

EPSI Bordeaux  
114 rue Lucien Faure  
33000 Bordeaux



Onepoint sud-ouest  
Cité Numérique  
2 rue Marc Sangnier  
33130 Bègles

**onepoint.**

# Mémoire Professionnel

Extraction de l'expertise métier au travers de la  
question de l'interprétabilité des réseaux de neurones  
récurrents : application au code informatique.

Marine LHUILLIER

Tutrice Entreprise : Ikram CHRAIBI KAADOUD

Tuteur Mémoire : Baptiste PESQUET

Promotion : 2019/2020

Soutenance : juillet 2020

## Résumé

Les connaissances de chaque individu sont multiples, variées et surtout se distinguent selon deux catégories. Nous retrouvons en chacun de nous des connaissances explicites, simples à formuler et partager mais aussi des connaissances dites implicites acquises par le vécu et l'expérience. Ce deuxième type de connaissance est en revanche très complexe à formaliser du fait de leur nature non consciente.

Dans la vie quotidienne, mais surtout en entreprise la gestion des connaissances et compétences de chaque collaborateur est une vraie nécessité. En effet, ces connaissances sont la clé du bon fonctionnement d'une équipe projet, sont vendues auprès des clients ou des prospects et utilisées en faire-valoir d'une certaine expertise. Néanmoins, il existe un véritable paradoxe entre l'utilisation au quotidien de ces connaissances par les collaborateurs et leur partage. La gestion de ces connaissances est un véritable levier et axe stratégique majeur en entreprise. Une problématique voit ainsi le jour autour de l'identification, de la formalisation et du transfert de ces compétences et connaissances implicites. Ainsi, nous présentons dans ce mémoire, nos travaux de recherche autour de l'élaboration d'une approche générique d'extraction de ces connaissances et notamment, l'expertise.

Nos travaux se situent ainsi dans un champ de recherche de l'intelligence artificielle, appelé le Machine Learning, qui nous permet de reproduire schématiquement et informatiquement le raisonnement humain en implémentant des réseaux de neurones artificiels. En effet, comme nous le verrons dans ce mémoire, notre réseau de neurones artificiels va apprendre des règles implicites, reflet de son raisonnement, à partir des données lui étant fournies et cela sans avoir été explicitement programmé pour le faire. Dans le cadre de nos travaux nous nous intéressons plus particulièrement à l'expertise des développeurs et experts Java à travers leur code. Nous pensons effectivement, qu'une représentation des connaissances implicites de ces développeurs se retrouve dans leur code. Ainsi, en utilisant des réseaux de neurones artificiels apprenant à partir de ces données, et de la littérature scientifique, nous proposons une approche générique d'extraction des règles à partir de ces réseaux de neurones, règles qui, dans le cas du code Java, représentent l'expertise de ces développeurs. Notre démarche exploratoire, liée à la dimension recherche de cette année d'alternance, répond à un besoin métier fort de notre entreprise et se découpe en plusieurs phases.

Dans un premier temps, nous avons implémenté notre propre réseau de neurones et lui avons fait apprendre sur des données dites artificielles afin de construire, valider et tester la robustesse de notre approche. Dans un second temps, nous sommes passés à l'apprentissage à partir de données réelles : le code Java. Enfin, la dernière grande phase de nos travaux a été d'implémenter une technique d'interprétabilité des réseaux de neurones artificiels afin d'extraire et représenter les règles implicites de notre réseau, c'est-à-dire expliciter l'expertise implicite recherchée. A travers ce mémoire, nous exposerons et partagerons nos travaux, les résultats préliminaires que nous avons obtenus ainsi que la suite de ceux-ci.

## Mots clés

Apprentissage autonome – réseaux de neurones récurrents – interprétabilité post-hoc - extraction de l'expertise – extraction de règles – connaissances implicites - sciences cognitives

## Abstract

The knowledge of each individual is multiple, varied and, above all, can be divided into two categories. We find in each of us explicit knowledge, simple to express and share, but also so-called implicit knowledge acquired through experience. This second type of knowledge, on the other hand, is very complex to formalize due to its non-conscious nature.

In daily life, but especially in companies, the knowledge management of each employee is a real necessity. Indeed, this knowledge is the key to the successful operation of a project team, is sold to customers or prospects and used as a showpiece of a certain expertise. Nevertheless, there is a real paradox between the daily use of this knowledge by employees and its sharing. The knowledge management is a real driver and a major strategic axis in a company. A problem is emerging around the identification, formulation and transfer of this knowledge and mainly implicit knowledge. Thus, in this paper, we present our research work on the development of a generic approach to the extraction of this knowledge and, in particular, expertise.

Our work is part of a field of research in artificial intelligence, called Machine Learning, which allows us to reproduce human reasoning schematically and computationally by implementing artificial neural networks. Indeed, as we will see in this paper, our artificial neural network will learn implicit rules hidden in the data provided to it without having been explicitly programmed to do so. In the scope of our work we are particularly interested in the expertise of Java developers and experts through their code. We believe that a representation of the implicit knowledge of these developers can be found in their code. Thus, using artificial neural networks learning sequences of java code, and the scientific literature, we propose a generic approach to extract rules from these neural networks, rules which in the case of Java code represent the expertise of these developers. Our exploratory approach, linked to the research dimension of this year of work-study, meets a strong business need of our company and is divided into several phases.

First, we implemented our own neural network and made it learn on so-called artificial data in order to build, validate and test the robustness of our approach. In a second step, we wanted to show the transposability of our approach on real data using : the java code. Finally, the last major phase of our work was to implement a technique of interpretability of artificial neural networks in order to extract and represent the implicit rules of our network, i.e. to make explicit the implicit expertise sought. Through this paper, we will expose and share our work, the preliminary results we obtained and the continuation of our work.

## Keywords

Machine learning - recurrent neural networks - post-hoc interpretability - expertise extraction - rule extraction - implicit knowledge - cognitive science

## Remerciements

Je souhaite remercier ici toutes les personnes qui m'ont accompagnée, soutenue et aidée dans la réalisation de ce mémoire.

Tout d'abord, je tiens à remercier mes parents et mes proches pour leur soutien sans faille, leur accompagnement durant ces 5 années d'études et l'intérêt particulier qu'ils ont porté aux travaux que j'ai menés cette année.

Je souhaite également remercier l'ensemble de l'équipe R&D de onepoint pour leur accueil chaleureux, leur humour et leur bonne humeur durant cette année d'alternance. La volonté de partager, transmettre leurs connaissances et l'écoute dont ils ont su faire preuve m'ont permis de découvrir le monde de la recherche et de m'ouvrir à de nouvelles disciplines scientifiques. Pour tous ces moments et nombreux échanges que j'ai particulièrement appréciés, merci. Parmi les membres de l'équipe, je tiens notamment à remercier Coralie Vennin pour la relecture de ce mémoire et plus particulièrement Alexandra Delmas pour sa relecture détaillée et ses nombreux bons conseils tout au long de cette année.

Je remercie aussi l'ensemble des collaborateurs de onepoint pour leur accueil, leur bonne humeur permanente et tous les échanges que nous avons pu avoir. Un grand merci à Erwan Le Bronec et l'équipe RH pour l'opportunité qu'ils m'ont offerte d'intégrer onepoint et l'équipe R&D pour ma dernière année d'alternance.

J'adresse également mes remerciements à l'ensemble des membres, actuels et anciens, des équipes pédagogiques et administratives de l'EPSI Bordeaux pour leur accompagnement et leurs enseignements durant ces 5 années d'études. Merci aux intervenants pour la qualité des cours dispensés, leur disponibilité et les nombreux retours d'expérience partagés.

Aussi, je remercie grandement Baptiste Pesquet enseignant en informatique et intelligence artificielle à l'ENSC (Ecole Nationale Supérieure de Cognitique) pour son accompagnement, son suivi et les précieux conseils qu'il m'a prodigués dans le cadre de ce mémoire en tant que tuteur pédagogique. Je remercie également Angèle Peyneau pour son soutien et son accompagnement à la préparation de la présentation orale de ce mémoire.

Enfin, je ne saurais exprimer assez de gratitude envers Ikram Chraïbi Kaadoud, ma tutrice et collègue de l'équipe R&D, pour l'opportunité qu'elle m'a offerte de découvrir les mondes de la recherche et de l'intelligence artificielle auxquels j'ai si bien accroché. Je la remercie tout particulièrement pour son écoute, sa disponibilité, son soutien mais surtout son accompagnement tout au long de cette année. Pour la rigueur scientifique qu'elle a su m'inculquer, la liberté qu'elle m'a laissée et la confiance dont elle a fait preuve, je dis merci. Je la remercie aussi pour les très nombreux échanges et moments d'émulations que nous avons eu. Merci pour cette collaboration qui m'a tant apporté, appris, fait découvrir et donné envie de faire de la recherche.

A cette année qui m'a énormément apporté professionnellement mais aussi personnellement.

## ATTESTATION DE NON-PLAGIAT (A inclure dans le mémoire professionnel)

### Je soussigné(e)

Nom : LHUILLIER..... Prénom : Marine.....

**Auteur du mémoire professionnel pour le Titre de niveau 7 RNCP – Expert en informatique et Système d'information**

Extraction de l'expertise métier au travers de la question de l'interprétabilité des réseaux de neurones récurrents : application au code informatique.

### Déclare :

- Que ce mémoire est un document original, fruit d'un travail personnel
- Avoir obtenu les autorisations nécessaires pour la reproduction d'images, d'extraits, de tableaux, figures ou graphiques
- Ne pas avoir contrefait, falsifié, copié tout ou partie de l'œuvre d'autrui afin de la faire passer pour mienne ;

### Atteste :

- Que toutes les sources d'information utilisées pour ce travail de réflexion et de rédaction sont référencées de manière exhaustive et claire dans la bibliographie/webographie de mon mémoire professionnel
- Etre informé(e) et conscient(e) que le fait de ne pas citer une source ou de ne pas la citer clairement et complètement est constitutif de plagiat, et que le plagiat est considéré comme une faute grave et sanctionné par la Loi.

### Date et Signature :

Fait le 26/06/2020..... A BORDEAUX.....

Signature :



# Table des matières

<b>Introduction.....</b>	<b>1</b>
<b>I.   Projet “Interprétabilité Appliquée au Code”, alias IAC.....</b>	<b>3</b>
I.A.   Contexte Entreprise .....	3
I.A.1.   Présentation de onepoint .....	3
I.A.2.   Cercle R&D .....	4
I.A.3.   L’équipe IAC .....	5
I.A.4.   IAC – gestion de projet .....	5
I.A.5.   Mes contributions et réalisations.....	6
I.B.   Contexte Recherche et Développement, R&D .....	7
I.B.1.   Démarche scientifique.....	7
I.B.2.   Sciences cognitives, la pluridisciplinarité au sein d’une entreprise .....	8
I.C.   Contexte scientifique .....	9
I.C.1.   Intelligence Artificielle .....	9
I.C.2.   Processus cognitifs et fonction de raisonnement .....	10
I.C.3.   Connaissances explicites / connaissances implicites.....	10
I.C.4.   Expertise métier : évolution des connaissances .....	11
I.C.5.   Machine Learning et expertise.....	12
I.D.   Besoin métier et objectifs.....	13
I.E.   Déroulement des travaux.....	13
<b>II.   Projet IAC – réseau de neurones artificiels.....</b>	<b>15</b>
II.A.   Réseaux de neurones artificiels, ANN .....	15
II.A.1.   Présentation générale .....	15
II.A.2.   Concept d’apprentissage .....	17
II.A.3.   Implémentation en Python 3.6 et Keras.....	20
II.B.   Réseaux de neurones récurrents, RNN .....	22
II.B.1.   Principe général.....	22
II.B.2.   Long Short Term Memory, LSTM .....	23
II.B.3.   Modèle implémenté pour le projet IAC.....	25
II.C.   Approche et données étudiées.....	26
II.C.1.   Notre approche .....	26
II.C.2.   Données artificielles : grammaire de Reber.....	27
II.C.3.   Données réelles : code Java .....	30

II.C.4.	Premiers résultats .....	31
<b>III.</b>	<b>Projet IAC – interprétabilité et extraction de l’expertise.....</b>	<b>34</b>
III.A.	Etat de l’art .....	34
III.A.1.	De la boîte noire à l’IA explicable .....	34
III.A.2.	Interprétabilité ou explicabilité.....	35
III.A.3.	Interprétabilité locale ou globale : différents usages.....	36
III.A.4.	L’interprétabilité au cœur du projet IAC .....	37
III.B.	Déroulé des travaux d’interprétabilité.....	38
III.B.1.	Extraction des états de la couche cachée du RNN-LSTM .....	38
III.B.2.	Clustering des données : k-means et silhouette score .....	40
III.B.3.	Obtention d’un automate .....	41
III.B.4.	Minimisation des automates .....	42
III.B.5.	Validation des automates .....	44
III.C.	Résultats de notre approche générique d’extraction de règles implicites et de l’expertise .....	45
III.C.1.	Application à la grammaire de Reber .....	46
III.C.2.	Application au code Java .....	50
III.D.	Discussion et suite du projet .....	57
<b>Conclusion</b>	.....	<b>59</b>
<b>Glossaire</b>	.....	<b>61</b>
<b>Liste des Figures</b>	.....	<b>63</b>
<b>Liste des tableaux</b>	.....	<b>65</b>
<b>Bibliographie</b>	.....	<b>66</b>
<b>Webographie</b>	.....	<b>68</b>
<b>Annexes</b>	.....	<b>69</b>

## Introduction

Un expert est une personne surentraînée ayant acquis des compétences et connaissances spécifiques par son expérience et son vécu. Ces connaissances spécifiques se caractérisent notamment par une migration des connaissances explicites vers un savoir-faire implicite, non conscient et non exprimable [Ettlinger et al., 2011]. Plus précisément, un expert, dans un domaine donné, est un individu qui possède des connaissances, processus et procédures implicites qui lui permettent de trouver, rapidement et automatiquement, un ensemble d'options possibles pour faire face à une situation précise selon le contexte courant mais aussi en s'appuyant sur ses expériences antérieures. Cela lui évite ainsi l'utilisation d'un processus itératif et laborieux de prise de décision tel qu'un junior aurait à le faire par exemple [Bootz, 2016].

Dans le domaine du développement informatique, cette notion d'implicite induit, du fait de sa nature, une problématique omniprésente autour de la verbalisation et la transmission du savoir-faire des développeurs. Ainsi, l'expertise et la gestion des connaissances associées représentent aujourd'hui un enjeu capital et un axe de développement stratégique pour les entreprises (e.g. formations, départs à la retraite, turnovers importants, ...). En effet, il leur faut constamment identifier, transférer, diffuser et formaliser ces connaissances implicites, i.e. le savoir-faire, afin de les expliciter dans un but de capitalisation [Nonaka et Takeuchi, 1995].

Mais comment faire pour partager cette expertise, ce savoir-faire si important au niveau projet, ces connaissances implicites qui sont non conscientes et de fait peu accessibles ? Comment réussir à les verbaliser, tout du moins essayer de les représenter d'une quelconque manière que ce soit ?

C'est dans ce contexte que se situent nos travaux de recherche et ce mémoire. En effet, nous allons étudier une nouvelle approche générique d'identification et d'extraction de l'expertise, et ce, à l'aide d'algorithmes de Machine Learning – des réseaux de neurones récurrents – et de techniques d'interprétabilité dites post-hoc (intervenant après l'apprentissage).

Dans le cadre de notre projet, nous nous intéressons tout particulièrement à l'expertise métier de développeurs Java, aux connaissances implicites liées et aux règles qui les régissent dans un but de capitalisation de la connaissance cachée dans les données.

Nous soulevons également une seconde problématique, cette fois-ci purement scientifique et propre à des travaux de recherche autour de la question de l'interprétabilité des réseaux de neurones artificiels – considérés aujourd'hui comme des « boîtes noires » – à laquelle nous tenterons d'apporter une réponse. Nous verrons ainsi ensemble comment, en étudiant le fonctionnement interne de ces « boîtes noires », nous tentons d'apporter un début de réponse à cette question scientifique. Nous verrons également comment ces premières réponses nous aideront dans un second temps à répondre à notre problématique métier sur l'extraction de l'expertise de développeurs Java.

Nous présentons ainsi dans ce document nos travaux en trois parties.



Dans un premier chapitre, nous exposerons et étudierons le contexte de l'entreprise onepoint<sup>1</sup>, le contexte de recherche et développement mais aussi le contexte scientifique dans lesquels se situent nos travaux dans le cadre du projet IAC – Interprétabilité Appliquée au Code. Notre problématique s'inscrit à la jonction de plusieurs disciplines scientifiques telles que les sciences cognitives pour la compréhension de la notion d'expertise d'un point de vue humain et l'informatique pour sa modélisation. Nous évoquerons également l'organisation du projet et ses étapes clés, avant de détailler ma contribution ainsi que mes différentes réalisations effectuées lors de mon année d'alternance.

Ensuite, dans un deuxième chapitre nous préciserons les étapes d'implémentation de notre réseau de neurones, cadre de notre étude sur l'extraction des connaissances implicites, i.e. l'expertise. Nous expliquerons l'implémentation, la phase d'apprentissage et de tests de celui-ci et les différentes métriques objectives implémentées. Enfin, nous détaillerons l'importance de valider notre approche à l'aide de différents types de données (artificielles et issues de code Java), d'un point de vue scientifique mais aussi en prévision des hypothèses et/ou conclusions métiers que nous pourrions exposer.

Dans un troisième chapitre, nous définirons la notion d'interprétabilité. Nous détaillerons ensuite l'implémentation d'une technique d'interprétabilité de réseaux de neurones, dite post-hoc. Puis nous soulignerons les limites de nos résultats obtenus et ferons le parallèle entre les réponses qu'ils nous apportent d'un point de vue scientifique mais aussi d'un point de vue métier sur l'extraction de l'expertise. Enfin, nous évoquerons les différents axes d'application potentiels et objectifs à plus long terme pouvant faire suite à nos travaux ; en lien direct avec notre domaine d'étude, pour l'entreprise onepoint mais aussi pour la communauté scientifique. Nous finirons par une conclusion sur la question de l'interprétabilité dans le domaine du Machine Learning et son impact sur les questions sociétales et éthiques.

---

<sup>1</sup> conformément à la charte onepoint, « onepoint » ne prend jamais de majuscule, sauf en début de phrase

## I. Projet “Interprétabilité Appliquée au Code”, alias IAC

Le projet « Interprétabilité Appliquée au Code » alias IAC, qui sert de cadre d’étude dans ce document, s’inscrit dans la lignée des travaux de recherche de ma tutrice, Ikram CHRAIBI KAADOUD, chercheuse en intelligence artificielle et sciences cognitives. Les travaux effectués lors de son doctorat portaient sur l’apprentissage de séquences par des réseaux de neurones récurrents appliqué au traçage de schémas techniques [Chraibi Kaadoud, 2018].

Le projet de recherche IAC est quant à lui axé sur la question de l’interprétabilité des réseaux de neurones récurrents appliquée au code Java. Il est important de souligner que sa nature particulière, i.e. exploratoire, nous confronte à des contraintes et objectifs différents de ceux définis dans des contextes de projets informatiques plus traditionnels. Ainsi, tout au long de ce mémoire nous traiterons d’éléments liés à l’aspect métier de notre sujet sur l’extraction de l’expertise Java mais aussi à l’aspect recherche scientifique prépondérant de nos travaux.

### I.A. Contexte Entreprise

#### I.A.1. Présentation de onepoint

Onepoint est une société française de services qui propose des solutions de transformation numérique pour les entreprises et administrations à travers le monde.

Depuis sa création en 2002 par David Layani, la société n’a cessé de croître atteignant aujourd’hui plus de 2300 collaborateurs répartis sur 14 sites dans le monde. Elle croise les regards d’experts et consultants de tous les domaines, lui permettant de rester compétitive à l’ère du digital avec une moyenne d’âge de 33 ans ; relativement élevée pour une Entreprise de Services du Numérique (ESN) mais preuve de cette expertise.

Fort d’un chiffre d’affaire actuel de 300 millions d’euros, onepoint est l’un des acteurs majeurs de la transformation du numérique et a pour ambition de continuer son développement. Cette croissance résulte notamment de son savoir-faire allant de la vision stratégique à l’innovation en passant par le design, le développement, l’architecture ou encore l’organisation et la culture. Ce savoir-faire permet ainsi à l’entreprise de proposer un accompagnement global à ses clients et prospects en devenant l’architecte de leur transformation.

Onepoint porte cinq grandes valeurs qui sont : l’authenticité, l’ouverture, le courage, l’élégance et l’engagement. Elles commandent ainsi une approche plus collaborative et créative fondée sur l’implication de chacun, sur l’échange et la collaboration qui favorisent l’idéation et l’innovation.

De plus, au sein de cette structure, le principe de communautés constitue le cœur de l’organisation et de ce travail collaboratif. Il s’agit d’un espace ouvert qui accueille et fait vivre ses expertises métiers, technologiques ou fonctionnelles au service du business et de l’épanouissement de chacun.

La communauté sud-ouest, composée des agences de Bordeaux et Toulouse, est elle-même organisée en pôles appelés « cercles » qui sont à ce jour au nombre de 17 : Agile, Architecture, Business, Change, Communication, Data, Delivery, Design, Green, QA, Média, RH, Sécurité, TechUp et **R&D**. C’est au sein de ce dernier que j’ai réalisé mon alternance.

Je suis ainsi arrivée au sein du cercle R&D de onepoint sud-ouest en septembre 2019 pour ma dernière année d'ingénierie à l'EPIS Bordeaux effectuée en alternance. J'ai eu l'opportunité d'être affectée à l'équipe R&D et tout particulièrement au projet IAC en tant qu'ingénieure recherche et développement en intelligence artificielle et plus particulièrement en Machine Learning et sciences cognitives.

#### I.A.2. Cercle R&D



Figure I.1 - Cercle R&D au sein de la communauté Sud-Ouest de onepoint

Le cercle R&D de onepoint, regroupant une quinzaine de collaborateurs, se compose aujourd'hui d'une équipe pluridisciplinaire : docteurs, consultants, alternants et stagiaires travaillant sur différentes thématiques telles que les sciences cognitives, l'intelligence artificielle (IA), les neurosciences ou encore la psychologie. Si aujourd'hui l'équipe de recherche de onepoint s'articule autour de ces grandes thématiques, elle continue d'être à l'écoute de nouveaux axes de recherches.

La mission du cercle R&D est d'étudier des mécanismes cognitifs permettant aux êtres humains d'interagir, de prendre des décisions et d'élaborer des stratégies. Le but étant de concevoir des solutions autonomes et adaptables pour les utilisateurs en prenant en compte leurs caractéristiques cognitives.

Les axes de recherche sont en relation avec des questions scientifiques interdisciplinaires liées à un besoin interne ou client dans le but de créer une solution ou une méthode applicative. Ce contexte est très important aux yeux de l'équipe car **il allie la recherche scientifique** tout en se projetant à plus long terme sur **des solutions applicatives et métiers** résultant de ces travaux.

Dans le cadre du cercle R&D de onepoint, une partie de cette mission consiste à réaliser des publications scientifiques et faire de la médiation scientifique. En d'autres termes, l'équipe souhaite partager, transmettre ses connaissances et communiquer autour de son travail à travers la participation à des conférences scientifiques et techniques, des *meet-up* de vulgarisation, mais aussi la publication d'articles scientifiques et de vulgarisation.

De manière plus concrète les objectifs de l'équipe sont les suivants :

- Mener à bien la recherche scientifique, des expérimentations et participer à la communauté scientifique internationale via des publications scientifiques ;
- Promouvoir la science en interne et en externe de l'entreprise à travers une activité de médiation et de vulgarisation scientifique ;

- Constituer une expertise utile en interne pour monter des projets ;
- Constituer une expertise pour le rayonnement en externe en participant et en décrochant des projets via des appels à projets, et participer à la vitrine du savoir-faire de l'entreprise.

### I.A.3. L'équipe IAC

L'équipe IAC faisant partie du cercle R&D se compose aujourd'hui de 3 membres :

- Ikram Chraïbi Kaadoud, docteure en informatique et chercheuse en intelligence artificielle et sciences cognitives. Elle est porteuse de ce projet et également ma tutrice ;
- Un développeur R&D Data/IA, en alternance de l'école Simplon Microsoft;
- Moi-même, ingénieure R&D en Machine Learning et sciences cognitives, en alternance.

Bien que l'équipe soit relativement petite, la richesse de nos diverses expériences passées mais aussi nos spectres de compétences différents nous ont permis d'avoir une bonne émulation ainsi qu'une certaine complémentarité, clés du bon avancement du projet.

### I.A.4. IAC – gestion de projet

L'organisation et la gestion de projet au sein de l'équipe IAC a évolué et s'est améliorée tout au long de mon année d'alternance. La gestion de projet et les rôles de chacun, et en particulier le mien, ont ainsi été adaptés.

L'équipe est assez autonome sur son fonctionnement interne, sur les rôles et l'organisation de chacun et ce, grâce à la vision de notre porteuse de projet et à la liberté qui nous a été laissée. C'est ainsi que j'ai eu l'opportunité, en parallèle de ma montée en compétences techniques sur le projet, de prendre part de plus en plus à la gestion du projet et à l'amélioration de certains de nos processus.

Concernant notre fonctionnement, nous nous inspirons de principes et bonnes pratiques des méthodes agiles<sup>2</sup>. Nous les avons bien entendu adaptées selon la taille de notre équipe mais aussi le contexte R&D dans lequel nous nous situons.

Ainsi, l'importance que nous accordons à la communication et la transparence au sein de l'équipe reflète très bien une des valeurs fondamentales des méthodes agiles, tout comme la nécessité de savoir s'adapter au changement. Cette dernière valeur fondamentale des méthodes agiles rejoint parfaitement la capacité de remise en question et l'adaptation dont il faut savoir faire preuve dans un projet de recherche.

Au niveau de notre organisation et gestion de projet, de nouvelles échéances ont été mises en place en cours d'année afin d'améliorer de nouveau nos processus tels que :

- Des « *weeklies* »: réunions hebdomadaires de trente minutes regroupant toute l'équipe. L'objectif étant de faire un bilan rapide sur les avancées de chaque membre de l'équipe durant la semaine écoulée tout en partageant (d'un point de vue macroscopique) une prévision des tâches à réaliser la semaine suivante.
- Des *ateliers bimensuels* : sessions de travail bimensuelles d'une demi-journée ou journée regroupant toute l'équipe. L'objectif étant de planifier le prochain sprint, i.e. itération, (d'une durée de deux à trois semaines maximum) et d'établir les objectifs à atteindre et réalisations à effectuer.

---

<sup>2</sup> Manifeste Agile : <https://manifesteagile.fr/>

- Des *ateliers spécifiques* : sessions de travail d'une durée choisie sur un sujet spécifique avec les membres de l'équipe concernés. La création de ce type d'atelier nous permettant de mieux cadrer et tracer les échanges sur des sujets importants et/ou bloquants rencontrés par un membre de l'équipe.

De plus, nous avons porté une attention particulière à la rédaction de comptes rendus de réunions et d'ateliers. Nous avons également conçu, en accord avec toute l'équipe IAC, différents plannings prévisionnels (cf. Annexe 1 de ce document) et rétroplannings afin de garder une continuité dans nos travaux.

#### I.A.5. Mes contributions et réalisations

Durant cette année d'alternance j'ai ainsi pu participer en premier lieu à la vie du cercle R&D et à la réalisation de ses différents objectifs au travers de projets communs tels que : l'écriture d'articles scientifiques et de vulgarisation, la réalisation de nombreux ateliers et réunions ou encore de workshops scientifiques internes où l'émulation scientifique et la pluridisciplinarité étaient les maîtres mots. De plus, j'ai eu l'opportunité de participer à l'animation de certaines réunions du cercle R&D et aussi des réunions et ateliers de l'équipe IAC.

J'ai également participé à la réalisation du projet Interprétabilité Appliquée au Code comme évoqué précédemment. Le projet IAC m'a permis de fortement monter en compétences dans le domaine du Machine Learning (ML), domaine dans lequel j'étais débutante en septembre dernier. J'ai ainsi découvert l'implémentation d'un réseau de neurones récurrents, la connaissance et la maîtrise d'un framework de ML mais aussi l'interprétation et l'analyse des résultats. Le plus grand objectif de nos travaux sur IAC n'étant pas l'implémentation du réseau mais de comprendre et rendre explicite son fonctionnement interne et son raisonnement.

J'ai aussi découvert le monde de la recherche et expérimenté le métier de chercheur, deux concepts qui m'étaient jusque-là inconnus. J'ai ainsi appris l'importance de la rigueur scientifique mais aussi l'ouverture d'esprit à de nouveaux concepts et disciplines. En effet, le contexte pluridisciplinaire du cercle R&D s'y prêtant bien, il m'a été possible d'élargir mes interrogations et axes de recherche à d'autres disciplines scientifiques liées. L'opportunité s'est donc présentée à moi de monter en compétences et découvrir de nombreux domaines qui m'étaient jusqu'alors peu connus, notamment les sciences cognitives.

D'un point de vue plus factuel, mes activités au sein du cercle R&D et du projet IAC, ont consisté en l'implémentation d'un réseau de neurones récurrents, sa phase d'apprentissage et de tests, la rédaction de notes techniques mais aussi à la mise en place d'une technique d'interprétabilité ainsi qu'à l'analyse et l'interprétation des résultats. J'ai également pris part à la gestion du projet IAC, à l'animation de réunions et à la présentation de mes travaux aux membres de l'équipe d'un point de vue opérationnel. De plus, j'ai contribué à la rédaction d'un article court et d'un abstract qui ont été soumis respectivement aux conférences RJCIA<sup>3</sup> 2020 et GDRIA<sup>4</sup> 2020, dont je suis actuellement en attente de réponse.

Au-delà de l'aspect plus technique de mes travaux, une partie non négligeable de ma mission a été de faire de la bibliographie dans le but d'étendre mes connaissances sur différents sujets liés à notre problématique projet, développer mon esprit critique vis-à-vis des approches proposées et

---

<sup>3</sup> Rencontres des Jeunes Chercheurs.ses en Intelligence Artificielle

<sup>4</sup> Groupement de Recherche sur les Aspects Formels et Algorithmiques de l'Intelligence Artificielle

d'apprendre à vulgariser l'ensemble des concepts très techniques acquis, à des collègues néophytes ou technophiles, du domaine ou non.

Je me suis ainsi constitué une culture scientifique large sur les thèmes de l'intelligence artificielle, du Machine Learning et des sciences cognitives mais aussi une culture plus spécifique sur l'interprétabilité des réseaux de neurones, nécessaire à la bonne compréhension des différents contextes dans lesquels se situe le projet IAC. Commençons par le contexte de Recherche et Développement.

## I.B. Contexte Recherche et Développement, R&D

D'après l'INSEE<sup>5</sup>, la notion R&D englobe toutes les activités créatives et systématiques entreprises en vue d'accroître la somme des connaissances et de concevoir de nouvelles applications à partir des connaissances disponibles [INSEE, 2019]. La R&D inclut ainsi la recherche fondamentale, la recherche appliquée et le développement expérimental.

Afin qu'une activité soit considérée comme relevant de la R&D selon l'INSEE, elle doit remplir 5 critères de base : comporter un élément de nouveauté, de créativité et d'incertitude tout en étant systématique et transférable et/ou reproductible.

Mais pour qu'une activité de recherche soit reconnue par la communauté scientifique à travers le monde, il faut ajouter à ces 5 critères prépondérants un protocole à suivre : **la démarche scientifique**.

### I.B.1. Démarche scientifique

*Afin de comprendre et d'expliquer le monde qui nous entoure, les scientifiques du monde entier suivent une méthodologie appelée la « démarche scientifique » [Galais et al., 2020].*

Elle se constitue d'un ensemble de protocoles, règles et lignes directrices qu'il est primordial de suivre à la lettre afin de mener une expérience scientifique valable. Cette rigueur permet ainsi de garantir la valeur potentielle des découvertes effectuées aux yeux de la communauté scientifique constituée de chercheurs et chercheuses du monde entier.

Ce point est très important en recherche car chaque travail de recherche est soumis au regard des pairs du domaine ciblé par les travaux. Leur rôle est d'analyser, de questionner et de critiquer les travaux, en les validant ou en les invalidant, et ce, afin de faire avancer les questions scientifiques.

D'un point de vue plus formel, la démarche scientifique se déroule en plusieurs étapes, comme présentées sur la figure I.2.

