, ce qui représente des taux de compression très importants.

3.3.4 Discussion

Nous avons sélectionné deux types d'auto-encodeurs sensibles à toutes les catégories d'anomalies. Nous avons testé plusieurs modèles récurrents et convolutionnels sur un jeu de données

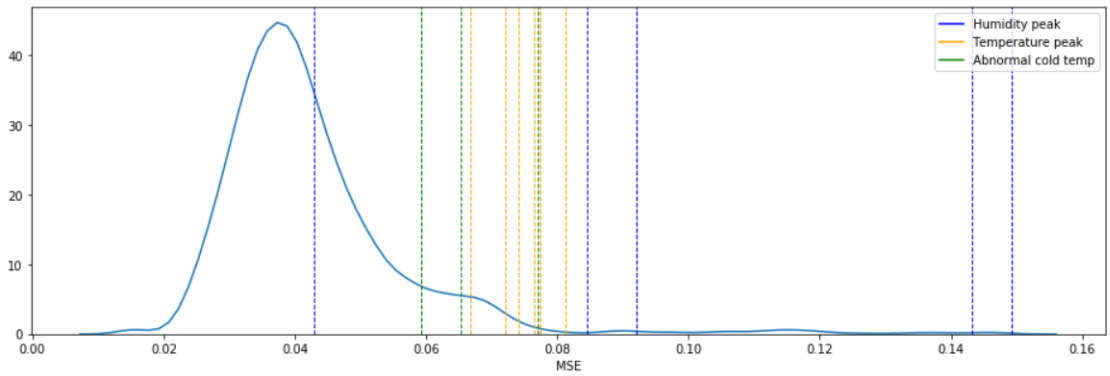

FIGURE 3.19 – Distribution des scores (distance calculée avec l'erreur quadratique) du jeu de données de test et positionnement des anomalies manuellement annotées pour le meilleur modèle d'auto-encodeur à convolution.

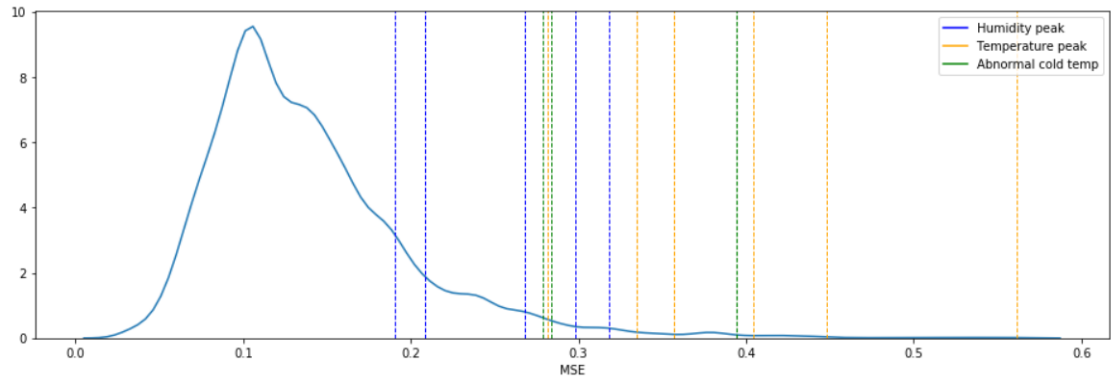

FIGURE 3.20 – Distribution des scores (distance calculée avec l'erreur quadratique) du jeu de données de test et positionnement des anomalies manuellement annotées pour le meilleur modèle d'auto-encodeur récurrent.

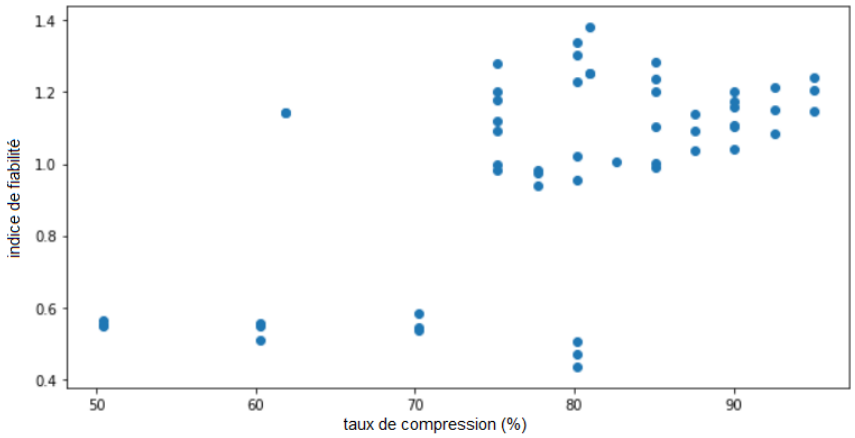

FIGURE 3.21 – Affichage de l'indice de fiabilité en fonction du pourcentage de réduction de dimension.

académiques conforme à notre besoin. Pour comparer les résultats, nous avons proposé deux indices qui traduisent la capacité des modèles à placer les anomalies dans la partie droite de la distribution de la distribution des scores.

L'étude des résultats obtenus nous a permis de déterminer que les auto-encodeurs récurrents semblent être les plus efficaces pour détecter les anomalies dans notre ensemble de données. Ce résultat peut s'expliquer par le fait que les auto-encodeurs à convolutions sont plus adaptés pour la détection et la prise en compte de motifs.

Dans notre cas, les motifs à apprendre correspondent à toutes les variations de chaque capteur correspondant aux journées typiques. On peut donc s'attendre à avoir de plus faibles performances si on augmente le nombre de capteurs à prendre en compte, et donc le nombre de motifs à apprendre.

En extrapolant ce résultat, on peut dire que les auto-encodeurs récurrents figurent, à ce jour, parmi les meilleurs candidats dans le domaine des réseaux de neurones appliqués à la détection d'anomalies dans les maisons connectés. Des tests supplémentaires sur différents jeux de données doivent encore être effectués pour corroborer cette affirmation.

Le temps d'entraînement de chaque modèle (cf. tableau 3.1), fortement corrélé à la taille de leur entrée et au nombre de neurones et généralement plus élevé avec les auto-encodeurs récurrents, est important. Ce résultat, sans surprise, est notamment lié à la complexité des modèles récurrents et aux optimisations faisant partie des CNNs (cf partie 3.1 et 3.2).

Bien que leur temps d'entraînement constitue pour le moment un obstacle à l'industrialisation de système implémentant de tels modèles, nous avons choisi de retenir les RNNs comme principale architecture.

L'accélération de leur temps d'entraînement constitue pour l'instant une perspective de recherche.

CHAPITRE 4

OPTIMISATION

Dans cette partie, nous exposons les travaux effectués dans le but d'améliorer les performances des réseaux de neurones auto-encodeurs utilisés dans le cadre de la détection d'anomalies. Nous développons les concepts d'incertitude, de réseaux de neurones bayésiens et d'approximation de réseaux de neurones bayésiens, ceux-ci étant à la base de nos contributions. Nos contributions incluent :

- De nouvelles fonctions de score prenant en compte l'incertitude ou reposant sur une approximation bayésienne
- La comparaison des performances de ces fonctions.

Les résultats présentés et méthodes proposées ont fait l'objet d'une publication [Legrand et al., 2019].

4.1 Incertitude et hypothèse

Nous commençons par illustrer le principe attendu d'incertitude dans les réseaux de neurones et les motivations derrière ce concept.

Si l'on veut mettre en place un classifieur d'images, par exemple un classifieur de races de chiens, nous pouvons utiliser un réseau de neurones avec en entrée une image ou une liste de caractéristiques et en sortie la race de chien associée. Un modèle très bien entraîné, avec de très bonnes performances, devrait, lors de la phase de test ou tout simplement lors de l'utilisation du modèle, renvoyer une race avec une confiance plutôt élevée.

Par contre, que devrait-il se passer si un utilisateur envoyait au classifieur une photo de chat en entrée ? Une telle entrée serait complètement en dehors de la distribution des données d'entraînement. Le modèle a été entraîné sur des photos de chiens de différentes races et a appris à bien les distinguer. Cet exemple simple peut être étendu à des cas d'utilisations plus sensibles, tels que les examens IRM avec des structures jamais observées par un système de diagnostic, ou des scènes de vidéo qu'un système de conduite autonome n'aurait jamais vu.

Un comportement possible, souhaité d'un modèle dans de tels cas, serait de renvoyer une prédiction, mais aussi de renvoyer une réponse avec l'information supplémentaire indiquant que le modèle n'a pas été formé pour traiter une telle information. Dans de nombreux domaines tels que la recherche en science ou encore le médical, des informations sur l'incertitude sont souvent utilisées, comme indiqué dans les travaux de Herzog et al. [Herzog and Ostwald, 2013], Krzywinski et al. [Herzog and Ostwald, 2013] ou encore Nuzzo [Nuzzo, 2014].

Il est essentiel dans ces domaines, de savoir quantifier la confiance dans les résultats produits par un modèle. De plus, lors de la conception d'un modèle, comprendre si celui-ci est peu confiant ou faussement trop confiant peut aider à en obtenir de meilleures performances. On peut notamment imaginer retravailler ou étendre les données d'entraînement.

De tels cas d'utilisation ont ainsi motivé les travaux dont nous tirons parti. En effet, il est difficile d'ignorer le parallèle que nous pouvons faire entre la capacité d'un modèle à pouvoir traiter une donnée (c'est-à-dire sa représentation au sein du jeu de données d'entraînement) et la rareté des anomalies au sein d'un jeu de données.

Nous formulons donc l'hypothèse suivante : La rareté et la divergence par rapport à la normalité au sein d'un jeu de données sont des caractéristiques inhérentes aux anomalies (cf : partie 2.1). Ainsi, d'après la définition que nous donnons de la confiance au sein d'un modèle prédictif, le traitement d'une anomalie devrait produire un résultat associé à une incertitude élevée.

Nous cherchons donc à tirer parti de cette incertitude dans le cadre de l'utilisation de réseaux de neurones auto-encodeurs pour la détection d'anomalie.

4.2 Application de l'incertitude aux auto-encodeurs

4.2.1 Réseaux de neurones bayésiens

Les réseaux de neurones bayésiens (Bayesian Neural Networks, BNNs) ont été suggérés pour la première fois dans les années 90 et ont été étudiés dans de nombreux travaux depuis lors [Neal, 1996] [MacKay, 1992]. Jusqu'à récemment, les BNNs étaient essentiellement des modèles théoriques difficiles à utiliser en pratique.

Les BNNs ont pour but d'offrir une version probabiliste des réseaux de neurones artificiels classiques en déduisant les distributions des paramètres du modèle plutôt que de traiter les poids comme de simples valeurs réelles à une dimension.

D'après la théorie, un tel modèle apporterait de la robustesse face au sur-apprentissage, pourrait offrir un moyen efficace d'estimer l'incertitude d'une prédiction et pourrait être entraîné avec des jeux de données bien plus petits que ceux utilisés couramment.

Cependant, l'inférence postérieure exacte de la distribution des poids est rarement possible. En effet, bien que les BNNs sont faciles à formuler, l'aspect non-linéaire constitue une barrière majeure à l'implémentation d'algorithmes efficaces intégrant l'aspect probabiliste.

Ces limitations ont motivé divers travaux portant sur l'approximation d'inférence. Ainsi, plusieurs méthodes d'inférence approximatives ont récemment été proposées pour les réseaux de neurones bayésiens dans le but de contourner cette barrière.

On retrouve parmi celles-ci la recherche stochastique [Paisley et al., 2012], la variationnelle Bayes [Kingma and Welling, 2014], la rétropropagation probabiliste [Hernández-Lobato and Adams, 2014], « Bayes par BackProp » [Blundell et al., 2015] et son extension. [Fortunato et al., 2017]. On retrouve aussi divers algorithmes s'étendant sur l'optimisation de l'α-divergence, notamment [Hernández-Lobato et al., 2016], [Li and Gal, 2017] (toutes ces approches sont détaillées dans [Gal, 2016]).

Tous les algorithmes mentionnés précédemment nécessitent différentes méthodes d'apprentissage pour un réseau de neurones, et différents facteurs et paramètres doivent être ajustés. Ce sont notamment la fonction d'évaluation de perte et l'algorithme d'apprentissage qui doivent être modifiés et ajustés selon la méthode et le cas d'utilisation.

En pratique, on aura généralement tendance à s'orienter vers des solutions spécifiques ne nécessitant pas de changer l'architecture du réseau de neurones et pouvant être directement appliquées au modèle précédemment formé.

En outre, la plupart des algorithmes d'inférence existants introduisent un nombre non-négligeable de paramètres supplémentaires, allant parfois jusqu'à doubler le nombre de paramètres initial. Ces algorithmes sont donc souvent délaissés, en particulier en cas de modèle conséquent.

Pour cette étude, nous utilisons une méthode d'approximation récente répondant aux problèmes précédemment mentionnés basée sur le Monte Carlo Dropout.

Cette méthode, proposée dans [Gal and Ghahramani, 2016a] et [Gal and Ghahramani, 2016b], nécessite peu ou pas de changement de l'architecture du modèle existant et fournit une estimation de l'incertitude pour un coût très réduit.

En effet, cette approche est basée sur l'utilisation du dropout standard (tel que présenté dans 2.4.6) appliqué à chaque couche cachée dans un réseau de neurones classique lors de la phase de test.

De plus, l'utilisation de cette méthode dans le milieu industriel a déjà mené à l'obtention de meilleurs résultats par rapport à ceux obtenu grâce à la méthode classique. En effet, Uber a montré un exemple notable de travail utilisant cette même technique. La prise en compte de l'incertitude estimée grâce au Monte Carlo dropout leur a permis de bénéficier d'une augmentation significative des performances de leurs systèmes de prédiction de trafic [Zhu and Laptev, 2017].

Nous rappelons que les modèles que nous utilisons sont inspirés des denoising auto-encodeurs (cf : Section 2.5) et bénéficient donc déjà de dropout appliqué à la couche en entrée. L'architecture résultante, dans notre cas, est alors un auto-encodeur avec du dropout appliqué à chaque couche.

4.2.2 Approximation bayésienne

Nous commençons dans cette partie par développer le moyen utilisé pour estimer l'incertitude du modèle.

La sortie de notre modèle peut être vue approximativement comme un échantillon aléatoire généré à partir de la distribution postérieure de tous les poids [Gal and Ghahramani, 2016a].

