Apprentissage par renforcement pour l'improvisation musicale automatique

Rémi Decelle

Mémoire d'ingénieur

Apprentissage par renforcement pour l'improvisation musicale automatique

Rémi DECELLE

Année 2016–2017

Stage de fin d'études réalisé dans le laboratoire INRIA

en vue de l'obtention du diplôme d'ingénieur de TELECOM Nancy

Maître de stage : Ken Déguernel et Emmanuel Vincent

Encadrant universitaire : Bruno Pinçon

Déclaration sur l'honneur de non-plagiat

Je soussigné(e),

Nom, prénom : Decelle, Rémi

Élève-ingénieur(e) régulièrement inscrit(e) en 3e année à TELECOM Nancy

Numéro de carte de l'étudiant(e) : 31419182

Année universitaire : 2016–2017

Auteur(e) du document, mémoire, rapport ou code informatique intitulé :

Apprentissage par renforcement pour l'improvisation musicale automatique

Par la présente, je déclare m'être informé(e) sur les différentes formes de plagiat existantes et sur les techniques et normes de citation et référence.

Je déclare en outre que le travail rendu est un travail original, issu de ma réflexion personnelle, et qu'il a été rédigé entièrement par mes soins. J'affirme n'avoir ni contrefait, ni falsifié, ni copié tout ou partie de l'œuvre d'autrui, en particulier texte ou code informatique, dans le but de me l'accaparer.

Je certifie donc que toutes formulations, idées, recherches, raisonnements, analyses, programmes, schémas ou autre créations, figurant dans le document et empruntés à un tiers, sont clairement signalés comme tels, selon les usages en vigueur.

Je suis conscient(e) que le fait de ne pas citer une source ou de ne pas la citer clairement et complètement est constitutif de plagiat, que le plagiat est considéré comme une faute grave au sein de l'Université, et qu'en cas de manquement aux règles en la matière, j'encourrais des poursuites non seulement devant la commission de discipline de l'établissement mais également devant les tribunaux de la République Française.

Fait à Nancy, le 21 septembre 2017

Signature :

Mémoire d'ingénieur

Apprentissage par renforcement pour l'improvisation musicale automatique

Rémi DECELLE

Année 2016–2017

Stage de fin d'études réalisé dans l'entreprise INRIA

en vue de l'obtention du diplôme d'ingénieur de TELECOM Nancy

Rémi DECELLE
38bis, rue Sellier
54000, NANCY
+33 (0)6 15 31 66 41
remi.decelle@telecomnancy.eu

TELECOM Nancy
193 avenue Paul Muller,
CS 90172, VILLERS-LÈS-NANCY
+33 (0)3 83 68 26 00
contact@telecomnancy.eu

INRIA
615, Rue du Jardin botanique
54600, Villers-lès-Nancy
+33 3 83 59 30 54

Maître de stage : Ken Déguernel et Emmanuel Vincent

Encadrant universitaire : Bruno Pinçon

Remerciements

Je tiens tout d'abord à remercier Emmanuel Vincent de m'avoir accepté au sein de l'équipe et de me proposer ce sujet passionnant, mélangeant la musique et l'informatique. Je le remercie aussi de son encadrement tout au long du stage.
Ken Déguernel et Nathan Libermann font partis également de me remerciements, tant pour leur accompagnement que leurs savoirs transmis. Sébastien Da Silva, enseignant à Telecom Nancy, et Olivier Festor, directeur de Telecom Nancy, sont également remerciés pour m'avoir accordé le droit d'effectuer ce stage de fin d'études.

Je remercie ensuite les enseignants de Telecom Nancy qui ont renforcé ma curiosité pour l'informatique, mais aussi l'administration de Telecom Nancy qui a suivi mon cursus durant ces 3 années d'école d'ingénieur.

Merci beaucoup aux personnes que j'ai croisées durant mon stage, voisins de bureau, stagiaires, doctorants,... avec qui j'ai pu partager des expériences, des connaissances et avec lesquelles j'ai pu discuter.