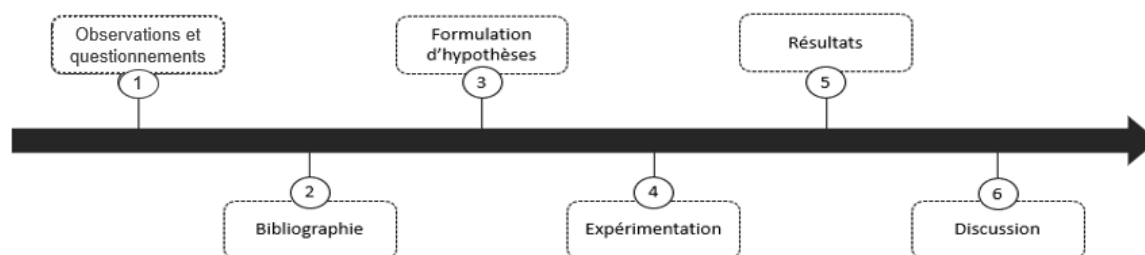


Figure I.2 - Les étapes de la démarche scientifique

<sup>5</sup> Institut National de la Statistique et des Etudes Economiques

En premier lieu, nous retrouvons l'observation et le questionnaire d'ordre général de la part du chercheur qui va lui permettre de soulever un axe de recherche.

Vient ensuite la deuxième phase de revue littéraire également appelée « bibliographie » ou « état de l'art ». Elle permet d'étudier de manière approfondie les travaux, articles scientifiques ou encore des revues spécialisées. Cette étude est réalisée par d'autres scientifiques dans le même domaine ou non mais qui apportent des éléments de réflexion autour de l'axe de recherche soulevé précédemment.

La troisième grande étape consiste en la formulation d'hypothèses. En effet l'ensemble des articles scientifiques et des travaux étudiés lors de l'état de l'art permettent déjà d'imaginer un élément de réponse au questionnaire de base. Il est important de noter à ce stade que ces hypothèses pourront aussi bien être validées ou infirmées par le biais des expérimentations. Les hypothèses offrent donc un cadre pour la construction du protocole d'expérimentation qui sera suivi.

L'expérimentation constitue donc la quatrième étape de la démarche scientifique. Il s'agit de mettre en place un protocole de travail visant à tester les hypothèses émises précédemment selon des paramètres précis afin de vérifier dans quelles mesures elles sont possibles à vérifier ou non.

L'avant dernière étape, et non des moindres, consiste en l'analyse des résultats qui suit l'expérimentation. De nouveau, cette analyse se plie à des protocoles très stricts et rigoureux. Tout comme pour la démarche scientifique dans son ensemble, la rigueur est de mise à cette étape car elle permet de déterminer l'exactitude et la robustesse des résultats selon un effet réel et non de hasard. Sans cette étape, les découvertes associées aux travaux ne pourront être validées par la communauté scientifique.

Enfin, la dernière étape est la discussion des résultats mais aussi de la démarche et des axes potentiels à suivre (perspectives, ouvertures mais aussi limites éventuelles) après travaux.

C'est donc dans ce contexte particulier de recherche et développement ainsi que de démarche scientifique que le cercle R&D de onepoint et donc l'équipe IAC effectue ses travaux.

### I.B.2. Sciences cognitives, la pluridisciplinarité au sein d'une entreprise

Le cercle R&D de onepoint s'inscrit donc dans une démarche de recherche axée autour de quatre grands domaines d'études et d'expertises qui sont :

- 1) l'expérience utilisateur ;
- 2) la psychologie et la cognition ;
- 3) la Data et l'IA ;
- 4) les neurosciences.

Les nombreux projets du cercle, bien que principalement rattachés à une de ces quatre grandes thématiques, nous amènent quelque fois à nous questionner et nous intéresser à d'autres disciplines scientifiques. Cette pluridisciplinarité de l'équipe rejoint ainsi l'approche des sciences cognitives qui s'articule autour de plusieurs disciplines scientifiques, présentées en figure I.3 : les neurosciences, la linguistique computationnelle, l'anthropologie cognitive, la psychologie cognitive, la philosophie de la cognition et l'informatique [Thagard, 2019].

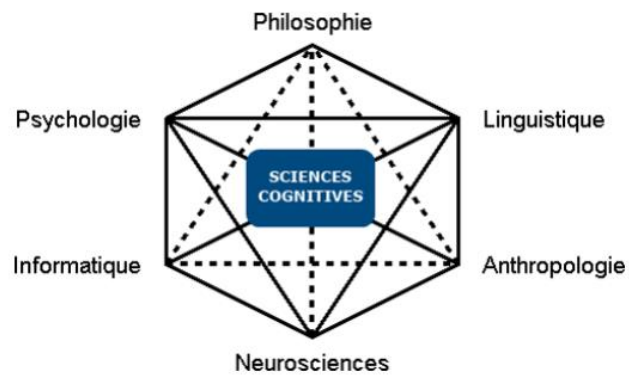


Figure I.3 - Les sciences composant les sciences cognitives

En revanche, pourquoi nous intéressons-nous aux sciences cognitives dans le cadre du projet IAC ?

Cette science de sciences représente en effet l'étude détaillée des mécanismes de la pensée humaine, animale ou artificielle afin de comprendre et d'expliquer le fonctionnement de l'esprit humain [Chraïbi Kaadoud et Delmas, 2018]. Ce fonctionnement est le regroupement d'un ensemble de processus mentaux associés à la connaissance et au traitement de l'information, appelé plus couramment la cognition. Or l'expertise est une notion intimement liée aux connaissances chez l'humain.

C'est donc dans ce contexte pluridisciplinaire porté sur la compréhension des mécanismes de la pensée et particulièrement la connaissance et son traitement que nous allons définir le contexte scientifique du projet IAC.

## I.C. Contexte scientifique

Présenter le contexte scientifique de nos travaux nous amène dans un premier temps à nous intéresser aux origines de l'intelligence artificielle et aux processus cognitifs humains.

### I.C.1. Intelligence Artificielle

La notion d'intelligence artificielle, a été définie par Marvin Lee Minsky<sup>6</sup> comme « *la construction de programmes informatiques qui s'adonnent à des tâches qui sont pour l'instant accomplies de façon plus satisfaisantes par des êtres humaines car elles demandent des processus mentaux de haut niveau tels que : l'apprentissage perceptuel, l'organisation de la mémoire et le raisonnement critique* » [COE, 2020].

Cette définition constitue ainsi la base de nos travaux de recherche : **construire un algorithme d'intelligence artificielle consiste à s'intéresser en premier lieu à ces processus mentaux humains** grâce aux sciences cognitives notamment, et d'autant plus lorsque notre objectif final est l'extraction de l'expertise humaine en utilisant ces technologies.

<sup>6</sup> Scientifique américain, pionnier de l'intelligence artificielle



### I.C.2. Processus cognitifs et fonction de raisonnement

Les processus cognitifs sont des traitements de l'information automatisés par le cerveau. Parmi ces nombreux processus cognitifs nous retrouvons par exemple la mémoire, l'attention, le langage, les fonctions visuo-spatiales, ou les fonctions exécutives. C'est au niveau des fonctions exécutives, processus cognitif de haut niveau, que nous retrouvons **la fonction de raisonnement** chez l'humain qui nous intéresse. Elle requière l'utilisation de plusieurs procédures des fonctions cognitives comme la fonction de planification ou de résolution de problèmes [Miyake et al, 2000].

Nous portons ainsi un intérêt particulier à la fonction de raisonnement car elle reflète bien d'un point de vue humain ce qu'il est nécessaire d'assimiler et d'interpréter dans le cadre de l'extraction de l'expertise au travers d'un algorithme de Machine Learning.

En effet, le raisonnement sollicite deux types de connaissances (implicites et explicites) qui vont permettre l'élaboration d'un plan d'actions à suivre pour obtenir un résultat comme la résolution d'un problème par exemple.

### I.C.3. Connaissances explicites / connaissances implicites

Il est important de **distinguer les connaissances explicites des connaissances implicites**, car c'est cette distinction qui soulève les fondements de notre problématique métier.

En effet, les **connaissances explicites**, associées à la mémoire déclarative, sont l'ensemble des connaissances qu'un individu possède et peut exprimer. L'expression et la transmission de celles-ci peut se faire verbalement mais également sur tout support de stockage tels que des manuels, des documents ou encore des mails [Smith, 2001].

Au contraire, les **connaissances implicites** sont quant à elle associées à la mémoire non déclarative. Il s'agit des connaissances dont un individu n'est pas conscient et sont de fait non exprimables [Schacter, 1987]. La figure I.4 présentent ces différents types de mémoire, chacune impliquée dans un type de connaissances : implicite, tel que faire du vélo, et explicite, l'apprentissage d'un numéro de téléphone par exemple.

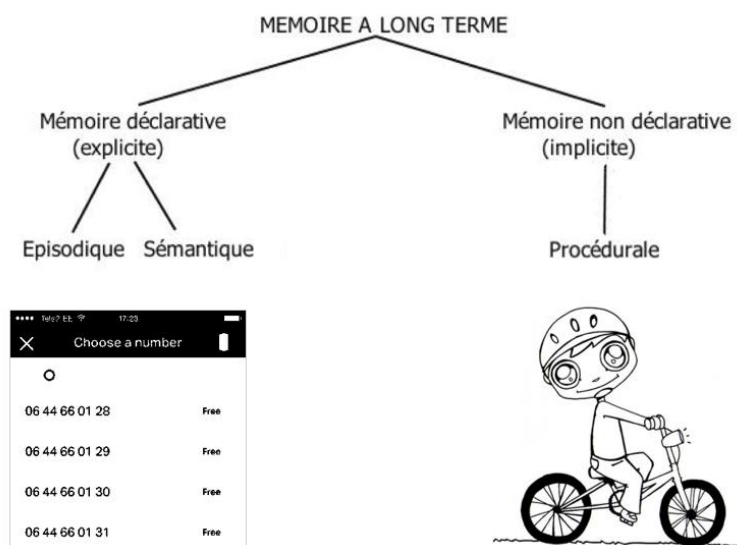


Figure I.4 - Différentes mémoires à long terme. Image adaptée de [Squire 1992] et extraite de [Chraïbi Kaadoud et Delmas, 2018]

L'apprentissage et l'acquisition de ce type de connaissances implicites par un individu se fait donc de **manière non consciente**, à son insu au travers de **son vécu, sa pratique, ses biais et son expérience**. Ce type de connaissance est alors très difficilement exprimable et donc peu accessible alors qu'il est le plus recherché des deux dans le monde scientifique mais aussi en entreprise [Malhotra, 2000; Crié, 2003].

Prenons désormais notre contexte métier des développeurs Java. Nous allons distinguer deux profils, celui d'un junior nouvellement arrivé sur un projet et celui d'un développeur sénior ayant une dizaine d'années d'expérience sur le même projet. Pour une même demande de résolution d'un bug par exemple, le développeur sénior fera, de manière non consciente, bien plus appel à ses connaissances implicites que nous appelons l'expertise. En effet, du fait de son expérience, il aura tendance à rapidement prendre en compte le contexte courant afin de formuler des hypothèses, des options et ainsi proposer, et même tester, des solutions en se basant sur son expérience et son vécu [Wiedenbeck, 1985]. A contrario, le développeur junior n'ayant pas ces connaissances implicites, à savoir cette connaissance du métier, passera par un processus itératif laborieux mobilisant ses connaissances explicites. Il essaiera d'utiliser par exemple les bonnes pratiques et concepts qu'il aura appris lors de sa formation pour résoudre le bug en question.

#### I.C.4. Expertise métier : évolution des connaissances

L'**expertise métier** peut être définie comme les connaissances et compétences d'un collaborateur acquises principalement par l'expérience dans un domaine spécifique [Daley, 1999].

Plusieurs critères existent afin d'encadrer plus précisément cette définition : la pratique avérée signe d'expérience, la capacité à rendre disponible un savoir de haut niveau ou encore la pratique dans un domaine stratégique en entreprise par exemple.

Cette notion d'expertise en entreprise reflète ainsi principalement les connaissances implicites d'un individu acquises au cours du temps. De ce fait, comme indiqué précédemment, un expert fera bien plus appel (de façon non consciente) à ses connaissances implicites pour traiter un problème qu'un junior par exemple. Cette différence est due notamment pour l'expert à l'évolution de ses connaissances de l'explicite à l'implicite comme le souligne la figure I.5 ci-dessous.

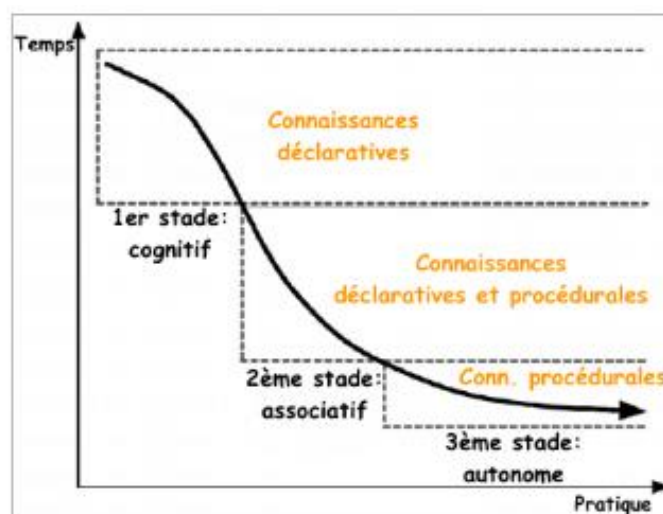


Figure I.5 - Evolution du temps nécessaire à la résolution de tâche au fur et à mesure de la montée en compétences (pratique) selon trois étapes, basée sur la théorie de Fitts et Posner [1967], Anderson [1983], Rasmussen [1986] et Van Lehn [1996]. Image adaptée de Kim et al. [2013] extraite de Chraïbi Kaadoud [2018]

Le premier stade cognitif représenté sur la figure précédente représente ainsi les connaissances et compétences explicites. Celles-ci vont évoluer avec le temps et devenir de plus en plus procédurales, i.e. séries d'étapes à réaliser, comme représentées au deuxième stade. Enfin, elles vont tendre vers un savoir-faire implicite, plus automatique encore (représenté par le troisième stade autonome) et ce, du fait de la pratique répétée et de l'expérience. Néanmoins, le terme « implicite » fait émerger une nouvelle problématique présente dans le monde de l'entreprise : comment verbaliser et transmettre ce type de connaissances ?

En effet, la richesse de certaines entreprises réside dans leurs connaissances, la valorisation et la création de ces dernières. Le management des connaissances défini par [Nonaka et Takeuchi \[1995\]](#) comme étant « *l'identification des connaissances implicites au sein d'une entreprise mais aussi leur transfert et formalisation afin de les expliciter dans un but de capitalisation* » est donc une problématique qui représente un axe de développement stratégique.

Enfin, en revenant sur la définition de l'IA exprimée par Marvin Lee Minsky, il est prépondérant de comprendre le raisonnement humain et les différents types de connaissances qui l'influencent (principalement les connaissances implicites) afin de pouvoir penser et trouver des solutions innovantes d'explicitation de ces connaissances en entreprise. Pour cela, le fait de s'appuyer sur des disciplines telles que l'informatique et le Machine Learning trouve tout son sens.

#### I.C.5. Machine Learning et expertise

Présentons désormais notre dernière grande notion : le **Machine Learning** (ML) également appelé Apprentissage Machine ou Apprentissage Automatique.

Il s'agit d'un sous-domaine de recherche de l'intelligence artificielle qui consiste à **donner la capacité d'apprentissage à une machine (un ordinateur) sans que celle-ci n'ait été explicitement programmée pour cela** [[Whatls, 2016](#)]. En d'autres termes, un algorithme de ML consiste à demander à notre machine d'effectuer une tâche précise sans que l'on code les différentes étapes et actions qu'elle devra réaliser pour y arriver. Cette tâche peut être variée telle que la classification d'images, la prédiction de symboles, etc.

L'intérêt d'utiliser des algorithmes de Machine Learning dans le cadre de nos travaux de recherche est qu'ils nous permettent de refléter, ou tout du moins de se rapprocher artificiellement, du raisonnement humain pour la résolution d'une tâche. En effet, les algorithmes de Machine Learning que nous utilisons, des réseaux de neurones récurrents (*Recurrent Neural Network*, RNN), vont, de manière autonome, développer une forme de raisonnement tel qu'un humain aurait à le faire, dans le but d'effectuer la tâche demandée. Pour ce faire, ils vont encoder des règles dites implicites sur lesquelles nous reviendrons plus en détail par la suite.

Dans le cadre de nos recherches, nous nous intéressons particulièrement à ces **règles implicites** ainsi qu'à leurs représentations lors de l'apprentissage de séquences de code Java écrits par des développeurs. Ainsi, l'utilisation de tels algorithmes nous permet deux choses. D'une part, en tant que chercheurs, de matérialiser des représentations des connaissances implicites humaines que nous pensons retrouver dans du code Java (approche métier d'extraction de l'expertise) et ce, dans un but de capitalisation de la connaissance en entreprise. D'autre part, nous essayons de répondre à un des grands enjeux scientifiques et sociétaux de l'intelligence artificielle : l'interprétabilité de nos modèles et algorithmes.

## I.D. Besoin métier et objectifs

Le projet IAC résulte ainsi d'un besoin interne chez onepoint portant sur la compréhension et la capitalisation de l'expertise métier implicite. Pour essayer d'y répondre, nous avons choisi de nous inspirer du domaine de l'interprétabilité des algorithmes d'intelligence artificielle et de Machine Learning. En effet, nous étudions et proposons un processus permettant d'extraire des règles implicites présentes dans le code, reflet de l'expertise des développeurs. Onepoint étant une ESN où le langage Java est majoritairement présent, nous avons choisi de nous focaliser sur celui-ci.

A l'instar des connaissances des développeurs, le code Java écrit par ces derniers contient une dimension explicite et une dimension implicite. D'une part, nous retrouvons au niveau de la dimension explicite les contraintes du langage, les contraintes techniques ou encore la formation qu'a reçu le collaborateur ayant écrit le code. D'autre part, nous pouvons retrouver l'expérience, le vécu et les habitudes des développeurs ayant influencé leur manière de coder.

Nous pouvons dire que la dimension implicite d'un code comporte une trace de l'expertise métier des développeurs qui ont participé à son élaboration.

Le projet IAC constitue ainsi un cadre d'étude de l'extraction de cette expertise métier à travers le code informatique. Les objectifs du projet ne sont pas figés tels qu'ils pourraient l'être sur des projets informatiques de production. En effet, les différentes avancées effectuées ainsi que la bibliographie scientifique nous amènent en permanence à reprioriser, changer et modifier nos objectifs à court terme. Néanmoins, nous pouvons évoquer des objectifs à plus long terme très importants et axés « métier » tels que la passation de connaissances basée sur du code entre collaborateurs, et à plus long terme la facilitation de la reprise d'anciens projets de développement en exploitant la représentation implicite de ces projets apprise par les réseaux de neurones.

Parmi ces objectifs, nous retrouvons également les objectifs à moyen terme fixés pour mon année d'alternance. Il s'agissait ici d'implémenter un réseau de neurones récurrents, d'en extraire les règles implicites et de proposer une méthode d'extraction de l'expertise métier dans un contexte applicatif nouveau : le code informatique. Pour répondre à ce besoin métier, nous avons mis en place une approche générique d'extraction de l'expertise présentée dans la section suivante.

## I.E. Déroulement des travaux

Les travaux menés dans le cadre du projet IAC afin de proposer une approche générique de processus d'extraction de règles implicites ont été effectués dans deux contextes d'application différents que sont la grammaire de Reber et le code Java.

Au moment de la conception de notre approche, notre besoin a été de pouvoir transposer les mêmes processus, principes et outils sur des données issues de contextes différents mais de natures similaires, i.e. des séquences. L'utilisation de données artificielles telles que la grammaire de Reber, qui correspond à notre premier contexte d'application, nous permet de tester, éprouver et valider notre implémentation et par extension notre approche sur des données déjà utilisées dans la littérature scientifique dont nous nous inspirons.

L'utilisation de données artificielles nous permet aussi de rester focaliser sur notre approche et le réseau sans se préoccuper de la qualité des données qui peut être bruitées, incomplètes ou influencées par des biais.

L'utilisation de code Java dans le second contexte, constitue l'application de notre démarche à un contexte métier et montre la transposabilité de l'approche à des données réelles.

Le protocole expérimental que nous avons ainsi mis en place se découpe en trois grandes étapes, représentées dans la figure I.6 ci-dessous : la phase d'apprentissage, la phase de test et enfin la phase d'interprétabilité.



**1) Contexte de données artificielles : grammaire de Reber**

**2) Contexte de données réelles : données Java**

*Figure I.6 - Représentation schématique de l'approche générique d'extraction de règles implémentée dans le cadre du projet IAC*

Dans la suite de ce document, nous nous concentrerons dans le chapitre suivant « II. Projet IAC – réseau de neurones artificiels » sur les concepts fondamentaux du Machine Learning et des réseaux de neurones artificiels, notamment les réseaux récurrents. Nous y évoquerons notamment les principes d'apprentissage et de test de ces types d'algorithmes. Nous poursuivrons ensuite sur une explication détaillée du réseau de neurones récurrents implémenté dans le cadre du projet IAC. Enfin, nous ferons le bilan des premiers résultats obtenus à la suite de l'apprentissage et du test de notre réseau de neurones récurrents dans les deux contextes cités précédemment.

Le dernier chapitre de ce mémoire « III. Projet IAC – interprétabilité et extraction de l'expertise » portera quant à lui essentiellement sur la question de l'interprétabilité des réseaux de neurones. Nous passerons donc en revue cette question selon un aspect scientifique et général puis nous nous focaliserons davantage sur l'extraction de règles cachées dans les données. Pour ce faire, nous étudierons plus particulièrement l'espace latent de notre réseau de neurones, porteur des représentations implicites de la connaissance. Toujours dans le cadre de notre approche générique, nous présenterons les résultats obtenus selon nos deux contextes d'applications que sont la grammaire de Reber et le code Java. Enfin, nous discuterons de la suite de nos travaux et des prochaines pistes de recherche que nous souhaitons traiter dans le cadre de l'extraction de l'expertise métier.

## II. Projet IAC – réseau de neurones artificiels

Evoquer l'extraction de l'expertise métier de développeurs Java au travers de la question de l'interprétabilité des réseaux de neurones récurrents nous conduit à définir deux notions prépondérantes : les réseaux de neurones récurrents d'une part et la notion d'interprétabilité d'autre part.

Dans ce chapitre, nous allons exposer ensemble les concepts fondamentaux des réseaux de neurones, puis la spécificité des réseaux de neurones récurrents (RNN) dans le domaine du Machine Learning et surtout de notre réseau. Nous présenterons ensuite les données étudiées lors de nos travaux, notre implémentation ainsi que les premiers résultats obtenus.

### II.A. Réseaux de neurones artificiels, ANN

Cette section a ici pour objectif de rappeler les concepts fondamentaux des réseaux de neurones artificiels et leur fonctionnement à travers un exemple d'architecture simple. Un lecteur maîtrisant déjà ces concepts pourra directement se rendre à la section suivante « II.B. Réseaux de neurones récurrents, RNN » s'il le souhaite.

#### II.A.1. Présentation générale

##### II.A.1.a. Réseau de neurones artificiels

Il existe de nombreux algorithmes de Machine Learning de natures diverses et variées tels que : les arbres de décision, les règles de classification, les algorithmes de régression ou encore les réseaux de neurones. C'est à ces derniers que nous nous sommes particulièrement intéressés dans le cadre de nos travaux.

Les **réseaux de neurones artificiels**, ANN pour *Artificial Neural Networks*, sont inspirés à l'origine du fonctionnement du cerveau humain dans le but d'essayer de répliquer artificiellement la façon dont les humains apprennent [Dormehl, 2019]. Il en existe plusieurs types selon la tâche que l'on souhaite réaliser. Afin d'explicitier les concepts fondamentaux et leur fonctionnement, prenons un réseau de neurones artificiels des plus classiques représentés en figure II.1.

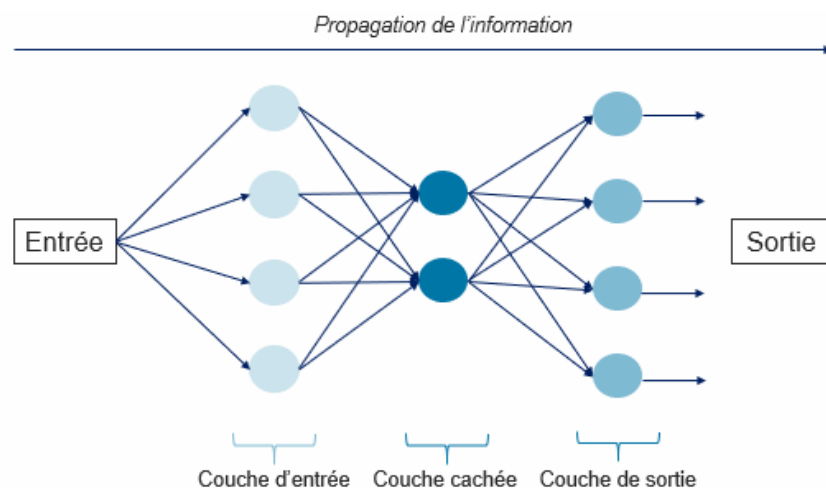


Figure II.1 - Réseau de neurones artificiels composé de 3 couches

Ce réseau de neurones artificiels se compose de 3 **couches** : couche d'entrée, couche cachée et couche de sortie. Chacune de ces couches est composée respectivement de quatre, deux et quatre **neurones formels**, i.e. neurones artificiels, représentés par les cercles sur la figure ci-dessus.

La couche d'entrée va être la seule à recevoir la donnée dans son format d'origine afin de la traiter grâce à ses neurones formels. Ensuite, cette donnée va être propagée jusqu'à une couche intermédiaire, la couche cachée<sup>7</sup> où les relations entre les variables vont être mises en exergue. Enfin, la donnée va passer par la dernière couche, la couche de sortie, qui se charge de fournir une donnée de sortie au réseau. La donnée, à chaque passage par une couche, va alors être traitée par les différents neurones formels de celle-ci. Ce principe de propagation de l'information est appelé **propagation avant**.

Il est important de noter que dans l'exemple de la figure II.1, tous les neurones formels d'une couche sont interconnectés aux neurones de la couche suivante, ces relations étant représentées par les flèches dans la figure II.1. Ainsi, chaque neurone de la couche cachée va recevoir en entrée des données provenant des quatre neurones de la couche précédente. A l'implémentation du réseau, des poids sont initialisés aléatoirement pour chacune de ces connexions.

#### II.A.1.b. Neurone formel

Un neurone formel (cf. figure II.2) est simplement **une représentation mathématique d'un neurone biologique** correspondant à une unité élémentaire d'un réseau de neurones artificiels. Il a pour but de traiter la donnée en la transformant grâce à une fonction d'activation, i.e. une fonction mathématique appelée aussi fonction de transfert, afin de fournir ce résultat aux neurones de la couche suivante, et ainsi de suite.

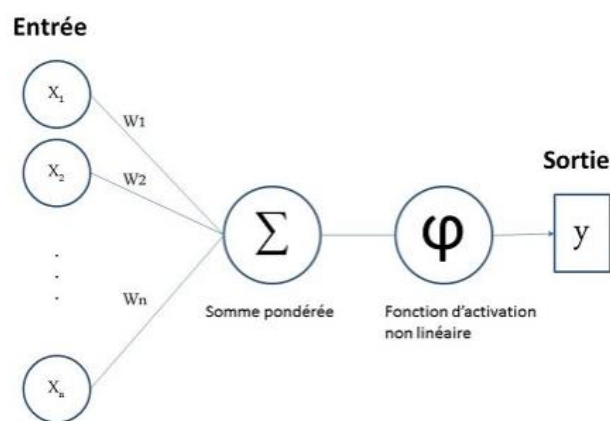


Figure II.2 – Représentation schématique du fonctionnement d'un neurone formel. Image tirée de [Giroux, 2019]

Chaque entrée fournie au neurone est représentée sur la figure par un « X » et correspond à une valeur numérique. **La connexion utilisée** (flèches sur la figure II.1) ici notée «  $W_i$  » **est assimilée à un poids** et représente le lien entre deux neurones formels. Les poids seront modifiés et ajustés au fur et à mesure de l'apprentissage et sont sauvegardés à la fin de cette phase.

<sup>7</sup> Bien que ce réseau se compose d'une seule couche cachée, des réseaux plus profonds composés de multiples couches cachées existent. L'étude et l'utilisation de ces réseaux profonds constituent le cœur du domaine du *Deep Learning* pour l'apprentissage profond.

D'autres paramètres sont également utilisés tels que le biais, mais nous ne les détaillerons pas ici. Pour plus d'informations sur le neurone formel et ses paramètres, le lecteur pourra se référer à [Giroux, 2019].

Ce qu'il est important de retenir quant au fonctionnement d'un réseau de neurones artificiels c'est qu'il traite la donnée en la transformant grâce à ses neurones formels au fur et à mesure qu'elle passe d'une couche à l'autre et ce jusqu'à la dernière couche, selon un sens de propagation de l'information.

## II.A.2. Concept d'apprentissage

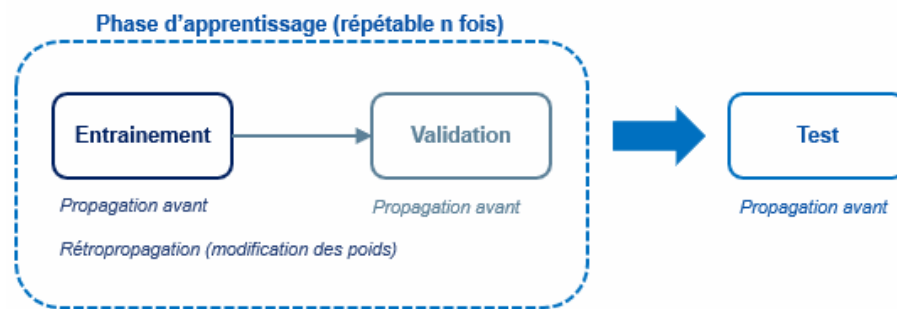


Figure II.3 - Représentation schématique du concept d'apprentissage d'un réseau de neurones artificiels. Phases d'entraînement, de validation et de test.

D'une manière similaire à celle qu'a notre cerveau humain d'apprendre de nouvelles informations grâce à l'expérience, **le réseau de neurones artificiels nécessite de nombreux essais et tentatives afin de s'améliorer et affiner ses résultats.**

Pour rappel, le Machine Learning consiste à implémenter un programme, ici un réseau de neurones, sans que celui-ci n'ait été programmé explicitement pour effectuer une tâche. En effet, la seule instruction qu'il a est d'apprendre et de créer lui-même son raisonnement à travers des règles implicites. C'est donc lors de la phase d'apprentissage (cf. figure II.3) que le réseau va apprendre à effectuer la tâche demandée, en analysant de nombreux exemples fournis pour s'entraîner.

Toutefois, on distingue trois méthodes principales d'apprentissage en Machine Learning qui sont : l'apprentissage supervisé, l'apprentissage non supervisé [App.NonSupervisé, 2020] et l'apprentissage par renforcement [Renforcement, 2020], représentés dans la figure II.4 ci-dessous. Dans le cadre de nos travaux et dans la suite de ce document, **nous traiterons exclusivement de l'apprentissage supervisé.**

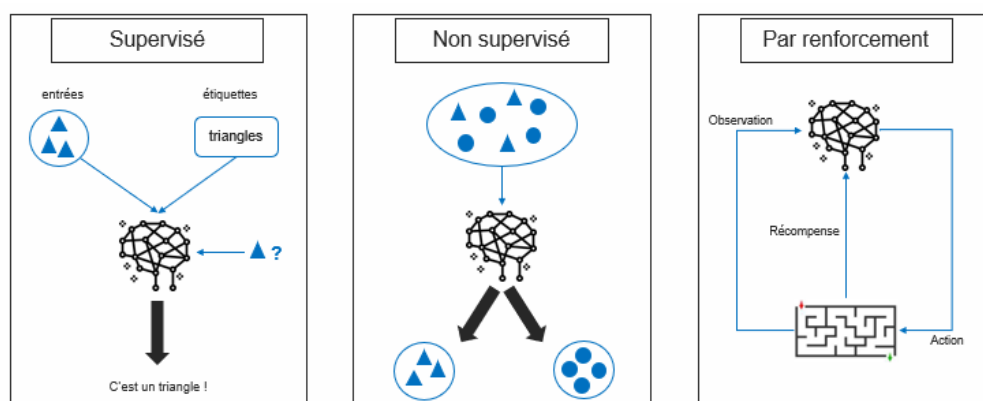


Figure II.4 - Différents types d'apprentissage en Machine Learning



### II.A.2.a. Phase d'apprentissage : apprentissage supervisé

L'**apprentissage supervisé** consiste à entraîner le réseau sur un ensemble de données étiquetées : des couples « entrée / sortie attendue ». Le réseau s'entraîne et donc, se modifie jusqu'à être capable d'obtenir la sortie attendue pour l'entrée fournie. Ce type d'apprentissage est principalement utilisé pour des tâches de classification (telle que la classification d'images de chats et de chiens), de régression (telle que la régression linéaire ou logistique) ou de prédiction (telle que la prédiction du comportement d'un acheteur sur un site d'achat en ligne) [App.Supervisé, 2020].