En conséquence, l'incertitude du modèle pour cet échantillon peut être estimée à l'aide de la variance de l'échantillon des prédictions du modèle en quelques répétitions. Dans notre cas, nous nous intéresserons au cas par cas à l'incertitude de prédiction de chaque donnée plutôt que sur l'incertitude globale du modèle.

Nous considérons un modèle $f^{\boldsymbol{W}}$, avec f la fonction d'approximation du modèle et $\boldsymbol{W}$ l'ensemble des paramètres dont dépend ce modèle (tel que décrit dans [Zhu and Laptev, 2017]).

Nous pouvons considérer que $\boldsymbol{W}$ représente les poids de chaque couche sous forme de matrice, chacune des lignes correspondant aux vecteurs des poids de chaque couche.

Les réseaux de neurones approximant les BNNs sont souvent initialisés en appliquant une distribution sur tous les paramètres.

Ainsi, pour chaque vecteur issu de la matrice des poids $\boldsymbol{W}_i$ (incluant les vecteurs de biais $\boldsymbol{b}_i$) pour une couche i, une distribution Gaussienne standard est souvent utilisée :

$$p(\boldsymbol{W}_i) = \mathcal{N}(\mathbf{0}, \boldsymbol{I})$$

Le modèle essaiera ensuite de faire correspondre la distribution de manière optimale aux données via entraînement standard, tel que présenté dans la partie 2.4.

Typiquement, la formulation des données générées par le modèle est la suivante : $p(y|f^{\boldsymbol{W}}(x))$.

Comme nous l'avons expliqué dans la partie 2.5, les auto-encodeurs se rapprochent bien plus d'un cas de régression multiple, où chaque attribut de l'entrée initiale doit être ré-estimé à partir d'un vecteur latent. Nous nous concentrons donc sur la formalisation inhérente au cas de la régression tel que présenté dans [Zhu and Laptev, 2017]. Dans un cas de régression, il est souvent assumé

$$y|\boldsymbol{W} \sim \mathcal{N}(f^{\boldsymbol{W}}(x), \sigma^2) \tag{4.1}$$

avec une certaine quantité de bruit inévitable σ.

Nous étendons maintenant cette formalisation au cas des auto-encodeurs.

Nous avons $y = \hat{x}$, $\hat{x}$ étant une approximation recréée pour $f^{\boldsymbol{W}}(x)$. Etant donné un jeu de n données, $\mathbf{X}$ = (x_1,..., x_n), l'inférence Bayésienne vise à trouver la distribution postérieure idéale pour les paramètres du modèle $p(\boldsymbol{W}|\boldsymbol{X}, \hat{\boldsymbol{X}})$. En d'autres termes, l'inférence Bayésienne cherche à déterminer les paramètres $\boldsymbol{W}$ qui sont les plus susceptibles d'avoir généré une sortie $\boldsymbol{x} \in \boldsymbol{X}$.

Comme nous l'avons précédemment mentionné, l'incertitude du modèle peut être estimée en déterminant Var($\hat{\boldsymbol{X}}^*|\boldsymbol{X}^*$), $\boldsymbol{X}^*$ étant un jeu de données de test, grâce au Monte Carlo dropout.

De la même manière, nous estimons l'incertitude de prédiction pour chaque donnée $x^* \in \boldsymbol{X}^*$ en déterminant Var($f^{\boldsymbol{W}}(x^*)$).

Le Monte Carlo dropout se déroule comme suit [Gal and Ghahramani, 2016a] :

Nous considérons l'auto-encodeur $f^{\boldsymbol{W}}$, $\boldsymbol{W}$ étant déterminé lors de l'entraînement sur un jeu de données différent.

Comme nous l'avons précédemment mentionné (cf : partie 4.2.1), nous avons aussi utilisé du dropout de manière standard lors de l'entraînement afin d'augmenter nos performances de base.

Pour chaque individu x^* du jeu de données de test, nous utilisons $f^{\boldsymbol{W}}$ pour produire un résultat $\hat{x}^*$.

Pendant ce passage au sein du modèle, du dropout est appliqué sur toutes les couches.

Nous répétons cette opération B fois, sachant que le masque de dropout aléatoire sera différent à chaque fois, celui-ci dépendant d'une probabilité p.

Nous obtenons donc, pour une donnée d'entrée, un jeu de B données ($\hat{x}_1^*$,...,$\hat{x}_b^*$) différentes.

De ce jeu de prédictions, deux valeurs portant des informations différentes peuvent être calculées.

La première correspond à « l'Approximation Bayésienne » qui, si le nombre d'itération B est suffisant, donnera une estimation fiable de la moyenne des valeurs des distributions que nous aurions eu en sortie avec un réseau de neurones Bayésien tel qu'exprimé par la formule 4.1 :

$$\bar{\hat{x}}^* = \frac{1}{B}\sum_{b=1}^{B} \hat{x}_b^* \tag{4.2}$$

La seconde valeur est l'incertitude de prédiction, que l'on peut estimer en calculant la variance des B approximations issues du Monte Carlo dropout précédemment calculée :

$$\text{Var}(f^{\boldsymbol{W}}(x^*)) = \frac{1}{B}\sum_{b=1}^{B}\left(\hat{x}_b^* - \bar{\hat{x}}^*\right)^2 \tag{4.3}$$

4.2.3 De nouvelles fonctions de score

Fonctions standards

Comme nous l'avons expliqué dans la partie 2.5, les auto-encodeurs font partie des algorithmes de détection d'anomalies qui produisent des scores, typiquement la distance entre l'entrée et la sortie.

Dans notre cas, nous utilisons la méthode classique Mean Squared Error (MSE, cf : 2.2.3).

La première fonction que nous évaluons au sein de notre étude est donc le score standard S., pouvant être formulé comme suit :

Pour chaque individu $\boldsymbol{x} \in \mathbb{R}^d$

$$\text{Score}_S(\boldsymbol{x}) = \frac{1}{d}\sum_{i=1}^{n}\left(\boldsymbol{x}_i - f^{\boldsymbol{W}}(\boldsymbol{x})_i\right)^2. \tag{4.4}$$

A notre connaissance, il n'existe pas de fonctions de score couramment utilisées dans le domaine de la détection d'anomalies basées sur l'approximation bayésienne.

Néanmoins, l'utilisation de l'approximation bayésienne au sein des réseaux de neurones existe déjà dans d'autres domaines, nous avons donc choisi de considérer l'approche que nous allons maintenant développer comme standard.

Comme nous l'avons vu dans la partie précédente, l'utilisation de l'approximation bayésienne nous permet d'avoir un résultat prenant en compte la distribution postérieure des valeurs de sortie que nous aurait fourni un BNN utilisant des paramètres $\boldsymbol{W}$ ajustés à leur distribution postérieure optimale.

Il est donc justifié de s'intéresser à l'utilisation de cette approximation dans le cadre de la détection d'anomalies.

Nous avons donc choisi d'évaluer une fonction de score basée sur la distance existante entre l'entrée d'un auto-encodeur et l'approximation bayésienne correspondante.

Notre seconde fonction de score, B.A. (Bayesian Approximation), peut être formulée comme suit :

$$\text{Score}_{BA}(\boldsymbol{x}) = \frac{1}{d}\sum_{i=1}^{n}\left(\boldsymbol{x}_i - \bar{\hat{\boldsymbol{x}}}_i\right)^2 \tag{4.5}$$

avec $\bar{\hat{\boldsymbol{x}}}$ correspondant à la formule 4.2.

L'étude de la fonction de score précédemment expliquée implique l'implémentation du Monte Carlo dropout. Il est donc trivial à partir de ce point de prendre en compte l'incertitude.

Fonctions de score incluant l'incertitude

Afin d'augmenter les performances des auto-encodeurs dans le contexte de la détection d'anomalies, tout facteur augmentant le score d'une anomalie ou baissant le score d'une donnée normale est une piste pouvant être exploitée.

Ainsi, l'hypothèse que nous avons formulée dans la partie 4.1 concernant l'augmentation de l'incertitude pour les anomalies peut être développée comme suit :

- La rareté et la divergence par rapport à la normalité au sein d'un jeu de données sont des caractéristiques inhérentes aux anomalies.

Ainsi, d'après les méthodes d'approximation d'inférence bayésienne que nous utilisons et la définition que nous donnons d'une anomalie, la distribution des paramètres $\boldsymbol{W}$ post-entraînement d'un auto-encodeur ne devrait avoir été suffisamment influencée que par les données les plus représentées dans la distribution du jeu d'entraînement.

Par conséquent, une anomalie donnée à un tel modèle devrait avoir une variance associée $\text{Var}(f^{\boldsymbol{W}}(x^*))$ plus haute que les données normales.

La prise en compte de cette estimation de l'incertitude devrait augmenter les performances des fonctions de score classiques.

Cependant, nous ne proposons pas d'utiliser uniquement l'incertitude comme fonction de score.

En effet, notre étude nous a révélé que l'hypothèse, selon le cas d'utilisation, n'est pas toujours vérifiée (si c'était le cas, les méthodes proposées seraient toujours les meilleures) et dépend de facteurs très variés, sur lesquels nous avons très peu d'influence en tant que simple utilisateur des données.

La conséquence est que, bien que dans certains cas l'incertitude seule serait suffisante pour constituer une fonction de score correcte, cette fonction montrerait des performances mauvaises voire nulles si l'hypothèse que nous formulons ne s'appliquait pas au cas d'utilisation.

De plus, lorsque l'hypothèse est vraie pour un cas, l'incertitude et l'erreur standard ne sont pas nécessairement corrélées. Ainsi, un modèle peut parfois donner une prédiction $\hat{x}$ assez éloignée de x (x étant une anomalie dans le cas présent) tout en montrant une certitude élevée et vice versa pour les données normales.

Nous préférons donc nous concentrer sur des fonctions enrichies qui tirent parti du mieux possible de l'erreur de recréation ainsi que de l'incertitude.

Nous proposons des fonctions incluant un terme de pondération modérée. Effectivement, l'incertitude est généralement une valeur basse parfois très proche de 0 en cas de certitude très forte. Pondérer une fonction standard par l'incertitude telle quelle, serait donc source de faux positifs et négatifs en cas de non corrélation avec l'erreur pour les mauvaises données.

Nous évaluons donc les fonctions standards précédemment présentées pondérées par un terme incluant l'incertitude de manière à ce que celles-ci ne soient pas pénalisées si l'incertitude de prédiction n'est pas adéquate.

La première fonction proposée est donc la fonction standard S. pondérée, W.S. (Weighted Standard), et est formulée comme suit :

$$\text{Score}_{WS}(\boldsymbol{x}) = \text{Score}_S(\boldsymbol{x}) * (1 + \frac{\text{Var}(f^{\boldsymbol{W}}(\boldsymbol{x}^*))}{reg}) \tag{4.6}$$

Le terme reg est utilisé pour que l'incertitude soit du même ordre de grandeur que la fonction pondérée (ici Score_S) afin que les deux aient la même importance lors de la détermination du seuil. Nous avons, au sein de nos expérimentation, un terme reg égal à la moyenne de toutes les incertitudes de prédictions.

L'incertitude étant potentiellement très petite, nous additionnons 1 au terme présenté précédemment afin de conserver l'information de la fonction de score initiale (ici, Score_S). Notre ambition est de vérifier que seul les scores d'anomalies soient impactés par l'incertitude, et donc décalés vers la droite de la distribution.

De la même manière, la dernière fonction que nous évaluons est la fonction de distance entre l'entrée et l'approximation bayésienne pondérée, W.B.A. (Weighted Bayesian Approximation) et est formulée comme suit :

$$\text{Score}_{WBA}(\boldsymbol{x}) = \text{Score}_{BA}(\boldsymbol{x}) * (1 + \frac{\text{Var}(f^{\boldsymbol{W}}(\boldsymbol{x}^*))}{reg}) \tag{4.7}$$

Ainsi, cette fonction ne dépend uniquement que de facteurs basés sur la méthode utilisée pour réaliser l'approximation d'inférence bayésienne. De la même manière que précédemment, nous avons fixé le terme reg, dans nos expérimentations, à la moyenne de toutes les incertitudes de prédictions.

Pour toutes nos évaluations, nous avons utilisé, lors du Monte Carlo dropout, un nombre d'itération B fixé arbitrairement à 100.

4.3 Etude empirique

Afin d'éprouver notre hypothèse et de déterminer si les fonctions de scores que nous avons proposées sont efficaces, nous avons réalisé une expérimentation sur des données réelles.

4.3.1 Données

Pour évaluer nos fonctions, nous avons sélectionné des jeux de données répondant à certains critères. Étant donné que les auto-encodeurs sont basés sur la réduction de dimension, nous avons sélectionné des jeux de données comportant au moins 15 caractéristiques et nous n'avons pas fixé de limite supérieure lors de notre recherche.

Nous disposons donc d'un ensemble de jeux de données variés ayant des dimensions allant de 16 à 617.

Les jeux de données sélectionnés sont généralement utilisés pour la classification binaire ou à multiple classes.

Nous avons, dans le cas des jeux de données utilisés pour la classification multi-classes, sélectionné et fusionné plusieurs classes après les avoir sous-échantillonnées pour créer notre ensemble d'anomalies. Dans le cas des jeux de données utilisés pour la classification binaire, nous avons sous-échantillonné l'une des deux classes. De cette manière, nous avons pu produire une expérimentation avec des taux d'anomalies complètement contrôlés et nous avons pu mettre en place un protocole d'évaluation efficace.

TABLE 4.1 – Résumé des différents jeux de données utilisés ainsi que de leurs caractéristiques. L'attribut « nb d'individus » est noté comme approximatif car il concerne la taille des jeux de données d'entraînement : nous avons testé diverses concentrations en anomalies, celui-ci n'est donc pas resté fixe.

Jeu de données	Nb de features	$\sim$ nb d'individus	Classe anomalie
LRS	101	360	classes 4 & 8
Isolet	617	7000	classes 23 & 26
Satimage	37	4000	classe 2
Ionosphere	34	200	classe 'bad'
WBC	30	320	classe 'malign'
Musk	166	1100	classe 'musk'
Letter Recognition	16	10000	classes 'M' & 'W'

Avec cette méthode d'utilisation des données de classification pour l'évaluation des méthodes de détection d'anomalies, nous sommes en conformité avec la littérature [Zimek et al., 2013].