Merci au Ministère de l'Enseignement Supérieur et de la Recherche, et au Ministère de l'Education Nationale qui ont, eux deux, financé la plus grande partie de mes études

Table des matières

Introduction

L'ordinateur comme aide à la décision musicale

Dès les années 60, le compositeur Iannis Xenakis se met à utiliser l'ordinateur pour créer des morceaux. Sa pièce intitulée *Achorripsis* et sa série dite des *ST* en sont le résultat. Pour cela, il utilise des données calculées par l'ordinateur IBM 7090.

Depuis les années 60, les ordinateurs ont énormément évolué, et sont devenus très performants. De telles performances permettent à l'intelligence artificielle de s'épanouir. Elle se développe pour l'aide à la prise de décisions et au diagnostic dans de nombreux domaines. Le traitement automatique des langues naturelles, pour la reconnaissance de la parole ou la synthèse de la parole, en est un. Emmanuel Kant disait que *la musique est la langue des émotions.*

Nous pouvons nous poser la question d'une intelligence artificielle capable de créations artistiques, et notamment capable de créer un morceau de musique par elle-même.

Regardons les capacités actuelles des ordinateurs pour la composition musicale. Par exemple, nous savons faire des compositions à la manière de Bach[1] ou bien encore dans le style des Beatles [1]. L'intervention de l'Homme est encore très présente dans le cas du style des Beatles, la machine produit uniquement la partition, l'interprétation étant aux mains des musiciens. Et remarquons que les mélodies restent simples dans le cas de BachBot.

Ici, nous nous intéressons à une autre forme de création musicale particulière : celle de l'improvisation. Elle diffère de la simple composition, dans le sens où une improvisation ne se limite pas à créer une musique, il s'agit également d'écouter le jeu des musiciens et le morceau dans lequel elle s'inscrit. Elle se doit d'être un échange collaboratif avec les musiciens. Former une synergie avec les musiciens. Elle est, pour ainsi dire, une composition en temps réel. L'improvisation a un caractère spontané, intuitif et interactif.

Mon stage s'insère dans le projet de l'Agence National de Recherche intitulé Dynamiques Créatives de l'Interaction Improvisée [2], qui vise à concevoir et mettre en œuvre un système d'improvisation musicale automatique capable d'interagir avec d'autres musiciens. Le système de l'état de l'art est celui d'OMax[2]. Ce système consiste à découper judicieusement le morceau, qui a été joué, en petit bouts, pour ensuite créer l'improvisation à partir de ces petits bouts selon des critères de sélection[3]. Grâce à l'apport fait par Ken Déguernel, qui consiste à améliorer les critères de sélection à l'aide de modèles probabilistes[4], le système développé par l'équipe dépasse celui de l'état de l'art. Mais ce système génère encore des fausses notes.

Le but essentiel de mon travail consiste à mettre en place un mécanisme qui permette au système de détecter ses erreurs et d'apprendre de ses erreurs afin de ne plus les reproduire. Mon travail se découpe donc en deux parties :

1. http ://www.flow-machines.com/
2. http ://repmus.ircam.fr/dyci2/home

- La première partie mettre en place un système de détection des fausses notes ;
- et la deuxième partie vise à modifier la génération de mélodies immédiatement pour que le système ne refasse pas les mêmes erreurs.

La détection de fausses notes est un challenge. Puisque, certes nous entendons assez facilement les fausses notes, mais il n'existe pas de formule magique pour traiter tous les cas. Il s'agit de trouver une formule qui s'en rapproche. Nous pourrions introduire des notions théoriques de la musique. Toutefois, cela réduirait la richesse et le caractère unique propre à l'improvisation.

Organisation du mémoire

À la suite de cette introduction, nous commencerons par définir des notions de musique (hauteur, rythme, ...). Nous en profiterons également pour introduire les notions fondamentales de l'apprentissage automatique, et notamment celle-ce de l'apprentissage profond.