Le principe de cette phase est d'entraîner le modèle à effectuer une tâche en lui fournissant une multitude d'exemples afin qu'il s'améliore et apprenne. Mais pour ce faire le réseau va avoir besoin de modifier certains de ses paramètres selon la réponse à une question très simple : la sortie attendue est-elle égale à la sortie obtenue ? En effet, le réseau a-t-il réussi à prédire que la forme géométrique passée en entrée était bien un triangle ? Si oui, le réseau ne va pas se mettre à jour, il va rester dans le même état jusqu'à l'exemple suivant et ainsi de suite. En revanche, s'il ne prédit pas le bon résultat, il va devoir **se modifier de manière autonome** afin d'ajuster son comportement et réaliser la prédiction adéquate au prochain exemple. Ce sont alors les connexions, i.e. liaisons entre les neurones, désignées par le terme de **poids** (cf. figure II.2) qui sont mises à jour (i.e. impactées) lors de la phase d'apprentissage pour permettre cette adaptation du comportement du réseau.

### II.A.2.b. Phase d'apprentissage : calcul de l'erreur et rétropropagation

Un réseau de neurones qui apprend est un réseau dont les poids sont modifiés. Il se configure de lui-même de façon autonome afin de s'améliorer pour répondre au mieux à la tâche demandée. Dans le cadre de l'apprentissage supervisé et de nos travaux, le but de notre réseau est d'effectuer des prédictions en faisant correspondre une donnée en sortie et une donnée attendue.

L'algorithme d'apprentissage que nous avons utilisé pour modifier ces poids est appelé la **rétropropagation de l'erreur**, autrement nommé *backpropagation*. Il s'agit d'un des algorithmes les plus utilisés en Machine Learning. Il permet de calculer l'impact de chaque poids entre chaque couche, en commençant par la dernière et en remontant vers la première de manière algorithmique efficace.

Le calcul de **l'erreur**, appelé *loss*, qui sera repropagée se fait à chaque pas d'apprentissage, c'est-à-dire à chaque fois qu'un exemple est présenté au modèle et que celui-ci fournit une sortie. Ce calcul se fait en comparant la sortie obtenue par le réseau, i.e. la prédiction, avec la sortie attendue. L'une des fonctions mathématiques les plus utilisées en Machine Learning et que nous avons choisie dans le cadre du projet IAC est l'**erreur quadratique moyenne**, représentée par la formule ci-après :

$$\text{erreur quadratique moyenne} = (\text{sortie attendue} - \text{sortie obtenue})^2$$

Schématiquement les grandes étapes de calcul de l'erreur et de rétropropagation de celle-ci sont :

- 1) La présentation d'un exemple au modèle en propagation avant afin qu'il effectue une prédiction (cf. figure II.5 – étape 1)
- 2) Le calcul de l'erreur globale à la sortie du réseau (cf. figure II.5 – étape 1)
- 3) La rétropropagation de l'erreur sur l'ensemble des poids du réseau selon la part de chacun d'entre eux, en commençant par la dernière couche en remontant jusqu'à la couche d'entrée (cf. figure II.5 – étapes 2 et 3)
- 4) La mise à jour de l'ensemble des poids du réseau, avant la répétition de l'ensemble de ces étapes pour le prochain pas de temps (cf. figure II.5 – étape 4)

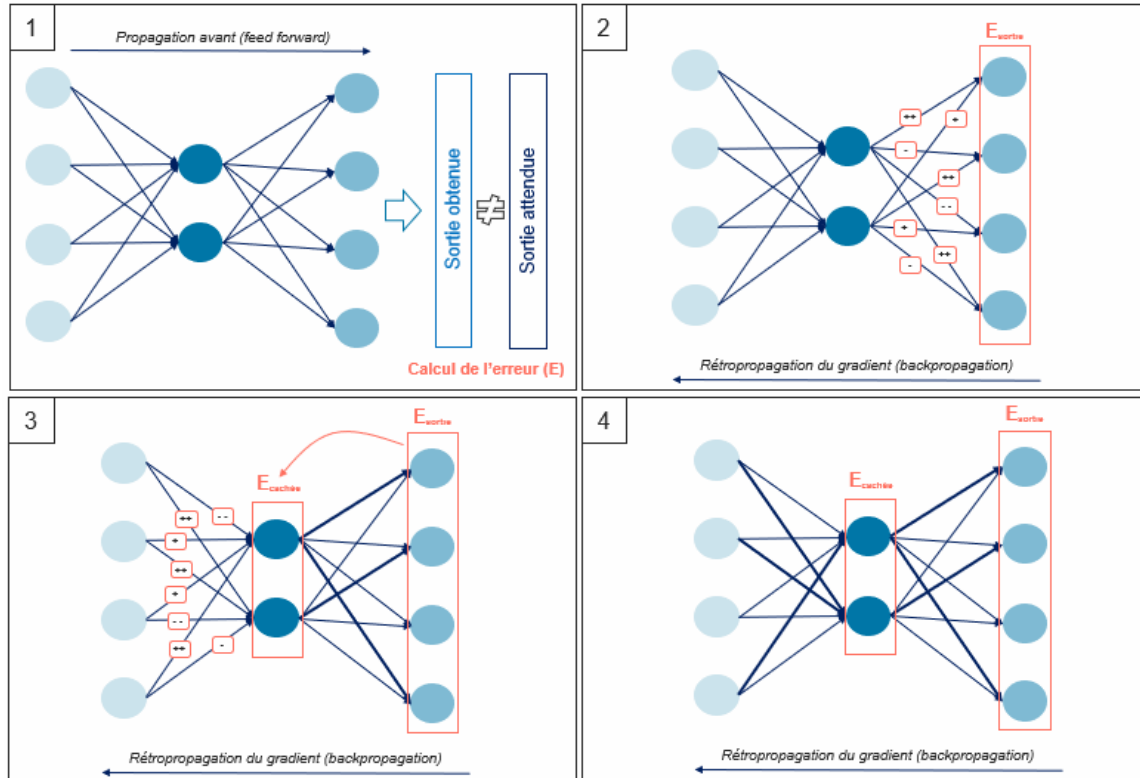


Figure II.5 – Représentation schématique des 4 étapes de la rétropropagation de l'erreur lors de la phase d'apprentissage d'un réseau de neurones

A noter qu'un poids ayant eu un impact majeur sur l'erreur sera de ce fait, plus lourdement modifié qu'un poids ayant eu un impact minime. L'objectif de la rétropropagation de l'erreur est de minimiser, au fur et à mesure des itérations, l'erreur globale calculée et donc d'avoir les meilleurs poids possibles.

Afin d'aider à la modification de ces poids, il existe en Machine Learning de nombreux **algorithmes d'optimisations**. Ces algorithmes ont pour rôle de favoriser la minimisation de l'erreur selon différentes stratégies, par exemples adaptatives. Dans le cadre du projet IAC, nous avons utilisé l'optimiseur Adam [Kingma et Ba, 2014], un des plus répandu en Machine Learning, qui s'est avéré pertinent dans le contexte de nos travaux.

#### II.A.2.c. Phase d'apprentissage : hyperparamètres

De plus, pour pouvoir entrainer notre modèle, il nous faut définir plusieurs paramètres primordiaux – les hyperparamètres [Hazan et al., 2017] – à commencer par **le taux ou pas d'apprentissage**, communément appelé **Learning Rate** en Machine Learning. Ce taux peut être assimilé à la vitesse d'apprentissage du réseau. La vitesse sert à la mise à jour des poids du réseau (i.e. leur valeur numérique) en ayant un impact plus ou moins fort. Plus le *Learning Rate* sera petit plus le réseau apprendra lentement et inversement, plus celui-ci sera grand plus le réseau apprendra vite. Néanmoins il est important de savoir le doser. En effet, l'ajustement des poids du réseau tend à atteindre un minimum, c'est-à-dire la configuration où l'erreur est minimale, et donc où nous considérons notre réseau comme le meilleur possible pour effectuer la tâche demandée. Dans ce cas précis, un Learning Rate trop grand ferait que le réseau pourrait se modifier trop « brutalement » et nous ne pourrions jamais atteindre ce minimum. Dans le cas contraire, un Learning Rate trop petit pourrait faire que notre réseau n'atteindrait jamais ce minimum non plus.

Un autre hyperparamètre inhérent à l'apprentissage d'un réseau est **le nombre d'apprentissages** que celui-ci va devoir effectuer : les **epochs**. Un epoch reflète une itération de l'apprentissage, c'est-à-dire un entraînement du réseau sur l'ensemble du corpus de données lui ayant été fourni en entrée. Plus le nombre d'*epochs* est grand plus le réseau verra et apprendra des règles liées données. Bien entendu, plus le nombre d'epochs est grand plus le temps d'apprentissage l'est également.

Le choix de tous ces hyperparamètres est essentiel lors de la conception d'un réseau de neurones artificiels car ils sont les premiers garants d'un apprentissage efficace et précis. Néanmoins, pour pouvoir affirmer qu'un apprentissage a été efficace, précis et qu'il est terminé, nous devons l'évaluer en implémentant différentes métriques pour les phases d'apprentissage, de validation et de test.

#### II.A.2.d. Phases de validation et test

Enfin, l'apprentissage d'un réseau de neurones ne peut être complet sans une phase de validation et de test. La **phase de validation** intervient à la fin de chaque *epoch* sur des données que le réseau n'a jamais rencontrées et donc jamais apprises. Lors de cette phase, le réseau va ainsi vérifier son apprentissage et générer des statistiques de validation tels que la précision ou le taux d'erreur par exemple, i.e. *accuracy*, qui nous permet de mesurer la fréquence à laquelle notre réseau effectue des prédictions correctes [Accuracy, 2020].

La **phase de test**, quant à elle, intervient après tous les epochs d'apprentissage. Il s'agit ici de récupérer le modèle final, i.e. le modèle entraîné qui peut être utilisé en l'état, et de tester une fois de plus son apprentissage sur de nouvelles données jamais rencontrées. Lors de cette phase, le modèle va donc prédire des résultats selon les données fournies en entrées. Il n'y a aucun ajustement de poids ou modification d'un quelconque paramètre du réseau car seule la propagation avant est effectuée (cf. figure II.3). L'objectif de la phase de test est de valider l'apprentissage de notre réseau de neurones et surtout de vérifier sa capacité à généraliser ce qu'il a appris sur de nouvelles données qu'il n'a jamais vu auparavant. Cette phase est primordiale car « *un modèle entraîné ne veut pas dire un modèle pertinent* » [Labadille, 2017].

#### II.A.3. Implémentation en Python 3.6 et Keras

Dans cette section, nous allons présenter l'implémentation du réseau de neurones artificiels précédemment exposé en figure II.1, avant de passer sur le réseau de neurones artificiels bien plus complexe qui a été implémenté dans le cadre du projet IAC. Notre objectif dans cette section est de présenter une implémentation d'un exemple simple de réseau de neurones artificiels afin de poser les bases de l'utilisation de la bibliothèque Keras.

Il existe de nombreux logiciels et bibliothèques, principalement écrits en langage Python ou C++, destinés à aider les développeurs à concevoir des réseaux de neurones artificiels. Dans le cadre de nos travaux, nous avons fait le choix d'utiliser le langage **Python en version 3.6** ainsi que la **bibliothèque Keras**.

Keras est une bibliothèque Python open source créée par Google AI et conçue pour permettre aux développeurs de créer et d'expérimenter rapidement des réseaux de neurones artificiels, en offrant des interactions optimisées avec des outils de Machine Learning.

La bibliothèque Keras peut être vue comme une interface connectée à une solution backend sans laquelle elle ne fonctionnerait pas, telle que Tensorflow<sup>8</sup>. La bibliothèque Keras permet notamment une gestion simplifiée et optimisée des ressources utilisées sur une machine (par exemple le GPU) lors des nombreux calculs effectués. Souhaitant pouvoir tester notre approche rapidement, nous avons donc fait le choix de cette technologie dans le cadre du projet IAC.

Reprenons désormais le réseau de neurones artificiels présenté en figure II.1. L'implémentation de celui-ci via Keras se fait comme suit :

```
model = Model()
model.add(Dense(4, input_dim=X, activation='sigmoid'))
model.add(Dense(2, activation='relu'))
model.add(Dense(4, activation='sigmoid'))
model.compile(loss='mean_squared_error', optimizer='adam', metrics=['accuracy'])
```

Figure II.6 - Implémentation avec Keras du réseau de neurones présenté en figure II.1

Comme le montre la figure II.6 ci-dessus, la première étape est de créer un objet « *model* » en faisant appel à la classe *Model* de Keras. Puis nous lui ajoutons successivement les différentes couches qui le compose en commençant par la couche d'entrée. A la création de chaque couche, il est alors impératif de spécifier la nature des neurones qui la composent. Ici, les couches de type *Dense* signifient l'utilisation de neurones formels qui sont entièrement connectés à l'ensemble des neurones de la couche précédente et suivante.

Pour chaque couche il faut alors à minima définir le nombre de neurones (1<sup>er</sup> paramètre). La couche d'entrée prend un deuxième paramètre obligatoire qui est la dimension de la donnée qui lui sera fournie, les autres couches prenant automatiquement en compte ce même format. Enfin, il est nécessaire de passer en paramètre la fonction d'activation qui sera utilisée par chaque neurone, ici les fonctions *sigmoïde* et *ReLU*.

Une fois le modèle créé selon la structure choisie, il faut le compiler en indiquant obligatoirement une fonction de perte et une fonction d'optimisation, respectivement les paramètres *loss* et *optimizer*. De plus, si nous le souhaitons, nous pouvons indiquer une liste de métriques qui seront utilisées (paramètre *metrics*) lors de la phase d'apprentissage. Dans l'exemple utilisé ici, nous avons choisi d'utiliser une métrique sur la précision des prédictions de notre modèle. A ce stade, le réseau est prêt à être entraîné puis testé.

---

<sup>8</sup> Outil d'apprentissage automatique développé par Google parmi les plus utilisés de la communauté IA

## II.B. Réseaux de neurones récurrents, RNN

Dans le cadre du projet IAC, nous avons donc implémenté notre propre réseau de neurones artificiels en nous basant sur une architecture de modèle déjà éprouvée lors de précédents travaux de recherche menés par [Chraïbi Kaadoud \[2018\]](#) sur l'extraction de règles dans des réseaux récurrents.

### II.B.1. Principe général

Notre approche repose sur les mêmes concepts fondamentaux présentés dans la section précédente, mais sur une structure de réseau plus complexe : **les réseaux de neurones récurrents**, autrement appelés **RNN** pour *Recurrent Neural Networks*.

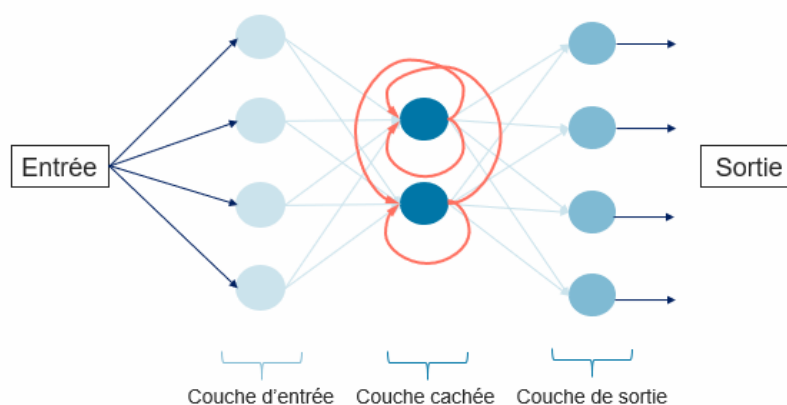


Figure II.7 - Structure d'un réseau de neurones récurrents, RNN

La spécificité de ces réseaux, présentés en figure II.7, est qu'ils possèdent, comme leur nom l'indique, des **connexions récurrentes** formant des cycles contrairement aux réseaux de neurones classiques. Le choix d'utiliser de tels réseaux vient de notre besoin de pouvoir tenir compte du temps et du passé lors de l'apprentissage du réseau sur des données séquentielles.

Rappelons-le, notre besoin métier est de trouver une méthode d'extraction de l'expertise métier de développeurs Java à l'aide des réseaux de neurones. Nous cherchons au travers du code écrit par ces derniers à extraire des règles implicites qui régissent la structure du code, la succession des éléments, etc. Néanmoins, ces règles ne sont pas simples d'un point de vue humain. En effet, les choix effectués par un développeur lors de la conception dépendent bien souvent de nombreux facteurs (tels que des règles métiers explicites à respecter), mais aussi de son expérience et de ses biais, ce qui aura obligatoirement une influence.

En partant de ce postulat, nous comprenons donc que les règles que nous cherchons à extraire sont très complexes et se construisent avec le temps au sein du réseau et après un apprentissage long de nombreuses données. Il nous était alors nécessaire de trouver une structure de réseau de neurones pouvant tenir compte de ces longues temporalités. De ce fait, l'utilisation de RNN semble être un choix adéquat dans la conception d'un réseau car il encode implicitement durant son apprentissage une représentation interne des **dépendances temporelles** présentes dans les séquences. Plus concrètement, les boucles situées sur les neurones de la couche cachée permettent au réseau d'avoir une mémoire d'une itération. Ce faisant, au prochain pas de temps lors de l'apprentissage, le réseau sera en mesure de se servir directement de ce qu'il a appris au pas de temps précédent pour fournir son résultat. En d'autres termes, le réseau peut garder une trace du passé récent, nécessaire pour faire la bonne prédiction à l'instant  $t$ .

En revanche, la particularité du réseau de neurones que nous avons implémenté pour le projet IAC est bien plus spécifique. Il nous fallait pouvoir augmenter la mémoire de notre réseau pour qu'il puisse prendre en compte un « passé » plus lointain encore, tel qu'un développeur pourrait le faire en se servant d'expériences passées récentes mais aussi plus anciennes. Ainsi, nous avons choisi d'utiliser des unités particulièrement complexes : les **LSTM**, pour *Long Short Term Memory*.

## II.B.2. Long Short Term Memory, LSTM

Les **LSTM** – *Long Short Term Memory*<sup>9</sup> – sont des unités de calcul complexes qui se constituent d'un système de portes et de cellules mémoires [Hochreiter et Schmidhuber, 1997]. Ils ont révolutionné le Machine Learning et le Deep Learning grâce à l'introduction de **cellules mémoires**. Ces unités permettent d'apporter au réseau un type de mémoire intermédiaire qui permet de garder les éléments pertinents et nécessaires à une bonne prédiction en mémoire sur de longs pas de temps notamment dans le cas de séquences longues.

En résumé et de manière simplifiée, un réseau qui utilise des LSTM possède **trois types de mémoires**. La première est la **mémoire à court terme**. Il s'agit d'une information ponctuelle liée au présent du réseau qui se reflète par l'activation des neurones calculée (leur valeur numérique) à chaque pas de temps. La deuxième mémoire est la **mémoire à moyen terme**, reflétée donc, dans les cellules mémoires comme nous l'avons indiqué précédemment. Enfin, la troisième mémoire est celle des poids du réseau. Elle représente l'expérience accumulée par le réseau lors de son apprentissage et peut être, par extension, assimilée à la **mémoire à long terme**. En effet, une fois l'apprentissage du réseau terminé, les poids ne sont plus mis à jour, ils sont alors sauvegardés pour être ensuite utilisés lors des phases de prédictions (cf. figure II.3).

Bien qu'ils aient été créés en 1997, ils n'avaient été jusqu'à récemment que très peu utilisés. Ils ont été remis au goût du jour dans les années 2010 avec l'explosion des volumes de données, le développement de nouveaux processeurs et les nouvelles puissances de calcul dont nous disposons. Néanmoins, le véritable engouement autour de ces unités de calcul particulières a été constaté à partir de 2015 où une véritable prise de conscience de leur potentiel a eu lieu. Depuis lors, les LSTM sont de plus en plus utilisés dans des travaux impliquant des séquences de tailles variables où les dépendances entre les éléments sont déterminantes pour les prédictions (exemple : la traduction).

Dans le cadre de nos travaux, et de plus en plus aujourd'hui dans la littérature scientifique, la version des unités LSTM utilisée est celle proposée par Gers en 1999 qui introduit la notion de porte d'oubli sur laquelle nous reviendrons juste après [Gers et al., 1999].

Bien que leur structure soit plus complexe que celle d'un réseau de neurones traditionnel, les grands principes restent inchangés et il s'avère que les LSTM réalisent des calculs simples. Le RNN que nous avons implémenté dans le cadre du projet se compose d'une couche cachée dotée « d'unités LSTM ». Une unité LSTM est donc un neurone artificiel particulier qui s'articule selon un système de portes et de cellules mémoires tel que présentée dans la figure II.8.

---

<sup>9</sup> Dans la littérature francophone, le terme *Long Short Term Memory* domine car peu de traductions françaises existent ou sont réellement utilisées. Par conséquent nous l'utiliserons dans son format anglais.

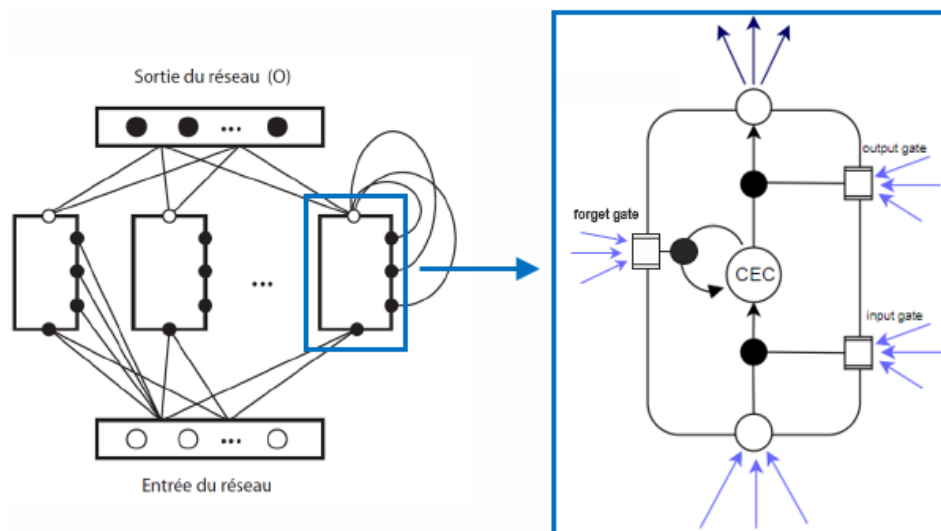


Figure 11.8 - Structure simplifiée d'une unité LSTM présente dans la couche cachée d'un RNN. Dans le réseau de neurone doté d'une couche cachée d'unités LSTM, chaque cercle blanc est lié par un poids à chaque cercle noir

Contrairement à un neurone formel, une unité LSTM se compose d'une entrée et d'une sortie, mais aussi de **trois portes** (entrée, oubli et sortie) et d'une unité de calcul linéaire dotée d'une connexion récurrente à elle-même, le **Carrousel d'Erreur Constant (CEC)**. La particularité des éléments d'une unité LSTM dans notre réseau récurrent est que chacun de ces éléments prend en entrée :

- Les données provenant des neurones de la couche précédente ;
- Les données fournies en sortie (au pas de temps  $t-1$ ) de toutes les autres unités LSTM de la couche cachée, mais également d'elle-même grâce aux connexions récurrentes.

De plus, chaque élément d'une unité LSTM a une fonction bien précise lors de la phase d'apprentissage.

La **porte d'entrée**, notée *input gate*, a pour rôle d'extraire l'information de la donnée courante et de la traiter grâce à une fonction d'activation. La deuxième porte, la **porte d'oubli**, ou *forget gate*, est la porte qui décide quelle information doit être conservée dans la mémoire de l'unité ou non, c'est-à-dire l'entretenir ou l'oublier. Cette information est stockée dans le **CEC** qui prend en compte les informations du pas de temps actuel, mais aussi les informations stockées relatives au pas de temps précédent. Cette structure porte aussi le nom d'état de la cellule.

Enfin, la dernière porte, la **porte de sortie** (*output gate*) décide quel sera le prochain état caché de la cellule, en prenant compte des précédentes informations calculées au sein de l'unité LSTM en transformant une dernière fois les données encodées qui seront fournies en sortie de celle-ci.

La sortie de la cellule, représentée par le dernier cercle blanc en haut de l'unité LSTM sur le schéma 11.8, correspond à l'état de l'unité de la cellule. Les états de toutes les unités LSTM de la couche cachée sont appelés **patterns d'activité** et seront utilisés lors de la phase d'interprétabilité.

Pour plus de détails sur les LSTM et leur principe computationnel, le lecteur pourra se référer à [Hochreiter et Schmidhuber \[1997\]](#) pour une explication du principe computationnel en anglais ou [Chraïbi Kaadoud \[2018\]](#) en français.



### II.B.3. Modèle implémenté pour le projet IAC

Le réseau de neurones artificiels que nous avons implémenté dans le cadre du projet IAC reprend donc les nombreux concepts précédemment exposés. Ainsi, celui-ci est un **réseau de neurones récurrents** composé des 3 couches suivantes : une **couche d'entrée de 7 neurones formels**, une **couche cachée de 8 unités LSTM** et une **couche de sortie de 7 neurones formels**.

Par souci de clarté et de compréhension nous avons décidé de simplifier la visualisation de celui-ci dans la figure II.9.

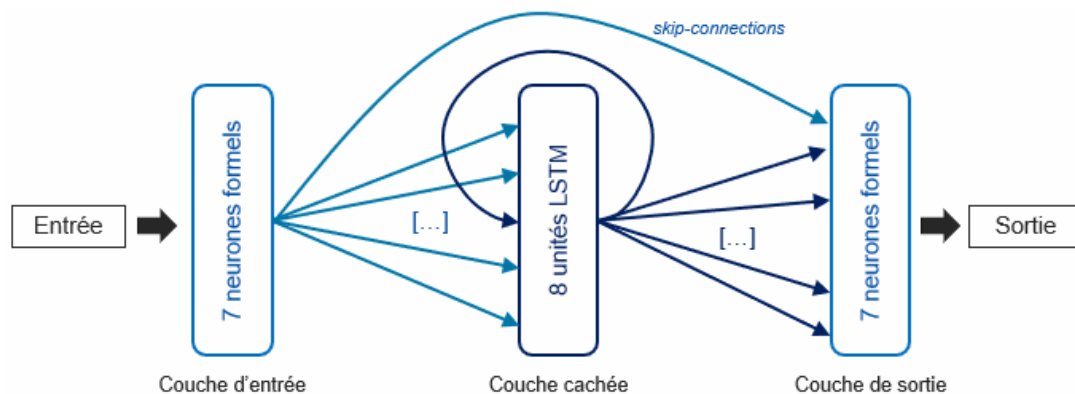


Figure II.9 - RNN-LSTM implémenté pour le projet IAC

En plus de l'interconnexion existante entre les neurones d'une couche et ceux de la couche suivante, nous avons choisi d'y ajouter des *skip-connections* représentées sur la figure II.9. Il s'agit de liaisons directes entre les neurones de la couche d'entrée et ceux de la couche de sortie. Cet ajout nous permet ainsi d'éliminer l'apparition possible d'un problème bien connu des réseaux de neurones : l'évanouissement du gradient. Bien que nous ne le présentions pas en détail, le principe est que la modification des poids du réseau n'a que très peu d'impact sur la sortie de celui-ci, provoquant ainsi un apprentissage non efficace.

L'implémentation, via Keras, de notre réseau de neurones récurrents doté de cellules LSTM, dont l'architecture est schématisée en figure II.9, est présentée dans la figure II.10 suivante.

```
input_tensor = Input(shape=(None, self.input_sequence_size))
input_layer = Dense(self.nb_neurons_for_input_layer, activation='sigmoid')(input_tensor)

lstm_layer, lstm_state_h, lstm_state_c = LSTM(units=self.nb_LSTM_cell_per_hidden_layer,
                                             return_sequences=True,
                                             return_state=True)(input_layer)

concat_input_and_hidden_layer_outputs = Concatenate()([input_layer, lstm_layer])

output_layer = Dense(self.nb_neurons_for_output_layer, activation='sigmoid',
                    name='output_layer')(concat_input_and_hidden_layer_outputs)

self.rnn_model = Model(inputs=[input_tensor], outputs=[output_layer, lstm_state_h, lstm_state_c])
```

Figure II.10 - Implémentation via Keras du RNN-LSTM du projet IAC



Comme nous pouvons le voir, il nous est nécessaire dans un premier temps de définir un vecteur, noté *input\_tensor* ayant pour rôle de transformer une dernière fois nos données au format requis par le modèle Keras.

Contrairement à l'exemple utilisé en figure II.6 pour montrer l'implémentation d'un modèle via Keras, notre architecture nous impose de créer en premier les différents éléments de notre réseau avant de les ajouter à notre modèle. De ce fait, nous avons créé la couche d'entrée de notre modèle, une couche de type *Dense*, à laquelle nous indiquons l'origine des données qu'elle recevra (dernier paramètre), dans notre cas : *l'input\_tensor*.

Nous créons ensuite notre couche cachée, nommée *lstm\_layer*, grâce à une couche de type LSTM fournie par la bibliothèque Keras. L'implémentation de cette couche est quelque peu complexe. En effet, afin de pouvoir exploiter les états des cellules LSTM, nous devons tout d'abord les récupérer. Pour ce faire, nous indiquons deux paramètres supplémentaires à la création de la couche : *return\_sequences* qui nous permet de stocker la sortie de la couche cachée et *return\_state* qui nous permet de récupérer l'état interne des cellules LSTM (i.e. les informations du CEC). Enfin, le dernier paramètre indique que les données fournies à cette couche cachée sont celles en sortie de la couche d'entrée.

Concernant l'implémentation des *skip-connections*, la solution mise en œuvre est la concaténation. Keras ne nous permettant pas d'implémenter deux entrées différentes pour une même couche (couche de sortie), nous avons tout d'abord dû concaténer les données venant de la couche d'entrée et de la couche cachée.

Nous avons ensuite créé la couche de sortie en précisant que les données fournies en entrée seront celles venant de la concaténation. Enfin, nous avons créé notre modèle auquel nous avons précisé l'entrée, *l'input\_tensor*, et les sorties qui sont respectivement la couche de sortie, la sortie de la couche cachée et enfin l'état interne des cellules LSTM. La visualisation graphique de notre modèle fournie par Keras est présentée en Annexe 2 de ce mémoire.

L'implémentation d'un réseau de neurones artificiels dépend dans un premier temps de la tâche que l'on souhaite lui faire réaliser mais aussi des données étudiées.

## II.C. Approche et données étudiées

### II.C.1. Notre approche

Dans le cadre du projet IAC, nous avons implémenté notre propre réseau de neurones récurrents dans le but de lui faire **effectuer des prédictions sur des séquences**. Nous allons ensemble comprendre ce choix.

Notre but est d'essayer d'extraire des représentations, i.e. des règles, des connaissances implicites des développeurs Java présentes dans leur code. De ce fait, notre premier objectif était de trouver un moyen pour que notre réseau de neurones construise lui-même ses propres règles implicites au fur et à mesure qu'il apprend, à partir des données qu'il rencontre.

Dans une tâche de prédiction, le réseau de neurones se doit de savoir prédire la sortie attendue à partir de la donnée qui lui est fournie en entrée. A chaque pas de temps lors de la phase d'apprentissage, le réseau apprend donc les corrélations qui existent entre l'entrée et la sortie attendue afin de s'améliorer encore et encore.

Dans le cadre du Machine Learning, à aucun moment nous n'indiquons au réseau de neurones les liens qui existent entre ces données, le but étant qu'il se crée lui-même une représentation de ceux-ci. Nous lui indiquons simplement qu'il doit prédire un résultat suivant l'entrée et nous le comparons avec la sortie attendue afin de modifier ses poids.

Néanmoins, une problématique émerge de ce concept : comment s'assurer que le réseau se soit construit un « bon raisonnement » et que les règles implicites qu'il s'est lui-même créées suivent celles que nous cherchons à extraire du code Java ?

Afin d'y répondre, nous avons choisi une approche en deux temps. Premièrement, nous avons décidé d'entraîner notre réseau de neurones récurrents sur des données dites artificielles, prouvées et éprouvées et ayant déjà été utilisées dans de précédents travaux de recherche [Omlin et Giles, 1996 ; Gers et al., 1999 ; Chraïbi Kaadoud, 2018]. Dans un second temps, après avoir validé cette première phase, nous avons fait apprendre notre réseau sur des données réelles : le code Java.