Les ensembles de données avec une classe binaire sont Musk [Chapman and Jain, 1995], Wisconsin Breast Cancer (WBC) [Wolberg et al., 1995] et Ionosphere [Sigillito, 1989]. Les autres jeux de données que nous avons utilisés sont le spectromètre à basse résolution (LRS) [Stutz, 1988], Isolet [Cole et al., 1988], Satimage [Srinivasan, 1993] et la reconnaissance de lettres [Slate, 1991].

Nous avons donc un ensemble hétérogène de jeux de données non-simulées, provenant de divers domaines d'application, ayant différentes dimensions et différents nombres d'échantillons.

Il convient toutefois de noter que ces jeux de données permettent l'utilisation d'un auto-encodeur «standard» (détaillé dans la partie 2.5). Nous n'avons donc pas axé notre étude sur les ensembles de données d'images, de vidéos, de séries temporelles, etc.

De ce fait, nous n'avons pas eu besoin d'architectures d'auto-encodeurs plus complexes tels que des auto-encodeurs récurrents ou convolutionnels (cf : partie 3.1 et 3.2). Ces ensembles de données sont des données réelles provenant du référentiel d'apprentissage automatique UCI [Dua and Graff, 2017] et ont déjà été utilisés dans la littérature pour des travaux sur la détection ou la classification des anomalies. Les détails concernant les classes de données utilisées en tant qu'anomalies, le nombre de caractéristiques, ainsi que le nombre d'individus de chaque jeu de données sont donnés dans le tableau 4.1.

Enfin, pour chaque cas, les données ont été centrées-réduites.

4.3.2 Méthodologie

Contrôle des données

Pour évaluer nos fonctions de score, nous avons, pour chaque jeu de données, testé des modèles entraînés avec des données contenant des concentrations d'anomalies que nous avons variées.

De cette manière, nous pouvons mieux évaluer les différentes fonctions de score car le niveau de difficulté de la tâche de détection d'anomalies augmente avec le nombre d'anomalies présentes dans le jeu de données d'apprentissage.

En raison des caractéristiques différentes des jeux de données d'origine, nous n'avons pas été en mesure d'utiliser des concentrations identiques pour chacun. En effet, il a fallu adapter les concentrations en anomalies aux capacités des auto-encodeurs utilisés afin de tester des niveaux de difficultés cohérents.

Par exemple, l'auto-encodeur entraîné avec le jeu de données Ionosphere avec une concentration d'anomalies de 18,8% est très efficace et produit des scores très satisfaisants, tandis que les scores provenant du meilleur auto-encodeur formé avec le jeu de données LRS avec une concentration d'anomalies de 2,2% le sont beaucoup moins.

De plus, la nature des jeux de données nous a bridé dans l'établissement des taux de concentrations en anomalies.

Par exemple, pour le jeu de données Letter Recognition, les 26 classes de base étaient représentées de manière équivalente. Il n'était donc pas possible de produire de jeux de données d'entraînement avec une concentration en anomalies supérieure à 7.69%.

Mise en place des modèles

Pour chaque concentration en anomalies de chaque jeu de données, nous avons testé de multiples modèles afin de ne garder que ceux répondant le plus efficacement à notre critère de sélection. Afin de ne pas biaiser l'expérience dans le sens des fonctions que nous proposons, nous avons limité notre critère de sélection à la fonction de score standard (S.).

Ainsi, dans nos résultats, nous montrons les performances des modèles ayant produit le meilleur score standard, pour une concentration d'anomalies et un jeu de données.

De plus, pour chaque concentration d'anomalies donnée, nous avons formé et testé plusieurs modèles avec les mêmes paramètres. En effet, plus le volume est petit ou plus la taille du jeu de données utilisé est petite, plus deux modèles avec les mêmes paramètres peuvent produire des résultats différents en raison, par exemple, de la nature aléatoire du dropout.

Afin d'être juste dans notre évaluation des modèles et des fonctions de score associées, nous utilisons le même nombre d'anomalies et de données standards pendant la phase de test.

Cela nous permet d'utiliser la méthode de calcul de l'aire en dessous de la courbe ROC (cf : partie 2.2.3) pour chaque modèle (chaque concentration d'anomalies pour chaque jeu de données) sans pour autant sous-estimer la valeur des anomalies en raison de leur nature rare. Cette méthode est largement utilisée dans la littérature pour évaluer les performances des modèles.

4.3.3 Résultats

De la même manière que pour l'expérimentation précédente, tous nos résultats ont été obtenus en entraînant et testant nos modèles sur une instance hébergée par Google Compute Engine et équipée d'un GPU NVidia Tesla V100. Les modèles ont été réalisés avec Tensorflow (cf : section 2.4.6) via le wrapper Keras.

Nous avons établi trois concentrations en anomalies différentes pour chaque jeu de données afin de proposer des niveaux de challenge équitables. Nous avons donc abouti à 21 cas d'étude.

Pour chacun de ces cas, nous avons testé la fonction de score standard ainsi que les fonctions de score initialement proposées.

Nous rappelons que notre hypothèse initiale est que l'incertitude de prédiction estimée avec $\text{Var}(f^{\boldsymbol{W}}(x^*))$ associée à une sortie $\hat{x}$ est plus élevée lorsque l'entrée x est une anomalie que lorsqu'elle est normale.

Nous trouvons que :

- La fonction de score standard (S.) n'est la meilleure que dans deux cas ($\sim$9,5%).
- La fonction de score basée sur l'approximation bayésienne (B.A.) est la meilleure dans 6 cas ($\sim$28,5%).
- Le score de la fonction utilisant l'erreur de base et l'incertitude (W.S.) est le meilleur dans sept cas ($\sim$33,3%).
- La fonction de score utilisant l'approximation bayésienne et l'incertitude (W.B.A.) est la meilleure dans six cas ($\sim$28,5%).

L'ensemble des résultats sont explicités dans le tableau 4.2.

Notre première constatation est que notre hypothèse initiale n'est pas toujours vérifiée, étant donné que nos fonctions n'ont pas été les meilleures dans tous les cas.

Néanmoins, notre seconde constatation est que les fonctions de scores proposées ont généralement eu de meilleurs résultats que la fonction de score classique lors de notre expérimentation.

Nous nous étendons sur l'évaluation des performances des fonctions de score dans la prochaine partie.

TABLE 4.2 – Evaluation des différentes fonctions de score. Les scores en gras sont les meilleurs pour un jeu de données et pour une concentration d'anomalies donnée.

Jeux de données		Aire sous la courbe ROC			
Nom	taux d'anomalies (%)	S.	B.A.	W.S.	W.B.A.
LRS	2.2	0.7557	0.7571	0.7561	**0.7577**
	5.4	0.6959	0.6932	**0.6971**	0.6942
	10	0.6408	**0.6506**	0.6375	0.6458
Isolet	1.2	0.9075	0.909	0.914	**0.9148**
	3.5	0.8743	0.8718	**0.8814**	0.8807
	6.8	0.8115	0.8101	**0.8193**	0.8173
Satimage	2.9	0.9386	0.936	**0.966**	0.9645
	5.7	0.9159	0.9061	**0.9469**	0.9383
	10.8	0.824	0.805	**0.8607**	0.8447
Ionosphere	4.6	0.9927	**0.9930**	0.9924	0.9913
	10.6	**0.9743**	0.9725	0.9718	0.9706
	18.8	0.985	**0.9868**	0.9831	0.9837
WBC	5.2	0.9212	0.9188	0.9268	**0.9316**
	9.7	0.9184	0.918	0.9212	**0.9236**
	15	0.7748	0.7796	0.788	**0.8036**
Musk	7.2	0.9868	**0.9871**	0.9854	0.9854
	10.1	0.9288	0.9277	**0.931**	0.93
	16.3	0.9269	0.9263	0.9285	**0.9289**
Letter Recognition	1.9	**0.8887**	0.887	0.8884	0.8866
	3.3	0.8572	**0.8583**	0.847	0.8463
	6.3	0.8327	**0.834**	0.8247	0.8256

4.3.4 Discussion

Ici, nous considérons les fonctions de score non-pondérées par l'incertitude et celles qui le sont comme deux catégories.

Pour un modèle donné, les scores de chaque catégorie sont pour la plupart similaires. En effet, bien que toutes les fonctions de scores soient très corrélées entres elles, les fonctions de la première catégorie présentent une corrélation de Pearson [Pearson, 1901] quasiment parfaite de $\sim$0.9984. Il en est de même pour les fonctions de la seconde catégorie, avec une corrélation de $\sim$0.9982.

Nous en déduisons que, généralement, la prise en compte de l'incertitude est le facteur déterminant de l'augmentation ou de la diminution de la performance, en fonction des modèles.

La catégorie prenant en compte l'incertitude de prédiction est celle ayant présenté le plus de fois un score plus élevé. La figure 4.1 illustre un exemple où nous pouvons voir que la taille de la section où se chevauchent la distribution normale et la distribution de l'anomalie est plus petite lorsque la fonction W.S est utilisée, par rapport à la fonction S.

Néanmoins, une investigation plus poussée nous incite à la prudence quant à la désignation des meilleurs fonctions. En effet, nous avons évalué pour chaque fonction de score la propension à s'éloigner du meilleur score propre à chaque cas.

$$distance = \sum_{cas}(max(scores_{cas}) - score_{cas})$$

Nous trouvons :

- Fonction standard (S.) : 0.1837
- Approximation Bayésienne (B.A.) : 0.2074
- Standard pondérée (W.S.) : 0.0681
- App. Bayésienne pondérée (W.B.A.) : 0.0702

Ainsi, nous constatons certains décalages. La fonction de score basée sur l'Approximation Bayésienne présente la plus grande distance totale alors que, dans notre expérimentation, elle a été la plus efficace dans $\sim$28,5% des cas. De plus, la fonction standard présente une distance totale inférieure mais n'a été la meilleure que dans $\sim$9,5% des cas.

De plus, cette expérimentation présente des limitations.

Par exemple, la forme des données utilisées (dimension, quantité) ne semble pas aider à déterminer quelle fonction de score sera la meilleure a priori.

Néanmoins, il semble que la capacité générale d'un auto-encodeur à discriminer les anomalies soit un bon indicateur. On peut voir sur la figure 4.2 que la catégorie qui n'utilise pas d'incertitude a généralement les meilleurs scores lorsque ceux-ci sont les plus élevés.

Ainsi, nous déduisons que l'utilisation de l'incertitude n'est pas nécessaire et aura tendance à réduire les performances des fonctions de score lorsque l'auto-encodeur présente de très bonnes performances.

Néanmoins, la tendance reste assez légère et nécessiterait une étude sur une échelle bien plus élevée pour affirmer ou infirmer cette tendance.

C'est pour ces raisons que nous encourageons donc le test de chacune des fonctions lors de l'établissement d'un auto-encodeur destiné à la détection d'anomalies.

Heureusement, le Monte Carlo dropout permet à la fois l'implémentation de l'Approximation Bayésienne et des fonctions pondérées proposées de manière triviale et pour un coût très faible en termes de temps et de complexité.

Nous rappelons que tous les auto-encodeurs utilisés ont été des auto-encodeurs simples. Notre hypothèse initiale n'a donc pas encore été formellement testée sur les auto-encodeurs récurrents dont nous nous servons dans le reste de cette thèse.

Nous étudions l'utilisation de l'incertitude pour des auto-encodeurs récurrents dans la prochaine partie ainsi que dans le chapite 5.

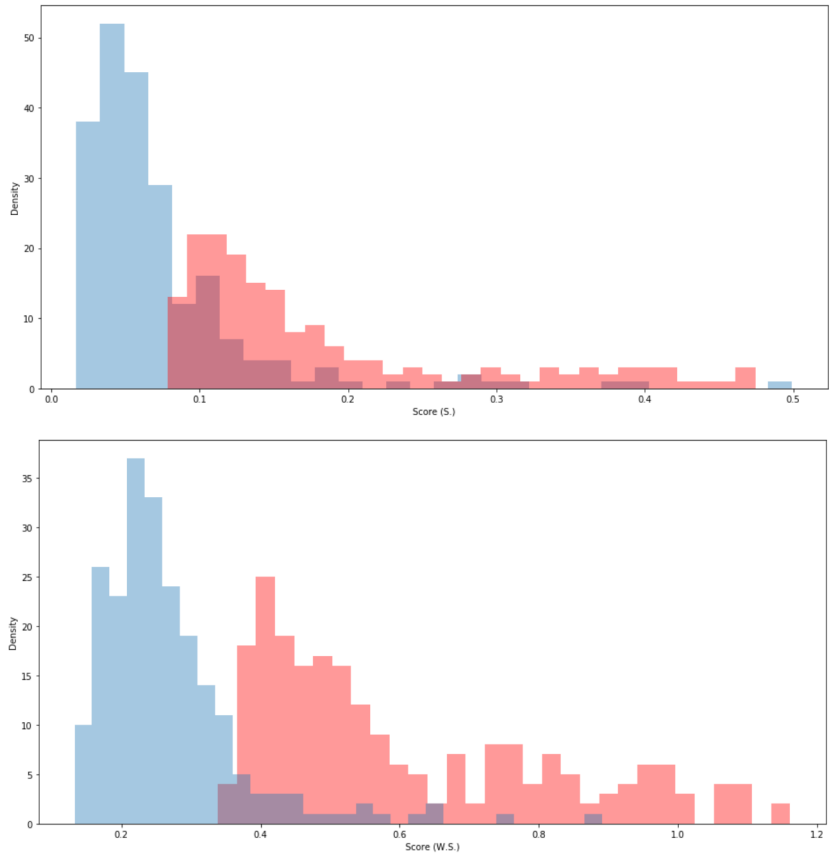

FIGURE 4.1 – Distribution des scores standards (en haut) et des scores standards pondérés (en bas) pour le jeu de données de test Satimage pour une concentration en anomalies lors de l'entraînement de 2.9%. Les distributions des scores des données standards sont bleues et les distributions des scores des anomalies sont en rouge.