Dans un second temps, nous décrirons en détail le système mis en place actuellement. Nous nous focaliserons aussi sur les tâches mises en œuvre durant le stage : la création d'une intelligence artificielle pour détecter les fausses notes, la modification du système actuel pour qu'il apprenne à ne pas refaire les mêmes erreurs. Dans cette partie nous détaillerons les choix de conceptions, leurs avantages, leurs limites.

Enfin nous analyserons les résultats, nous débattrons des modèles choisis et nous proposerons de nouvelle piste de réflexion. Dans cette partie, j'indiquerais également ma méthode de travail employée durant mon stage.

L'annexe A décrit les données d'apprentissage et de tests utilisés.

1 Le contexte : Projet ANR DYCI2

Mon stage s'inscrit dans le cadre du projet ANR intitulé DYCI2. Ce chapitre va présenter ce projet : ses objectifs, ses motivations, sa composition. Nous verrons aussi le domaine de recherche du projet dans lequel mon stage se place, à savoir celui de l'apprentissage interactif.

1.1 Présentation du projet

Le projet DYCI2 vise à explorer les interactions entre l'homme et des agents artificiels dans le domaine de l'improvisation musicale. Son but est de concevoir des modèles d'écoute, d'apprentissage, de modélisation et d'interaction capables de s'adapter et de comprendre la complexité de l'improvisation musicale. Ces agents musicaux doivent être créatifs, autonomes et capables de prendre part à une improvisation de façon interactive et musicalement viable (dans le sens qu'ils ne jouent pas au hasard).

1.1.1 Motivations

Le projet part d'une observation : l'immense majorité des interactions humaines sont improvisées. Et pourtant, toutes ces interactions sont structurées, cohérentes et ont du sens.
Par exemple, la majorité des improvisations, que les groupes de musique font, ne sont pas écrites note à note à l'avance. Dans le cas contraire, nous perdons la notion même d'improvisation. Les musiciens ne prévoient pas l'ensemble de l'improvisation ; celle-ci a un début, un développement et une fin.

En somme, ces interactions humaines mettent en jeu de nombreux comportements. La perception est la première, nous captons et recevons les actions d'autrui. Puis l'expression, nous agissons en conséquence de ce que nous avons perçu. L'environnement réagit ainsi aux actions que nous faisons, et ce schéma boucle sur lui-même. Nous recevons de nouvelles informations, nous agissons, l'environnement réagit, etc. Tout cela en une fraction de seconde.
En reprenant l'exemple de l'improvisation, les musiciens s'écoutent mutuellement. Puis, si un musicien décide de faire une variation à un moment donné, les autres membres du groupe réagissent à celle-ci et adaptent leur jeu musical en conséquence. Ce procédé se fait jusqu'à la fin de l'improvisation.

1.1.2 Composition du projet

En partant de ce constat, le projet veut modéliser le style et l'interaction à l'aide d'agents numériques. Le projet se décompose en trois domaines de recherche :

- l'écoute informée ;
- l'apprentissage interactif des structures musicales ;
- les dynamiques d'interaction improvisée.

L'écoute informée veut mettre en place un système en temps réel d'écoute et d'analyse de la scène sonore dans laquelle l'agent artificiel devra interagir. C'est l'équivalent de notre comportement de perception.

L'apprentissage interactif des structures musicales vient après l'écoute informée. Il vise, à partir des données reçues de l'écoute informée, à mettre en avant des modèles capables de capturer les structures musicales (style de musique, rythmique associé, ...). Ces structures serviront de points d'appui pour générer de nouvelles séquences musicales. L'objectif est de mieux rendre compte de la complexité du langage musical. Le comportement équivalent serait celui de l'expression.