## II.C.2. Données artificielles : grammaire de Reber

Bien que notre étude soit principalement axée sur l'extraction de l'expertise métier de développeurs Java à travers leur code, il nous a été nécessaire de nous concentrer dans un premier temps sur un autre type de données : la **grammaire de Reber**.

En effet, il est admis dans la communauté informatique qu'il existe une multitude de façon d'arriver à une solution pour un même problème donné, et encore plus lorsqu'il s'agit de développement informatique. En effet, deux développeurs vont, très certainement, proposer pour un même besoin deux codes différents. Les résultats tout aussi similaires, si ce n'est identiques, découleront de ce fait de deux approches distinctes. Ainsi, une première problématique liée au code Java s'est posée : comment s'assurer que notre réseau apprenait les bonnes informations à partir des données ? Effectivement, comment tirer des conclusions et affirmer des faits issus d'un réseau dont nous ne pouvons prouver le bon fonctionnement ?

Il nous a donc fallu trouver une approche nous permettant de présenter des résultats basés sur une démarche rigoureuse et complète. Ainsi, l'utilisation d'une grammaire artificielle, la grammaire de Reber, s'est justifiée afin de valider notre approche.

### II.C.2.a. Présentation

**Une grammaire est un ensemble de règles**, i.e. un ensemble de transitions, propres à un langage permettant de décrire ses structures et son fonctionnement. Bien que l'apprentissage de certaines de ces règles puisse se faire de manière explicite, de nombreuses études ont prouvé qu'une personne, en acquérant des connaissances, apprenait en parallèle les régularités structurelles de manière implicite. Partant de ce postulat, Reber [1967] a décidé, lors de ses travaux sur l'apprentissage implicite, de créer une grammaire artificielle composée de symboles, formant des séquences, dont nous nous sommes servis aujourd'hui.

La **grammaire de Reber**, **RG** pour *Reber Grammar*, est une grammaire à états finis composée de **7 symboles** : **B, T, P, X, S, V et E**. Sa structure ressemble à un graphe orienté composé de nœuds et d'arcs de transition qui possèdent à la fois un sens et une étiquette (cf. figure II.11).

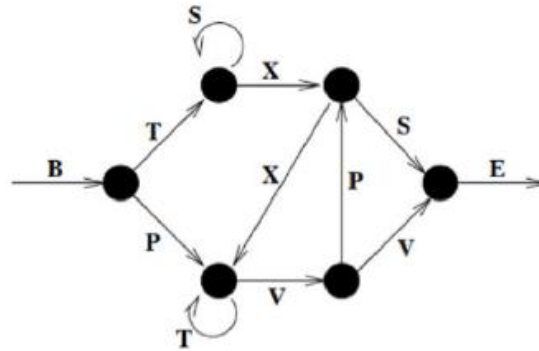


Figure II.11 - Grammaire de Reber représentée sous forme de graphe orienté

Le parcours de cette grammaire se fait comme indiqué par le graphe dans un certain ordre et sens, ce qui nous amène, en suivant les transitions, à effectuer des **séquences de symboles**. Ces séquences sont régies par des relations de causalité (règles) entre les symboles qui la composent, qu'ils soient proches (se suivent) ou non (aux extrémités de la séquence).

Par exemple, BTXSE est une séquence reconnue par la grammaire. En revanche, BTPVVVSE est une séquence incorrecte car la succession des symboles ne suit pas les règles de la grammaire. A noter que les séquences générées commenceront obligatoirement par un « B » pour *Begin* et finiront par un « E » pour *End*. De plus, à chaque nœud, la probabilité qu'une transition soit choisie plutôt qu'une autre est de 50%.

Cette grammaire est d'autant plus intéressante qu'elle propose, hormis pour les symboles de début et de fin, de pouvoir passer par un même symbole plusieurs fois mais dans un **contexte différent**. Par exemple, le symbole T peut venir après un B mais aussi après un P ou encore un X. Cette particularité permet donc d'induire des règles de transition implicites plus complexes qu'une simple succession de symboles.

Ce contexte, justement, est extrêmement important dans le cadre d'étude du projet IAC. En effet, cette prise en compte du contexte dans une séquence, que l'on qualifie de passé et qui peut s'avérer plus ou moins lointain, influence les règles implicites que le réseau encode. De plus, les séquences étudiées provenant de la grammaire de Reber et composées de 7 symboles possibles sont de longueur variable. La présence de boucles dans le graphe orienté en étant la cause. Ainsi, la prise en compte du passé que fera notre réseau lors de la phase d'apprentissage le mènera à **encoder une représentation implicite de ces règles** très complexes. De ce fait, le réseau sera amené à traiter des séquences simples de 5 ou 6 symboles, comme des séquences de grande taille telle que BT...SSSSXXTT...TVPXTVVE dont le passé sera potentiellement très grand selon le nombre de répétitions de S et de T.

#### II.C.2.b. Transformation des données

Comme indiqué précédemment, la grammaire de Reber nous a permis de générer des séquences composées des symboles B, T, P, X, S, V et E. Afin de pouvoir utiliser des séquences issues de la grammaire de Reber en données de notre réseau de neurones, nous avons dû les transformer en vecteurs numériques afin d'entraîner notre réseau à effectuer des prédictions.

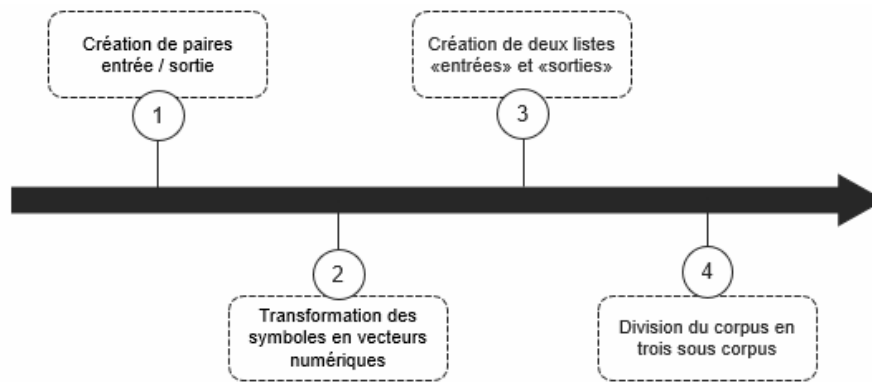


Figure II.12 - Etapes de transformation des données issues de la grammaire de Reber

La première étape (cf. figure II.12 – étape 1) de cette transformation a donc consisté à créer des paires : « symbole d’entrée / symbole attendu » à partir des séquences. Prenons pour exemple la séquence BTXSE, représentée sous forme de paires de symboles dans le tableau 1.

Paire n°	Symbole actuel	Symbole attendu
0	B	T
1	T	X
2	X	S
3	S	E

Tableau II.1 - Exemple de paires issues de la séquence BTXSE dans le cadre de la grammaire de Reber

Le symbole actuel est le symbole qui sera fourni au réseau au pas de temps « t » et le symbole attendu sera celui qui est attendu en sortie du réseau, mais aussi celui d’entrée du réseau au pas de temps suivant « t+1 ». Ce schéma de transformation est alors répété pour l’ensemble des séquences du corpus de données utilisé. A noter que le dernier symbole de chaque séquence est un « E », ce symbole n’est donc jamais utilisé en entrée du réseau car il ne demande aucune prédiction.

Une fois ceci fait, il est nécessaire de transformer chaque symbole, initialement au format texte, en un vecteur numérique de taille 7, qui sera fourni à la couche d’entrée du réseau de neurones (cf. figure II.12 – étape 2). Pour ce faire, il faut ordonner les symboles en attribuant à chacun d’eux une position dans le vecteur. Ainsi, nous avons utilisé une technique de « *one-hot encoding* » qui permet de générer des vecteurs binaires avec une seule unité non nulle correspondant à un symbole. Ces associations sont représentées dans le tableau ci-après.

Symbole au format texte	Vecteur binaire associé
B	[1, 0, 0, 0, 0, 0, 0]
T	[0, 1, 0, 0, 0, 0, 0]
S	[0, 0, 1, 0, 0, 0, 0]
X	[0, 0, 0, 1, 0, 0, 0]
V	[0, 0, 0, 0, 1, 0, 0]
P	[0, 0, 0, 0, 0, 1, 0]
E	[0, 0, 0, 0, 0, 0, 1]

Tableau II.2 - Association des symboles issue de la grammaire de Reber à des vecteurs binaires

L'étape suivante (cf. figure II.12 – étape 3) a été de mettre dans deux listes distinctes les symboles d'entrée et les symboles attendus transformés au format vectoriel. Ces deux listes seraient pour notre exemple précédent les suivantes : liste d'entrées = [B, T, X, S] et liste de sorties attendues = [T, X, S, E]. Enfin, la dernière étape principale de transformation de ces données a été de convertir ces vecteurs numériques en matrices à une dimension ; format utilisable par notre réseau.

Finalement, nous avons divisé notre corpus de données, composé de séquences, en trois sous-corpus : un premier utilisé lors de la phase d'apprentissage, un deuxième pour la phase de validation et un dernier utilisé pour la phase de test (cf. figure II.12 – étape 4). La répartition des données dans chacun de ces trois corpus peut s'effectuer selon plusieurs techniques ou méthodes. Nous avons choisi, dans le cadre de la grammaire de Reber, d'utiliser un corpus initial de 125 000 séquences réparties comme suit : 100 000 dans le corpus d'apprentissage et 12 500 dans chacun des corpus de validation et de test. Les pourcentages de répartition, 80/10/10, correspondent aux standards des travaux de Machine Learning qui préconisent 80% du corpus initial destiné à l'apprentissage et 20% pour la phase de test.

### II.C.3. Données réelles : code Java

Contrairement à la grammaire de Reber, des données issues de code Java ne sont pas, du fait de leur nature, structurées dans leur ensemble selon des règles explicites. Nous retrouvons d'une part une dimension explicite constituée des contraintes liées au langage, à la structure, etc. La seconde dimension, qui nous intéresse le plus dans le cadre de nos travaux est implicite. Il s'agit dans ce cas de « reflets » de l'expérience, du vécu et des habitudes des développeurs ayant élaboré le code. A titre d'exemple, pour une même tâche donnée deux développeurs vont implémenter deux codes distincts.

Afin de pouvoir tester notre approche et effectuer un apprentissage pertinent sur notre réseau de neurones, nous avons choisi de **vérifier la transposabilité de notre approche** dans un cas de données réelles. Le code Java utilisé est un corpus provenant de multiples projets Java disponibles sur GitHub<sup>10</sup>, qui a été nettoyé afin de garder des lignes de code d'une qualité supérieure à la moyenne (c'est-à-dire sans lignes de code « mort » ou inutilisées).

Néanmoins, afin d'utiliser ce code, il nous a été nécessaire de le transformer dans un format interprétable par notre réseau de neurones. Cette étape de transformation a été réalisée par un collaborateur onepoint du cercle R&D. Pour ce faire, il a mis en place une stratégie (que nous ne détaillerons pas ici car non implémentée par nous-mêmes) liée au traitement automatique du langage (NLP pour *Natural Language Processing*) afin de transformer le code Java initial en séquences de mots Java, autrement appelés *tokens* dans le domaine du NLP. A titre d'exemple, et dans un souci de clarté, la figure II.13 ci-dessous illustre le passage du code Java initial en séquences de mots.

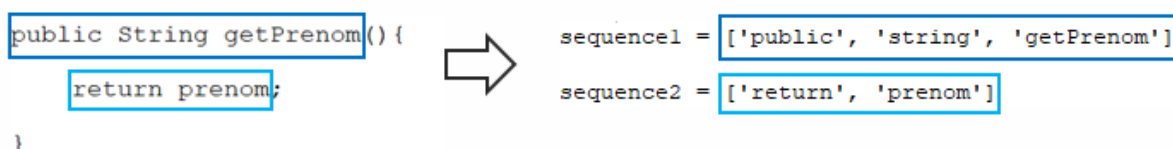



Figure II.13 - Transformation d'un accesseur Java en séquences de mots Java

<sup>10</sup> Corpus Java : <http://groups.inf.ed.ac.uk/cup/javaGithub/>

Une fois toutes les lignes de code Java transformées en séquences de mots, il nous a fallu les **encapsuler**. En effet, une séquence est définie comme un enchainement de symboles ou de mots ayant un début et une fin. Or, à partir de mots Java, il est impossible de déterminer le début et la fin d'une séquence. Le début d'une séquence qui correspond à une ligne de code pourrait éventuellement être défini par l'un des nombreux mots clés du langage comme par exemple : *private*, *public*, *protected*, etc. mais cela demanderait bien trop de ressources pour pouvoir tous les tester. Concernant la fin d'une séquence, il est impossible de définir de manière binaire si un mot est la fin d'une séquence ou non. De ce fait, nous avons décidé d'ajouter arbitrairement un mot de début et un mot de fin à chaque séquence Java préalablement récupérée.

Pour des questions de clarté et de cohérence avec les séquences issues de la grammaire de Reber, et afin d'être certains de ne pas dénaturer les séquences Java, nous avons choisi arbitrairement les symboles B et E pour designer *Begin* et *End*, respectivement en symbole de début et de fin de séquence. Par conséquent, en reprenant l'exemple de la figure II.13, les séquences deviennent alors :

```
sequence1 = ['public', 'string', 'getPrenom']
sequence2 = ['return', 'prenom']
```



```
sequence1 = ['B', 'public', 'string', 'getPrenom', 'E']
sequence2 = ['B', 'return', 'prenom', 'E']
```

Figure II.14 - Ajout de symboles de début et de fin à des séquences de mots Java

Contrairement aux séquences issues de la grammaire de Reber que nous avons évoquées précédemment, la transformation des séquences Java en vecteurs numériques a été réalisée par un collaborateur extérieur à l'équipe IAC. Nous avons donc directement pu utiliser la correspondance numérique de chaque mot. Ainsi, chaque mot Java de chaque séquence a été transformé en un vecteur numérique de taille 64.

A ces vecteurs initiaux, deux nouvelles valeurs ont été ajoutées afin de pouvoir identifier de façon numérique les symboles B et E. Les vecteurs numériques sont tous alors de taille 66.

#### II.C.4. Premiers résultats

	Grammaire de Reber	Code Java
<b>Corpus d'apprentissage</b>		
Nombre de séquences	3 000	1 000
Nombre de paires	56 678	2 543
<b>Corpus de validation</b>		
Nombre de séquences	1 000	250
Nombre de paires	16 169	1 209
<b>Corpus de test</b>		
Nombre de séquences	1 000	250
Nombre de paires	22 435	1 283

Tableau II.3 - Caractéristiques des corpus utilisés pour les phases d'apprentissage, de validation et de test pour la grammaire de Reber et le code Java. Une paire est constituée du symbole d'entrée et du symbole attendu en sortie

Afin de mener à bien nos travaux de recherche en suivant notre approche générique, nous avons donc utilisé deux corpus de données : la grammaire de Reber, notre corpus de données artificielles et le code Java, notre corpus de données réelles. La taille des corpus de données utilisés est présentée dans le tableau II.3 ci-dessus. Bien que nous ayons testés dans le cadre de nos travaux différentes tailles de corpus de données, dans la section suivante nous nous concentrons sur les résultats les plus probants pour plus de clarté et de cohérence.

Nous avons ainsi conçu notre réseau de neurones récurrents afin qu'il apprenne à effectuer des prédictions en tenant compte du passé, i.e. contexte, et des séquences déjà apprises. Lors de la phase d'apprentissage de notre réseau, nous avons effectué de nombreuses simulations selon différents paramétrages et avons choisi de présenter dans ce mémoire nos meilleurs et plus intéressants résultats.

Afin d'évaluer la pertinence de l'apprentissage de notre réseau nous avons mis en place différentes métriques précédemment présentées qui sont notamment, la *loss* et l'*accuracy*. Pour rappel, le signe d'un bon apprentissage est une courbe de *loss* qui décroît et une courbe d'*accuracy* qui croît. Nous avons calculé à la fin de chaque epoch ces deux métriques à divers endroits de notre réseau comme par exemple la sortie de celui-ci ou la sortie de la couche cachée. Par souci de clarté, nous faisons le choix de ne présenter ici que les courbes obtenues au niveau de la sortie de notre réseau.

En premier lieu, nous avons étudié les courbes d'erreurs, i.e. *loss*, de notre réseau selon deux *Learning Rate* pour en mesurer son impact. Ces courbes présentées en figure II.15 démontrent l'importance du choix du *Learning Rate*. Comme nous pouvons l'observer, les courbes de couleur rouge représentent les courbes d'erreurs calculés lors de l'apprentissage pour un *Learning Rate* de 0,1 (cf. figure II.15 – a) et de 0,01 (cf. figure II.15 – b) dont le comportement change significativement selon celui-ci.

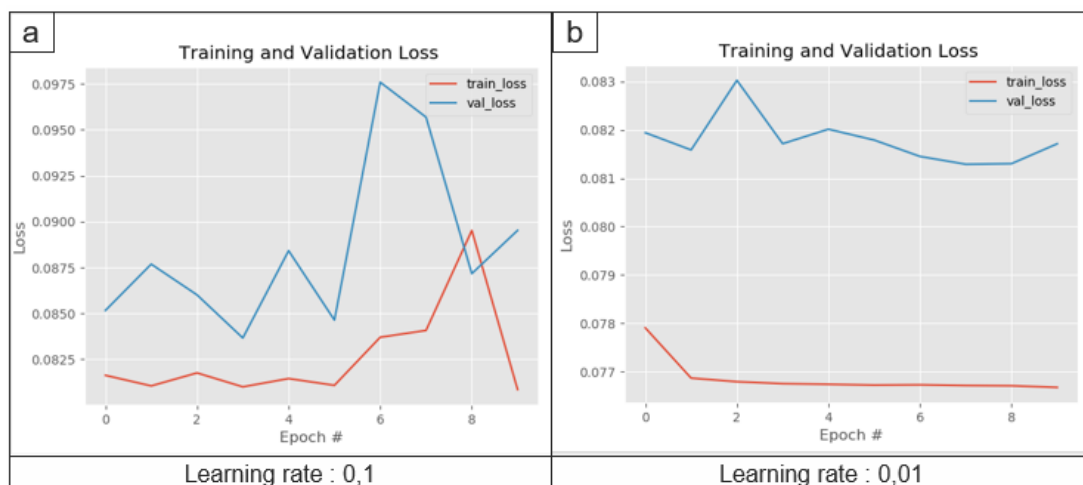


Figure II.15 - Apprentissage sur la grammaire de Reber : Courbes d'erreur (*loss*) du RNN pour un apprentissage sur 3000 séquences pour un learning rate de 0,1 (a) et de 0,01 (b) sur 10 epochs.

Nous le voyons, dans le cadre du *Learning Rate* le plus élevé, nous pouvons affirmer que l'apprentissage de notre réseau est mauvais car la courbe de *loss* augmente mais surtout varie fortement d'un *epoch* à l'autre. En revanche, la courbe obtenue pour un *Learning Rate* de 0,01 est très satisfaisante : nous observons que l'erreur diminue à chaque epoch de manière régulière à partir de la fin du premier *epoch*. Partant de cette observation, nous avons une première métrique significative indiquant un bon apprentissage du réseau. Nous avons en parallèle tenu compte des courbes d'*accuracy* de notre RNN lors de son apprentissage et celles-ci nous ont confortées dans notre conclusion (cf. figure II.16).

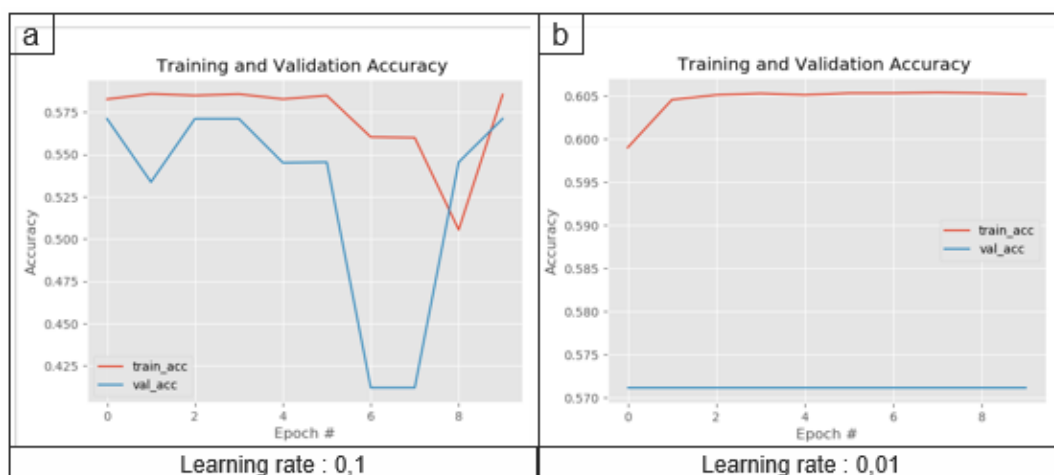


Figure II.16 - Apprentissage sur la grammaire de Reber : Courbes de précision (accuracy) du RNN pour un apprentissage sur 3000 séquences pour un learning rate de 0,1 (a) et de 0,01 (b) sur 10 epochs.

Dans le cadre de nos travaux nous avons également implémenté des métriques pour la phase de test de notre réseau telles que le critère de Gers ou le pourcentage de prédictions correctes que nous ne présenterons pas dans ce mémoire, car des simulations et analyses sont toujours en cours.

L'implémentation de métriques sur les phases d'apprentissage, de validation et de test a été réalisée dans un second temps sur notre RNN ayant appris à partir du code Java. Les principes d'analyses de notre démarche générique restent les mêmes. Ainsi, nous présentons en figure II.17 ci-dessous les meilleurs résultats obtenus pour un Learning Rate de 0,001.

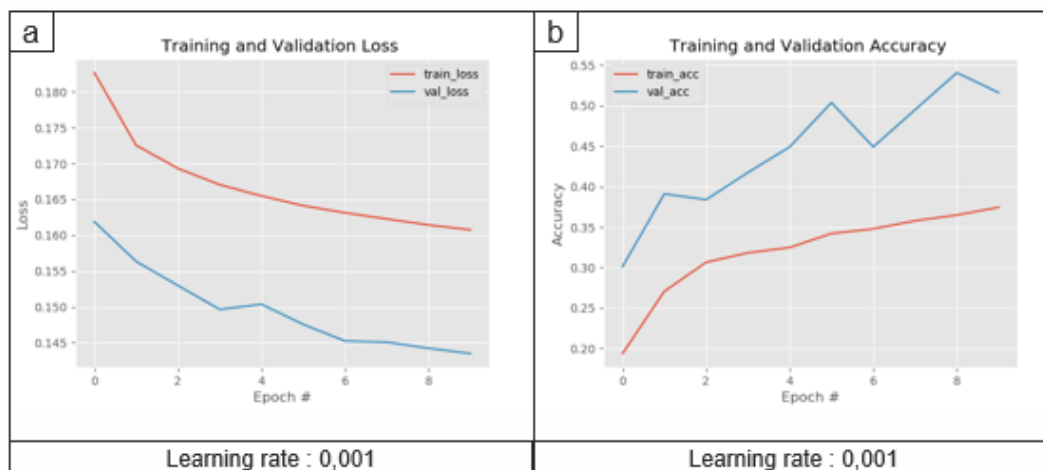


Figure II.17 - Apprentissage sur le code Java : Courbes de loss (a) et d'accuracy (b) du RNN pour un apprentissage sur 1000 séquences pour un learning rate de 0,001 sur 10 epochs.

Comme nous pouvons l'observer, les courbes de *loss* et d'*accuracy*, comme pour la grammaire de Reber, reflètent un très bon comportement de notre réseau lors de son apprentissage.

Avec ces premiers résultats prometteurs, nous avons décidé d'avancer dans nos démarches. Nous nous sommes donc concentrés sur la notion d'**interprétabilité des réseaux de neurones** et sur l'implémentation d'une de ces techniques afin d'**exploiter l'espace latent** du réseau implémenté précédemment. Nous allons aborder et détailler ces notions ainsi que les travaux portant sur eux dans le chapitre suivant.



### III. Projet IAC – interprétabilité et extraction de l’expertise

L’utilisation d’algorithmes de Machine Learning, et notamment les réseaux profonds liés au *Deep Learning*, soulève de nos jours de plus en plus de problématiques d’ordre éthique ou juridique [Hebling, 2019]. Ces problématiques sont principalement dues à l’**opacité** de certains algorithmes de Machine Learning tels que les réseaux de neurones artificiels. Elles amènent donc de plus en plus de chercheurs sur des **questions d’interprétabilité** et **d’explicabilité** de de ces algorithmes.

Dans le chapitre suivant, nous allons ainsi définir les notions d’explicabilité mais surtout d’interprétabilité des réseaux de neurones récurrents. Nous verrons ensemble la technique d’interprétabilité dite **post-hoc** que nous avons choisie d’implémenter dans le cadre du projet IAC. Nous présenterons ensuite les premiers résultats obtenus sur la grammaire de Reber puis sur les données issues du code Java. Nous verrons comment, à travers la question de l’interprétabilité des réseaux de neurones, nous avons pu tirer les premières conclusions sur l’extraction de l’expertise métier à l’aide de ces mêmes techniques. Enfin, nous discuterons de nos résultats mais aussi des futurs tâches et axes à traiter dans le cadre du projet.

#### III.A. Etat de l’art

##### III.A.1. De la boîte noire à l’IA explicable

« La principale différence entre l’IA des années 1970 et celle d’aujourd’hui est qu’il ne s’agit plus d’une approche déterministe » [Vérine et Mir, 2019].

En effet, l’implémentation des algorithmes d’IA a fortement évolué ces dernières années. Précédemment fondée en majorité sur les choix des développeurs, l’agencement des formules et des équations mathématiques utilisées, ces algorithmes et leurs prises de décision étaient fortement biaisés par les développeurs et leur propre façon de penser. En revanche, ces règles sont maintenant bien plus fondées sur les propres règles cachées dans les données à partir desquelles les algorithmes vont apprendre. Bien entendu, l’implémentation sera toujours influencée par le développeur mais la prise de décision de l’algorithme ne le sera pas tout autant. De ce fait, la prise de décision automatisée par ces algorithmes s’est opacifiée.

Ainsi représentée par la figure III.1 ci-dessous, un réseau de neurones est assimilé à une « boîte noire » dont nous connaissons les données en entrée et en sortie mais pas précisément le fonctionnement interne. Plus on étudie les couches cachées profondes d’un réseau (représentée sur notre figure par le cadre le plus foncé), plus la représentation des données devient abstraite et complexe à déchiffrer tout comme les règles implicites implémentées le sont également.

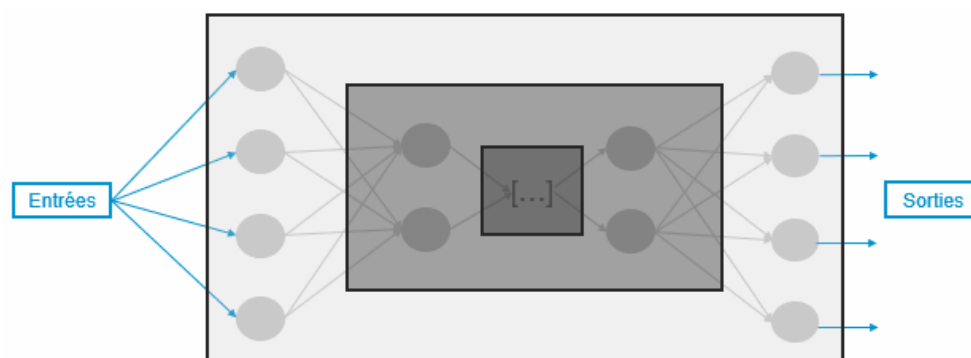


Figure III.1 - Représentation schématique d'un réseau de neurones selon le principe d'une « boîte noire ».

Ce phénomène est notamment dû à l'approche que les nombreux chercheurs et développeurs en IA ont eu pendant des années : la performance de ces algorithmes a été mise au centre des préoccupations. Néanmoins, il y a aujourd'hui une vraie **réorientation des problématiques** liées au Machine Learning, notamment le **besoin de les expliquer** [Crawford, 2019]. Les algorithmes d'IA étant de plus en plus utilisés dans des secteurs critiques comme la médecine ou la finance, il devient indispensable de réduire les biais moraux et éthiques présents dans les prises de décision.

Les prises de décision basées sur l'IA posent des questions cruciales et générales, notamment en termes d'acceptabilité de ces outils. Le besoin de confiance et de transparence est ainsi très présent et très prisé.

Effectivement, comment avoir confiance en la décision prise par un algorithme d'IA si nous ne pouvons expliquer comment celle-ci l'a été ? C'est dans ce contexte que de nouveaux travaux de recherche axés cette fois-ci autour de **l'IA interprétable et explicable** ont vu le jour. De plus, l'entrée en vigueur de la RGPD<sup>11</sup> et particulièrement l'article 22-1 qui stipule qu'une décision ne peut être fondée exclusivement sur un traitement automatisé, ont fortement accéléré les recherches dans le domaine.

### III.A.2. Interprétabilité ou explicabilité

Le besoin de transparence et de confiance dans les algorithmes de Machine Learning tels que les réseaux de neurones mais aussi leur complexité a fait émerger deux concepts : l'interprétabilité et l'explicabilité. Souvent associés dans la littérature, nous prenons ici le soin de les différencier et de les définir selon les termes suivants.

**L'interprétabilité** consiste à fournir une information représentant le raisonnement de l'algorithme de Machine Learning dans un format interprétable par un expert du Machine Learning ou des données. Le résultat fourni est fortement lié aux données utilisées et nécessite une connaissance de celles-ci, mais aussi du modèle utilisé [Gilpin et al., 2018].

**L'explicabilité** quant à elle consiste à fournir une information dans un format sémantique complet se suffisant à lui-même et accessible à un utilisateur, qu'il soit néophyte ou technophile, et quel que soit son expertise en termes de Machine Learning [Gilpin et al., 2018].

En d'autres termes, l'interprétabilité répond à la question « comment » un algorithme prend-il une décision tandis que l'explicabilité tend à répondre à la question « pourquoi ». A noter que l'interprétabilité est la première étape à réaliser afin de faire de l'explicabilité. Un modèle explicable est donc interprétable mais l'inverse ne l'est pas automatiquement.

Dans le cadre du projet IAC nous nous concentrons tout particulièrement sur le premier concept qu'est l'interprétabilité. Nous essayons de fournir des résultats compréhensibles pour nous, experts des données et connaisseurs de notre modèle. Néanmoins, l'interprétabilité des réseaux de neurones artificiels est un champ de recherche vaste avec de nombreux prismes. Nous avons donc fait le choix dans nos travaux d'implémenter une technique d'interprétabilité en particulier **exploitant l'espace latent** de notre réseau après apprentissage : **l'interprétabilité post-hoc** afin d'accéder à la mémoire implicite du réseau de neurones.

---

<sup>11</sup> Règlement Général sur la Protection des Données

### III.A.3. Interprétabilité locale ou globale : différents usages

Pour rappel, les réseaux de neurones artificiels font certes partie des algorithmes de Machine Learning les plus précis cependant ils sont les moins transparents et interprétables (cf. figure III.2).

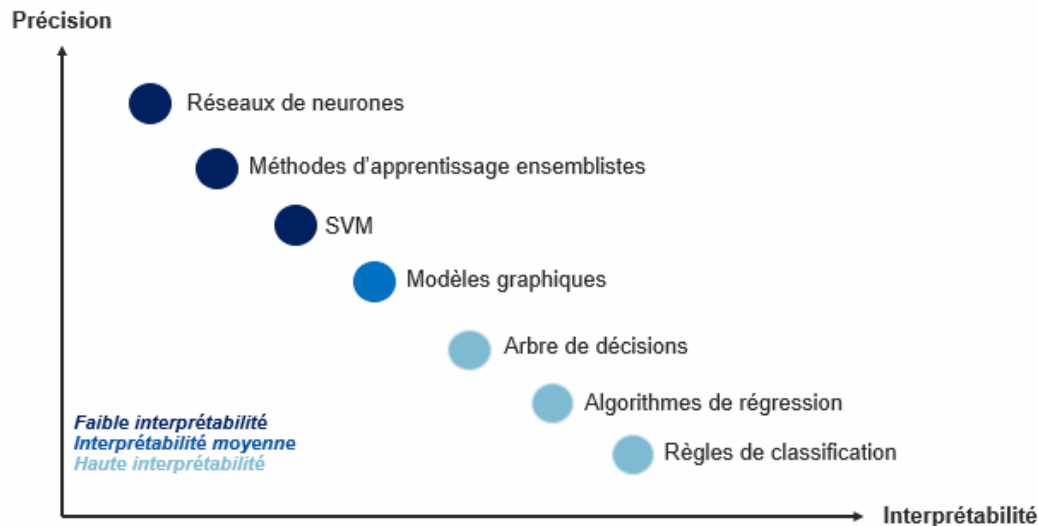


Figure III.2 – Niveaux d'interprétabilité des algorithmes de Machine Learning. Image adaptée à partir de [Dam et al. \[2018\]](#)

Lorsque l'on évoque l'interprétabilité des réseaux de neurones artificiels il est alors primordial de savoir ce que nous cherchons à expliquer et l'usage que nous souhaitons en faire. En effet, les techniques qui en découlent se distinguent selon plusieurs critères.