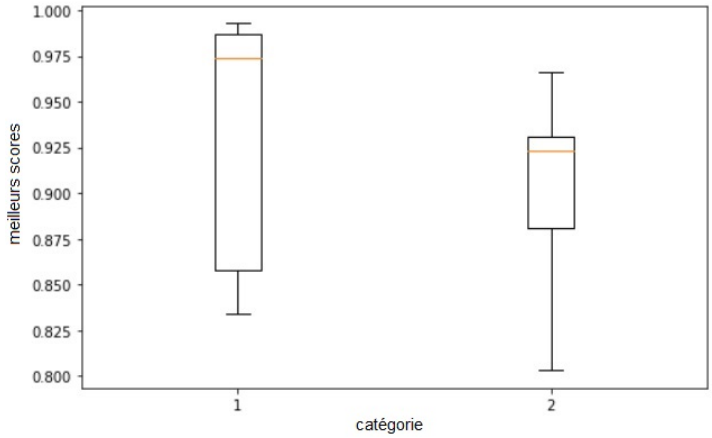

FIGURE 4.2 – Diagrammes en boîtes des meilleurs scores pour chaque catégorie. Nous incluons dans la catégorie 1 les scores non-pondérés (standards et approximations bayésiennes). La catégorie 2 correspond aux fonctions pondérées que nous proposons. Le jeu de données LRS a été ignoré pendant la mise en place de ce diagramme étant donné les performances très basses de toutes les fonctions de score.

4.4 Cas d'utilisation

Au cours de cette thèse, nous avons eu l'occasion de mettre en pratique ce que nous avons recommandé dans la précédente partie sur un cas concret. En effet, les fonctions proposées ont pu être testées sur des données réelles lors d'une preuve de concept effectuée en partenariat avec l'entreprise Enedis [1]. La preuve de concept a concerné la détection d'anomalies dans des données liées aux compteurs connectés Linky. Pour des raisons de confidentialité, nous ne pouvons entrer dans le détail du cas d'utilisation ou des résultats.

Cette preuve de concept a été l'occasion de tester pour la première fois nos méthodes avec des auto-encodeurs récurrents dotés de cellules LSTM (cf : partie 3.2). Les individus du jeu de données utilisé étaient représentés par 14 caractéristiques. Ces caractéristiques incluent 4 données temporelles concernant la saisonnalité des données obtenues selon le procédé décrit section 3.3.1. En effet, un périodogramme a révélé deux phénomènes de saisonnalité avec des périodes de 8h et de 24h. Nous avons utilisé des fenêtres d'une journée. Les données ayant été remontées toutes les 5 minutes, la dimension totale de nos fenêtres correspondaient à $288 * 14$.

En comparant toutes les fonctions de score présentées dans la partie 4.2.3, il s'est avéré que la fonction basée sur l'approximation bayésienne s'est révélée être la meilleure. Ce n'est néanmoins pas cette méthode qui a été retenue pour le cas d'utilisation initial. En effet, les scénarios anormaux métiers à détecter ne se reflétaient pas entièrement en tant qu'anomalies statistiques dans les données fournies. La solution finale adoptée a donc été une méthode basée sur des règles métiers bien spécifiques, cette méthode offrant de meilleurs résultats. On peut citer ce cas comme un exemple de limite des auto-encodeurs utilisés dans le cadre de la détection d'anomalies.

[1] https://www.enedis.fr/

CHAPITRE 5

APPLICATION AUX BÂTIMENTS CONNECTÉS

5.1 RNN variationnel

Nous n'avons pas encore, au cours de cette thèse, abordé la question du dropout (cf : section 2.4.6) appliqué aux unités récurrentes utilisées notamment lors de notre première expérimentation (cf : chapitre 3).

Les réseaux de neurones récurrents (RNN), sujets à de nombreux travaux et de nombreuses optimisations, constituent une branche conséquente de l'apprentissage automatique. Cependant, leur tendance au sur-apprentissage est restée, jusqu'en 2016, une des difficultés lors de l'utilisation de ceux-ci. En effet, le dropout classique s'est révélé inadapté lorsqu'il est appliqué à des couches récurrentes.

Ainsi, lors de notre première expérimentation, nous n'avons pas utilisé de dropout au sein des unités récurrentes, rendant nos modèles potentiellement sensibles au sur-apprentissage.

Afin de compléter nos travaux, nous avons cherché, au sein de la littérature, un moyen de pallier ce phénomène afin d'étendre notre connaissance concernant l'utilisation des auto-encodeurs récurrents pour la détection de scénarios anormaux au sein des bâtiments connectés.

En 2016 ont été publiés, pour la première fois à notre connaissance, des travaux concernant l'utilisation de « tied weights » lors du dropout au sein de cellules récurrentes [Gal and Ghahramani, 2016b].

L'utilisation traditionnelle du dropout au sein des unités récurrentes applique, à chaque timestep, un masque différent aux matrices de poids $\boldsymbol{U}$, $\boldsymbol{V}$ et $\boldsymbol{W}$ décrites dans la section 3.2.2. Le concept de « tied weights » vise à considérer les mêmes masques de dropout à chaque timestep, comme illustré figure 5.1. Ce concept est donc applicable à toutes sortes de cellules récurrentes (LSTM, GRU... cf : partie 3.2.4).

Au sein des travaux de Gal et al. [Gal and Ghahramani, 2016b], ce principe est utilisé directement au sein de RNNs variationnels, visant à produire une approximation bayésienne telle que

décrite dans la partie 4.2.2. Ainsi, nous pouvons tester les fonctions de score que nous proposons chapitre 4 sur des données issues de bâtiments connectés avec un auto-encodeur LSTM.

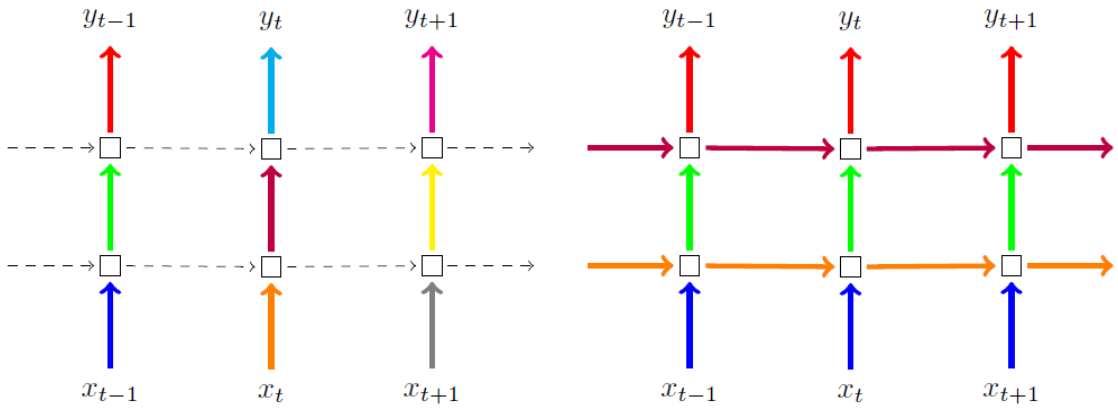

FIGURE 5.1 – Représentation de la technique de dropout suivant l'interprétation bayésienne (à droite) par rapport à la technique standard (à gauche). Cette représentation est tirée de l'article « A Theoretically Grounded Application of Dropout in Recurrent Neural Networks » [Gal and Ghahramani, 2016b]. Chaque carré représente une unité RNN, avec des flèches horizontales représentant la dépendance temporelle (autrement dit, la récurrence des unités). Les flèches verticales représentent l'entrée et la sortie à chaque timestep. Les connexions colorées de manière différente correspondant à différents masques de dropout. Les lignes en pointillés correspondent aux connexions standard sans dropout. Les techniques actuelles (suppression dite « naïve », à gauche) utilisent différents masques à chaque timestep, sans suppression des couches récurrentes. La technique proposée par Gal (Variation RNN, à droite) utilise le même masque de dropout dans le temps, y compris sur les connexions récurrentes.

5.2 Expérimentation

Afin de tester nos méthodes d'optimisation de score dans le contexte de bâtiments connectés, nous avons réalisé une expérimentation assez similaire à celle présentée chapitre 3 sur toutes les maisons du jeu de données REFIT (cf : partie 3.3.1).

5.2.1 Données

Comme nous l'avons mentionné précédemment, cette expérimentation porte non plus sur une unique maison de REFIT mais sur toutes. Ainsi, chaque maison a subi un tri similaire à celui subi par la maison 0 lors de la première expérimentation au niveau des capteurs. Cependant, lors de ce tri, nous avons été plus permissif et avons permis la prise en compte d'une catégorie de capteurs de température spécifique aux radiateurs.

De plus, de même que lors de la première expérimentation, les données ont été fenêtrées, partiellement labélisées, puis enrichies (cf : partie 3.3.1) et centrées-réduites (cf : partie 3.3.1).

Certaines données ont subi un pré-traitement supplémentaire. En effet, parmi les maisons du jeu de données, certaines catégories de capteurs existantes (consommation de gaz, température en surface des radiateurs, température de l'air, humidité...) étaient sur-représentées. Nous considérons que, lorsqu'une catégorie inclut un nombre de capteurs ayant un ordre de grandeur supérieur à la catégorie la moins représentée, alors celle-ci est sur-représentée. Au sein des maisons concernées, les catégories étaient les températures de surface de radiateurs, les températures de l'air, et dans de rare cas, l'humidité.

La conséquence potentielle d'une telle sur-représentation est la perte de sensibilité aux anomalies concernant les catégories moins représentées. Par exemple, si on prend le cas de la catégorie de capteurs de consommation de gaz, celle-ci n'est représentée qu'une seule fois au sein de chaque maison. Si l'on imagine un scénario anormal induisant une hausse anormale de la consommation, comme par exemple une fuite, sans affecter les capteurs des autres catégories, il y a des chances bien plus importantes que ce scénario soit ignoré si les autres catégories sont sur-représentées. En effet, les données anormales seraient « noyées » dans la normalité des données des capteurs non affectés.

Nous proposons donc, pour chaque catégorie sur-représentée, une première compression afin de répartir avec plus d'équitabilité l'importance d'un résultat au sein des différentes catégories de capteurs.

Afin d'étudier la possible réduction de dimension applicable à chaque catégorie et pour chaque maison, nous avons calculé les matrices de corrélations des données.

Ces matrices ont révélé que de nombreux capteurs, au sein d'une même catégorie, présentaient entre eux de très haut taux de corrélations. On peut citer comme exemple, au sein de la maison 4, la catégorie correspondant aux capteurs de température de l'air disséminés dans celle-ci (cf : figure 5.2), ou encore, au sein de la maison 10, la catégorie correspondant à tous les capteurs d'humidité dans l'air (cf : figure 5.3).

Ces informations, concernant les hauts taux de corrélations présents dans les données d'une même catégorie, nous ont permis de nous orienter vers un mode de compression de données linéaire. Nous avons donc utilisé la méthode ACP (cf : section 2.3.1) afin de transformer les catégories sur-représentées en composantes principales. Les taux de corrélations n'étant pas les mêmes pour chaque catégorie et pour chaque maison, nous avons adopté la méthodologie suivante. Pour chaque catégorie :

- Nous commençons par la réduire en une unique composante par la méthode ACP.
- Nous recréons une approximation des données originale en utilisant cette composante et la transformation inverse de la méthode ACP.
- Nous évaluons la perte d'information en comparant les données approximées et les données originales.
- Tant que l'erreur moyenne des données est supérieure à 0.15 (rappelons que les données sont centrées-réduites et ont donc un écart-type de 1), nous répétons les étapes précédentes en incrémentant le nombre de composantes.

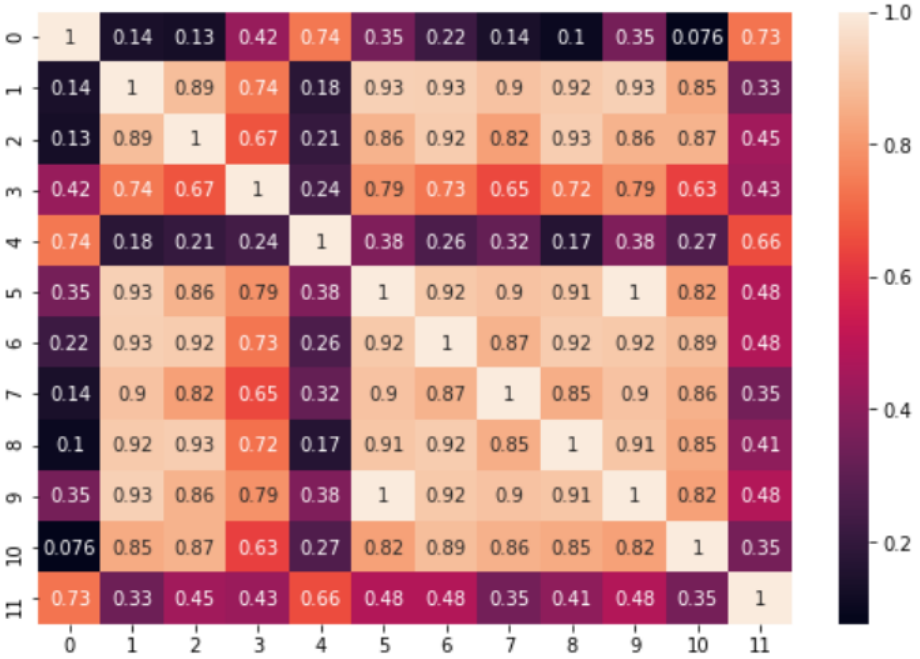

FIGURE 5.2 – Matrice de corrélation superposée à la heatmap correspondante concernant les données des capteurs de la catégorie « température de l'air » dans la maison 4. Chaque ligne et chaque colonne correspondent à un capteur de cette catégorie. On constate entre de nombreux capteurs des taux de corrélation élevés (>0.8).