Enfin, les dynamiques d'interaction improvisée ont pour objectif d'articuler l'agent numérique. C'est-à-dire qu'il doit s'adapter à son environnement (le jeu des musiciens, le morceau sur lequel il doit improviser, etc.). Ce domaine s'intéresse à la question du modèle de mémoire à attribuer à un tel agent pour lui permettre de remplir sa fonction d'adaptation et celle d'anticipation. La notion de mémoire n'est pas celle de la mémoire vive d'un ordinateur. Dans ce contexte, la mémoire correspond à une structure de séquences musicales annotées dans laquelle nous piocherons les séquences en les transformant et en les réagençant de façon à créer une improvisation [5]. Ici, nous avons affaire au comportement qui permet d'interagir, car écouter et agir ne suffissent pas. Une notion de mémoire est importante pour éviter les incohérences.

1.2 L'apprentissage interactif

Mon stage est lié au domaine de l'apprentissage interactif, dont la charge revient à l'équipe MULTISPEECH de l'Institut National de Recherche en Informatique et en Automatique (INRIA). Cette équipe fait ses recherches dans le traitement automatique de la parole et du langage.

L'approche adoptée par l'équipe pour la modélisation de séquences musicales mélange une approche formelle et une approche probabiliste [4]. La première approche est basée sur la logique mathématique et la seconde sur les probabilités. La volonté de fusion de ces deux approches s'explique. L'approche formelle a su montrer son efficacité, notamment pour sa rapidité dans les calculs. L'approche probabiliste a su montrer son efficacité dans le domaine du langage parlé [6]. L'objectif est d'explorer la possibilité d'obtenir un modèle de séquences musicales plus longues, qu'il est difficile d'obtenir dans un cadre purement probabiliste, tout en bénéficiant de la robustesse [1] au sur-apprentissage qu'offrent les approches probabilistes. Le modèle doit aussi s'adapter au style des musiciens.
En vue de découvrir et de réaliser un tel modèle, le projet DYCI2 ne part pas de rien. Il fonde sa recherche sur un paradigme particulier : le paradigme d'OMax.

1. La robustesse est la capacité du système à ne pas varier de comportement malgré une petite modification dans les données ou les paramètres choisis.

2 État de l'art : *OMax*

OMax est un logiciel capable d'improviser grâce à un apprentissage et une génération en direct à partir d'un corpus et/ou du jeu d'un musicien. Il n'a, a priori, aucune connaissance, ni aucune règle pour analyser la musique donnée en entrée. *OMax* a été développé par Assayag et al. [2] de l'Institut de Recherche et Coordination Acoustique/Musique (IRCAM).
Dans ce chapitre, nous allons explorer l'univers du paradigme *OMax* : ses principes de base, son architecture d'un point de vue global et le modèle qui se cache derrière la création des improvisations. Ce cheminement nous amènera à détailler la problématique.

2.1 Les principes de base

OMax est un système qui se base sur le fait qu'il ne connaît a priori rien à la musique. *OMax* est capable d'effectuer quatre actions en parallèle. Ces actions sont très semblables à celles évoquées dans la composition du projet DYCI2, à savoir : écoute, apprentissage et interaction. En voici la liste :

- l'écoute d'un flux audio en continu ;
- l'extraction d'informations à partir de l'écoute faite ;
- la modélisation des informations extraites ;
- la génération d'une nouvelle structure musicale à l'aide du modèle construit.

2.1.1 Réinjection stylistique

Pour mettre en évidence un modèle qui analyse une séquence musicale et capte certains traits relatifs au style [7],*OMax* va puisser les ressources nécessaires dans des musiques existantes et dans le jeu de l'improvisateur. Si les ressources sont en quantité suffisante, il va les recombiner pour imaginer une improvisation. De cette sorte, il y a une *réinjection* d'un passé, certes altéré, mais innovateur. C'est une forme de *feedback*. Effectivement, tout ce qui a été joué est susceptible d'être réinjecté dans le processus de création.