Tout d'abord, il est nécessaire de savoir quel type de comportement nécessite d'être analysé. Cherchons-nous à fournir une explication au comportement de la totalité du modèle ou en revanche son comportement sur un résultat en particulier ?

La première approche, nommée **interprétabilité globale** tend donc à fournir une explication sur le comportement global du réseau et ce, sur l'ensemble des données qu'il a apprises. Il permet de rendre le processus de prise de décision transparent pour toutes les données et s'avère être un moyen précieux d'évaluer la pertinence de ce que le modèle a appris [[Clapaud, 2019](#)].

D'autre part, l'**interprétabilité locale** tend à fournir une explication pour un résultat précis, c'est-à-dire pour une décision en particulier sur une échelle très réduite. Elle est particulièrement pertinente lorsqu'il est nécessaire d'analyser un cas en particulier pour des besoins clients par exemple [[Guidotti, 2018](#)].

Le deuxième usage qu'il est nécessaire de déterminer consiste à définir le moment où l'extraction des connaissances du réseau sera faite. Si la phase d'interprétabilité est effectuée pendant l'apprentissage de celui-ci on parle de méthode pédagogique, en revanche, si **l'extraction se fait à posteriori de l'apprentissage** on parle de méthode de décomposition et, dans nos travaux de recherche en particulier, nous nous intéresserons au domaine de l'**interprétabilité post-hoc**.

#### III.A.4. L'interprétabilité au cœur du projet IAC

Dans le cadre du projet IAC notre but est d'accéder à la mémoire implicite de notre réseau de neurones afin d'en extraire les règles et représentations qu'il encode. Nous avons donc choisi d'implémenter **une technique d'interprétabilité post-hoc**, i.e. après apprentissage, afin d'extraire et d'analyser ces règles implicitement encodées par le réseau lors de son apprentissage. Néanmoins, lorsque nous essayons de définir le type d'interprétabilité que nous effectuons, i.e. une interprétabilité globale ou locale, la prise de position n'est pas tranchée, et ce, bien que nous cherchions à comprendre le raisonnement du réseau dans sa globalité, en nous concentrons uniquement sur l'espace latent de celui-ci et nous rapprochons de l'interprétabilité locale. Ce non-positionnement stricte n'est en revanche pas préjudiciable dans la suite de nos travaux.

Plus concrètement, nous cherchons à **extraire les informations situées au niveau de l'espace latent** de notre réseau de neurones. Ce champ d'étude est plus communément appelé *Rules Extraction* en Machine Learning et l'intérêt y est double. En effet, nous cherchons d'un point de vue scientifique à faire avancer les questionnements sur l'interprétabilité des réseaux de neurones artificiels en accédant à leur mémoire implicite, en complément duquel nous essayons de répondre à une problématique métier. Nous essayons d'extraire l'expertise métier des développeurs Java à travers le code informatique qu'ils ont pu écrire. Pour ce faire, nous pensons qu'utiliser les résultats issus de notre première approche est pertinente.

Lorsque notre réseau de neurones se crée son propre raisonnement au fur et à mesure qu'il apprend des données et améliore ses prédictions, nous supposons que celui-ci a été capable **d'encoder les règles cachées** directement **depuis les données** dans sa propre mémoire implicite. Ainsi, en utilisant une grammaire telle que la grammaire de Reber, nous avons souhaité valider notre intuition en retrouvant les règles de la grammaire d'origine dans les résultats fournis par notre méthode d'interprétabilité sans que nous les ayons explicitement spécifiées à notre réseau. En partant de cette hypothèse, nous pensons donc être en mesure de retrouver les règles cachées dans les données issues du code Java que nous ne connaissons pas explicitement. Concrètement, nous pensons qu'il nous est possible de récupérer les représentations des règles implicites issues du réseau pour extraire et formaliser les connaissances implicites des développeurs cachées dans leur code.

### III.B. Déroulé des travaux d'interprétabilité

A l'instar de la phase d'apprentissage de notre réseau, la phase d'interprétabilité du projet est également effectuée selon une approche générique.

Les grandes étapes de notre phase d'interprétabilité et du processus d'extraction des règles implicites à partir de notre réseau de neurones récurrent, présenté en figure III.3, a donc été réalisé dans deux contextes applicatifs que sont la grammaire de Reber et le code Java.

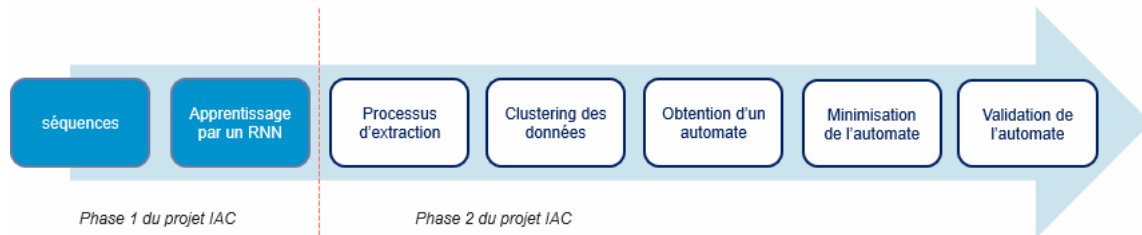


Figure III.3 - Présentation schématique de l'approche générique du processus d'extraction des règles à partir d'un RNN

A ce stade de nos travaux, nous avons donc un réseau de neurones entraîné sur les données adéquates dont il nous est possible de récupérer les représentations implicites situées au niveau de l'espace latent (cf. figure III.3 – phase 1). Pour cela nous allons essentiellement nous concentrer autour de l'analyse des vecteurs numériques enregistrés au niveau de la couche cachée lors de la phase de test. L'extraction de ces informations, ainsi que leur transformation, grâce à une technique d'interprétabilité post-hoc est ainsi possible. La phase d'interprétabilité du projet se décompose en cinq grandes parties selon la littérature [Omlin et Giles, 1996 ; Jacobsson, 2005 ; Wang et al., 2017].

Dans un premier temps nous allons extraire les informations situées au niveau de l'espace latent de notre réseau de neurones récurrents. Dans un deuxième temps nous allons regrouper les données selon leur caractéristiques communes à l'aide d'une technique de *clustering*. Le troisième temps consistera à générer un automate, i.e. un graphe, à partir de ces données. Puis, dans un quatrième temps, nous verrons ensemble en quoi consiste l'étape de minimisation de cet automate. Enfin, le dernier et cinquième temps de ce processus consistera à valider notre automate.

#### III.B.1. Extraction des états de la couche cachée du RNN-LSTM

##### III.B.1.a. Espace latent

Ainsi présenté précédemment, l'espace latent d'un réseau de neurones (cf. figure III.4) est le porteur majeur des représentations implicites des données qui se situent au niveau des couches cachées d'un réseau de neurones. Elles reflètent les règles que le réseau a encodé lors de son apprentissage, autrement dit ces règles constituent les représentations implicites des données [Cleermans, 1993]. Il s'agit de vecteurs numériques composés de huit valeurs (cf. figure III.4 – b) représentant respectivement chaque sortie, i.e. **pattern d'activité**, des unités LSTM de la couche cachée.

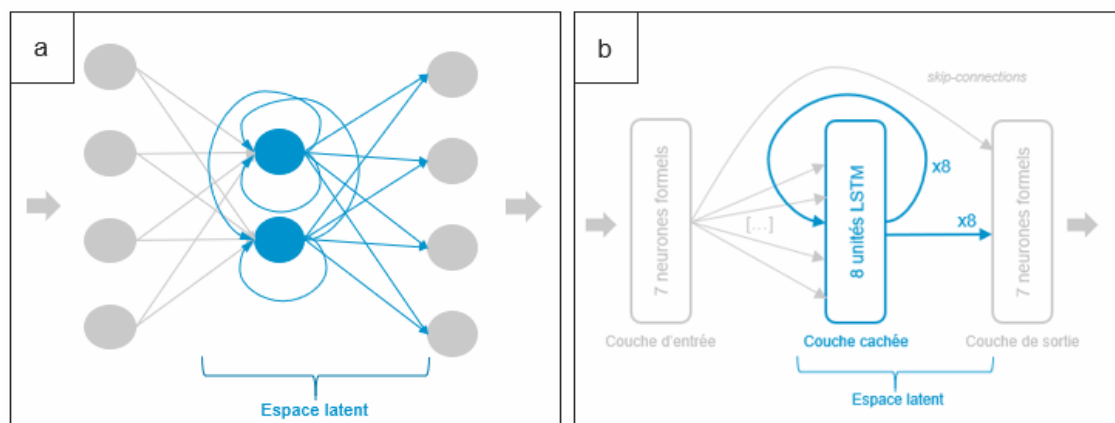


Figure III.4 - Représentation schématique de l'espace latent d'un RNN doté de 2 cellules cachées en (a) et de l'espace latent du RNN implémenté pour le projet IAC en (b)

### III.B.1.b. Extraction de l'espace latent

Le processus d'extraction des règles consiste à récupérer les informations situées au niveau de l'espace latent de notre réseau de neurones récurrents. Pour rappel, une règle dans une séquence est une relation de causalité entre les symboles qui la composent, qu'ils soient proches (se suivent) ou non (aux extrémités de la séquence). Extraire les règles, revient donc à mettre en évidence ces relations de causalités entre les symboles à travers des changement d'état, i.e. l'étude du comportement, d'un réseau de neurones qui a encodé implicitement ces règles. Ainsi, lors de cette étape, notre but est d'extraire les **représentations implicites des données** telles qu'elles le sont au sein du réseau, c'est-à-dire au format numérique.

Nous avons choisi de l'appliquer une fois l'apprentissage de notre réseau terminé afin d'obtenir des représentations internes stables, ce qui n'est pas le cas durant l'apprentissage car le réseau évolue toujours [Servan-Schreiber et al., 1988] et par conséquent, sa représentation implicite aussi. Durant la phase de test, nous avons donc enregistré à chaque pas de temps de nombreuses informations, dont une partie est listée ci-après :

- Les états internes de la couche cachée, i.e. les patterns d'activité des unités LSTM qui seront utilisés par la suite ;
- Le vecteur numérique fourni en entrée au réseau ;
- Le symbole de la grammaire de Reber ou mot Java associé au format texte ;
- La prédiction effectuée par le réseau au format numérique ;
- La sortie attendue ;
- La séquence en cours de traitement ;
- Le « passé » de la séquence composé des symboles (ou mots) déjà rencontrés par le réseau aux pas de temps précédents de la séquence en cours.

La liste exhaustive des données récupérées et sauvegardées dans un dictionnaire au format *json* est disponible en Annexe 3.

Les patterns d'activité des unités LSTM de la couche cachée désormais sauvegardés dans un fichier, nous devons les partitionner et les regrouper. Ce besoin est dû à la complexité que représente la tâche d'exploration de l'espace latent du réseau et de ses représentations implicites. Il y a une réelle nécessité de pouvoir regrouper et ordonner les données afin de réduire ce temps de traitement.

### III.B.2. Clustering des données : k-means et silhouette score

Le clustering est une méthode d'analyse des données qui consiste à regrouper les données en groupes, i.e. *clusters*, selon une ou plusieurs caractéristiques communes. Il porte également le terme de partitionnement ou regroupement des données.

En nous appuyant sur la littérature et de précédents travaux menés par Omlin et Giles [1996] sur l'interprétabilité des réseaux de neurones et l'extraction de règles à partir de réseaux de neurones, nous avons choisi d'utiliser comme **technique de clustering**, l'algorithme des *k-means* qui est la technique la plus répandue dans la littérature [Zeng et al., 1993]. Côté implémentation, nous avons utilisé, dans le cadre de nos travaux, l'implémentation de l'algorithme fournie par la librairie de machine Learning Scikit-learn<sup>12</sup>, recommandée par la communauté scientifique en machine Learning.

L'algorithme des *k-means* permet ainsi de segmenter les données en un nombre de groupes (*k*), appelé aussi *clusters*, en minimisant la distance entre les données à l'intérieur d'un même groupe [K-means, 2020]. Elle permet donc de regrouper les données selon leurs **caractéristiques communes** de manière efficace. L'utilisation de cet algorithme a ainsi été fait sur les patterns d'activité de la couche cachée précédemment récupérés. Une visualisation simple et servant à titre d'exemple de clusters obtenus après l'utilisation de cet algorithme est disponible sur la figure III.5, ci-après.

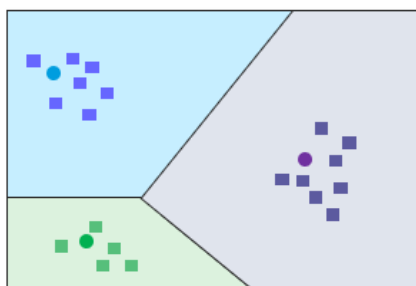


Figure III.5 - Exemple de visualisation d'un clustering obtenu avec l'algorithme des *k-means*, pour  $k=3$

Le nombre de groupes obtenus à la fin de ce processus reste néanmoins à notre charge. Il est alors possible de choisir ce nombre de manière purement arbitraire ou de s'appuyer sur des algorithmes tels que le calcul du **silhouette score**. Il s'agit d'une métrique à calculer qui représente et permet d'évaluer la qualité du clustering. Plus concrètement, la valeur calculée reflète la pertinence des regroupements effectués au sein des différents groupes : les données dans un même groupe sont proches les unes des autres et le plus éloignées possibles de celles des autres groupes. Situé sur la plage de valeur  $[-1 ; 1]$ , un *silhouette score* optimal est égal à 1.

Dans le cadre de nos travaux, ce qui nous intéresse le plus étant la représentation que nous pouvons avoir de la mémoire implicite de notre réseau de neurones, nous avons choisi de réitérer cette étape et donc les étapes suivantes en faisant varier la valeur de *k*. Dans le contexte de la grammaire de Reber, l'intervalle choisi a été  $[2 ; 19]$  et dans le contexte des données issues du code Java nous avons choisi l'intervalle  $[2 ; 26]$ .

La plage de valeur choisie dans chacun de ces contextes applicatifs résulte du type et des caractéristiques des données que nous étudions. De nombreuses expérimentations ont été nécessaires afin d'affiner les valeurs de *k* à étudier. Le choix définitif de la valeur de *k* que nous garderons pour conclure sur nos travaux sera expliqué dans une section ultérieure.

<sup>12</sup> Bibliothèque Python destinée au Machine Learning. <https://scikit-learn.org/>

### III.B.3. Obtention d'un automate

Une fois le regroupement des données effectué, nous pouvons passer à la construction **d'un automate à états finis** qui peut être représenté sous la forme d'un graphe orienté. L'algorithme complet défini par [Omlin et Giles \[1996\]](#) que nous avons utilisé est disponible en Annexe 4 de ce document.

La première étape, présentée sur la figure III.6 – étape a, consiste à définir un nœud initial neutre, noté -1, à notre automate qui se situe en dehors des clusters précédemment récupérés. Nous récupérons ensuite les patterns d'activité de la couche cachée du premier pas de temps et cherchons le cluster associé. Ce cluster devient alors le deuxième nœud, i.e. état, de notre graphe, représenté à l'étape b de la figure III.6 ci-dessous.

Nous ajoutons ensuite la transition entre ces deux nœuds partant du nœud initial vers le nœud issu du cluster (cf. figure III.6 – étape c). L'étiquette associée à cette transition correspond au symbole fourni en entrée à notre RNN lors de la phase de test et ayant permis d'obtenir le pattern d'activité traité dans l'étape précédente. Plus concrètement, la première transition dans le cadre de la grammaire de Reber serait le symbole « B ». Une fois ceci fait, nous passons au deuxième pas de temps et ainsi de suite jusqu'à traiter l'ensemble des pas de temps et donc l'ensemble des patterns d'activité issus de la couche cachée (cf. figure III.6 – étapes d, e et f).

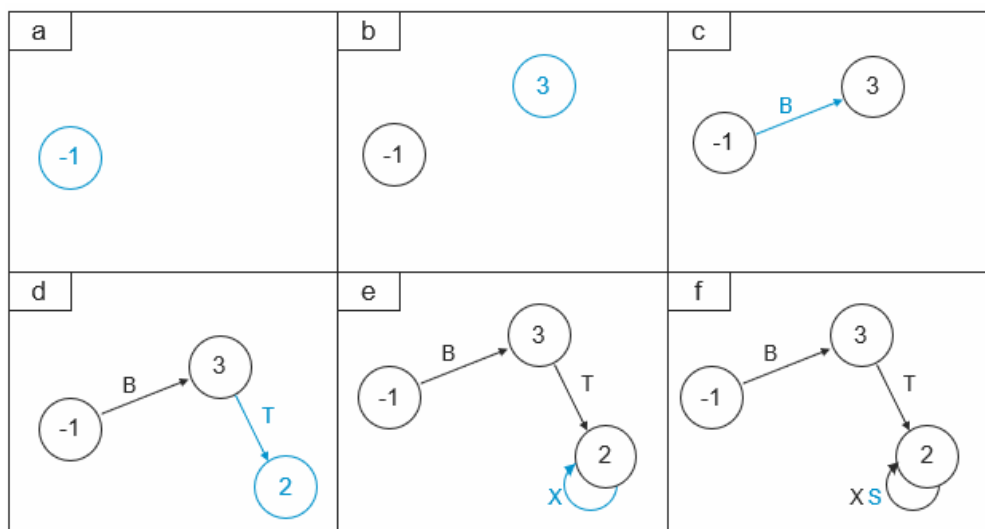


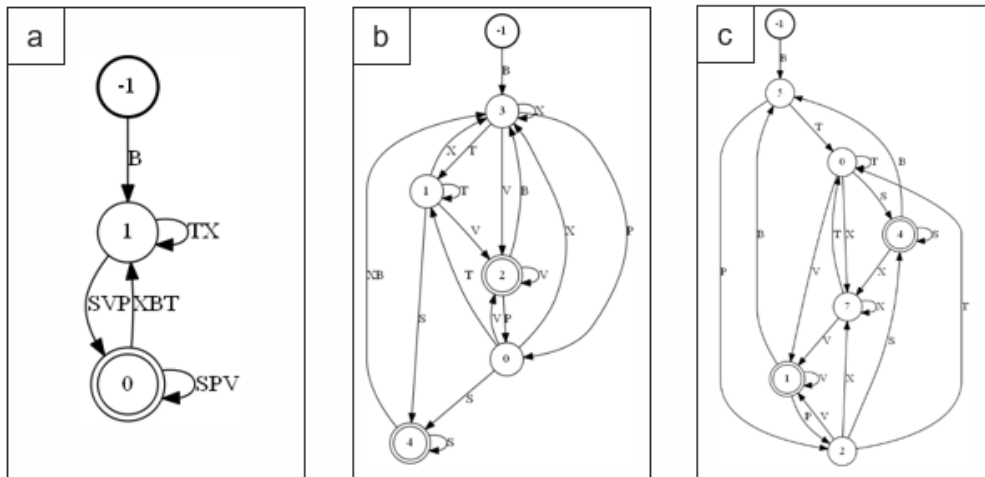
Figure III.6 - Exemple de génération d'automate avec la liste de clusters [3, 2] et la liste de patterns [B, T, X, S] associés aux pas de temps [t0, t1, t2, t3].

A chaque nouveau pas de temps, une première vérification est effectuée afin de déterminer si un nœud correspondant au cluster associé existe. Si la réponse est négative, alors ce nœud est créé et ajouté au graphe, tout comme la transition associée. En revanche, si le nœud existe déjà, alors il est réutilisé. Selon ce même principe, si une transition entre deux nœuds existe déjà alors l'étiquette associée est mise à jour. Un cas particulier existe néanmoins : il intervient lorsque pour deux pas de temps successifs, soit deux patterns qui se suivent, le cluster associé est le même. Dans ce cas, une transition récurrente (une boucle) sur le nœud courant est alors ajoutée à l'automate (cf. figure III.6 – étapes e et f).

Enfin, la dernière étape consiste à identifier le ou les nœuds finaux, représentés par un double cercle dans nos graphes. Un nœud final est le dernier état par lequel passe le réseau lorsqu'il reçoit la dernière paire d'une séquence. A ce moment, le réseau reçoit l'entrée de cette dernière paire et arrive



L'ensemble du processus précédemment expliqué et dont des exemples de résultats sont disponibles en figure II.7 est répété pour chaque valeur de  $k$  utilisée lors du *clustering*. De plus, l'ensemble des données et informations présentes sur ces graphes est également sauvegardé dans des fichiers au format *json* et au format *dot* (format préférentiel pour la sauvegarder de graphe) dont des extraits sont disponibles en Annexe 5.



Néanmoins, le processus d'extraction des activités de la couche cachée n'est pas terminé à cette étape. En nous appuyant une fois de plus sur la littérature et les précédents travaux menés, nous devons désormais transformer notre graphe afin de le minimiser.

The diagram illustrates the construction of a Deterministic Finite Automaton (DFA) from a neural network. The process is shown in four stages:

- Automate à états finis (NFA):** A neural network with 4 input nodes, 2 hidden nodes, and 4 output nodes is shown. Below it, a Non-deterministic Finite Automaton (NFA) is depicted with states -1, 1, and 0. Transitions are labeled with 'B', 'TX', 'SVP', and 'XBT'. State 0 is the final state.
- Est-il déterministe ? (Decision):** A decision point asking 'Est-il déterministe ?' (Is it deterministic?). The answer is 'non' (no), indicated by a blue arrow pointing to the next stage.
- Automate déterministe complet (Complete DFA):** A complete Deterministic Finite Automaton (DFA) is shown with states -1, 1, 0, and s. Transitions are labeled with 'B', 'TX', 'SVP', and 'XBT'. State s is the final state.
- Automate déterministe minimisé (Minimized DFA):** A minimized Deterministic Finite Automaton (DFA) is shown with states ('-1'), ('1'), and ('0'). Transitions are labeled with 'B', 'TX', 'SVP', and 'XBT'. State ('0') is the final state.

42

Le principe de la **minimisation** consiste à réduire la taille et la complexité de l'automate précédemment obtenu. En effet, celui-ci peut s'avérer très complexe et donc difficile à interpréter. Néanmoins, il est impossible de le minimiser en l'état. Le processus de minimisation des automates ou graphes se découpe en plusieurs étapes et qui s'avère assez complexe est présenté en figure III.8 ci-dessus. Ainsi, par souci de clarté, nous ne présenterons pas en détails les algorithmes utilisés mais en expliquerons simplement les concepts.

La **minimisation des automates** est nécessaire car elle nous permet de nous aligner sur la démarche scientifique la plus répandue et validée par la communauté de chercheurs dans ce domaine. Le second intérêt est que cette minimisation nous permet de diminuer la charge computationnelle dans le cadre des tests, de l'analyse et du stockage des graphes.

Dans un premier temps, nous devons **déterminer la nature de notre automate** : celui-ci est-il **déterministe** ou non ? Un automate **non déterministe** (cf. figure III.9 – a) est un automate dont plusieurs transitions partant d'un même nœud source peuvent avoir la même étiquette (symbole). En revanche, un **automate déterministe** (cf. figure III.9 – b) n'aura qu'une seule transition possible d'un nœud à un autre comportant ce symbole.

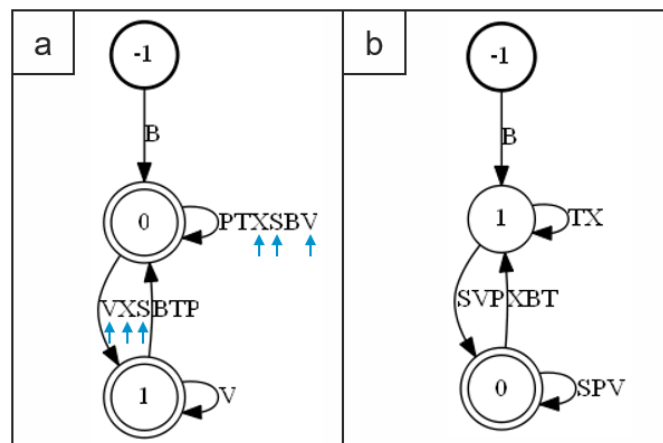


Figure III.9 - Exemple d'un automate non déterministe représenté en (a) et d'un automate déterministe représenté en (b) dans le contexte de la grammaire de Reber pour  $k = 2$  selon l'algorithme des  $k$ -means

Ainsi, si notre automate à états finis obtenu à l'étape précédente s'avère être non déterministe, il nous est nécessaire de le transformer en **automate déterministe**, autrement appelé **DFA** pour *Deterministic Finite Automaton* en anglais, afin d'obtenir une seule transition pour un seul symbole.

La deuxième étape consiste à créer un **automate complet** (cf. figure III.10 - a) à partir de l'automate déterministe précédent. Pour cela, un nouveau nœud va être ajouté à l'automate. Ce nœud, appelé « nœud poubelle » dans la littérature, que nous symbolisons par la lettre « s » pour *sink*, va être rattaché au reste des nœuds par les transitions manquantes. Chaque nœud du graphe doit présenter autant de transitions (vers d'autres nœuds) que de symboles ou mots présents dans les données fournies en entrée au RNN. Prenons un exemple en utilisant les symboles de la grammaire de Reber : B, T, X, S, P, V et E, qui ne sera pas utilisé car non présent dans les données d'entrée (cf. figure II.11) Imaginons que le nœud numéro 1 dispose déjà des transitions B, T et S vers d'autres nœuds du graphe, il lui manque alors trois transitions afin de couvrir l'ensemble des symboles. Afin d'y remédier nous créons ces transitions manquantes vers le nœud S, i.e. les transitions X, P et V.

Bien que le processus de minimisation ne soit pas obligatoire, c'est une étape établie dans la littérature scientifique dans le processus d'extraction de règles à partir d'un RNN, nous avons choisi de le mettre en place afin de réduire l'automate obtenu à l'étape précédente en le plus petit automate

possible. Pour ce faire, nous nous basons sur les travaux et l'algorithme standard de minimisation proposé par [Hopcroft et al. \[1979\]](#). Cet algorithme nous permet ainsi de réduire la complexité de l'automate en réduisant le nombre de nœuds et de transitions. Afin de terminer notre phase de minimisation, nous allons maintenant passer à la validation de notre automate minimisé (cf. figure III.10 – b).

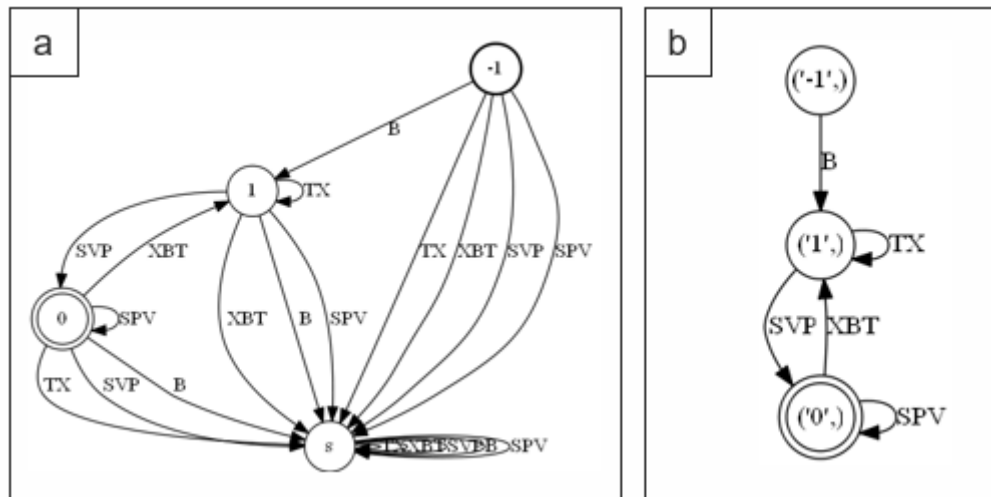


Figure III.10 – Exemple d'un automate déterministe complet (a) avec l'automate déterministe minimisé associé (b) dans le contexte de la grammaire de Reber pour  $k=3$  selon l'algorithme des  $k$ -means

### III.B.5. Validation des automates

Avant de passer à l'étape de **validation de nos automates**, faisons un rapide rappel des étapes effectuées précédemment (cf. figure III.8). Nous avons ensemble vu qu'il était possible de générer un automate à états finis à partir des états internes des unités LSTM de la couche cachée. Nous avons ensuite vérifié si cet automate était déterministe ou non. Si ce dernier ne l'était pas, nous l'avons converti en un automate déterministe (DFA). Ensuite, nous l'avons complété en ajoutant toutes les transitions manquantes vers un « nœud poubelle ». Enfin, nous l'avons minimisé à l'aide de l'algorithme de Hopcroft [\[Hopcroft, et al., 1979\]](#).

Néanmoins, comme évoqué dans la section précédente, **un automate minimisé doit être validé** avant d'être utilisé. Cette phase de validation va être répétée, tout comme les précédentes, sur l'ensemble des automates générés selon les différentes valeurs de  $k$  lors de l'algorithme des  $k$ -means.

La validation de notre automate minimisé est primordiale car elle nous permet d'une part de **vérifier le bon apprentissage** par notre réseau de neurones récurrent des règles implicites situées dans les données et d'autre part de vérifier que notre processus d'extraction s'est bien déroulé. En effet, si la grammaire de Reber et l'automate minimisé extrait valident les mêmes séquences cela signifie qu'ils représentent tout deux **le même langage** [\[Berstel et al., 2010\]](#). Nous pouvons ainsi affirmer que nous avons réussi à extraire une représentation latente proche de la grammaire d'origine à partir des règles cachées dans les données.

L'étape de validation consiste ainsi, dans le cadre de nos travaux et au vu des données dont nous disposons au moment des simulations, à utiliser les séquences de tests utilisées pour l'extraction, en les fournissant à notre automate minimisé afin de voir s'il les accepte ou non. En d'autres termes, nous

essayons de passer d'un symbole à un autre en suivant les nœuds et transitions de notre automate. Nous pouvons ainsi déterminer si la séquence est acceptée ou non par notre automate. Les séquences issues du corpus de test utilisé pour la phase de test du RNN sont donc correctes et nous cherchons avec notre automate à **obtenir le meilleur pourcentage de séquences grammaticales acceptées**. Nous effectuons ensuite un second test, cette fois-ci avec des séquences aléatoires composées des mêmes symboles que les séquences grammaticales mais qui ne suivent pas la grammaire d'origine. Lors de ce test nous cherchons cette fois-ci à minimiser et à rester aussi proche que possible d'un pourcentage nul de séquences acceptées.

Dans le cadre de la grammaire de Reber, la phase de validation permet de valider notre approche générique. En effet, si les règles présentent dans la grammaire d'origine et celles extraites de notre réseau de neurones sous la forme d'un automate sont les mêmes, alors ainsi expliqué précédemment, cela signifie qu'ils reconnaissent le même langage. Une autre conclusion que nous pouvons tirer est que notre réseau de neurone a bien appris les règles de la grammaire d'origine à partir des représentations implicites de celles-ci cachées dans les données. Ainsi, si notre démarche fonctionne pour cette grammaire, nous pouvons affirmer qu'elle fonctionnera également pour le code Java dont nous ne connaissons pas les règles d'origine. Nous aurons alors de bonnes raisons de penser que les règles extraites à partir du code Java sont correctes et ceci est, rappelons-le, notre hypothèse de recherche. **Les propos précédents illustrent bien notre démarche scientifique : la validation de notre approche générique dans un contexte artificiel et sa transposabilité dans un contexte de données réelles.**

### III.C. Résultats de notre approche générique d'extraction de règles implicites et de l'expertise

Dans la suite de nos travaux d'interprétabilité présentés dans ce mémoire, nous allons présenter les **résultats obtenus** pour la meilleure valeur de  $k$  obtenue avec l'algorithme des *k-means*. En calculant le pourcentage de séquences acceptées par notre automate minimisé et en le mettant en exergue avec le *silhouette score* nous pouvons déterminer la meilleure valeur de  $k$ . Plus concrètement, **l'automate minimisé ayant obtenu le meilleur pourcentage de séquences reconnues et le meilleur silhouette score est celui qui aura le mieux encodé les règles implicites cachées dans les données.**

Les caractéristiques des données corpus de données utilisées pour la génération des automates présenté dans le cadre de ce mémoire sont disponibles ci-dessous dans le tableau III.1.