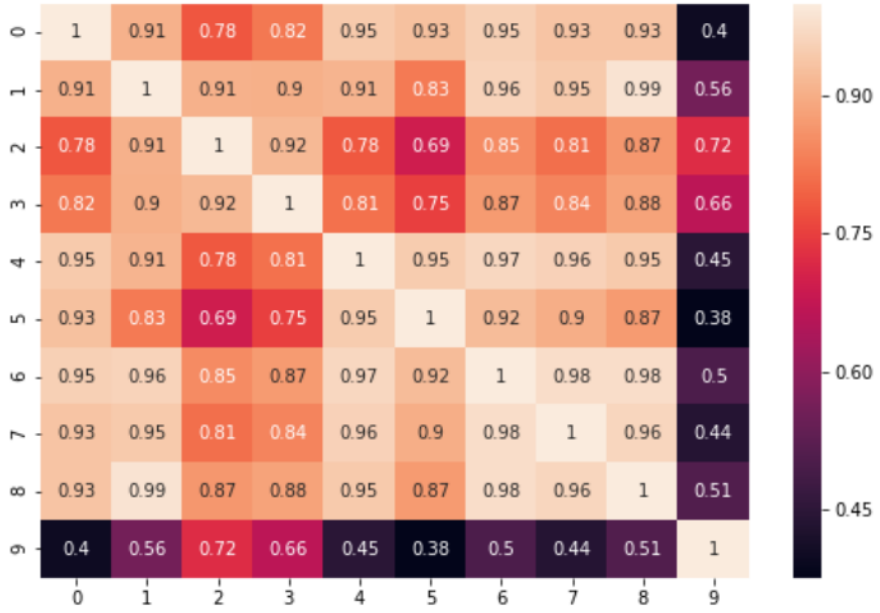

FIGURE 5.3 – Matrice de corrélation superposée à la heatmap correspondante concernant les données des capteurs de la catégorie « humidité » dans la maison 10. On constate entre de nombreux capteurs des taux de corrélation élevés (>0.8).

Ainsi, chaque catégorie est réduite de manière à retenir une quantité d'informations acceptable. Cette méthode nous a permis de réduire les données de $\sim$30% à $\sim$75%, selon le cas traité.

Nous avons donc utilisé, pour chaque maison, des fenêtres de 48 timesteps (l'expérimentation présentée chapitre 3 nous ayant indiqué que les modèles prenant en entrée des fenêtres d'une demie journée semblent présenter les meilleurs résultats pour les RNNs), fois le nombre de capteurs une fois trié, les catégories ayant été compressées si besoin (nous indiquons dans les résultats quelles maisons ont subi ce traitement).

5.2.2 Méthodologie

Encore une fois, nous nous sommes appuyés sur les résultats présentés partie 3.3.3 afin de nous guider et de rendre cette expérimentation réalisable en termes de temps. En effet, déterminer les paramètres optimaux pour un réseau de neurones, comme nous l'avons fait pour les deux catégories d'auto-encodeurs comparés lors la première expérimentation, est une tâche chronophage. Le temps d'entraînement de la multitude de modèles testés a aussi contribué à allonger le procédé. Le temps d'entraînement des RNNs utilisés lors de cette expérimentation est du même ordre que ceux entraînés dans le cadre de la première expérimentation (cf : tableau 3.1 section 3.3.3), soit de l'ordre du millier de secondes (allant jusque $\sim$3000 secondes pour le modèle le plus volumineux). De plus, pour les méthodes que nous proposons, nous avons utilisé un nombre d'itérations fixé à 100 pour le Monte Carlo dropout (de la même manière qu'énoncé section 4.2.3). Ainsi, même si l'évaluation classique d'un individu du jeu de donnée de test est usuellement rapide (de l'ordre de quelques secondes, étant donné la complexité et la taille de nos modèles), ce temps d'évaluation a été multiplié par 100 pour les deux fonctions de score que nous proposons et pour la fonction basée sur l'approximation bayésienne.

Grâce aux résultats issus de la première expérimentation, nous avons pu drastiquement diminuer le nombre de modèles à tester afin de nous concentrer sur les paramètres les plus intéressants a priori.

Ainsi, nous avons, pour chaque maison, réalisé le test de deux auto-encodeurs récurrents LSTM, l'un compressant les données à 80%, et l'autre à 85%. Afin de déterminer quelle fonction de score était la meilleure, nous avons ré-utilisé les méthodes d'indice de fiabilité et de fiabilité absolue présentées section 3.3.2.

Au cours de notre expérimentation, nous avons constaté que certains scores d'anomalies pouvaient fausser le résultat. En effet, les fonctions de score pondérant les erreurs par l'incertitude de prédiction ont parfois tendance à produire un score très supérieur aux fonctions de score classiques malgré un faible nombre d'anomalies annotées (exemple figure 5.4), et ce malgré un terme de régulation (cf : terme reg dans la formule 3.14 présentée section 3.3.2) basé sur la moyenne de toutes les erreurs.

Nous avons donc effectué deux comparaisons. L'une en considérant, pour chaque fonction de score et pour chaque modèle donné, tous les scores d'anomalies disponibles, et l'autre ne considérant que les scores d'anomalies compris dans l'écart type des scores d'anomalies du cas courant.

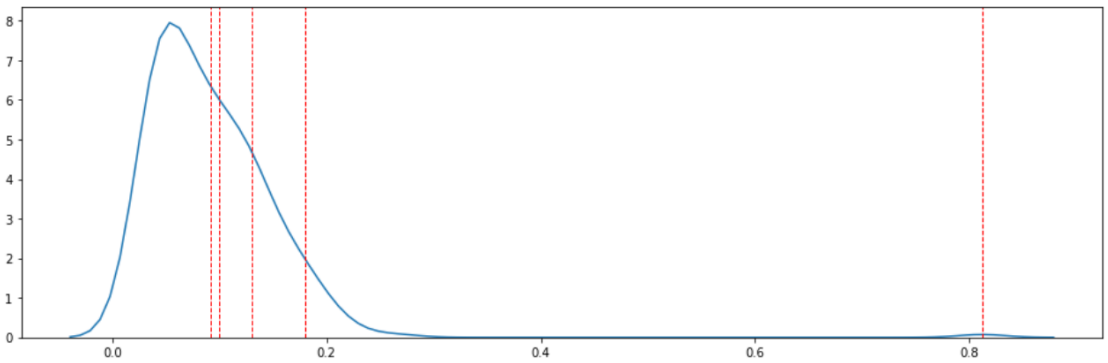

FIGURE 5.4 – Distribution des scores classiques (S.) de la maison 6 et placement des anomalies. On constate une anomalie très excentrée, augmentant significativement les indices de mesures et biaisant la comparaison des fonctions de score.

5.2.3 Résultats

Tous nos résultats peuvent être trouvés dans les tableaux 5.1 et 5.2. Lorsque l'on compare les fonctions avec tous les scores d'anomalies disponibles, on constate que les fonctions de score proposées (W.S. et W.B.A., cf : section 4.2.3) produisent systématiquement les indices les plus élevés :

- W.S. - I.F. meilleur dans $\sim$47.3% des cas, I.F.A. meilleur dans $\sim$50%
- W.B.A. - I.F. meilleur dans $\sim$52.7% des cas, I.F.A. meilleur dans $\sim$50%.

Lorsque l'on ignore les scores extrêmes lors de l'évaluation de chaque fonction, les résultats sont différents :

- S. - I.F. meilleur dans $\sim$7.89% des cas, I.F.A. meilleur dans $\sim$7.89%
- B.A. - I.F. meilleur dans $\sim$21.05% des cas, I.F.A. meilleur dans $\sim$18.4%
- W.S. - I.F. meilleur dans $\sim$34.4% des cas, I.F.A. meilleur dans $\sim$31.58%
- W.B.A. - I.F. meilleur dans $\sim$39.5% des cas (dont un cas à égalité avec B.A.), I.F.A. meilleur dans $\sim$42.1%.

Nous constatons tout d'abord que les fonctions classiques (S. et B.A.) présentent dans certains cas des indices plus élevés que les fonctions proposées, bien que celles-ci soient encore les meilleures dans la plupart des cas.

TABLE 5.1 – Evaluation des différentes fonctions de score selon l'indice de fiabilité (I.F.) et l'indice de fiabilité absolue (I.F.A.). Les indices en gras sont les meilleurs I.F. et I.F.A. pour une maison et pour un taux de réduction donné. Les maisons annotées (ACP) sont les maisons comprenant des catégories de capteurs ayant subi une compression linéaire avant la mise en place de l'auto-encodeur (les nom des maisons servent à faire la correspondance avec le tableau suivant).

Maison	% réduction de dimension	S.		B.A.		W. S.		W.B.A.	
		I. F.	I. F. A.	I. F.	I. F. A.	I. F.	I. F. A.	I. F.	I. F. A.
house_1	85	0.594	0.328	0.644	0.442	0.693	0.409	**0.745**	**0.518**
	80	0.544	0.204	0.589	0.352	0.63	0.258	**0.671**	**0.421**
house_2	85	1.614	0.319	1.539	0.357	**4.578**	0.901	4.446	**1.006**
	80	1.754	0.397	1.657	0.327	**4.87**	**1.035**	4.687	0.86
house_3	85	3.759	0.997	3.818	0.706	9.506	**2.756**	**9.587**	1.974
	80	3.222	0.136	3.195	0.283	**7.628**	0.418	7.564	**0.796**
house_4 (ACP)	85	0.68	0.339	0.602	0.242	**0.842**	**0.407**	0.747	0.303
	80	0.589	0.209	0.55	0.185	**0.706**	**0.246**	0.648	0.221
house_5 (ACP)	85	0.356	0.281	0.397	0.269	0.403	**0.314**	**0.446**	0.303
	80	0.371	0.332	0.424	0.324	0.419	**0.367**	**0.473**	0.355
house_6 (ACP)	85	0.416	0.349	0.429	0.326	0.481	**0.39**	**0.49**	0.37
	80	0.433	0.343	0.449	0.372	0.5	0.4	**0.508**	**0.413**
house_7	85	1.414	1.012	1.384	0.961	**2.188**	**1.555**	2.138	1.485
	80	1.622	1.051	1.553	0.968	**2.512**	**1.647**	2.407	1.497
house_8	85	0.269	0.036	0.288	0.043	0.317	0.042	**0.336**	**0.053**
	80	0.379	0.042	0.28	0.036	**0.425**	**0.047**	0.324	0.041
house_9	85	3.029	2.354	3.013	2.311	5.893	**4.579**	**5.964**	4.523
	80	2.636	2.1	2.673	2.163	4.792	3.858	**4.829**	**3.908**
house_10 (ACP)	85	0.33	0.089	0.387	0.106	0.387	0.099	**0.448**	**0.12**
	80	0.255	0.038	0.292	0.055	0.298	0.042	**0.335**	**0.063**
house_11	85	0.723	0.577	0.686	0.537	**0.829**	**0.665**	0.789	0.617
	80	0.526	0.409	0.577	0.411	0.577	0.447	**0.64**	**0.457**
house_12	85	0.5	0.259	0.459	0.23	**0.561**	**0.283**	0.517	0.262
	80	0.557	0.373	0.504	0.336	**0.609**	**0.403**	0.559	0.365
house_13	85	2.595	2.103	2.486	1.993	**3.112**	**2.516**	2.992	2.412
	80	2.964	2.453	2.788	2.307	**3.492**	**2.878**	3.307	2.717
house_14 (ACP)	85	0.294	0.103	0.315	0.104	0.369	0.128	**0.388**	**0.13**
	80	0.215	0.094	0.233	0.102	0.254	0.11	**0.277**	**0.119**
house_15	85	0.242	0.192	0.261	0.199	0.272	0.214	**0.289**	**0.221**
	80	0.225	0.202	0.251	0.212	0.245	0.221	**0.273**	**0.232**
house_16 (ACP)	85	0.608	0.182	0.534	0.201	**0.689**	0.209	0.609	**0.253**
	80	0.574	0.247	0.565	0.227	**0.645**	**0.281**	0.632	0.259
house_17	85	0.2	0.007	0.211	0.008	0.243	0.008	**0.25**	**0.009**
	80	0.124	0.004	0.176	0.006	0.153	0.005	**0.208**	**0.007**
house_18	85	2.185	1.172	2.189	1.076	4.117	**2.244**	**4.124**	2.027
	80	2.26	0.972	2.182	0.904	**3.95**	**1.698**	3.817	1.604
house_19	85	0.501	0.275	0.491	0.322	**0.568**	0.321	0.558	**0.368**
	80	0.464	0.173	0.455	0.2	**0.514**	0.191	0.506	**0.222**

TABLE 5.2 – Evaluation des différentes fonctions de score selon l'indice de fiabilité (I.F.) et l'indice de fiabilité absolue (I.F.A.). Ce tableau correspond à une évaluation excluant, pour chaque fonction de score, les scores d'anomalies dépassant l'écart type du cas courant.

Maison	% réduction de dimension	S.		B.A.		W. S.		W.B.A.	
		I. F.	I. F. A.	I. F.	I. F. A.	I. F.	I. F. A.	I. F.	I. F. A.
house_1	85	0.646	0.56	0.639	0.573	**0.743**	0.651	0.733	**0.657**
	80	0.56	0.477	0.616	0.533	0.651	0.545	**0.695**	**0.599**
house_2	85	**0.133**	0.032	0.132	**0.037**	-0.088	-0.021	-0.077	-0.021
	80	0.101	**0.029**	**0.102**	0.026	-0.125	-0.034	-0.113	-0.026
house_3	85	**0.298**	**0.081**	0.293	0.056	-0.194	-0.057	-0.202	-0.042
	80	0.3	0.013	**0.319**	**0.029**	-0.054	-0.003	-0.035	-0.004
house_4 (ACP)	85	0.452	0.294	0.532	0.359	0.561	0.282	**0.617**	**0.423**
	80	0.421	0.265	0.438	0.302	**0.589**	**0.359**	0.489	0.34
house_5 (ACP)	85	0.199	0.178	0.238	0.217	0.221	0.195	**0.262**	**0.234**
	80	0.187	0.184	0.271	0.261	0.213	0.206	**0.299**	**0.287**
house_6 (ACP)	85	0.278	0.246	0.381	0.337	0.306	0.263	**0.424**	**0.369**
	80	0.239	0.204	0.281	0.252	0.256	0.224	**0.299**	**0.263**
house_7	85	0.263	0.22	0.273	0.223	0.264	0.221	**0.279**	**0.227**
	80	0.256	0.202	**0.262**	0.199	0.255	**0.203**	**0.262**	0.198
house_8	85	0.522	0.497	0.535	0.484	0.596	**0.56**	**0.607**	0.55
	80	0.672	0.649	0.536	0.513	**0.742**	**0.717**	0.599	0.568
house_9	85	0.232	0.185	**0.239**	**0.189**	0.066	0.052	0.087	0.068
	80	0.225	0.183	**0.25**	**0.205**	0.111	0.091	0.129	0.106
house_10 (ACP)	85	0.134	0.058	**0.249**	**0.089**	0.15	0.064	0.231	0.067
	80	0.022	0.005	**0.113**	**0.037**	0.08	0.014	0.057	0.015
house_11	85	0.497	0.41	0.429	0.371	**0.527**	**0.442**	0.47	0.4
	80	0.376	0.337	0.466	0.402	0.45	0.397	**0.499**	**0.432**
house_12	85	0.508	0.418	0.501	0.424	**0.567**	0.464	0.566	**0.472**
	80	0.544	0.499	0.5	0.439	**0.596**	**0.546**	0.548	0.475
house_13	85	2.237	2.1	2.147	2.011	**2.494**	**2.341**	2.408	2.256
	80	2.563	2.406	2.401	2.259	**2.799**	**2.633**	2.657	2.494
house_14 (ACP)	85	0.23	0.151	0.241	0.181	0.255	0.181	**0.266**	**0.206**
	80	0.181	0.136	0.188	0.147	0.196	0.15	**0.203**	**0.166**
house_15	85	0.264	0.258	0.249	0.205	**0.297**	**0.294**	0.273	0.227
	80	0.216	0.215	0.302	0.289	0.248	0.242	**0.345**	**0.32**
house_16 (ACP)	85	0.603	0.428	0.717	0.631	0.678	0.483	**0.81**	**0.715**
	80	0.651	0.581	0.703	0.539	**0.727**	**0.648**	0.703	0.547
house_17	85	0.3	0.2	0.235	0.049	**0.336**	**0.222**	0.263	0.068
	80	0.285	0.209	0.324	0.244	0.313	0.226	**0.357**	**0.27**
house_18	85	**-0.005**	**-0.004**	-0.011	-0.007	-0.045	-0.032	-0.049	-0.031
	80	-0.022	-0.013	**-0.019**	**-0.01**	-0.057	-0.033	-0.053	-0.03
house_19	85	0.646	0.57	0.588	0.544	**0.698**	**0.62**	0.668	0.618
	80	0.588	0.476	0.552	0.486	**0.653**	0.542	0.613	**0.548**

TABLE 5.3 – Moyennes des différences des évaluations des fonctions avant et après filtre des scores d'anomalies excentrés pour chaque indice.