En somme, l'improvisateur pendant son improvisation écoute les autres musiciens. Il doit s'écouter lui-même. *OMax* va devenir une forme de mémoire externe en créant un avatar de l'improvisateur. Cet avatar devient un partenaire de jeu à part entière. Ainsi, l'improvisateur va devoir aussi prendre cet avatar en compte dans son jeu. Le musicien improvise avec une copie d'un passé de lui-même. Toute la réinjection stylistique est représentée dans la Figure 2.1, issue de [8].

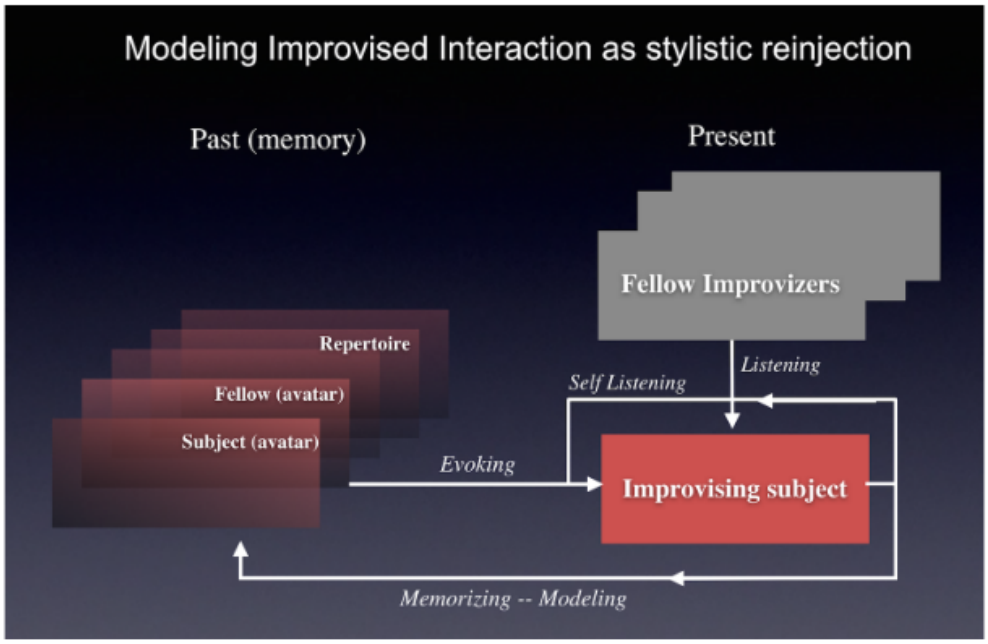

Figure 2.1 – *Schéma montrant le processus de réinjection stylistique.*

2.1.2 Vue d'ensemble de l'architecture

Pour comprendre le fonctionnement d'OMax, nous allons le décomposer en deux parties. Une première partie est chargée de l'écoute et de l'apprentissage. La seconde partie se consacre à l'improvisation même.
L'analyse, c'est-à-dire l'écoute et l'apprentissage, faite par OMax peut se décomposer en trois couches. La première couche va extraire des informations issues du son capturé à l'aide du traitement du signal. C'est la partie dite de *détection*. Vient ensuite une partie de *segmentation*. Elle consiste à agréger les informations issues de la détection afin de créer des unités symboliques et cohérentes.
Le processus de détection est fait à l'aide de l'algorithme YIN [9]. Il permet d'estimer en temps-réel la fréquence fondamentale de chaque son capturé. Chaque son possède un ensemble continu de fréquences. Une séquence de fréquences discrète est envoyée vers la partie de segmentation qui va faire le lien avec la note correspondante. Par exemple, une fréquence fondamentale de 440 Hz correspond au *La* symbolique des diapasons.
Enfin la dernière couche est la couche de modélisation. Elle va créer un modèle à partir de cette représentation symbolique. Cette modélisation est faite par un *oracle des facteurs*, décrit en détail dans la partie 2.2.2.