	Grammaire de Reber	Code Java
<b>Corpus de test utilisé pour la génération des automates</b>		
Nombre de séquences	13	22
Nombre de patterns	400	100

Tableau III.1 - Caractéristiques des corpus utilisés pour la génération des automates extraits à partir des données collectées lors de la phase de test du RNN

### III.C.1. Application à la grammaire de Reber

Dans cette section nous allons exposer le **processus d'extraction des règles implicites** à partir de notre réseau de neurones récurrents ayant appris sur des données issues de la **grammaire de Reber**. Nous présenterons nos premiers résultats et tirerons les premières conclusions.

Comme nous l'avons fait depuis le début de nos travaux, l'utilisation de la grammaire de Reber est ici une étape primordiale afin de valider notre approche générique. En effet, nous n'aurions pas pu faire de l'interprétabilité directement sur un réseau entraîné sur du code Java, et ce, du fait de la nature de ces données. Comme nous l'avons vu précédemment, le processus d'extraction de règles nous permet d'obtenir **un automate minimisé représentant les règles implicites encodées par le réseau** lors de son apprentissage sur des données issues de la grammaire de Reber.

La première étape de nos travaux a ainsi consisté à générer un automate à états finis à partir des états de la couche cachée de notre réseau. Au vu des contraintes de temps imposées, de la complexité notable de graphes générés à partir de volumes de données très importants et de la nature exploratoire de nos travaux, nous avons décidé de n'utiliser que les 400 premiers patterns d'activité de la couche cachée de notre réseau de neurones.

Bien que l'utilisation d'une partie seulement des patterns d'activité puisse provoquer le non-traitement de certaines informations et donc de règles, nous avons été dans l'obligation de procéder à ce choix. Pour des questions de clarté et de cohérence, nous ne présenterons ici que les meilleurs résultats obtenus pour cette grammaire avec une valeur de  $k=6$  pour l'algorithme de *clustering k-means* pour les 400 patterns.

Lors de la génération de nos automates nous distinguons deux formats d'étiquettes pour les transitions : les étiquettes longues et les étiquettes courtes. Les **étiquettes longues** vont comporter l'ensemble des symboles représentés par la transition du graphe avec un pas de temps associé. Par exemple si le symbole B est représenté sur une transition pour les pas de temps 1, 6 et 10, alors l'étiquette longue sera : « B1B6B10 ».

Au contraire, **l'étiquette courte** est utilisée pour simplifier la lecture du graphe et le minimiser car nous ne gardons qu'une seule occurrence du symbole et enlevons donc les pas de temps associés. Ainsi notre étiquette longue « B1B6B10 » devient simplement « B ». De plus, si des symboles différents sont présents sur une même transition, le principe reste le même : « B1B6B10S14 » pour l'étiquette longue et « BS » pour l'étiquette courte.

Par souci de lisibilité nous n'utiliserons dans les figures ci-après que des graphes dotés d'étiquettes courtes (cf. figure III.11). Les graphes générés avec des étiquettes longues tendent à être trop volumineux et à devenir illisibles au-delà d'une taille de patterns. Nous avons donc fait le choix dans ce document de n'utiliser que des automates à étiquettes courtes. Néanmoins un extrait d'un automate à étiquettes longues est disponible en Annexe 6 de ce document.

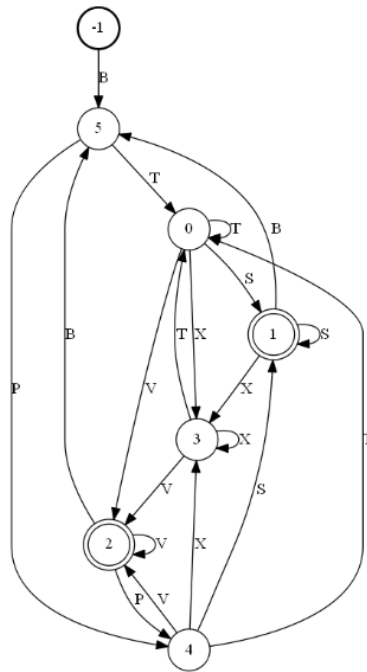


Figure III.11 - Automate à états finis pour  $k\text{-means}=6$  extrait à partir d'un RNN entraîné sur la grammaire de Reber (400 patterns)

Comme nous pouvons le voir sur le graphe ci-dessus, tous les symboles de la grammaire de Reber utilisés pour la phase d'apprentissage sont représentés (hormis le E, jamais fourni en entrée à notre réseau). Ce graphe n'étant que peu exploitable en l'état nous passons à la phase 2. Ainsi que vu précédemment, la deuxième étape consiste à déterminer la nature de notre graphe.

Le graphe utilisé ici étant déjà déterministe, sa représentation ne change donc pas et nous passons à l'étape suivante qui consiste à le compléter. Nous avons donc ajouté à notre graphe l'ensemble des transitions manquantes vers un « poubelle », noté S dans le graphe suivant :

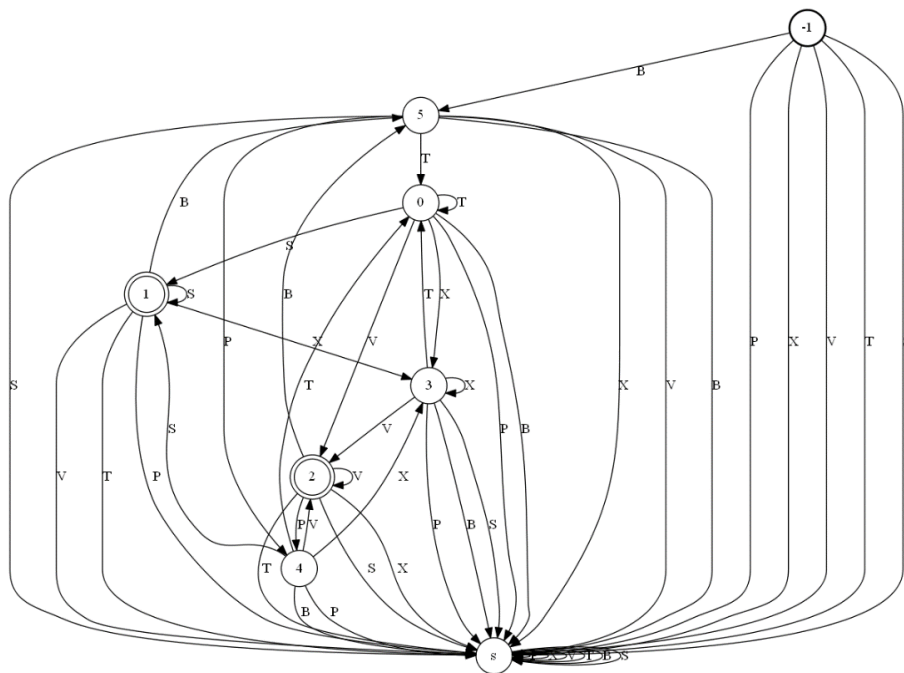


Figure III.12 - Automate déterministe complet généré à partir de l'automate à états finis représenté en figure III.11.

Ici représenté par la figure III.12, le fait de compléter un automate déterministe revient dans un premier temps à le complexifier pour mieux le minimiser ensuite. L'étape de minimisation effectuée à l'aide de l'algorithme de Hopcroft<sup>13</sup> nous permet d'obtenir notre dernier graphe présenté en figure III.13 ci-après.

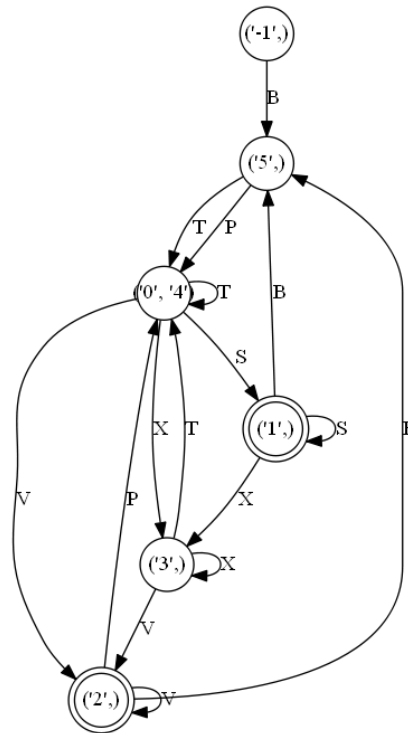


Figure III.13 - Automate minimisé obtenu à partir de l'automate complet présenté en figure III.12

L'**automate minimisé** que nous obtenons ici s'avère être **lisible** et **interprétable**. Nous pouvons par exemple observer que des nœuds ayant les mêmes transitions ont été regroupés en un seul et unique nœud, rendant ainsi l'automate plus simple et lisible.

En revanche, afin de nous assurer que cet automate représente bien les règles de la grammaire de Reber, il nous faut le tester afin de le valider. Pour cela, nous utilisons deux stratégies telles que définies précédemment. La première consiste donc à tester notre automate sur des séquences grammaticales issues de la grammaire de Reber. La deuxième consiste à faire l'inverse, c'est-à-dire tester notre automate avec des séquences composées des symboles de la grammaire de Reber mais non grammaticales. Les résultats de ces deux tests sont recensés dans le tableau suivant :

	Séquences grammaticales	Séquences non grammaticales
Nombre de séquences testées	100	100
Nombre de séquences acceptées	94	3
% de séquences acceptées	94	3

Tableau III.2 - Pourcentages de séquences grammaticales et non grammaticales acceptées par l'automate minimisé.  
Application à la grammaire de Reber

<sup>13</sup> Algorithme de Hopcroft <http://i.stanford.edu/pub/cstr/reports/cs/tr/71/190/CS-TR-71-190.pdf>

Comme nous le montre ce tableau les résultats obtenus sont très bons. En effet notre automate **accepte 94% de séquences grammaticales** issues de la grammaire de Reber contre seulement 3% des séquences non grammaticales.

Nous avons également choisi de faire une représentation graphique des pourcentages de séquences acceptées pour l'ensemble des valeurs de  $k$ . Cette représentation disponible ci-dessous en figure III.14 nous indique effectivement que la meilleure valeur de  $k$  est 6 selon nos critères. Bien que les valeurs de  $k$  supérieures à 6 montrent également les mêmes comportements, nous ne les prenons pas en compte. En effet, pour les valeurs supérieures à 6 lors de l'utilisation du *k-means*, nous avons constaté que le nombre de clusters différents était toujours de 6 et que les automates étaient identiques ; cela signifie qu'un regroupement des données en un nombre plus élevé de groupes n'était pas possible. En d'autres termes, le meilleur clustering possible sur des données issues de la grammaire de Reber afin de générer des automates se fait avec 6 clusters distincts.

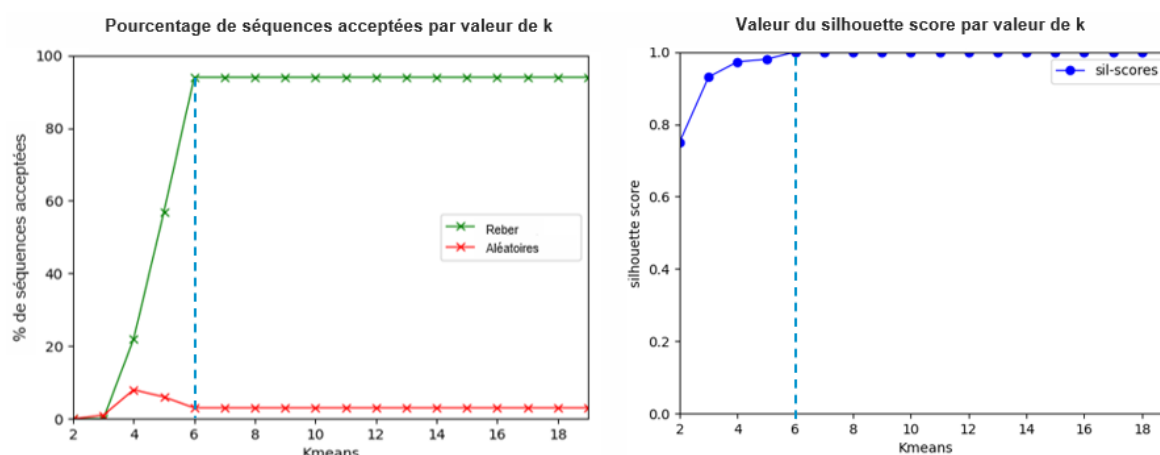


Figure III.14 - Pourcentages de séquences de la grammaire de Reber et aléatoires acceptées par les automates pour  $k$  appartenant à l'intervalle [2, 19] et valeur des silhouette scores calculés associés pour chaque valeur de  $k$

Nous avons également choisi de tenir compte de la valeur des *silhouette scores* (cf. figure III.14) calculés pour chaque valeur de  $k$  dans le cadre de l'algorithme des *k-means*. Nous remarquons qu'il y a un lien certain entre les pourcentages calculés de séquences acceptées et la qualité du *clustering* effectué selon les différentes valeurs de  $k$ .

Nous pouvons donc affirmer à ce stade via notre automate minimisé et grâce à la phase de validation, que **notre réseau de neurones récurrents a été capable d'encoder les bonnes règles implicites de la grammaire d'origine** : la grammaire de Reber. En effet un fort pourcentage de séquences grammaticales acceptées corrélé à un faible pourcentage de séquences non grammaticales acceptées prouve que le réseau a encodé les bonnes règles. Un automate extrait à partir d'informations du RNN étant capable de reconnaître des séquences issues de la grammaire d'origine démontre que celui-ci et la grammaire de Reber reconnaissent le même langage. Le faible pourcentage de séquences non grammaticales reconnues, ici seulement 3%, prouve que le réseau sait reconnaître des séquences ne suivant pas ces règles. Nous tendons bien entendu à obtenir un pourcentage égal à zéro dans ce second cas. L'explication la plus plausible concernant l'acceptation de certaines séquences aléatoires est que celles-ci, au moment de leur génération, ont par hasard reflétées les règles de la grammaire d'origine.

Grâce à ces premiers résultats obtenus sur la grammaire de Reber **nous venons de valider notre approche générique** et pouvons désormais tester sa transposabilité sur un autre contexte d'application : le code Java.



### III.C.2. Application au code Java

Dans la section suivante nous allons nous concentrer sur l'exploitation de l'espace latent de notre réseau de neurones récurrents entraîné cette fois-ci sur les **données issues du code Java**. Contrairement à la grammaire de Reber dont nous connaissons toutes les règles d'origine, nous ne connaissons pas l'ensemble de ces règles dans le cadre du code Java. Bien que **le code Java** que nous utilisons **comporte une dimension explicite** telle que les contraintes du langage de programmation Java, les contraintes techniques, etc. il contient également **une dimension implicite** due à l'expérience, au vécu et aux biais des développeurs l'ayant écrit. De ce fait, et contrairement à la grammaire de Reber, nous ne disposons pas de l'ensemble des règles qui le régit. Ainsi, l'approche exploratoire et la transposabilité de nos travaux prend tout son sens.

L'ensemble des séquences utilisées pour illustrer nos résultats dans le cas des données Java est disponible en Annexe 7 de ce document.

La première étape du processus de génération de l'automate à états finis est identique à celle opérée sur la grammaire de Reber. Bien que nous ayons effectué l'ensemble du processus selon différentes valeurs de  $k$  dans le cadre du *k-means*, nous avons choisi d'exposer ici les résultats pour  $k$  égal à 19. Ce choix s'est opéré selon le pourcentage de séquences acceptées et donc reconnues par nos automates comme le montre la figure III.15 ci-dessous.

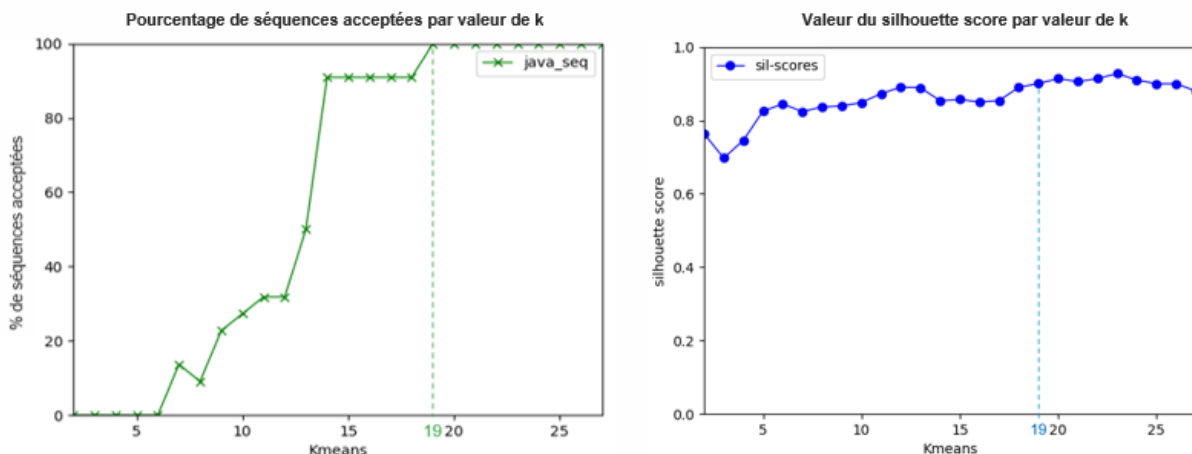
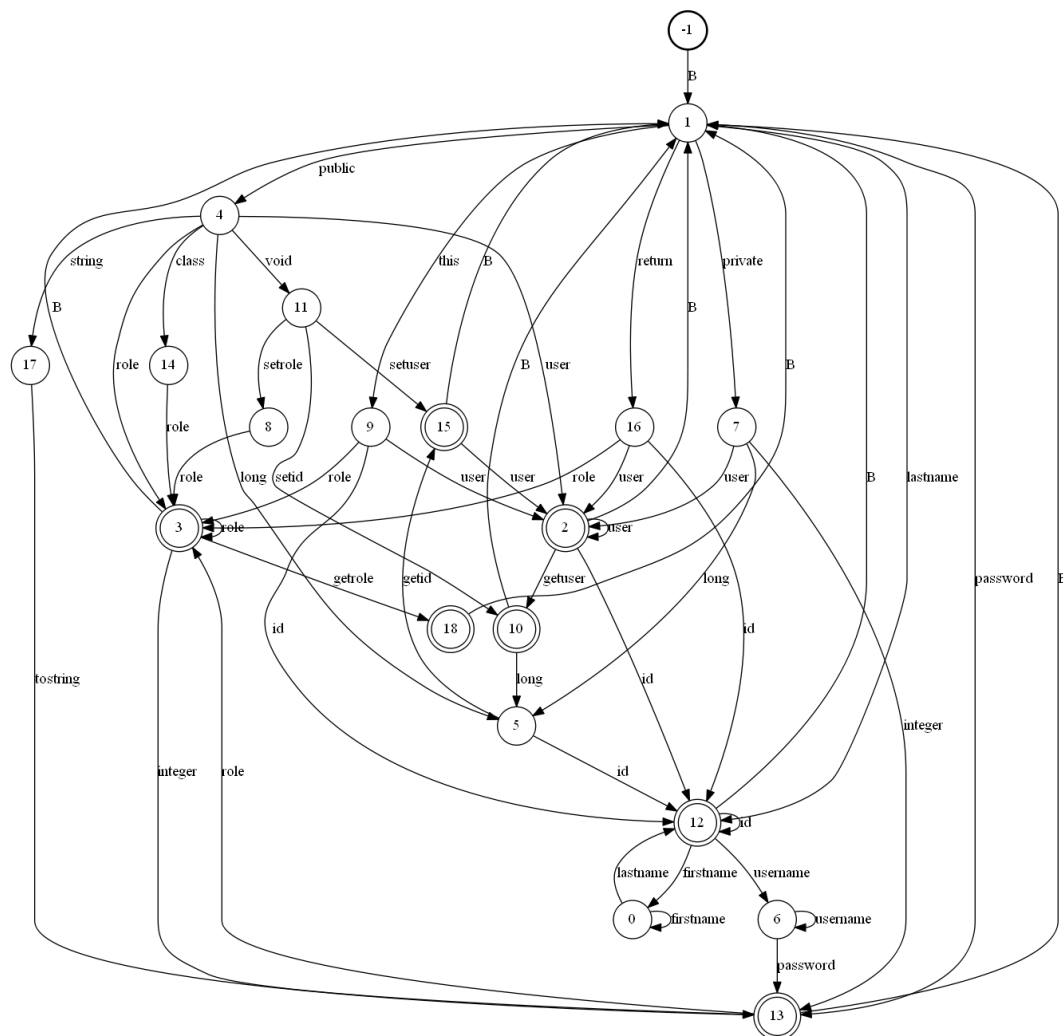


Figure III.15 - Pourcentages de séquences de code Java acceptées par les automates pour  $k$  appartenant à  $[2, 26]$  et valeur des silhouette scores calculés associés pour chaque valeur de  $k$

Nous avons également remarqué qu'il pouvait y avoir une corrélation nette entre les pourcentages de séquences Java reconnues obtenus précédemment et la valeur des *silhouette scores* calculés pour chaque valeur de  $k$ . Pour rappel, le **silhouette score reflète la qualité du clustering**. Plus cette valeur est proche de 1, meilleur est le regroupement des données. La définition d'un seuil minimum pour considérer une valeur comme un bon *silhouette score* dépend majoritairement du type de données étudiées et de l'ensemble des valeurs obtenues. Une autre observation aidant à choisir ce seuil repose sur l'observation d'un « plateau » au niveau de ces valeurs : si celles-ci restent stables sur plusieurs  $k$  successifs alors on peut estimer avoir obtenu les meilleurs *silhouettes scores* précédemment et de ce fait définir notre seuil en conséquence.

Dans le cadre de nos travaux, les valeurs de *silhouette scores* présentées en figure III.15 étaient très satisfaisantes au vu des pourcentages de séquences acceptées.

A l'instar de l'extraction des patterns d'activité de la couche cachée de notre RNN dans le contexte de la grammaire de Reber, nous avons choisi ici de n'utiliser qu'une partie des données issues du code Java. Le traitement d'un trop grand volume de données sous forme d'automate les rendant illisibles. De ce fait, nous avons décidé de n'utiliser et donc de ne présenter que des automates générés à partir des 100 premiers patterns d'activité de la couche cachée de notre RNN pour un total de 22 séquences. Malgré ce choix restrictif les automates obtenus n'en restent pas moins complexes à analyser.



Comme nous pouvons l'observer ce premier automate généré s'avère bien plus complexe que celui obtenu à partir de la grammaire de Reber. Cette complexité notable vient tout d'abord du nombre de nœuds présents : 20 dans cet automate contre 7 pour l'automate obtenu à partir de la grammaire de

Reber. Une seconde explication vient de la **plus grande variété de mots Java** dont nous disposons que de symboles présents dans la grammaire de Reber. Par exemple, dans la partie du corpus Java composé des 100 premiers patterns d’activité que nous avons utilisé il y a 24 mots différents.

Pour rappel, en suivant notre approche générique de processus d’extraction (cf. figure III.8), nous avons généré notre automate à états finis, puis avons vérifié sa nature déterministe ou non et enfin, nous avons obtenu un automate complet. Néanmoins celui-ci est trop complexe pour être présenté sous sa forme de graphe. Nous faisons ici le choix de présenter une partie de ce graphe, reflétant sa grande complexité à travers son **format dot**. La figure III.17 ci-dessous illustre l’ensemble des transitions partant du nœud 1 de cet automate, à gauche dans son format d’automate à états finis et à droite dans son format d’automate complet.

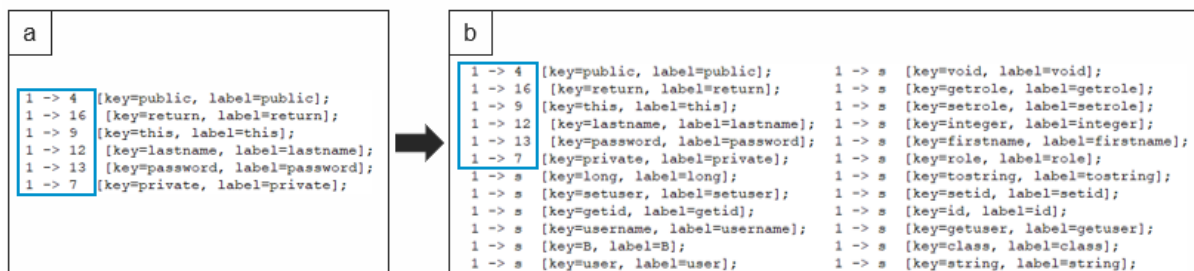


Figure III.17 - Transitions partant du nœud 1 sur l’automate à états finis (a) et sur l’automate complet (b) généré à partir de celui-ci pour  $k=19$  présentées au format dot

Comme nous pouvons le voir dans le cadre « a » de la figure III.17 les transitions partant du nœud 1 sur l’automate à états finis obtenus sont au nombre de 6. En revanche, ces mêmes transitions sur l’automate déterministe complet, présentées dans le cadre « b », sont au nombre de 23. Cette différence majeure, qui sera la même pour chaque nœud du graphe d’origine, augmente donc considérablement la complexité.

Enfin, notre dernière étape de minimisation selon notre approche générique a été de transformer cet automate complet en un automate minimisé. Pour ce faire, nous avons réitéré le même processus que celui effectué sur l’automate issu de la grammaire de Reber. Le résultat obtenu est présenté ici en figure III.18.

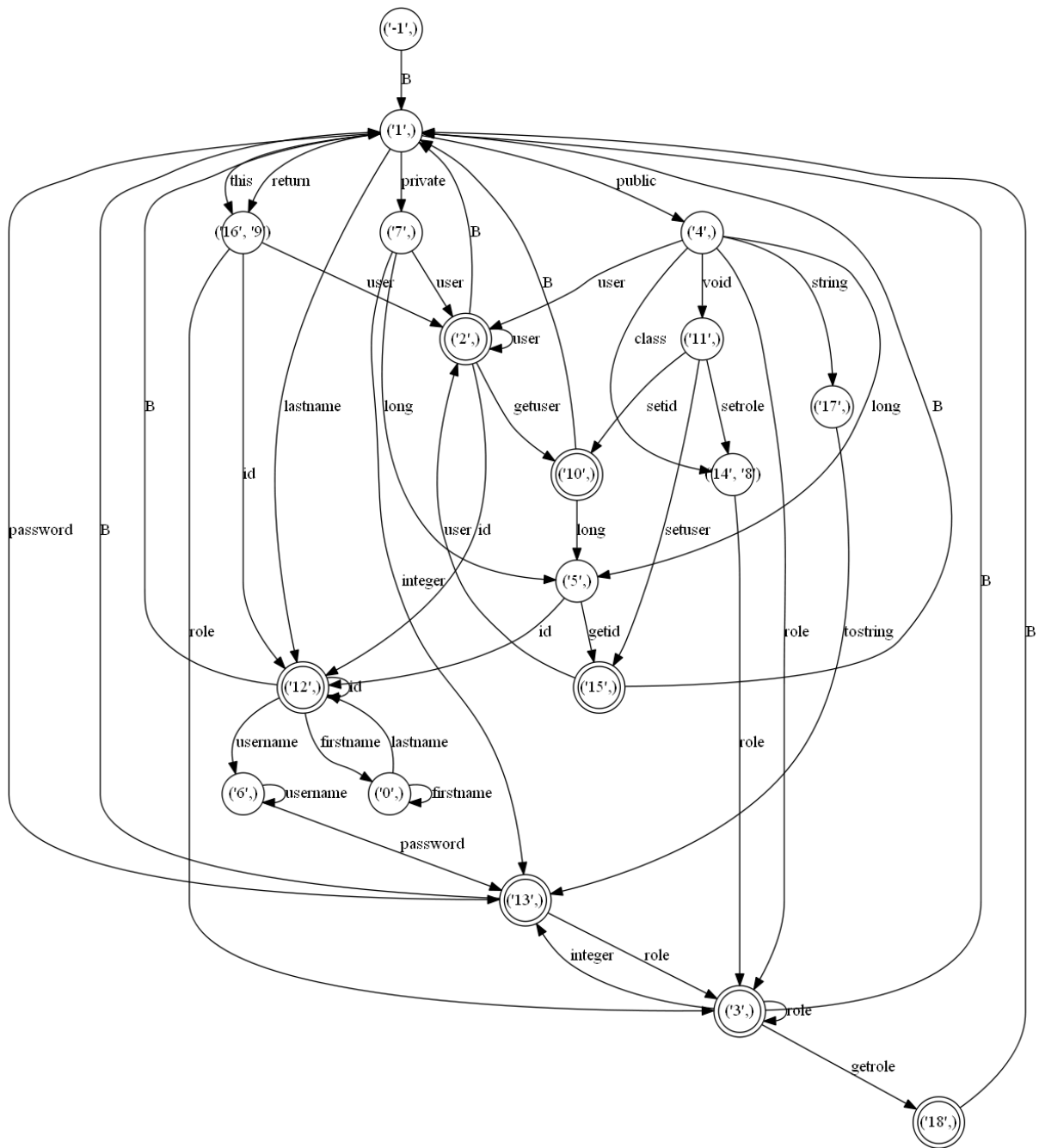


Figure III.18 - Automate minimisé obtenu à partir de l'automate à états finis présenté en figure III.16

Cet automate minimisé étant trop complexe pour être expliqué dans sa totalité, nous faisons le choix de n'exposer et d'expliquer nos résultats obtenus qu'à partir d'extraits de ce graphe.

L'extrait numéro 1 que nous avons choisi (cf. figure III.19) représente une partie des transitions partant du nœud numéro 1 vers les autres nœuds du graphe.

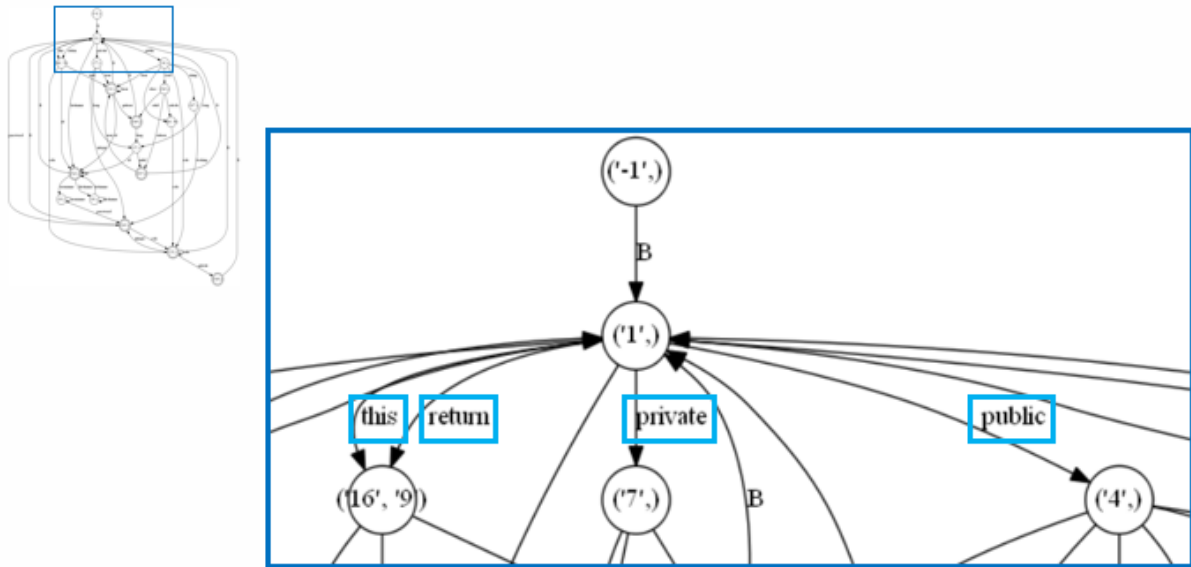


Figure III.19 - Extrait numéro 1 de l'automate minimisé présenté en figure III.18

En prêtant attention aux étiquettes des transitions (encadrées en bleu sur la figure ci-dessus) nous remarquons qu'elles représentent uniquement des **mots clés du langage Java**. Ces mots clés sont utilisés notamment pour définir l'accessibilité d'une classe ou pour retourner une variable à la fin d'une méthode. Il est très intéressant de remarquer que ces mots clés Java de **nature similaire** sont reflétés par des transitions partant d'un même nœud. Pour rappel, **les nœuds représentent ici des états du réseau**. Ces états ont été définis selon le *clustering* effectué afin de regrouper les patterns d'activité de la couche cachée de notre RNN selon des caractéristiques communes.

Ainsi, à l'aide de cette représentation graphique qu'est notre automate, nous pouvons dire que notre réseau de neurones a su encoder **une représentation implicite de ces mots**, qui se trouvaient être similaires, tout du moins assez ressemblants au moment de son apprentissage. Nous pouvons considérer cette information comme **une règle du langage Java**, certes explicite, que le réseau s'est créé lors de son apprentissage.