Fonction de score	S.		B.A.		W.S.		W.B.A.	
Indice	I. F.	I. F. A.	I. F.	I. F. A.	I. F.	I. F. A.	I. F.	I. F. A.
Différence	0.677	0.372	0.656	0.345	1.442	0.68	1.422	0.632

Comme nous le montrons dans le tableau 5.3, les fonctions de scores proposées sont significativement plus affectées que les fonctions classiques lorsque l'on filtre les scores d'anomalies excentrés avant l'évaluation.

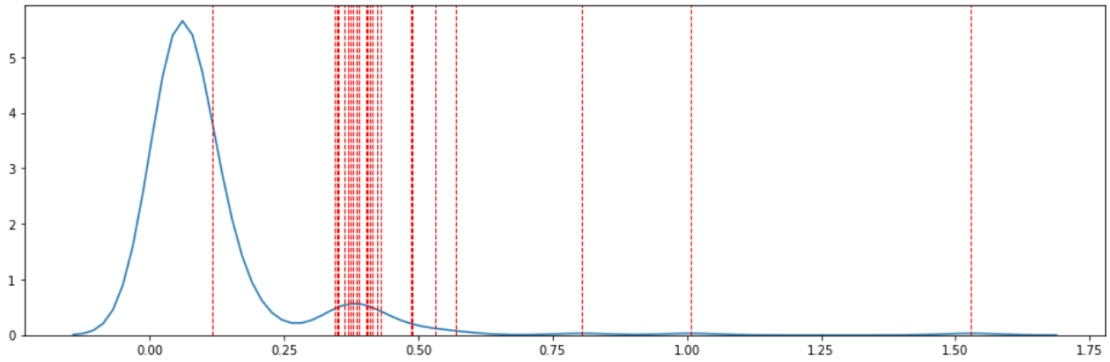

FIGURE 5.5 – Distribution des scores de la fonction W.B.A. de la maison 13 et placement des anomalies. On constate une très nette concentration des anomalies annotées en retrait des données saines.

5.2.4 Discussion

Le fait que les fonctions proposées soient plus affectées que les fonctions classiques par un filtre des scores d'anomalies excentrés suggère qu'une des particularités de nos fonctions soit d'augmenter de manière significative les scores des anomalies, en particulier lorsque celles-ci sont flagrantes de base (cf : figure 5.4).

De plus, malgré le filtre de ces scores, nos fonctions produisent les meilleurs scores dans la plupart des cas. Ce résultat nous montre que, même dans le cas de RNNs variationnels, et en particulier lorsqu'ils sont utilisés pour traiter les données issues de bâtiments connectés, nos fonctions ont un réel intérêt.

Notre expérimentation présente néanmoins certaines limites. En effet, la labélisation de 120 jours pour chaque maison est une tâche fastidieuse sensible à l'erreur qui devrait être confiée, dans l'idéal, à un expert du domaine. De plus, certaines maisons semblent quasiment dépourvues de scénarios anormaux. Nous avons donc cherché à labéliser en tant qu'anomalies des fenêtres seulement légèrement différentes des autres, augmentant sensiblement leurs chances d'être considérées comme normales par nos modèles. C'est un des facteurs expliquant les mauvaises performances de certains cas (notamment des indices négatifs pour la maison 2, 3 et 18 tableau 5.2). Un autre facteur est aussi la sous-représentation de certaines données : bien que

nous ayons réduit au préalable les dimensions des catégories sur-représentées, les anomalies reflétées par un ou deux capteurs ont un faible impact et tendent à être ignorées.

L'absence de jeux de données complètement labélisées a aussi été un obstacle à l'utilisation de technique d'évaluation classique (comme le calcul de l'aire en dessous de la courbe ROC utilisée lors de la deuxième expérimentation, cf : partie 4.3.3). Nous n'avons donc pas pu évaluer les modèles au sens strict, seulement étudier des comparaisons de scores issus de réseaux de neurones.

Notre approche permet néanmoins d'avoir une idée sur la capacité des méthodes que nous proposons à discriminer les données saines des anomalies lorsque celle-ci ont pu être labélisées visuellement sans hésitation. Nous prenons l'exemple illustré par la figure 5.5 : on constate que la fonction W.B.A. a été particulièrement efficace dans le placement des scores d'anomalies pour la maison 13 (on ne constate qu'un seul score d'anomalie manifestement mal placé). Etant donné que ces scores correspondent à un échantillon d'anomalies (établi afin qu'il soit, dans la mesure du possible, représentatif), on peut supposer que les performances concrètes du modèle associé à cette fonction de score soient correctes. De plus, la capacité de nos fonctions à produire des scores élevés en cas d'anomalies très manifestes nous permet de leur accorder une confiance plus élevée en termes de faux négatifs.

CHAPITRE 6

CONCLUSION

Au cours de cette thèse, nous nous sommes intéressés aux traitements des scénarios anormaux au sein des bâtiments connectés. Nous nous sommes particulièrement concentrés sur la détection de ceux-ci au travers des données issues des objets connectés ayant une capacité de détection.

Dans ce contexte, nous avons investigué les moyens et techniques présents dans la littérature afin de contribuer à la mise en place d'un éventuel futur système générique de détection de scénarios anormaux. Le but d'un tel système serait de limiter le plus possible dans le temps les dégâts occasionnés par ces scénarios.

Nous avons exposé nos réflexions quant au choix de la technique de base à considérer. Tenant compte du nombre considérable d'objets connectés à venir et la réduction progressive du coût des objets connectés, nous avons fait le choix de nous concentrer sur les techniques de réduction de dimensions. Nous avons souligné les particularités des données de bâtiments et les particularités des anomalies pouvant apparaître dans celles-ci. Ainsi, nous avons choisi de nous orienter vers les réseaux de neurones auto-encodeurs. Une première validation a été réalisée dans le cadre d'une première expérimentation à moindre échelle.

Nous avons étudié les performances d'auto-encodeurs récurrents et d'auto-encodeurs à convolutions, deux architectures étant apparues dans la littérature au cours de cette thèse, mais n'ayant jusque là pas encore été comparées (chapitre 3). Ces deux techniques ont donc été appliquées à un jeu de données de maison connectée correspondant à 4 mois d'acquisition de données (jeu de données REFIT), ce qui nous a permis de montrer que, malgré des temps d'entraînements sensiblement plus élevés, les auto-encodeurs récurrents semblaient plus adaptés. Cette expérimentation a été l'occasion de proposer une méthode de comparaison d'auto-encodeurs traduisant les facteurs visuels que nous utilisons afin d'évaluer le placement de scores d'anomalies sur une distribution globale.

Le temps d'entraînement des nombreux modèles réalisés lors de ces travaux a été une des difficultés majeures rencontrées. On peut donc citer, parmi les perspectives de recherche, tout travail concernant l'optimisation des réseaux de neurones artificiels, et en particulier l'optimisation de l'algorithme de rétro-propagation.

Une proposition d'amélioration des méthodes classiques a ensuite été proposée et corroborée par des études empiriques. Cette proposition d'amélioration inclut un facteur basé sur l'incertitude dans les réseaux de neurones, une notion que des travaux ont rendu utilisable en 2016. Une étude a été réalisée, appliquant nos méthodes à des auto-encodeurs standards sur divers jeux de données et dans des contextes variés (chapitre 4). Bien que l'expérimentation ne nous ait pas permis de déterminer quelle fonction serait la meilleure a priori, nous avons obtenu des résultats prometteurs et avons souligné que l'implémentation de notre méthode était peu coûteuse en termes de temps de développement et de temps de traitement. Les résultats de cette expérimentation nous ont poussé à étendre cette étude à notre cas d'utilisation initial (chapitre 5), les données de bâtiments connectés.

Dans ce contexte, nous avons pu tirer parti des informations recueillies lors de notre première expérimentation et orienter notre méthodologie. Nous avons donc comparé les méthodes que nous proposons aux méthodes classiques sur un nombre conséquent de maisons connectées. Bien que les résultats obtenus aient renforcé notre volonté d'encourager les utilisateurs à prendre en compte les méthodes que nous proposons, cette expérimentation a souligné certaines limites. En effet, l'hétérogénéité et la répartition des données au sein des catégories existantes sont entièrement soumises aux capteurs placés dans les maisons. Ainsi, nous avons rencontré dans le jeu de données REFIT des maisons présentant des catégories de capteurs complètement sur-représentées par rapport à d'autres. Nous avons donc souligné que les données de capteurs au sein d'une même catégorie affichent souvent de très hauts taux de corrélation et sont donc susceptibles d'être réduites linéairement. Néanmoins, ce phénomène de sur-représentation des catégories reste un axe sujet à de nombreuses améliorations.

L'établissement d'un jeu de données aussi complet que REFIT comprenant une labélisation complète des anomalies, idéalement des différents types d'anomalies, permettrait des investigations plus complètes. On peut notamment imaginer avec beaucoup moins de difficulté un état des lieux comparant toutes les techniques de détection d'anomalies au sein de séries temporelles multi-variées appliquées à l'IoT.

En outre, étant donné les résultats des méthodes proposées face aux méthodes classiques, l'application de celles-ci à d'autres cas d'utilisation que l'IoT constitue une direction de recherche pertinente.

Globalement, les thématiques de cette thèse touchent des domaines qui sont, de nos jours, en évolution rapide (l'IoT et le traitement des données issues de bâtiments, apprentissage automatique et réseaux de neurones...). Les perspectives évoluent donc au fur et à mesure des nouvelles avancées dans ces domaines.

BIBLIOGRAPHIE

[Abadi et al., 2016] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I. J., Harp, A., Irving, G., Isard, M., Jia, Y., Józefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D. G., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P. A., Vanhoucke, V., Vasudevan, V., Viégas, F. B., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2016). Tensorflow : Large-scale machine learning on heterogeneous distributed systems. *CoRR*, abs/1603.04467.

[Abbas, 2019] Abbas, J. (2019). *https://fr.123rf.com/profile_joshyabb*.

[Akbar et al., 2015] Akbar, A., Carrez, F., Moessner, K., and Zoha, A. (2015). Predicting complex events for pro-active iot applications. *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*, pages 327–332.

[Alberini et al., 2014] Alberini, C. M., Baxter, D. A., Bean, A. J., Beierlein, M., Brady, S., Brophy, P., Byrne, J. H., Colman, D. R., DeFelipe, J., Deutch, A. Y., Dienel, G. A., Elder, G. A., Gireud, M., Giuffrida, A., Heidelberger, R., Jan, L. Y., Jan, Y. N., Hof, P. R., Kaeser, P. S., Kidd, G., Klann, E., Knierim, J. J., Kullmann, D. M., LaBar, K. S., LeDoux, J. E., Lipscombe, D., McCormick, D. A., Nicholson, B., Roberts, J. L., Roth, R. H., Sáez, J. C., Schafe, G. E., Schulman, H., Shouval, H., Sirisaengtaksin, N., Smolen, P. D., Sosa, M. A. G., Thompson, R. F., Tien, J., Toro, C. P., Trapp, B. D., Young, D. M., Waxham, M. N., Zucker, R. S., and de Vellis, J. (2014). From molecules to networks (third edition). Academic Press, Boston, third edition edition.

[Aleskerov et al., 1997] Aleskerov, E., Freisleben, B., and Rao, B. (1997). Cardwatch : a neural network based database mining system for credit card fraud detection. *Proceedings of the IEEE/IAFE 1997 Computational Intelligence for Financial Engineering (CIFEr)*, pages 220–226.

[Almotiri et al., 2017] Almotiri, J., Elleithy, K., and Elleithy, A. (2017). Comparison of autoencoder and principal component analysis followed by neural network for e-learning using handwritten recognition. *2017 IEEE Long Island Systems, Applications and Technology Conference (LISAT)*, pages 1–5.

[Araya, 2016] Araya, D. B. (2016). Collective contextual anomaly detection for building energy consumption. *https://ir.lib.uwo.ca/etd/4027*.

[Baldi et al., 1999] Baldi, P., Brunak, S., Frasconi, P., Soda, G., and Pollastri, G. (1999). Exploiting the past and the future in protein secondary structure prediction. *Bioinformatics*, 15(11) :937–946.

[Baum and Haussler, 1989] Baum, E. B. and Haussler, D. (1989). What size net gives valid generalization ? pages 81–90. Morgan-Kaufmann.

[Bellman, 1961] Bellman, R. (1961). *Adaptive Control Processes : A Guided Tour*. Princeton University Press.