La deuxième partie, la partie d'improvisation, est effectuée en parallèle. L'improvisation va naviguer dans le modèle afin de créer une improvisation. Cette improvisation est encore à l'état de symboles qu'il faut alors transformer en un son : c'est la partie *rendu*. Cette architecture est représentée dans la Figure 2.2.

2.2 Le modèle : Oracle des Facteurs

Nous avons précédemment dit que la modélisation est faite à l'aide d'un oracle des facteurs. Dans cette partie, nous allons détailler ce dont il s'agit. Avant de voir l'oracle des facteurs, nous al-

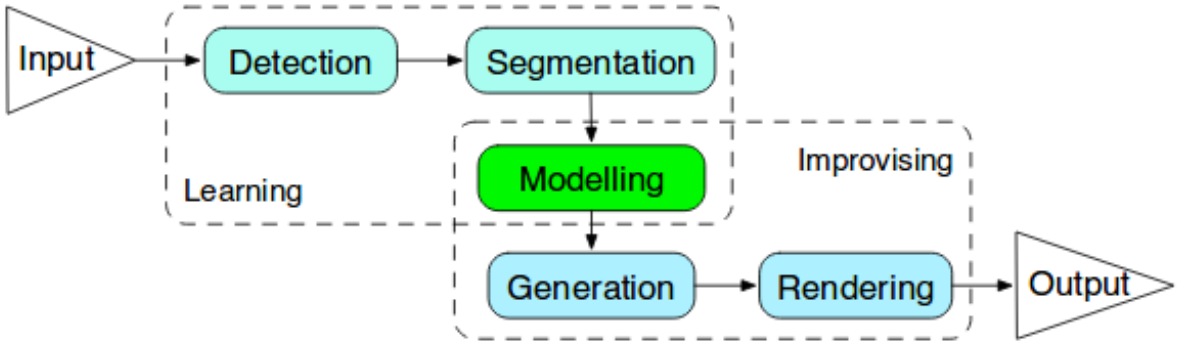

Figure 2.2 – *Architecture d'OMax. Image issue de la documentation d'OMax [8].*

lons devoir brièvement rappeler quelques notions de la théorie des langages. Une fois ces notions rappelées, nous préciserons comment utiliser l'oracle des facteurs pour créer une improvisation.

2.2.1 Alphabet, mots, facteurs, automates : des notions simples de la théorie des langages.

Pour créer un langage il nous faut des symboles : les *lettres*. L'ensemble de ces lettres forme un *alphabet*, que l'on note Σ. Nous prenons comme exemple la musique, nous pouvons créer un alphabet composé des notes de musique. Notre alphabet est :

$$\Sigma = \{Do, Re, Mi, Fa, Sol, La, Si\}.$$

Un alphabet ne suffit pas, il nous faut pouvoir construire des mots. Un *mot* de longueur k est une suite de k lettres issues de l'alphabet. Un exemple de mot basé sur notre alphabet musical est :

$$\omega = DoDoReMi.$$

L'ensemble des mots issus de l'alphabet Σ est noté Σ^*. La définition d'un facteur est la suivante :

$$x \text{ est facteur } y \text{ ssi } y \text{ est de la forme } uxv, \text{ avec } u, v \in \Sigma^*$$

Cela se traduit par : un facteur est une partie d'un mot. Voici tous les facteurs du mot ω que nous avons défini :

$$Do, Re, Mi, DoDo, DoRe, ReMi, DoDoRe, DoReMi, DoDoReMi$$

La notion de suffixe est la même que dans le langage courant. Formellement,

$$x \text{ est suffixe } y \text{ ssi } y \text{ est de la forme } ux, \text{ avec } u \in \Sigma^*$$

Les suffixes du mot ω sont :

$$Do, DoDo, DoDoRe, DoDoReMi$$

L'oracle des facteurs est un automate qui permet de reconnaître au moins l'ensemble des facteurs d'un mot. Il se peut qu'il reconnaisse d'autres mots. Mais qu'est-ce qu'un automate ? C'est une machine qui reproduit une séquence d'actions. Un automate $A = (\Sigma, S, I, F, \delta)$ est composé :

- d'un alphabet Σ ;
- d'un ensemble d'états S ;

- d'états initiaux I ;
- d'états finaux F ;
- des transitions δ, qui représentent la possibilité de lire une lettre à partir d'un état donné.