Un deuxième constat possible à partir de ce premier extrait concerne **la structure des lignes de code Java**. Pour rappel, chacune des lignes de code a été transformée en une séquence de mots Java afin d'être utilisée en entrée de notre réseau. Le nœud numéro 1 de notre automate représente donc ici le premier état d'une séquence. Bien qu'il ne soit pas le nœud initial de notre automate, nous décidons ici de le considérer comme tel lorsque nous parlons de séquences Java. En effet, nous avons ajouté à chaque séquence de code Java un symbole supplémentaire, B, afin de nous permettre d'identifier leur début et de permettre à notre réseau d'effectuer un apprentissage correct et complet. En revanche, lors de l'analyse de nos résultats, nous ne prenons pas en compte ce symbole B. Ainsi, nous pouvons voir que les quatre premières transitions possibles, et donc les quatre premiers mots Java utilisables en début de séquence d'après notre automate et au vu des données étudiées, sont *return*, *this*, *private* et *public*. Connaissant la structure utilisée lors de l'écriture du code Java, nous remarquons par exemple que ces mots clés sont ceux pouvant être utilisés en début de ligne afin d'instancier une classe ou une méthode. Nous pouvons donc affirmer que **notre réseau a su encoder correctement cette règle explicite propre à la structure du langage Java**.

Enfin, nous avons choisi de calculer la **fréquence d'utilisation** de ces mots clés parmi les données que nous avons utilisées et leur représentation dans notre automate. Les résultats sont présentés dans le tableau III.3 ci-dessous. Durant les deux derniers mois précédents le rendu de ce mémoire, nous avons souhaité explorer une autre piste de recherche. Notre hypothèse de départ a été la suivante : il existe un lien entre les transitions présentes sur un graphe et le nombre d'occurrences de chaque mot présents dans les données d'origine. Autrement dit, les biais statistiques présents dans les données de départ devraient se retrouver dans les extractions réalisées. La nature exploratoire de nos travaux et le temps imparti ne nous a malheureusement pas permis d'aller plus loin dans l'exploration de cette piste. Néanmoins, nous avons tout de même souhaité la partager dans ce mémoire car elle semble intéressante au vu de notre sujet sur l'extraction de l'expertise.

Mot clé Java (transition)	Nombre d'occurrences
This	4
Return	4
Private	3
Public	10
Autres mots (19)	79
Total	100

Tableau III.3 - Nombre d'occurrences des mots clés Java présents dans les données sur un total de 100 mots

Nous le constatons, le mot clé *public* est bien plus utilisé que le mot *private* par exemple. Ce constat peut nous amener à supposer que nous venons ici de mettre en exergue **une règle implicite concernant l'accessibilité du code**. Cette règle pouvant refléter une partie des connaissances métier des développeurs ayant écrit ce code Java. En effet aucune règle directement liée au langage Java ne stipule l'utilisation plus forte d'un type d'accessibilité d'une classe ou d'une méthode à une autre.

Un deuxième extrait de notre automate minimisé intéressant à analyser est présenté sur la figure III.20 ci-dessous.

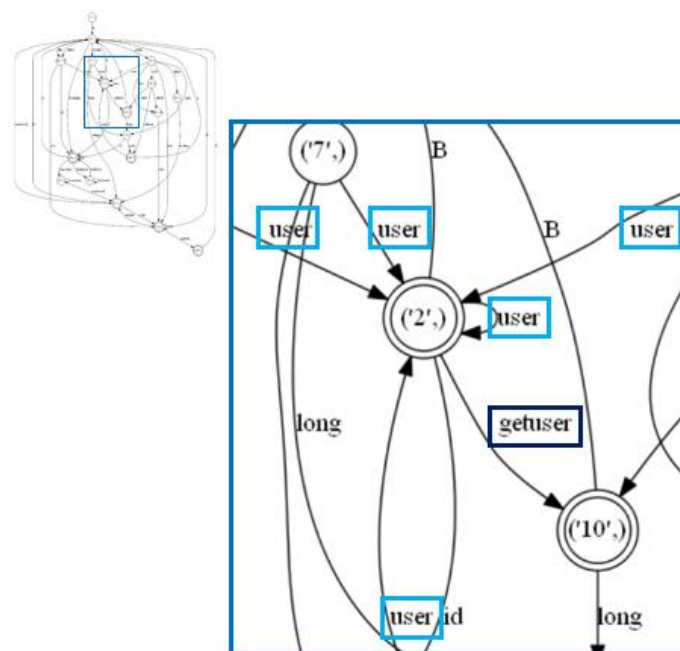


Figure III.20 - Extrait numéro 2 de l'automate minimisé présenté en figure III.18

Dans ce deuxième extrait présenté ci-dessus, nous portons notre attention en particulier sur le nœud 2. Comme nous pouvons le voir ce nœud, qui représente un état de notre réseau de neurones, est le point de **convergence de toutes les transitions porteuses de l'information « user »** encadrées en bleu clair sur la figure ci-dessus. Ce point de convergence nous fait supposer que notre RNN, au moment de son apprentissage, a encodé une **représentation implicite similaire** de cette information et ce malgré **des contextes différents**, représentés par les différentes séquences dans lesquelles le mot « user » est issu. Le mot « user », représentant ici un objet Java « Utilisateur » dans le code d'origine, est utilisé et représenté selon différents contextes dans les séquences. Ainsi il nous est permis de dire que notre réseau a su retrouver **une similitude autour de la donnée « user »** et ce, malgré différents contextes dans lesquels elle se situait. Notre réseau n'analysant pas la sémantique des séquences ni même des mots Java. Comme nous pourrions le supposer en tant que développeur à la vue des lignes de code Java, ce mot « user » est certainement assimilé au même objet Java. En revanche notre réseau ne possédant pas cette connaissance sémantique a tout de même réussi à encoder cet aspect que nous considérons comme **une règle métier implicite**.

En observant la figure III.21 ci-dessous, nous remarquons dans la représentation fournie par notre automate un ensemble de nœuds proches les uns des autres concernant les caractéristiques liées à l'objet « user » telles que le prénom, le nom de famille, le mot de passe ou l'identifiant. Cette seconde zone de l'automate, fortement liée à celle présentée en figure III.20, conforte nos suppositions. En effet, cette fois-ci les mots Java sont variés (e.g. *username*, *firstname*, *lastname*, etc.). Ainsi, comme nous le voyons, notre automate reflète bien des règles implicites apprises par notre réseau à partir des données telles **qu'un lien certain entre des caractéristiques d'un objet Java** et ce, sans en avoir connaissance au début de l'apprentissage.

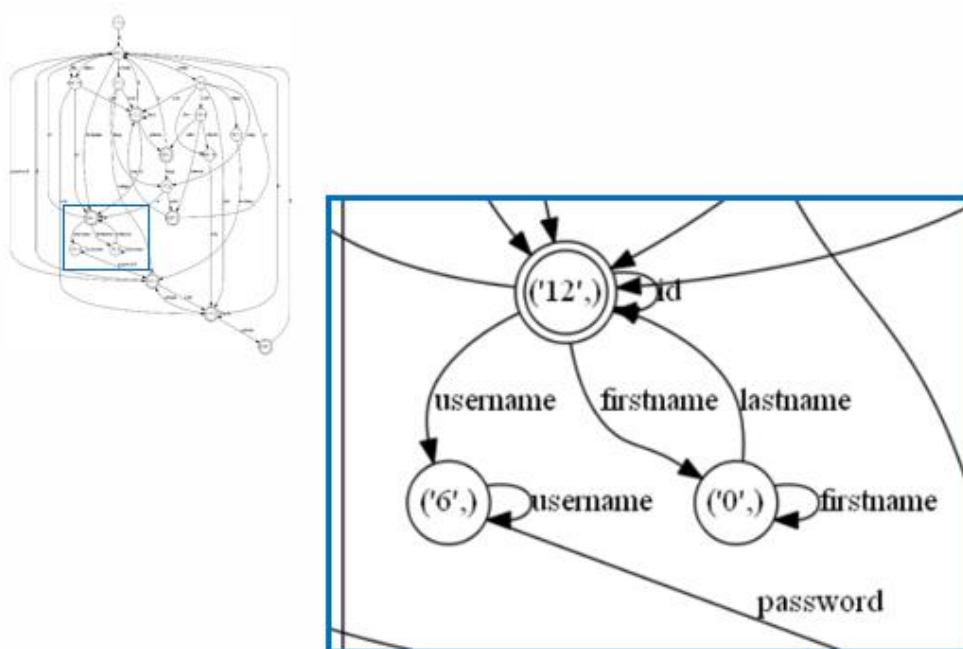


Figure III.21 - Extrait numéro 3 de l'automate minimisé présenté en figure III.18

Comme nous avons pu l'observer, ces **premiers résultats sont très encourageants et prometteurs**. Bien évidemment, lorsque nous parlons d'expertise métier des développeurs, nous remarquons qu'à ce jour, au vu de la démarche exploratoire que nous avons mis en œuvre, nous n'avons pas eu le temps de faire davantage d'analyses et d'exploration sur les représentations extraites.



Il serait aussi possible de réaliser de nouvelles simulations sur d'autres extraits du corpus Java d'origine. Néanmoins ces résultats nous confortent dans notre hypothèse de départ : il est possible d'extraire les règles implicites et donc de l'expertise dans des séquences de code Java.

D'autres exemples d'automates générés à partir des patterns d'activité ayant appris sur le même corpus de données Java sont disponibles en Annexe 8 de ce document.

### III.D. Discussion et suite du projet

Le projet IAC tel que nous venons de le présenter n'en est qu'à ses prémices. En effet notre démarche exploratoire est longue, non seulement du fait de sa nature (i.e. la recherche scientifique) mais aussi par les nombreuses analyses et simulations qu'il nous faut effectuer avant de pouvoir affirmer ou infirmer notre hypothèse de départ.

Néanmoins, comme nous avons pu le constater dans la section précédente, nous avons aujourd'hui des **résultats encourageants**. Nous sommes tout d'abord parvenus à implémenter un réseau de neurones récurrents qui apprend sur des données issues de la grammaire de Reber et sur des données issues de code Java. Ces premières données nous ont ainsi permis de **valider notre approche** et de nous assurer une base solide afin de **faire de l'interprétabilité sur des données issues de code Java**. Les premiers résultats obtenus au niveau de l'apprentissage de notre RNN mais aussi des automates que nous avons générés à partir du code Java nous prouvent la **faisabilité** de notre approche, la **solidité** de notre apprentissage et la **généricité** de celle-ci puisqu'elle a été transposée de manière réussie sur deux domaines d'application différents.

En ce qui concerne la suite du projet IAC, et surtout la phase d'interprétabilité, nous avons actuellement **deux axes majeurs et objectifs à moyen terme**. Le premier axe concerne davantage le niveau d'extraction de l'expertise que nous souhaitons effectuer. Le second axe concerne quant à lui un autre champ de recherche autour du traitement naturel du langage (NLP).

**Le premier axe**, que nous souhaitons explorer, consiste à revenir aux bases du projet, au niveau des données. En effet la transformation des lignes de code Java en vecteurs numériques dans le but de les fournir en entrée à notre réseau de neurones reste une zone d'ombre. Bien que nous connaissions la technique employée pour cette transformation nous n'avons à l'heure actuelle aucune certitude sur les impacts et effets qu'elle a eu. De nombreuses questions concernant la perte d'informations au moment de la transformation restent en suspens. Par exemple, la transformation des lignes de codes doit-elle se faire de manière séquentielle ou en prenant en compte le contexte de ces lignes ? En clair, le fait de transformer ligne par ligne est-il différent de transformer méthode par méthode ou classe par classe ? Voici quelques-unes des nombreuses questions auxquelles nous nous devons de répondre afin d'avoir une interprétation la plus objective possible de nos résultats.

**Le second axe** à moyen terme que nous cherchons à traiter concerne les prédictions de notre réseau de neurones. Bien que nous n'ayons pas détaillé ce processus dans ce mémoire, lorsque notre réseau a fini sa phase d'apprentissage et que nous passons à la phase de test, celui-ci effectue des prédictions vis-à-vis des données reçues en entrée. Ce processus, au niveau de la grammaire de Reber, est simple à gérer. En effet, les prédictions sont des vecteurs numériques d'une taille égale aux vecteurs numériques fournis en entrée qui prédisent des probabilités (celles que chaque caractère soit le prochain symbole). Chaque vecteur numérique prédit est transformé en vecteur binaire grâce, une nouvelle fois, à une technique de « *one-hot encoding* » qui permet de générer des vecteurs binaires



avec une seule unité non nulle correspondant à un symbole. Ainsi, pour chaque vecteur numérique, l'unité ayant la probabilité la plus élevée en son sein deviendra un « 1 » et les autres valeurs seront automatiquement mises à « 0 ». En appliquant ce principe, nous pouvons alors retrouver le symbole prédit par le réseau dans son format textuel. En revanche nous ne pouvons pas effectuer ce même processus pour les données Java du fait de leur nature. En effet, chaque mot Java a été transformé en un vecteur numérique et non binaire. Ainsi il nous est actuellement impossible de retrouver le symbole prédit par le réseau lors de la phase de test à partir de ces mêmes vecteurs numériques.

Une grande question se pose alors : comment transformer la prédiction de notre réseau de neurones en un mot Java et quel est l'impact de l'algorithme choisit coté NLP sur la robustesse de notre approche dans ce contexte ? Pour y répondre, nous devons chercher des solutions faisant partie du domaine du traitement du langage naturel TALN soit NLP pour *Natural Language Processing*. Ce domaine spécifique du Machine Learning consiste à générer du texte dans un langage naturel à partir de code informatique ou de valeurs numériques. Malheureusement nous n'avons jusqu'alors pas eu l'opportunité de mettre en place de telles techniques sur le projet IAC en raison de la complexité de leur implémentation et du temps imparti.

## Conclusion

**La gestion des connaissances et de l'expertise** est un axe de développement stratégique en entreprise. En effet, la valorisation de ces connaissances est primordiale afin de mener au mieux les projets actuels mais aussi repris par une entreprise. La bonne gestion des nombreuses connaissances métiers associées à chaque projet mais aussi à chaque collaborateur est un facteur important et preuve d'une bonne expertise en entreprise.

Néanmoins une problématique persiste au niveau de l'**identification**, de la **transmission** et de fait, de la **capitalisation de ces connaissances**. Tout d'abord il est important de différencier les deux types de connaissances présentes chez l'Homme. D'une part nous retrouvons les **connaissances explicites** qui sont exprimables et donc facilement transférables d'un individu à l'autre. De l'autre nous avons les **connaissances implicites** qui, comme nous l'avons vu, sont non conscientes mais essentielles. En effet c'est l'alliance de ces deux types de connaissances présentes en chacun de nous qui influence notre raisonnement. Les connaissances implicites, acquises par le vécu et l'expérience sont utilisées dans nos prises de décision. Ce phénomène, clairement identifié pour les développeurs et experts en informatique, influence donc particulièrement leur façon de coder. En partant de ce postulat, nous avons supposé qu'une trace, i.e. **une représentation de ces connaissances, se trouvait dans leur code**. Ne pouvant accéder directement aux connaissances implicites des collaborateurs, qui sont par définition inaccessibles (car non verbalisables), nous avons souhaité proposer une nouvelle approche d'identification et d'extraction de ces connaissances à travers le code informatique.

Nous avons ainsi vu ensemble comment l'utilisation de techniques de Machine Learning dans le cadre d'une **démarche exploratoire** peut nous aider à identifier ces connaissances implicites. Les réseaux de neurones artificiels, et plus particulièrement les réseaux de neurones récurrents, nous permettent de **simuler informatiquement le phénomène d'apprentissage implicite d'un individu** à travers l'apprentissage de code informatique. Ce code informatique, porteur des biais, du vécu et de l'expérience des développeurs l'ayant implémenté, est une source considérable de données que nous pouvons traiter et analyser. Nous avons ainsi réussi dans un premier temps à implémenter de bout en bout un réseau de neurone aux résultats encourageants ayant appris des séquences de code Java.

Nous nous sommes ensuite positionnés dans un domaine de recherche du **Machine Learning** en pleine expansion : **l'interprétabilité**. Ce domaine nous a ainsi permis d'analyser le comportement de notre réseau et d'évaluer son apprentissage. Nous avons ainsi, en implémentant une technique d'interprétabilité post-hoc de réseau de neurones, réussi à **extraire des connaissances implicites des développeurs présents dans leur code sous forme de règles**. Il nous est possible, à ce stade de nos travaux, de constater que ces règles que nous retrouvons au plus profond de notre réseau représentent des règles de structure inhérentes au langage Java mais aussi des habitudes des développeurs et des règles métiers liées aux contraintes projet.

Notre démarche exploratoire nous demandant beaucoup de ressources, d'expérimentations mais surtout de temps, ne nous permet pas aujourd'hui d'obtenir sur du court terme des solutions applicatives et fonctionnelles de nos résultats (rappelons que les travaux présentés dans ce mémoire se sont déroulé sur une période de 8 à 9 mois). La recherche étant ainsi faite, il s'agit ici d'un **engagement à plus long terme**. Néanmoins, les premiers résultats obtenus sont probants et valorisables de différentes manières comme par exemple une preuve de concept (POC), des publications scientifiques ou de médiation ou l'implémentation d'une approche générique. **D'un point de vue scientifique**, nos travaux contribuent à l'avancée de la recherche dans le domaine du Machine Learning et de l'interprétabilité des réseaux de neurones. Nous avons en effet prouvé qu'il était possible de fournir des interprétations de l'apprentissage d'un réseau à partir de données réelles, en l'occurrence, un nouveau domaine d'application : le code Java. Ces premiers résultats nous

permettent aussi de constater que **la recherche a toute sa place en entreprise** et permet de **proposer des solutions innovantes** à même d'aider à la capitalisation des connaissances et de l'expertise.

De manière plus concrète, en continuant nos travaux et en approfondissant nos recherches, nous serons capables **d'extraire et donc de formaliser l'expertise des développeurs** Java (dans un premier temps) afin d'aider les équipes projets. Les résultats attendus permettraient ainsi d'accompagner les développeurs à prendre conscience de leur expertise à travers une analyse de leur code via nos techniques. Ils nous permettraient également de rechercher et formaliser les connaissances métiers liées non plus à un seul développeur mais à une équipe projet dans son ensemble. Cette analyse permettrait notamment d'améliorer ou tout du moins de faciliter l'intégration d'un nouveau collaborateur sur le projet en fournissant une représentation des règles métiers de celui-ci.

Un dernier axe de recherche, résultant des travaux effectués lors du projet IAC qui nous intéresse particulièrement consiste en **la génération automatique d'explications ou de documentations**. Il s'agirait ici d'ajouter un nouveau module à nos travaux. Celui-ci reprendrait les informations extraites à partir de la couche cachée de notre réseau, tel que nous le faisons déjà aujourd'hui. En revanche, contrairement à nos travaux d'interprétabilité il s'agirait ici de se focaliser sur des recherches en **explicabilité des réseaux de neurones**. Ce module serait constitué de techniques de **traitement naturel du langage (NLP)** un autre domaine du Machine Learning et de techniques liées au domaine des mécanismes de l'attention afin de déterminer les parties du code pertinentes à documenter. Il nous permettrait, à partir des informations récupérées après apprentissage de notre réseau, de générer automatiquement des mots ou des phrases formant une explication des données recueillies. Ces explications pourraient ainsi nous permettre de fournir une documentation d'une part, sur le comportement de notre réseau et donc **d'aider à la compréhension de ses mécanismes de raisonnement** et, d'autre part, de **fournir des explications sur les connaissances implicites cachées dans les données**. Nous pensons que ce module serait complémentaire pour nos travaux actuels et que l'association interprétabilité / explicabilité serait un levier fort pour la capitalisation de l'expertise en entreprise.

Enfin, nous ne pouvons pas parler de Machine Learning sans penser à **l'éthique et la transparence des algorithmes d'intelligence artificielle**. Les algorithmes d'IA et leurs prises de décisions sont influencés par les données qu'ils apprennent. En effet, une forte problématique se pose autour **des biais présents dans les données** mais aussi **des biais cognitifs présents chez les développeurs** implémentant les algorithmes. De mauvais jeux de données peuvent amener à des conséquences fâcheuses. Si un algorithme d'IA apprend des données des règles erronées ou biaisées alors ses décisions le seront tout autant et provoqueront ce que l'on appelle des **biais de sélection**. Un exemple tristement célèbre de l'impact de ces biais est la description fournie par Google en 2015 sur une photo d'un couple de personnes afro-américaines comme étant des gorilles. Comme nous le constatons, l'impact de ces biais peut s'avérer très grave surtout dans des domaines critiques tels que la finance, la défense ou encore la santé. De nos jours, de nombreux travaux existent dans ce domaine afin d'étudier les biais. Néanmoins il est important de souligner, que **prendre connaissance de ces biais ne signifie pas obligatoirement que nous savons comment les supprimer**. L'intérêt que nous portons au domaine de l'interprétabilité mais aussi de l'explicabilité des algorithmes d'IA prend tout son sens dans ces **questions sociétales**.

Ainsi, nous espérons, à l'aide de nos travaux, aider à **démystifier l'IA** et à rendre ces « **boîtes noires** » **plus transparentes, intelligibles et compréhensibles**.

---

*"What all of us have to do is to make sure we are using AI in a way that is for the benefit of humanity, not to the detriment of humanity."*

Tim Cook, CEO Apple Inc.

---

## Glossaire

**ANN - Artificial Neural Network** : réseau de neurones artificiels utilisé en Machine Learning

**Apprentissage supervisé** : entraînement d'un modèle de Machine Learning sur un ensemble de données étiquetées via un couple entrée / sortie désirée

**Apprentissage non supervisé** : entraînement d'un modèle de Machine Learning sur un ensemble de données non étiquetées dont leur nature n'est pas prédéterminée

**Apprentissage par renforcement** : entraînement d'un modèle de Machine Learning selon une observation réalisée au sein de l'environnement et d'un système de récompense

**Automate** : construction mathématique abstraite composée d'états, de symboles et de transitions pouvant être représentée sous la forme d'un graphe

**Citation de Tim Cook** : « *Ce que nous devons tous faire, c'est nous assurer que nous utilisons l'IA d'une manière qui est dans l'intérêt de l'humanité, et non au détriment de l'humanité.* »

**Clustering** : méthode de partitionnement de données qui consiste à les regrouper selon une ou plusieurs caractéristiques communes

**Connaissances explicites** : ensemble des connaissances qu'un individu possède et peut exprimer

**Connaissances implicites** : connaissances dont un individu n'est pas conscient et sont, de fait, non exprimables

**Expert** : personne surentraînée ayant acquis des compétences et connaissances spécifiques par son expérience et son vécu

**Expertise métier** : connaissances et compétences d'un collaborateur acquises principalement par l'expérience dans un domaine spécifique

**Explicabilité** : champ de recherche qui consiste à fournir une information dans un format sémantique complet, se suffisant à lui-même et accessible à tout utilisateur

**Grammaire de Reber** : grammaire à états finis composée de 7 symboles (B, T, P, X, S, V et E) conçue initialement pour des expériences de psychologie cognitive et désormais utilisée pour entraîner des algorithmes de Machine Learning lors de tâches de modélisation de l'apprentissage implicite ou de tâches d'apprentissage supervisé de séquences.

**IA - Intelligence Artificielle** : ensemble des théories et des techniques mises en œuvre en vue de réaliser des machines capables de simuler l'intelligence

**IAC - Interprétabilité Appliquée au Code** : acronyme du projet servant de cadre d'étude à ce mémoire

**Interprétabilité** : champ de recherche dédié à la compréhension des rouages et le comportement du modèle de Machine Learning dans un format interprétable par un expert

**Interprétabilité post-hoc** : technique d'interprétabilité appliquée sur un réseau de neurones artificiels après apprentissage de celui-ci et qui se focalise sur l'étude de l'espace latent d'un réseau de neurones

**K-means** : algorithme de clustering permettant de segmenter les données en un nombre de groupes (k) en minimisant la distance entre les données situées à l'intérieur d'un même groupe (cluster)

**LSTM - Long Short Term Memory** : unités de calcul dans un réseau de neurones artificiels qui permet de gérer la mémoire à court terme mais aussi à long terme grâce à un système de portes et de cellules mémoires

**ML - Machine Learning** : également nommé apprentissage automatique ou apprentissage machine, il s'agit d'un sous-domaine de recherche de l'intelligence artificielle qui consiste à donner la capacité d'apprentissage à une machine (un ordinateur) sans que celle-ci ait été explicitement programmée pour cela

**Neurone artificiel** : également nommé neurone formel. Il s'agit d'une unité de calcul dans les réseaux de neurones artificiels

**One hot encoding** : technique de génération de vecteurs binaires avec une seule unité non nulle

**R&D - Recherche et Développement** : activités créatives et systématiques entreprises en vue d'accroître la somme des connaissances et de concevoir de nouvelles applications

**Raisonnement** : processus cognitif permettant de poser un problème de manière réfléchie en vue d'obtenir un ou plusieurs résultats

**RNN - Recurrent Neural Network** : réseau de neurones artificiels comportant des connexions récurrentes (boucles)

**Sciences cognitives** : étude détaillée des mécanismes de la pensée humaine, animale ou artificielle afin de comprendre et d'expliquer le fonctionnement de l'esprit humain

## Liste des Figures

### Chapitre I. Projet « Interprétabilité Appliquée au Code », alias IAC

<b>Figure I.1</b> - Cercle R&D au sein de la communauté Sud-Ouest de onepoint .....	4
<b>Figure I.2</b> - Les étapes de la démarche scientifique .....	7
<b>Figure I.3</b> - Les sciences composant les sciences cognitives.....	9
<b>Figure I.4</b> - Différentes mémoires à long terme. Image adaptée de Squire [1992] et extraite de [Chraïbi Kaadoud et Delmas, 2018] .....	10
<b>Figure I.5</b> - Evolution du temps nécessaire à la résolution de tâche au fur et à mesure de la montée en compétences (pratique) selon trois étapes, basée sur la théorie de Fitts et Posner [1967], Anderson [1983], Rasmussen [1986] et Van Lehn [1996]. Image adaptée de Kim et al. [2013] extraite de Chraïbi Kaadoud [2018] .....	11
<b>Figure I.6</b> - Représentation schématique de l'approche générique d'extraction de règles implémentée dans le cadre du projet IAC.....	14

### Chapitre II. Projet IAC - réseau de neurones artificiels

<b>Figure II.1</b> - Réseau de neurones artificiels composé de 3 couches .....	15
<b>Figure II.2</b> – Représentation schématique du fonctionnement d'un neurone formel. Image tirée de [Giroux, 2019] .....	16
<b>Figure II.3</b> - Représentation schématique du concept d'apprentissage d'un réseau de neurones artificiels. Phases d'entraînement, de validation et de test. ....	17
<b>Figure II.4</b> - Différents types d'apprentissage en Machine Learning .....	17
<b>Figure II.5</b> – Représentation schématique des 4 étapes de la rétropropagation de l'erreur lors de la phase d'apprentissage d'un réseau de neurones .....	19
<b>Figure II.6</b> - Implémentation avec Keras du réseau de neurones présenté en figure II.1 .....	21
<b>Figure II.7</b> - Structure d'un réseau de neurones récurrents, RNN .....	22
<b>Figure II.8</b> - Structure simplifiée d'une unité LSTM présente dans la couche cachée d'un RNN. Dans le réseau de neurone doté d'une couche cachée d'unités LSTM, chaque cercle blanc est lié par un poids à chaque cercle noir .....	24
<b>Figure II.9</b> - RNN-LSTM implémenté pour le projet IAC .....	25
<b>Figure II.10</b> - Implémentation via Keras du RNN-LSTM du projet IAC.....	25
<b>Figure II.11</b> - Grammaire de Reber représentée sous forme de graphe orienté .....	28
<b>Figure II.12</b> - Etapes de transformation des données issues de la grammaire de Reber .....	29
<b>Figure II.13</b> - Transformation d'un accesseur Java en séquences de mots Java .....	30
<b>Figure II.14</b> - Ajout de symboles de début et de fin à des séquences de mots Java .....	31
<b>Figure II.15</b> - Apprentissage sur la grammaire de Reber : Courbes d'erreur (loss) du RNN pour un apprentissage sur 3000 séquences pour un learning rate de 0,1 (a) et de 0,01 (b) sur 10 epochs. ....	32
<b>Figure II.16</b> - Apprentissage sur la grammaire de Reber : Courbes de précision (accuracy) du RNN pour un apprentissage sur 3000 séquences pour un learning rate de 0,1 (a) et de 0,01 (b) sur 10 epochs. ....	33
<b>Figure II.17</b> - Apprentissage sur le code Java : Courbes de loss (a) et d'accuracy (b) du RNN pour un apprentissage sur 1000 séquences pour un learning rate de 0,001 sur 10 epochs. ....	33

### Chapitre III. Projet IAC - interprétabilité et extraction de l'expertise

<b>Figure III.1</b> - Représentation schématique d'un réseau de neurones selon le principe d'une « boîte noire ».....	34
<b>Figure III.2</b> – Niveaux d'interprétabilité des algorithmes de Machine Learning. Image adaptée à partir de Dam et al. [2018] .....	36

<b>Figure III.3</b> - Présentation schématique de l'approche générique du processus d'extraction des règles à partir d'un RNN.....	38
<b>Figure III.4</b> - Représentation schématique de l'espace latent d'un RNN doté de 2 cellules cachées en (a) et de l'espace latent du RNN implémenté pour le projet IAC en (b) .....	39
<b>Figure III.5</b> - Exemple de visualisation d'un clustering obtenu avec l'algorithme des k-means, pour k=3 .....	40
<b>Figure III.6</b> - Exemple de génération d'automate avec la liste de clusters [3, 2] et la liste de patterns [B, T, X, S] associés aux pas de temps [t0, t1, t2, t3]. .....	41
<b>Figure III.7</b> - Exemple d'automates à états finis générés à partir d'un RNN sur la grammaire de Reber. a) k=2 b) k=5 et c) k=9 dans le cadre de l'algorithme des k-means.....	42
<b>Figure III.8</b> - Représentation schématique du processus d'extraction des règles à partir d'un RNN, des étapes de génération d'un automate à états finis à l'obtention d'un automate déterministe minimisé .....	42
<b>Figure III.9</b> - Exemple d'un automate non déterministe représenté en (a) et d'un automate déterministe représenté en (b) dans le contexte de la grammaire de Reber pour k = 2 selon l'algorithme des k-means .....	43
<b>Figure III.10</b> – Exemple d'un automate déterministe complet (a) avec l'automate déterministe minimisé associé (b) dans le contexte de la grammaire de Reber pour k=3 selon l'algorithme des k-means .....	44
<b>Figure III.11</b> - Automate à états finis pour k-means=6 extrait à partir d'un RNN entraîné sur la grammaire de Reber (400 patterns).....	47
<b>Figure III.12</b> - Automate déterministe complet généré à partir de l'automate à états finis représenté en figure III.11. ....	47
<b>Figure III.13</b> - Automate minimisé obtenu à partir de l'automate complet présenté en figure III.12. 48	
<b>Figure III.14</b> - Pourcentages de séquences de la grammaire de Reber et aléatoires acceptées par les automates pour k appartenant à l'intervalle [2, 19] et valeur des silhouette scores calculés associés pour chaque valeur de k .....	49
<b>Figure III.15</b> - Pourcentages de séquences de code Java acceptées par les automates pour k appartenant à [2, 26] et valeur des silhouette scores calculés associés pour chaque valeur de k.....	50
<b>Figure III.16</b> - Automate à états finis pour k=19 selon l'algorithme des k-means. Extraction réalisée à partir des 100 premiers patterns d'activité d'un RNN entraîné sur le code Java .....	51
<b>Figure III.17</b> - Transitions partant du nœud 1 sur l'automate à états finis (a) et sur l'automate complet (b) généré à partir de celui-ci pour k=19 présentées au format dot.....	52
<b>Figure III.18</b> - Automate minimisé obtenu à partir de l'automate à états finis présenté en figure III.16 .....	53
<b>Figure III.19</b> - Extrait numéro 1 de l'automate minimisé présenté en figure III.18 .....	54
<b>Figure III.20</b> - Extrait numéro 2 de l'automate minimisé présenté en figure III.18 .....	55
<b>Figure III.21</b> - Extrait numéro 3 de l'automate minimisé présenté en figure III.18 .....	56

## Liste des tableaux

### Chapitre II. Projet IAC - réseau de neurones artificiels

<b>Tableau II.1</b> - Exemple de paires issues de la séquence BTXSE dans le cadre de la grammaire de Reber .....	29
<b>Tableau II.2</b> - Association des symboles issue de la grammaire de Reber à des vecteurs binaires .....	29
<b>Tableau II.3</b> - Caractéristiques des corpus utilisés pour les phases d'apprentissage, de validation et de test pour la grammaire de Reber et le code Java. Une paire est constituée du symbole d'entrée <b>et</b> du symbole attendu en sortie .....	31

### Chapitre III. Projet IAC - interprétabilité et extraction de l'expertise

<b>Tableau III.1</b> - Caractéristiques des corpus utilisés pour la génération des automates extraits à partir des données collectés lors de la phase de test du RNN .....	45
<b>Tableau III.2</b> - Pourcentages de séquences grammaticales et non grammaticales acceptées par l'automate minimisé. Application à la grammaire de Reber .....	48
<b>Tableau III.3</b> - Nombre d'occurrences des mots clés Java présents dans les données sur un total de 100 mots .....	55



## Bibliographie

**ANDERSON, JR**, 1983. Cognitive science series. the architecture of cognition.

**BERSTEL, Jean, BOASSON , Luc, CARTON, Olivier et FAGNOT, Isabelle**, 2010. « Minimization of Automata », dans Automata: from Mathematics to Applications, European Mathematical Society, 2010 (arXiv 1010.5318)

**BOOTZ, JP**, 2016. Comment définir et gérer l'expert.

**CHRAIBI KAADOUD, Ikram**, 2018. Apprentissage de séquences et extraction de règles de réseaux récurrents : application au traçage de schémas techniques. Autre [cs.OH]. Université de Bordeaux, 2018. Français. (NNT : 2018BORD0032). (tel-01771685)

**CLEERMANS, Axel**, 1993. Mechanisms of implicit learning : Connectionist models of sequence processing. MIT press.

**CRIE, Dominique**, 2003. De l'extraction des connaissances au knowledge management. Revue française de gestion, (5) :59–79.