[Bengio et al., 2013] Bengio, Y., Courville, A., and Vincent, P. (2013). Representation learning : A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8) :1798–1828.

[Bloomenthal and Shoemake, 1991] Bloomenthal, J. and Shoemake, K. (1991). Convolution surfaces. *ACM SIGGRAPH Computer Graphics*, 25(4) :251–256.

[Blundell et al., 2015] Blundell, C., Cornebise, J., Kavukcuoglu, K., and Wierstra, D. (2015). Weight uncertainty in neural networks. *Proceedings of the 32nd International Conference on Machine Learning*.

[Bourlard and Kamp, 1988] Bourlard, H. and Kamp, Y. (1988). Auto-association by multilayer perceptrons and singular value decomposition. *Biological cybernetics*, 59(4-5) :291–294.

[Breiman, 2001] Breiman, L. (2001). Random forests. *Machine learning*, 45(1) :5–32.

[Breunig et al., 2000] Breunig, M. M., Kriegel, H.-P., Ng, R. T., and Sander, J. (2000). Lof : identifying density-based local outliers. *ACM sigmod record*, 29(2) :93–104.

[Chandola et al., 2009] Chandola, V., Banerjee, A., and Kumar, V. (2009). Anomaly detection : A survey. *ACM computing surveys (CSUR)*, 41(3) :15.

[Chapman and Jain, 1995] Chapman, D. and Jain, A. (1995). Musk (version 2) data set. *https://archive.ics.uci.edu/ml/datasets/Musk+(Version+2)*.

[Chatfield et al., 2014] Chatfield, K., Simonyan, K., Vedaldi, A., and Zisserman, A. (2014). Return of the devil in the details : Delving deep into convolutional nets. *British Machine Vision Conference*.

[Chellapilla et al., 2006] Chellapilla, K., Puri, S., and Simard, P. (2006). High performance convolutional neural networks for document processing. *Tenth International Workshop on Frontiers in Handwriting Recognition*.

[Chen, 2012] Chen, Y.-K. (2012). Challenges and opportunities of internet of things. *17th Asia and South Pacific design automation conference*, pages 383–388.

[Cho et al., 2014] Cho, K., van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using RNN encoder–decoder for statistical machine translation. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734.

[Chung et al., 2014] Chung, J., Gulcehre, C., Cho, K., and Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *NIPS 2014 Workshop on Deep Learning, December 2014*.

[Ciresan et al., 2011] Ciresan, D. C., Meier, U., Masci, J., Gambardella, L. M., and Schmidhuber, J. (2011). Flexible, high performance convolutional neural networks for image classification. *Twenty-Second International Joint Conference on Artificial Intelligence*.

[Cole et al., 1988] Cole, R., Fanty, M., and Dietterich, T. (1988). Isolet data set. *https://archive.ics.uci.edu/ml/datasets/isolet*.

[Collobert et al., 2002] Collobert, R., Bengio, S., and Marithoz, J. (2002). Torch : A modular machine learning software library.

[Cook, 2013] Cook, S. (2013). *CUDA Programming : A Developer's Guide to Parallel Computing with GPUs*. Morgan Kaufmann Publishers Inc.

[Cortes and Vapnik, 1995] Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine learning*, 20(3) :273–297.

[Dietterich, 2000] Dietterich, T. G. (2000). Ensemble methods in machine learning. *International workshop on multiple classifier systems*, pages 1–15.

[Ding et al., 2008] Ding, H., Trajcevski, G., Scheuermann, P., Wang, X., and Keogh, E. (2008). Querying and mining of time series data : Experimental comparison of representations and distance measures. *Proc. VLDB Endow.*, 1(2) :1542–1552.

[Dua and Graff, 2017] Dua, D. and Graff, C. (2017). UCI machine learning repository. *http://archive.ics.uci.edu/ml*.

[Duchi et al., 2011] Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul) :2121–2159.

[Estivill-Castro, 2002] Estivill-Castro, V. (2002). Why so many clustering algorithms : A position paper. *SIGKDD Explor. Newsl.*, 4(1) :65–75.

[Fan et al., 2018] Fan, C., Xiao, F., Zhao, Y., and Wang, J. (2018). Analytical investigation of autoencoder-based methods for unsupervised anomaly detection in building energy data. *Applied energy*, 211 :1123–1135.

[Firth et al., 2017] Firth, S., Kane, T., Dimitriou, V., Hassan, T., Fouchal, F., Coleman, M., and Webb, L. (2017). REFIT Smart Home dataset. *https://figshare.com/articles/REFIT_Smart_Home_dataset/2070091*.

[Fortunato et al., 2017] Fortunato, M., Blundell, C., , and Vinyals, O. (2017). Bayesian recurrent neural networks. *arXiv preprint arXiv :1704.02798*.

[Gal, 2016] Gal, Y. (2016). *Uncertainty in deep learning*. PhD dissertation, University of Cambridge.

[Gal and Ghahramani, 2016a] Gal, Y. and Ghahramani, Z. (2016a). Dropout as a bayesian approximation : Representing model uncertainty in deep learning. *international conference on machine learning*, pages 1050–1059.

[Gal and Ghahramani, 2016b] Gal, Y. and Ghahramani, Z. (2016b). A theoretically grounded application of dropout in recurrent neural networks. *Advances in neural information processing systems*, pages 1019–1027.

[Gartner, 2017] Gartner (2017). Gartner says 8.4 billion connected "things" will be in use in 2017, up 31 percent from 2016. *https://www.gartner.com/en/newsroom/press-releases/2017-02-07-gartner-says-8-billion-connected-things-will-be-in-use-in-2017-up-31-percent-from-2016*.

[Gartner, 2018] Gartner (2018). *Gartner Symposium/ITxpo 2018 November 4-8 in Barcelona, Spain*.

[Gers et al., 2000] Gers, F. A., Schmidhuber, J. A., and Cummins, F. A. (2000). Learning to forget : Continual prediction with lstm. *Neural Comput.*, 12(10) :2451–2471.

[Glorot and Bengio, 2010] Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256.

[Goix, 2016] Goix, N. (2016). How to Evaluate the Quality of Unsupervised Anomaly Detection Algorithms ? *ICML Anomaly Detection Workshop*.

[Goix et al., 2015] Goix, N., Sabourin, A., and Clémençon, S. (2015). On anomaly ranking and excess-mass curves. *Artificial Intelligence and Statistics*, pages 287–295.

[Goodfellow et al., 2016a] Goodfellow, I., Bengio, Y., and Courville, A. (2016a). *Deep Learning*, chapter Autoencoders. The MIT Press.

[Goodfellow et al., 2016b] Goodfellow, I., Bengio, Y., and Courville, A. (2016b). *Deep learning*, chapter Convolutional Networks. MIT press.

[Goodfellow et al., 2016c] Goodfellow, I., Bengio, Y., and Courville, A. (2016c). *Deep learning*, chapter Sequence Modeling : Recurrent and Recursive Nets. MIT press.

[Graves, 2012] Graves, A. (2012). *Supervised Sequence Labelling with Recurrent Neural Networks*, chapter Long Short-Term Memory. The MIT Press.

[Graves et al., 2008] Graves, A., Liwicki, M., Fernández, S., Bertolami, R., Bunke, H., and Schmidhuber, J. (2008). A novel connectionist system for unconstrained handwriting recognition. *IEEE transactions on pattern analysis and machine intelligence*, 31(5) :855–868.

[Graves and Schmidhuber, 2005] Graves, A. and Schmidhuber, J. (2005). Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural networks*, 18(5-6) :602–610.

[Gubbi et al., 2013] Gubbi, J., Buyya, R., Marusic, S., and Palaniswami, M. (2013). Internet of things (iot) : A vision, architectural elements, and future directions. *Future generation computer systems*, 29(7) :1645–1660.

[Hahnloser et al., 2000] Hahnloser, R. H., Sarpeshkar, R., Mahowald, M. A., Douglas, R. J., and Seung, H. S. (2000). Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. *Nature*, 405(6789) :947.

[Hanley and McNeil, 1982] Hanley, J. A. and McNeil, B. J. (1982). The meaning and use of the area under a receiver operating characteristic (roc) curve. *Radiology*, 143(1) :29–36.

[Haussler, 1999] Haussler, D. (1999). Convolution kernels on discrete structures. Technical report, Technical report, Department of Computer Science, University of California ....

[Haveliwala, 2003] Haveliwala, T. H. (2003). Topic-sensitive pagerank : A context-sensitive ranking algorithm for web search. *IEEE transactions on knowledge and data engineering*, 15(4) :784–796.

[Hernandez and Reiff-Marganiec, 2014] Hernandez, M. E. P. and Reiff-Marganiec, S. (2014). Classifying smart objects using capabilities. *2014 International Conference on Smart Computing*, pages 309–316.

[Hernández-Lobato and Adams, 2014] Hernández-Lobato, J. M. and Adams, R. P. (2014). Probabilistic backpropagation for scalable learning of bayesian neural networks. *Proceedings of the 32nd International Conference on Machine Learning*.

[Hernández-Lobato et al., 2016] Hernández-Lobato, J. M., Li, Y., Rowland, M., Hernández-Lobato, D., Bui, T. D., and Turner, R. E. (2016). Black-box α-divergence minimization. *Proceedings of the 33rd International Conference on Machine Learning*.

[Herzog and Ostwald, 2013] Herzog, S. and Ostwald, D. (2013). Experimental biology : sometimes bayesian statistics are better. *Nature*, 494(7435) :35.

[Hill and Lewicki, 2007] Hill, T. and Lewicki, P. (2007). Statistics methods and applications. *statsoft*.

[Hinton et al., 2012a] Hinton, G., Deng, L., Yu, D., Dahl, G., Mohamed, A.-r., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Kingsbury, B., et al. (2012a). Deep neural networks for acoustic modeling in speech recognition. *IEEE Signal processing magazine*, 29.

[Hinton et al., 2012b] Hinton, G., Srivastava, N., and Swersky, K. (2012b). Neural networks for machine learning - lecture 6a : Overview of mini-batch gradient descent. *Cited on*, 14 :8.

[Hinton, 1990] Hinton, G. E. (1990). Preface to the special issue on connectionist symbol processing. *Artificial Intelligence*, 46(1-2) :1–4.

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

[Holden et al., 2015] Holden, D., Saito, J., Komura, T., and Joyce, T. (2015). Learning motion manifolds with convolutional autoencoders. *SIGGRAPH Asia 2015 Technical Briefs*, page 18.

[Huang et al., 2011] Huang, G.-B., Zhou, H., Ding, X., and Zhang, R. (2011). Extreme learning machine for regression and multiclass classification. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 42(2) :513–529.

[Hush and Horne, 1993] Hush, D. R. and Horne, B. G. (1993). Progress in supervised neural networks. *IEEE signal processing magazine*, 10(1) :8–39.

[Ioannou, 2018] Ioannou, Y. (2018). Learning with backpropagation. *https://blog.yani.io/sgd/*.

[Johnson, 1967] Johnson, S. C. (1967). Hierarchical clustering schemes. *Psychometrika*, 32(3) :241–254.

[Khan et al., 2012] Khan, R., Khan, S. U., Zaheer, R., and Khan, S. (2012). Future internet : the internet of things architecture, possible applications and key challenges. *2012 10th international conference on frontiers of information technology*, pages 257–260.

[Kingma and Welling, 2014] Kingma, D. P. and Welling, M. (2014). Auto-encoding variational bayes. *The International Conference on Learning Representations*.

[Kleene, 1956] Kleene, S. C. (1956). Representation of events in nerve nets and finite automata. pages 3–41. Princeton University Press, Princeton, NJ.

[Knox, 1974] Knox, C. (1974). Cross-correlation functions for a neuronal model. *Biophysical Journal*, 14(8) :567–582.

[Kohonen, 1982] Kohonen, T. (1982). Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, 43(1) :59–69.

[Kramer, 1991] Kramer, M. A. (1991). Nonlinear principal component analysis using autoassociative neural networks. *AIChE Journal*, 37(2) :233–243.

[Krizhevsky et al., 2012a] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012a). Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, pages 1097–1105.

[Krizhevsky et al., 2012b] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012b). Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, pages 1097–1105.

[Kumar, 2005] Kumar, V. (2005). Parallel and distributed computing for cybersecurity. *IEEE Distributed Systems Online*, 6(10).

[Lawrence and Giles, 2000] Lawrence, S. and Giles, C. L. (2000). Overfitting and neural networks : conjugate gradient and backpropagation. *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing : New Challenges and Perspectives for the New Millennium*, 1 :114–119.

[LeCun, 1987] LeCun, Y. (1987). *Modèles connexionnistes de l'apprentissage*. PhD thesis, These de Doctorat, Université Paris 6.

[LeCun et al., 1989] LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4) :541–551.

[Legrand et al., 2018] Legrand, A., Niepceron, B., Cournier, A., and Trannois, H. (2018). Study of autoencoder neural networks for anomaly detection in connected buildings. *2018 IEEE Global Conference on Internet of Things (GCIoT)*, pages 1–5.

[Legrand et al., 2019] Legrand, A., Trannois, H., and Cournier, A. (2019). Use of uncertainty with autoencoder neural networks for anomaly detection. *2019 IEEE Second International Conference on Artificial Intelligence and Knowledge Engineering (AIKE)*, pages 32–35.

[Li et al., 2015] Li, S., Da Xu, L., and Zhao, S. (2015). The internet of things : a survey. *Information Systems Frontiers*, 17(2) :243–259.

[Li and Gal, 2017] Li, Y. and Gal, Y. (2017). Dropout inference in bayesian neural networks with alpha-divergences. *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2052–2061.

[Liew, 2019] Liew, L. (2019). What is overfitting in trading ? *https://algotrading101.com/learn/what-is-overfitting-in-trading/*.

[Liu and Shih, 2005] Liu, D.-R. and Shih, Y.-Y. (2005). Integrating ahp and data mining for product recommendation based on customer lifetime value. *Inf. Manage.*, 42(3) :387–400.

[Liu et al., 2008] Liu, F. T., Ting, K. M., and Zhou, Z.-H. (2008). Isolation forest. *2008 Eighth IEEE International Conference on Data Mining*, pages 413–422.

[Liu and Baiocchi, 2016] Liu, X. and Baiocchi, O. (2016). A comparison of the definitions for smart sensors, smart objects and things in iot. *2016 IEEE 7th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, pages 1–4.