**DALEY, B. J.**, 1999. Novice to Expert: An Exploration of How Professionals Learn. Adult Education Quarterly, 49(4), 133–147.

**DAM Hoa K et al.**, 2018. Explainable Software Analytics. International Conference on Software Engineering 2018 (pp. 53-56).

**ETTLINGER, Marc, MARGULIS, Elizabeth H et WONG, Patrick CM**, 2011. Implicit memory in music and language. Frontiers in psychology, 2.

**FITTS, Paul M et POSNER, Michael I**, 1967. Human performance. brooks. Cole, Belmont, CA, 5 :7–16.

**GERS, Felix A, SCHMIDHUBER, Jürgen et CUMMINS, Fred**, 1999. Learning to forget : Continual prediction with lstm.

**GILPIN, Leilani H., BAU, David, YUAN, Ben Z., et al.**, 2018 Explaining explanations: An overview of interpretability of machine learning. In : 2018 IEEE 5th International Conference on data science and advanced analytics (DSAA). IEEE, 2018. p. 80-89.

**GRUNDSTEIN, Michel**, 1995. La capitalisation des connaissances de l'entreprise, système de production des connaissances. Dans Actes du Colloque L'Entreprise Apprenante et les Sciences de la Complexité, Aix en Provence, France, pages 22–24.

**HAZAN, Elad, KLIVANS, Adam, et YUAN, Yang**, 2017. Hyperparameter optimization: A spectral approach. arXiv preprint arXiv:1706.00764.

**HELBING, Dirk**, 2019 Societal, economic, ethical, and legal challenges of the digital revolution: from big data to deep learning, artificial intelligence, and manipulative technologies. In : Towards Digital Enlightenment. Springer, Cham, 2019. p. 47-72.

**HOCHREITER, Sepp et SCHMIDHUBER, Jürgen**, 1997. Long short-term memory. Neural computation, 9(8):1735-1780.

**HOPCROFT, JE et al.**, 1979. Ullman. jd (1979) introduction to automata theory, languages, and computation.

**JACOBSSON, Henrik**, 2005. Rule extraction from recurrent neural networks : A taxonomy and review. Neural Computation, 17(6) :1223–1263.

- KIM, JongW, RITTER, Frank E et KOUBEK, Richard J**, 2013. An integrated theory for improved skill acquisition and retention in the three stages of learning. *Theoretical Issues in Ergonomics Science*, 14(1) :22–37.
- KINGMA, Diederik P. et BA, J. Adam**, 2014. A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2019, vol. 434.
- MALHOTRA, Yogesh**, 2000. Knowledge management and new organization forms : A framework for business model innovation. *Knowledge management and virtual organizations*, 2(1) :13–27.
- MIYAKE, Akira, FRIEDMAN, Naomi P, EMERSON, Michael J, WITZKI, Alexander H et HOWERTER, Amy**, 2000. The Unity and Diversity of Executive Functions and Their Contributions to Complex “Frontal Lobe” Tasks: A Latent Variable Analysis. *Cognitive Psychology* 41, 49–100 (2000) doi:10.1006/cogp.1999.0734
- NONAKA, Ikujiro et TAKEUCHI, Hirotaka**, 1995. The knowledge-creating company : How Japanese companies create the dynamics of innovation. Oxford university press.
- OMLIN, Christian, W et GILES, C Lee**, 1996. Extraction of rules from discrete-time recurrent neural networks. *Neural networks*, 9(1) :41–52.
- RASMUSSEN, Jens**, 1986. Information processing and human-machine interaction. an approach to cognitive engineering.
- SCHACTER, Daniel L**, 1987. Implicit memory : History and current status. *Journal of experimental psychology : learning, memory, and cognition*, 13(3) :501.
- SERVAN-SCHREIBER, D, CLEERMANS, A et McCLELLAND, JL**, 1988. Encoding sequential structure in simple recurrent networks. School of Computer Science, Carnegie-Mellon University, Pittsburgh, PA, CMU-CS-88-183.(November 1988).
- SMITH, Elizabeth A**, 2001. The role of tacit and explicit knowledge in the workplace. *Journal of knowledge Management*, 5(4) :311–321
- SQUIRE, Larry R**, 1992. Declarative and nondeclarative memory : Multiple brain systems supporting learning and memory. *Journal of cognitive neuroscience*, 4(3) :232–243.
- VANEHN, Kurt**, 1996. Cognitive skill acquisition. *Annual review of psychology*, 47(1) :513–539.
- WIEDENBECK, Susan**, 1985. Novice/expert differences in programming skills. *International Journal of Man-Machine Studies*, 1985, vol. 23, no 4, p. 383-390.

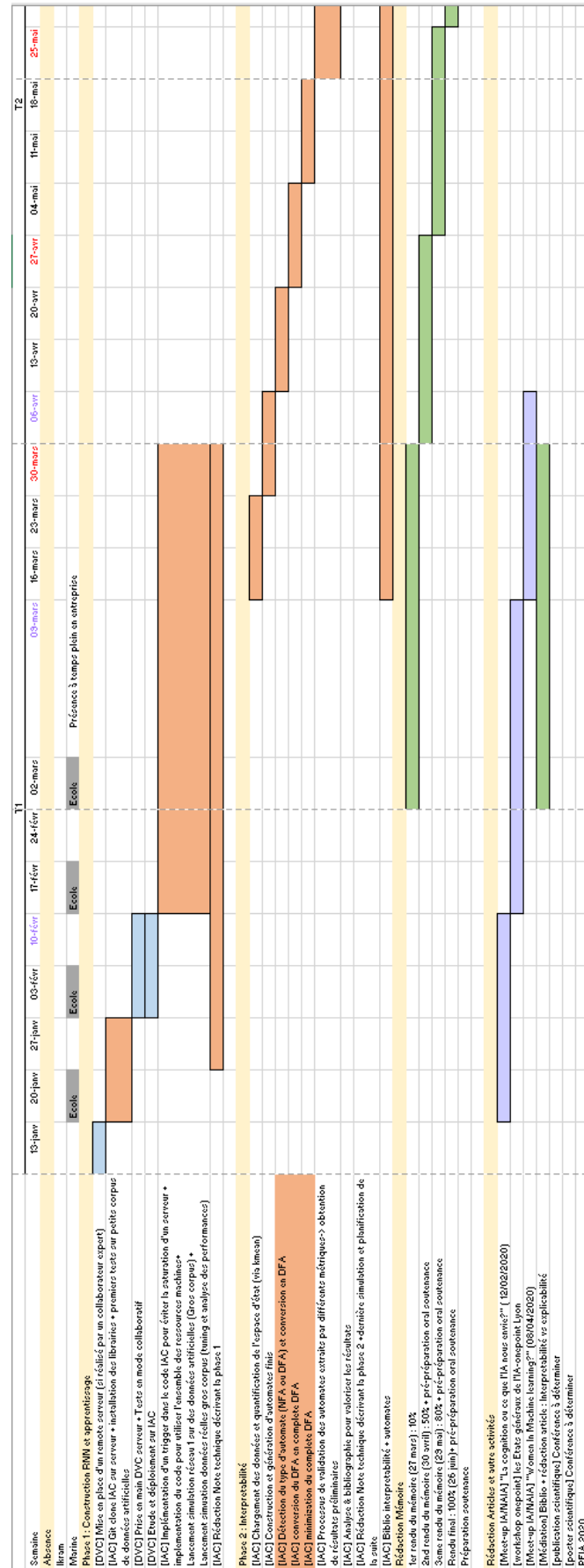
## Webographie

- Accuracy**, 2020. Accuracy (error rate). URL : <https://deeptai.org/machine-learning-glossary-and-terms/accuracy-error-rate>
- App.Supervisé**, 2020. Supervised Learning. URL : <https://machinelearningmastery.com/types-of-learning-in-machine-learning/>
- App.NonSupervisé**, 2020. Unsupervised Learning. URL: <https://machinelearningmastery.com/types-of-learning-in-machine-learning/>
- CHRAIBI KAADOU, Ikram et DELMAS, Alexandra**, 2018. La cognition ou qu'est-ce que les sciences cognitives ? URL : <http://www.scilogs.fr/intelligence-mecanique/la-cognition-ou-quest-ce-que-les-sciences-cognitives/>
- CLAPAUD, Alain**, 2019. Explicabilité des IA : quelles solutions mettre en œuvre ? Publié sur LeMagIT. URL : <https://www.lemagit.fr/conseil/Explicabilite-des-IA-queelles-solutions-mettre-en-uvre>
- COE**, 2020. History of Artificial Intelligence. Publié sur le site du Conseil de l'Europe. URL : <https://www.coe.int/en/web/artificial-intelligence/history-of-ai>
- DORMEHL**, 2019. What is an artificial neural network ? URL : <https://www.digitaltrends.com/cool-tech/what-is-an-artificial-neural-network/>
- GALAIS Thomas, CHRAIBI KAADOU, Ikram et DELMAS, Alexandra**, 2020. Que comprendre par recherche scientifique ? URL : <https://www.groupeonepoint.com/fr/points-de-vue/que-comprendre-par-recherche-scientifique>
- GIRAUX**, 2019. Comment fonctionne un réseau de neurones. URL : <https://aiandi.fr/comment-fonctionne-un-reseau-de-neurones/>
- INSEE**, 2019. Définition : Recherche et développement / R&D / Travaux de recherche et développement. URL : <https://www.insee.fr/fr/metadonnees/definition/c1174>
- CRAWFORD Kate**, 2019. « les biais sont devenus le matériel brut de l'IA ». URL : <https://www.lemonde.fr/blog/internetactu/2019/10/03/kate-crawford-les-biais-sont-devenus-le-materiel-brut-de-lia/>
- K-means**, 2020. K-means clustering. URL : <https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html>
- LABADILLE, Pierre**, 2017. Réseaux de neurones artificiels ou la révolution du web moderne. URL : <https://www.imagile.fr/reseaux-de-neurones-artificiels-revolution-web-moderne/>
- Renforcement**, 2020. Reinforcement Learning. URL : <https://machinelearningmastery.com/types-of-learning-in-machine-learning/>
- THAGARD, Paul**, 2019. "Cognitive Science", The Stanford Encyclopedia of Philosophy (Spring 2019 Edition), Edward N. Zalta (ed.), URL : <https://plato.stanford.edu/archives/spr2019/entries/cognitive-science>.
- VERINE, Alexandre et MIR, Stéphan**, 2019. L'interprétabilité du machine learning : quels défis à l'ère des processus de décision automatisés ? URL : <https://www.wavestone.com/fr/insight/interpretabilite-machine-learning/>
- WhatIs**, 2016. Machine Learning definition. Publié sur le site WhatIs.com URL : <https://whatIs.techtarget.com/fr/definition/Machine-Learning>

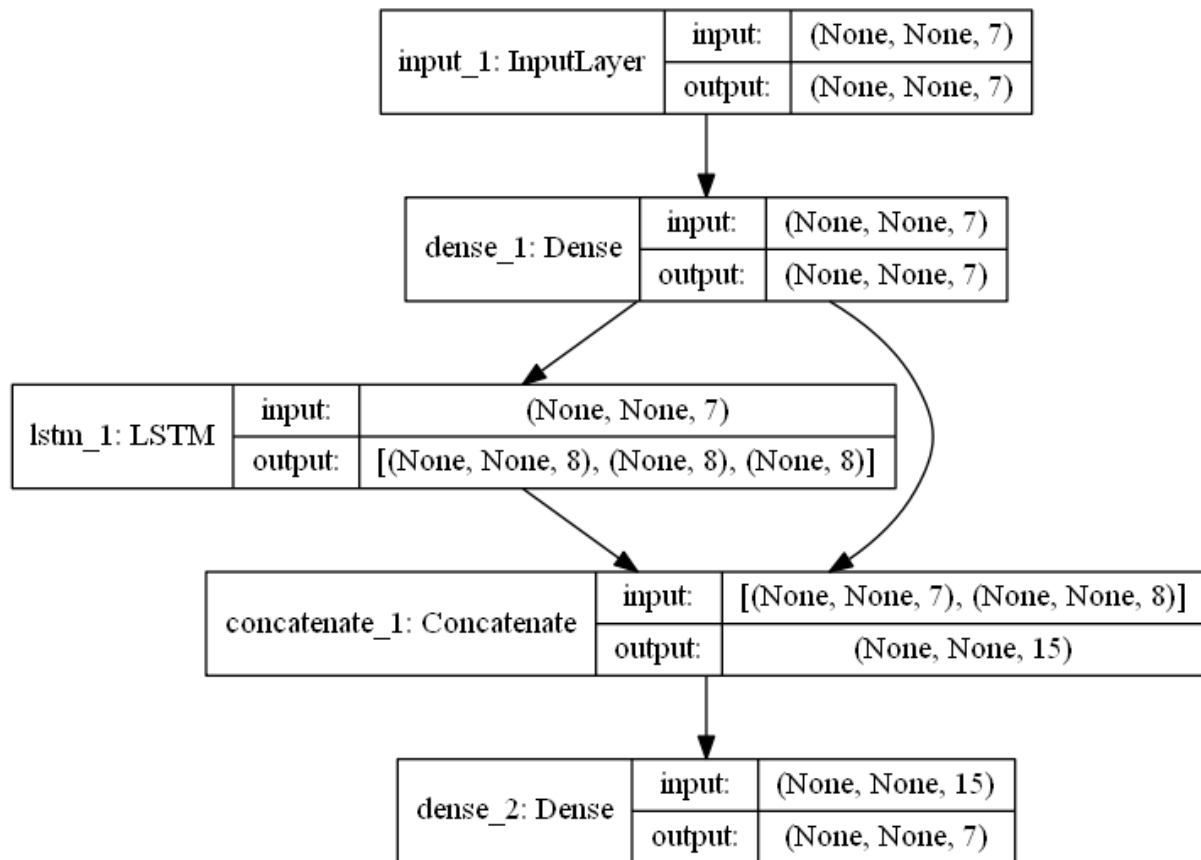
## Annexes

<b>Annexe 1.</b> Extrait du planning prévisionnel du projet IAC sous la forme d'un diagramme de Gantt.....	I
<b>Annexe 2.</b> Représentation graphique du RNN implémenté pour le projet IAC. Génération effectuée via Keras.....	II
<b>Annexe 3.</b> Liste des données et informations sauvegardées lors des phases d'apprentissage, de validation et de test du RNN.....	III
<b>Annexe 4.</b> Algorithme d'extraction des règles sous la forme d'un automate (GA) doté d'étiquettes longues à partir des patterns d'activité d'un RNN-LSTM. Image extraite de Chraïbi Kaadoud [2018].....	IV
<b>Annexe 5.</b> Extraits des fichiers de sauvegarde d'un automate à états finis pour k-means=2 au format JSON et DOT.....	V
<b>Annexe 6.</b> Extrait de l'automate à états finis à étiquettes longues généré à partir d'un automate à états finis. Contexte de la grammaire de Reber pour k=6 sur les 400 premiers patterns d'activité.....	VI
<b>Annexe 7.</b> Séquences de mots Java et lignes de code Java correspondant aux 100 premiers patterns utilisés pour l'extraction de règles sous la forme d'automates.....	VII
<b>Annexe 8.</b> Extraction des patterns d'activité sous la forme d'un automate minimisé sur du code Java supplémentaire reflétant une fonction complexe.....	VIII

## 1



Annexe 2. Représentation graphique du RNN implémenté pour le projet IAC. Génération effectuée via Keras



Annexe 3. Liste des données et informations sauvegardées lors des phases d'apprentissage, de validation et de test du RNN.

Données	Format de fichier
<b>Le modèle</b> <ul style="list-style-type: none"> <li>L'architecture du modèle permettant de recréer le modèle et ses poids</li> <li>Les poids du modèle</li> <li>La configuration utilisée pour l'apprentissage (perte, optimiseur)</li> <li>L'état de l'optimiseur permettant de reprendre l'entraînement au moment où la sauvegarde a eu lieu</li> </ul>	HD5
<b>Les poids</b> <ul style="list-style-type: none"> <li>L'ensemble des poids du modèles sous un autre format</li> </ul>	TXT
<b>Toutes les pertes (<i>history losses</i>)</b> <ul style="list-style-type: none"> <li>Pour chaque paire entraînée</li> <li>Sur tous les epochs effectués</li> </ul>	NPY
<b>Les pertes par epoch pour l'apprentissage</b> <ul style="list-style-type: none"> <li>Moyenne de la <i>Loss</i> pour chaque epoch</li> </ul>	NPY
<b>Les pertes par epoch pour la validation</b> <ul style="list-style-type: none"> <li>Moyenne de la <i>Loss</i> pour chaque epoch</li> </ul>	NPY
<b>La précision par epoch pour l'apprentissage</b> <ul style="list-style-type: none"> <li>Moyenne de l'<i>accuracy</i> pour chaque epoch</li> </ul>	NPY
<b>La précision par epoch pour la validation</b> <ul style="list-style-type: none"> <li>Moyenne de l'<i>accuracy</i> pour chaque epoch</li> </ul>	NPY
<b>Les prédictions effectuées lors du test</b>	TXT
<b>L'ensemble des métriques</b> des phases d'apprentissage, de validation et de test	JSON
<b>Les sorties attendues</b> (vecteurs binaires)	JSON
<b>Les séquences utilisées pour l'apprentissage</b>	JSON
<b>Les données fournies en entrée</b> (vecteurs binaires)	JSON
<b>Les sorties obtenues</b> (vecteurs numériques)	JSON
<b>Les valeurs des CEC</b> de la couche cachée (vecteurs numériques)	JSON
<b>Les patterns d'activité</b> de la couche cachée (vecteurs numériques)	JSON

Annexe 4. Algorithme d'extraction des règles sous la forme d'un automate (GA) doté d'étiquettes longues à partir des patterns d'activité d'un RNN-LSTM. Extrait de [Chraibi Kaadoud \[2018\]](#).

---

```

Pré-conditions : # Apprentissage et test du réseau de neurones —
RNN_LSTM.apprentissage(corpus_apprentissage)
# liste_etiquettes : liste des symboles présentés au réseau durant les tests
liste_patterns_activité, liste_etiquettes = RNN_LSTM.test(corpus_test)

Fonction extraction_règles (liste_patterns_activité,
liste_etiquettes, k) :
# Clustering —
liste_clusters = k_means(k, liste_patterns_activité)

# Creation automate —
GA = {} # Dictionnaire
noeud_courant = -1
GA['noeuds'].ajouter(noeud_courant)
GA['arcs'] = [] # Liste de dictionnaires
Pour tout pattern h d'indice i de la liste_patterns_activité faire
    cluster_associe = liste_clusters[i]
    Si cluster_associe ∉ GA['noeuds'] alors
        GA['noeuds'].ajouter(cluster_associe)
    Fin Si
    arc = {} # Dictionnaire
    arc['id'] = (noeud_courant, cluster_associe)
    Si arc ∉ GA['arcs'] alors
        nouvel_arc = arc
        nouvel_arc['poids'] = 1
        nouvel_arc['etiquette'] = liste_etiquettes[i]
        GA['arcs'].ajouter(nouvel_arc)
    Sinon
        arc['poids'] = arc['poids'] + 1
        arc['etiquette'] = arc['etiquette'] + liste_etiquettes[i]
        GA['arcs'].mise_à_jour(arc)
    Fin Si
    # Mise à jour de la valeur du noeud courant
    noeud_courant = cluster_associe
Fin Pour

return GA

Fin Fonction

```

---



## Annexe 5. Extraits des fichiers de sauvegarde d'un automate à états finis pour k-means = 2 au format JSON et DOT.

```
{
  "nodes": [
    "-1",
    "1",
    "0"
  ],
  "transitions": [
    {
      "id": [
        "-1",
        "1"
      ],
      "long_label": "B0",
      "short_label": "B",
      "weight": 1
    },
    {
      "id": [
        "1",
        "1"
      ],
      "long_label": "P1T5X6T9T14S15X16S17B18P19T2",
      "short_label": "PTXSBV",
      "weight": 178
    },
    {
      "id": [
        "1",
        "0"
      ],
      "long_label": "V2X7V10S12V21S23V30X35S38V41",
      "short_label": "VXS",

```

Annexe 5.a – extrait du fichier de sauvegarde JSON d'un automate à états finis pour k-means=2

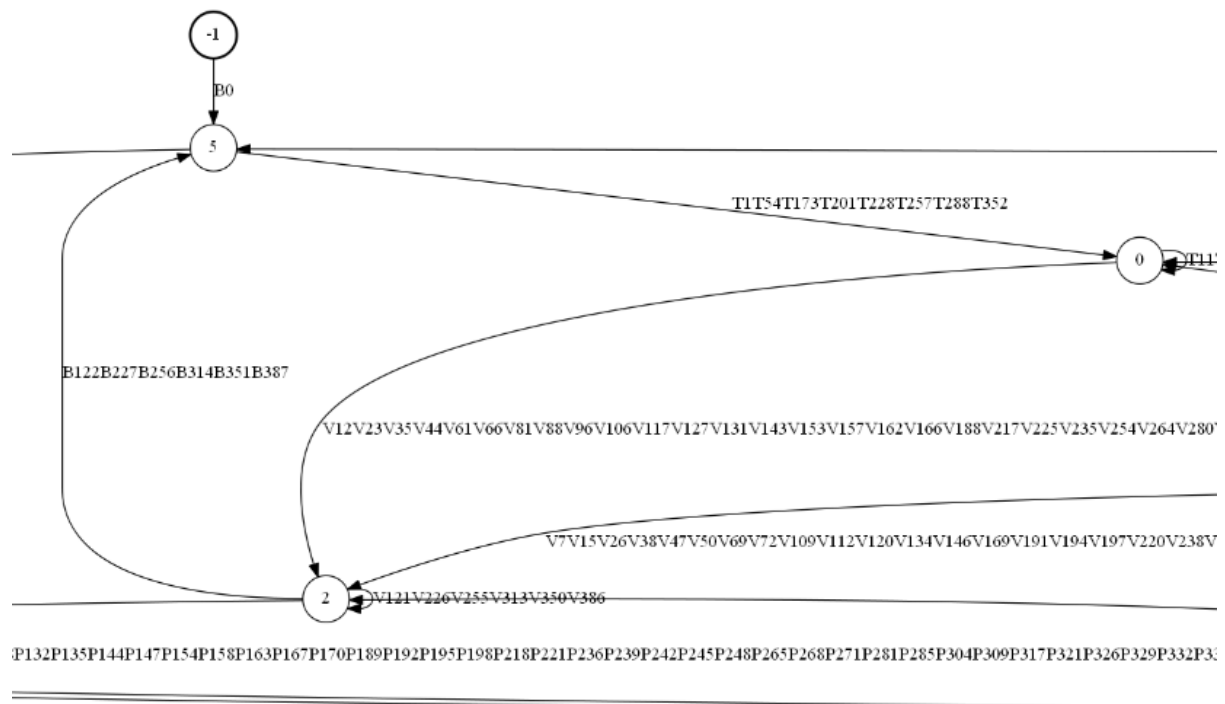
La première partie du fichier définit les nœuds de l'automate (*nodes*). Elle est suivie des transitions représentées par plusieurs informations qui sont : un identifiant (*id*) représentant les nœuds liés par cette transition, l'étiquette longue (*long\_label*), l'étiquette courte (*short\_label*) et enfin le poids de la transition (*weight*) qui indique le nombre de fois où celle-ci est utilisée lors du parcours du graphe.

```
digraph "FSM" {
  "-1" [final=False, fixedsize=True, initial=True, penwidth=2, shape=circle, sink=False];
  1 [final=True, fixedsize=True, initial=False, penwidth=1, shape=doublecircle, sink=False];
  0 [final=True, fixedsize=True, initial=False, penwidth=1, shape=doublecircle, sink=False];
  "-1" -> 1 [key=B, label=B];
  1 -> 1 [key=PTXSBV, label=PTXSBV];
  1 -> 0 [key=VXS, label=VXS];
  0 -> 0 [key=V, label=V];
  0 -> 1 [key=BTP, label=BTP];
}
```

Annexe 5.b – Extrait du fichier de sauvegarde DOT d'un automate à états finis pour k-means=2

Le fichier Dot présente les nœuds du graphes et leur nature. Il définit ensuite les transition identifiées par les deux nœuds impliqués (ex : « -1 » -> 1), l'identifiant de la transition (*key*) et l'étiquette courte (*label*).

**Annexe 6.** Extrait de l'automate à états finis à étiquettes longues généré à partir d'un automate à états finis. Contexte de la grammaire de Reber pour k=6 sur les 400 premiers patterns d'activité



Un exemple d'une transition extraite de cet automate partant du nœud 2 au nœud 5 est « B122 B227 B256 B314 B351 B387 ». Comme nous pouvons l'observer, les étiquettes des transitions présentes sur cet extraits sont très longues et complexes, de ce fait il est impossible de visualiser graphiquement cet automate.

Annexe 7. Séquences de mots Java et lignes de code Java correspondant aux 100 premiers patterns utilisés pour l'extraction de règles sous la forme d'automates.

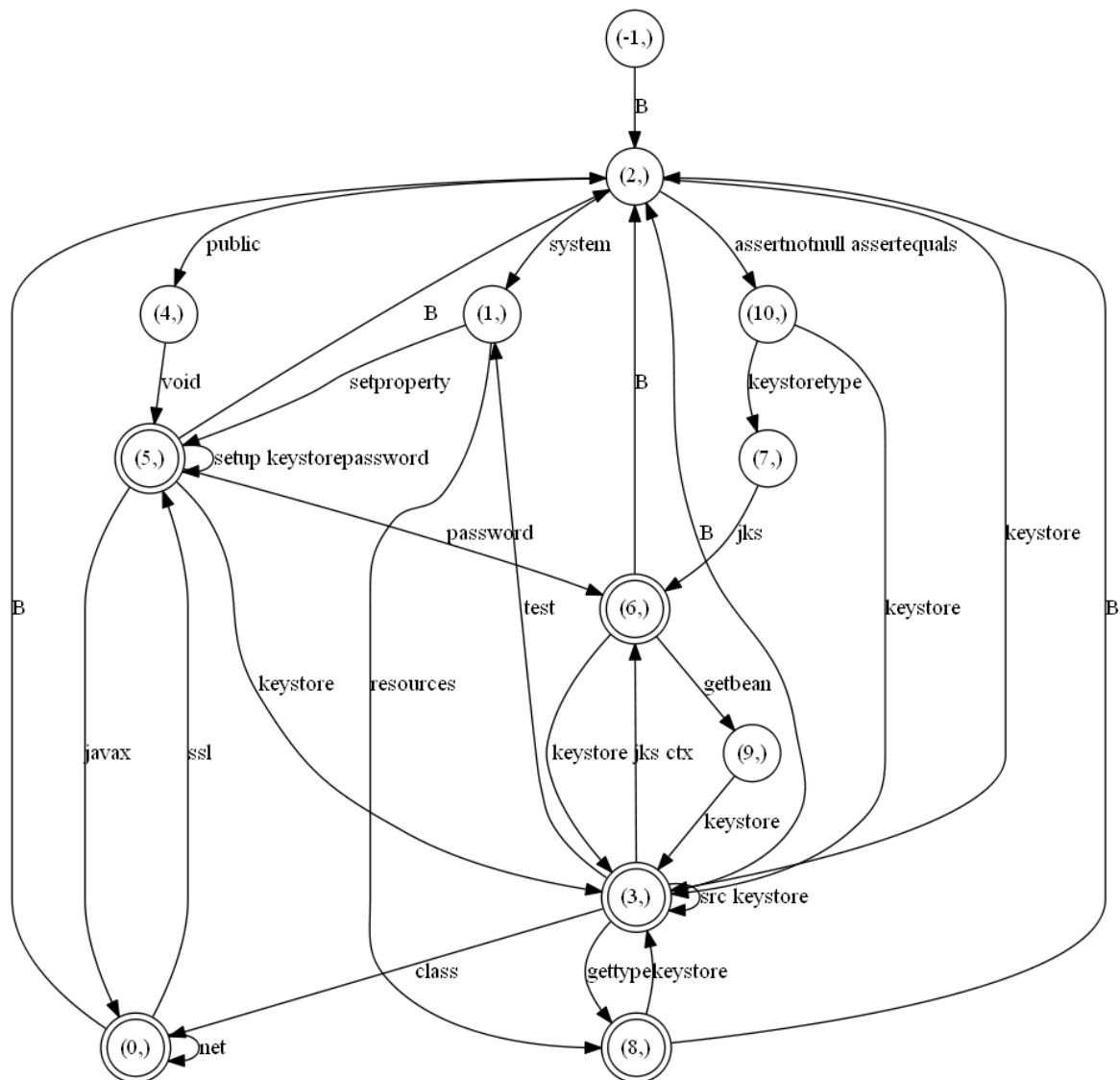
Lignes de code Java	Séquences associées
<pre>public Role getRole() {     return role ; }</pre>	["B", "public", "role", "getrole", "E"]
	["B", "return", "role", "E"]
<pre>public void setRole(Role role) {     this.role = role; }</pre>	["B", "public", "void", "setrole", "role", "role", "E"]
	["B", "this", "role", "role", "E"]
<pre>public String toString() {     return "User [id=" + id + ", firstName="         + firstName + ", lastName="         + lastName + ", username=" + username         + ", password=" + password         + ", role=" + role + "]"; }</pre>	["B", "public", "string", "toString", "E"]
	["B", "return", "user", "id", "id", "firstname", "firstname", "lastname", "E"]
	["B", "lastname", "username", "username", "password", "E"]
	["B", "password", "role", "role", "E"]
<pre>public class Role {     private Long id;     private User user;     private Integer role;      public Role() {}      public Role(Integer role) {         this.role = role;     } }</pre>	["B", "public", "class", "role", "E"]
	["B", "private", "long", "id", "E"]
	["B", "private", "user", "user", "E"]s
	["B", "private", "integer", "role", "E"]
	["B", "public", "role", "E"]
	["B", "public", "role", "integer", "role", "E"]
	["B", "this", "role", "role", "E"]
<pre>public Long getId() {     return id; }</pre>	["B", "public", "long", "getId", "E"]
	["B", "return", "id", "E"]
<pre>public void setId(Long id) {     this.id = id; }</pre>	["B", "public", "void", "setId", "long", "id", "E"]
	["B", "this", "id", "id", "E"]
<pre>public User getUser() {     return user; }</pre>	["B", "public", "user", "getUser", "E"]
	["B", "return", "user", "E"]
<pre>public void setUser(User user) {     this.user = user; }</pre>	["B", "public", "void", "setuser", "user", "user", "E"]
	["B", "this", "user", "user", "E"]

**Annexe 8.** Extraction des patterns d'activité sous la forme d'un automate minimisé sur du code Java supplémentaire reflétant une fonction complexe.

```
public void setup() {
    System.setProperty("javax.net.ssl.keyStore",
        "src/test/resources/keystore.jks");
    System.setProperty("javax.net.ssl.keyStorePassword",
        "password");

    KeyStore keyStore = ctx.getBean(KeyStore.class);
    assertNotNull(keyStore);
    assertEquals("keyStoreType", "JKS", keyStore.getType());
    assertNotNull(keyStore1);
    assertNotNull(keyStore2);
}
```

Annexe 8.a – Fonction Java d’origine servant à l’extraction d’automates



Annexe 8.b – Automate minimisé extrait à partir de la fonction Java présentée en Annexe 8.a