[Luo et al., 2013] Luo, H., Carrier, P. L., Courville, A., and Bengio, Y. (2013). Texture modeling with convolutional spike-and-slab rbms and deep extensions. *Artificial Intelligence and Statistics*, pages 415–423.

[MacKay, 1992] MacKay, D. J. (1992). A practical bayesian framework for backpropagation networks. *Neural computation*, 4(3) :448–472.

[Malhotra et al., 2016] Malhotra, P., Ramakrishnan, A., Anand, G., Vig, L., Agarwal, P., and Shroff, G. (2016). LSTM-based Encoder-Decoder for Multi-sensor Anomaly Detection. *ICML Anomaly Detection Workshop*.

[Marchi et al., 2015] Marchi, E., Vesperini, F., Eyben, F., Squartini, S., and Schuller, B. (2015). A novel approach for automatic acoustic novelty detection using a denoising autoencoder with bidirectional lstm neural networks. *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1996–2000.

[Martens, 2010] Martens, J. (2010). Deep learning via hessian-free optimization. *Proceedings of the 27th International Conference on International Conference on Machine Learning*, pages 735–742.

[McCulloch and Pitts, 1943] McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4) :115–133.

[Minerva et al., 2015] Minerva, R., Biru, A., and Rotondi, D. (2015). Towards a definition of the internet of things (iot). *IEEE Internet of Things Initiative*.

[Minsky, 1967] Minsky, M. L. (1967). *Computation : Finite and Infinite Machines*. Prentice-Hall.

[Mitchell, 1997] Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition.

[Mitra et al., 2016] Mitra, P., Ray, R., Chatterjee, R., Basu, R., Saha, P., Raha, S., Barman, R., Patra, S., Biswas, S. S., and Saha, S. (2016). Flood forecasting using internet of things and artificial neural networks. *2016 IEEE 7th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, pages 1–5.

[Mood et al., 1974] Mood, A., Graybill, F., and Boes, D. (1974). *Introduction to the Theory of Statistics*. International Student edition. McGraw-Hill.

[Mrissa et al., 2015] Mrissa, M., Médini, L., Jamont, J., Le Sommer, N., and Laplace, J. (2015). An avatar architecture for the web of things. *IEEE Internet Computing*, 19(2) :30–38.

[Naphade et al., 2011] Naphade, M., Banavar, G., Harrison, C., Paraszczak, J., and Morris, R. (2011). Smarter cities and their innovation challenges. *Computer*, 44(6) :32–39.

[Neal, 1996] Neal, R. M. (1996). *Bayesian Learning for Neural Networks*. Springer-Verlag, Berlin, Heidelberg.

[Nesterov, 1983] Nesterov, Y. E. (1983). A method for solving the convex programming problem with convergence rate o (1/kˆ 2).

[Ning and Wang, 2011] Ning, H. and Wang, Z. (2011). Future internet of things architecture : like mankind neural system or social organization framework ? *IEEE Communications Letters*, 15(4) :461–463.

[Nuzzo, 2014] Nuzzo, R. (2014). Scientific method : statistical errors. *Nature News*, 506(7487) :150.

[Oniga and Sütő, 2014] Oniga, S. and Sütő, J. (2014). Human activity recognition using neural networks. *Proceedings of the 2014 15th International Carpathian Control Conference (ICCC)*, pages 403–406.

[Paisley et al., 2012] Paisley, J., Blei, D. M., and Jordan, M. I. (2012). Variational bayesian inference with stochastic search. *Proceedings of the 29th International Coference on International Conference on Machine Learning*, pages 1363–1370.

[Pearson, 1901] Pearson, K. (1901). Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11) :559–572.

[Pimentel et al., 2014] Pimentel, M. A., Clifton, D. A., Clifton, L., and Tarassenko, L. (2014). A review of novelty detection. *Signal Processing*, 99 :215–249.

[Press and Rybicki, 1989] Press, W. H. and Rybicki, G. B. (1989). Fast algorithm for spectral analysis of unevenly sampled data. *The Astrophysical Journal*, 338 :277–280.

[Pyle, 1999] Pyle, D. (1999). *Data Preparation for Data Mining*. Morgan Kaufmann Publishers Inc.

[Raina et al., 2009] Raina, R., Madhavan, A., and Ng, A. Y. (2009). Large-scale deep unsupervised learning using graphics processors. *Proceedings of the 26th annual international conference on machine learning*, pages 873–880.

[Ritchie Ng, 2019] Ritchie Ng, J. F. (2019). Deep learning wizard. *https://www.deeplearningwizard.com/deep_learning/boosting_models_pytorch/lr_scheduling/*.

[Rosenblatt, 1958] Rosenblatt, F. (1958). The perceptron : a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6) :386.

[Roweis and Saul, 2000] Roweis, S. T. and Saul, L. K. (2000). Nonlinear dimensionality reduction by locally linear embedding. *science*, 290(5500) :2323–2326.

[Rumelhart et al., 1986] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning Representations by Back-propagating Errors. *Nature*, 323(6088) :533–536.

[Russell and Norvig, 2009] Russell, S. and Norvig, P. (2009). *Artificial Intelligence : A Modern Approach*. Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd edition.

[Sakurada and Yairi, 2014] Sakurada, M. and Yairi, T. (2014). Anomaly detection using autoencoders with nonlinear dimensionality reduction. *Proceedings of the MLSDA 2014 2nd Workshop on Machine Learning for Sensory Data Analysis*, page 4.

[Sarkar et al., 2014] Sarkar, C., Nambi, S. N. A. U., Prasad, R. V., and Rahim, A. (2014). A scalable distributed architecture towards unifying iot applications. *2014 IEEE World Forum on Internet of Things (WF-IoT)*, pages 508–513.

[Sarkar et al., 2014] Sarkar, C., SN, A. U. N., Prasad, R. V., Rahim, A., Neisse, R., and Baldini, G. (2014). Diat : A scalable distributed architecture for iot. *IEEE Internet of Things journal*, 2(3) :230–239.

[Schölkopf et al., 1999] Schölkopf, B., Williamson, R., Smola, A., Shawe-Taylor, J., and Platt, J. (1999). Support vector method for novelty detection. *Proceedings of the 12th International Conference on Neural Information Processing Systems*, pages 582–588.

[Schuster and Paliwal, 1997] Schuster, M. and Paliwal, K. K. (1997). Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11) :2673–2681.

[Sejnowski and Rosenberg, 1988] Sejnowski, T. J. and Rosenberg, C. R. (1988). Neurocomputing : Foundations of research. chapter NETtalk : A Parallel Network That Learns to Read Aloud, pages 661–672. MIT Press, Cambridge, MA, USA.

[Sigillito, 1989] Sigillito, V. (1989). Breast cancer wisconsin (diagnostic) data set. *https://archive.ics.uci.edu/ml/datasets/ionosphere*.

[Silva and Almeida, 1990] Silva, F. M. and Almeida, L. B. (1990). Acceleration techniques for the backpropagation algorithm. *European Association for Signal Processing Workshop*, pages 110–119.

[Slate, 1991] Slate, D. J. (1991). Letter recognition data set. *https://archive.ics.uci.edu/ml/datasets/letter+recognition*.

[Smola and Schölkopf, 2004] Smola, A. J. and Schölkopf, B. (2004). A tutorial on support vector regression. *Statistics and computing*, 14(3) :199–222.

[Sola and Sevilla, 1997] Sola, J. and Sevilla, J. (1997). Importance of input data normalization for the application of neural networks to complex industrial problems. *IEEE Transactions on nuclear science*, 44(3) :1464–1468.

[Spence et al., 2001] Spence, C., Parra, L., and Sajda, P. (2001). Detection, synthesis and compression in mammographic image analysis with a hierarchical image probability model. *Proceedings IEEE Workshop on Mathematical Methods in Biomedical Image Analysis (MMBIA 2001)*, pages 3–10.

[Sporns et al., 2005] Sporns, O., Tononi, G., and Kötter, R. (2005). The human connectome : a structural description of the human brain. *PLoS computational biology*, 1(4) :e42.

[Srinivasan, 1993] Srinivasan, A. (1993). Statlog (landsat satellite) data set. *https://archive.ics.uci.edu/ml/datasets/Statlog+%28Landsat+Satellite%29*.

[Srivastava et al., 2014] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout : a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1) :1929–1958.

[Stankovic, 2014] Stankovic, J. A. (2014). Research directions for the internet of things. *IEEE Internet of Things Journal*, 1(1) :3–9.

[Steinkraus et al., 2005] Steinkraus, D., Buck, I., and Simard, P. (2005). Using gpus for machine learning algorithms. *Eighth International Conference on Document Analysis and Recognition (ICDAR'05)*, pages 1115–1120.

[Stutz, 1988] Stutz, J. (1988). Low resolution spectrometer data set. *https://archive.ics.uci.edu/ml/datasets/Low+Resolution+Spectrometer*.

[Sutton and Barto, 2018] Sutton, R. S. and Barto, A. G. (2018). *Reinforcement Learning : An Introduction*. The MIT Press, second edition.

[Swingler, 2012] Swingler, K. (2012). Lecture 2 : Single layer perceptrons. *http://www.cs.stir.ac.uk/courses/ITNP4B/lectures/kms/2-Perceptrons.pdf*.

[Taigman et al., 2014] Taigman, Y., Yang, M., Ranzato, M., and Wolf, L. (2014). Deepface : Closing the gap to human-level performance in face verification. *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1701–1708.

[Theano Development Team, 2016] Theano Development Team (2016). Theano : A Python framework for fast computation of mathematical expressions. *arXiv e-prints*.

[Tolias et al., 2015] Tolias, G., Sicre, R., and Jégou, H. (2015). Particular object retrieval with integral max-pooling of cnn activations. *arXiv preprint arXiv :1511.05879*.

[Van Der Maaten et al., 2009] Van Der Maaten, L., Postma, E., and Van den Herik, J. (2009). Dimensionality reduction : a comparative. *J Mach Learn Res*, 10(66-71) :13.

[Vincent et al., 2008] Vincent, P., Larochelle, H., Bengio, Y., and Manzagol, P.-A. (2008). Extracting and composing robust features with denoising autoencoders. *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103.

[Vincent et al., 2010] Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., and Manzagol, P.-A. (2010). Stacked denoising autoencoders : Learning useful representations in a deep network with a local denoising criterion. *J. Mach. Learn. Res.*, 11 :3371–3408.

[Volpi and Tuia, 2016] Volpi, M. and Tuia, D. (2016). Dense semantic labeling of subdecimeter resolution images with convolutional neural networks. *IEEE Transactions on Geoscience and Remote Sensing*, 55(2) :881–893.

[Werbos, 1974] Werbos, P. J. (1974). *Beyond Regression : New Tools for Prediction and Analysis in the Behavioral Sciences*. PhD thesis, Harvard University.

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

[Widrow and Hoff, 1960] Widrow, B. and Hoff, M. E. (1960). Adaptive switching circuits. *1960 IRE WESCON Convention Record, Part 4*, pages 96–104.

[Wolberg et al., 1995] Wolberg, W. H., Street, W. N., and Mangasarian, O. L. (1995). Breast cancer wisconsin (diagnostic) data set. *https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic)*.

[Xie et al., 2012] Xie, J., Xu, L., and Chen, E. (2012). Image denoising and inpainting with deep neural networks. *Advances in neural information processing systems*, pages 341–349.

[Xu et al., 2015] Xu, D., Ricci, E., Yan, Y., Song, J., and Sebe, N. (2015). Learning deep representations of appearance and motion for anomalous event detection. *Procedings of the British Machine Vision Conference 2015*.

[Zeiler and Fergus, 2013] Zeiler, M. D. and Fergus, R. (2013). Stochastic pooling for regularization of deep convolutional neural networks. *arXiv preprint arXiv :1301.3557*.

[Zhu and Laptev, 2017] Zhu, L. and Laptev, N. (2017). Deep and confident prediction for time series at uber. *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*, pages 103–110.

[Zimek et al., 2013] Zimek, A., Gaudet, M., Campello, R. J., and Sander, J. (2013). Subsampling for efficient and effective unsupervised outlier detection ensembles. *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 428–436.

Détection, Anticipation, Action face aux risques dans les bâtiments connectés

RESUME : Cette thèse vise à exploiter la future masse de données qui émergera du nombre conséquent d'objets connectés à venir. Concentrés sur les données de bâtiments connectés, ces travaux ont pour but de contribuer à un système générique de détection d'anomalies. La première année fut consacrée à définir la problématique, le contexte et à recenser les modèles candidats. La piste des réseaux de neurones auto-encodeurs a été privilégiée et justifiée par une première expérimentation. Une deuxième expérience plus conséquente, prenant plus en compte l'aspect temporel et traitant de toutes les classes d'anomalies a été menée en deuxième année. Cette expérimentation a visé à étudier les améliorations que peuvent apporter la récurrence face à la convolution au sein d'un auto-encodeur utilisé dans le cadre de bâtiments connectés. Les résultats de cette étude ont donné lieu à une publication. La dernière année a été consacrée à l'amélioration de l'utilisation d'auto-encodeurs en incluant au fonctionnement original de l'auto-encodeur une estimation de l'incertitude. Ces tests, menés sur divers jeux de données connus dans un premier temps puis sur un jeu de données de bâtiment connecté dans un second temps, ont montré une amélioration des performances et ont été publiés.

MOTS-CLEFS : Internet des objets – Réseau de neurones – Auto-encodeur – Détection d'anomalies – Incertitude

ABSTRACT : This thesis aims to exploit the future mass of data that will emerge from the large number of connected objects to come. Focusing on data from connected buildings, this work aims to contribute to a generic anomaly detection system. The first year was devoted to defining the problem, the context and identifying the candidate models. The path of autoencoder neural networks has been selected and justified by a first experiment. A second, more consistent experiment, taking more into account the temporal aspect and dealing with all classes of anomalies was conducted in the second year. This experiment aimed to study the improvements that recurrence can bring in response to convolution within an autoencoder used in connected buildings. The results of this study were published. The last year was devoted to improving the use of auto-encoder by proposing to include an estimate of uncertainty in the original operation of the auto-encoder. These tests, conducted on various known datasets initially and then on a connected building dataset later, showed improved performance and were published.

KEY-WORDS : Internet of things – Neural networks – Autoencoder – Anomaly detection – Uncertainty