Un exemple d'automate construit sur notre alphabet musical Σ et qui reconnaît toutes les mélodies (mots) commençant par Do et finissant par Do (démonstration faite en Appendice A). Cet automate est présenté dans la Figure 2.3.

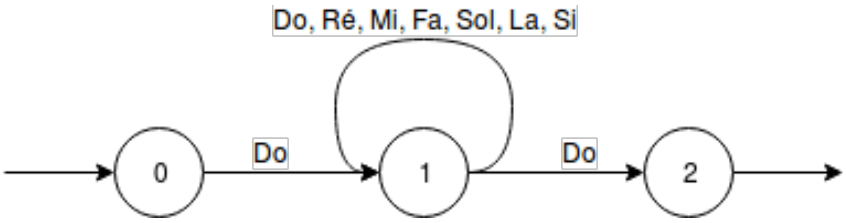

Figure 2.3 – *Automate construit sur l'alphabet des notes de musique. Cet automate reconnaît les mélodies (mots) commençant par Do et finissant par Do.*

Pour lire un automate, nous démarrons d'un état initial (ici 0), puis nous regardons les transitions possibles. Nous lisons la lettre que possède la transition que nous voulons faire, et nous arrivons dans l'état pointé par la transition. Et ainsi de suite jusqu'à arriver à un état final (ici 2). Le mot final qui est lu est la suite des lettres lues à chaque transition faite. Par exemple, pour le mot *DoReMiDo* nous passons par les états 0-1-1-2.

2.2.2 Oracle des facteurs

L'oracle des facteurs est un automate qui permet de reconnaître au moins l'ensemble des facteurs d'un mot. Il est issu de la bio-informatique et du traitement algorithmique de textes, il doit être capable de chercher des motifs dans le génome. Cette structure a beaucoup été étudiée [10] et a montré son efficacité pour l'analyse et la construction de motifs musicaux. L'oracle est une structure permettant la mise en œuvre d'algorithmes efficaces issus de la théorie des automates [5].
L'oracle des facteurs est par construction capable de dire, si oui ou non, un mot est un facteur du mot à partir duquel il est construit. Mais il est aussi capable de découvrir les facteurs qui sont répétés dans ce même mot. Ces répétitions sont appelées motifs.
Parmi d'autres représentation possibles (arbres suffixes, automate des suffixes, ...), l'oracle offre un excellent compromis [3] :

- il est déterministe, un état ne peut pas avoir deux transitions avec la même lettre ;
- sa création est linéaire, c'est-à-dire qu'il faut au maximum un nombre d'états égal à la longueur du mot ;
- il est homogène, chaque transition pointant vers un état donné possède le même symbole (économie d'espace) ;
- il permet la connection entre les motifs, à l'aide de *liens suffixes* ;
- il est possible de le construite en ligne (rapide lors d'ajout d'un état) et donc est très performant pour le temps-réel ;

Les liens suffixes introduisent la notion de plus long suffixe commun, ces liens permettant de lier ces suffixes entre eux.
Nous n'allons voir ni construction de l'oracle des facteurs (voir [10]), ni toutes les propriétés pour

son application musicale (voir [3] et [11]). Mais nous allons simplement regarder la façon dont nous pouvons parcourir l'oracle afin de créer une mélodie.

Tout d'abord introduisons un oracle des facteurs. Cet oracle est représenté par la Figure 2.4. Il est construit sur le mot *MiReReMiReRe*, issu toujours de notre alphabet musical. Les flèches en traits pleins dénotent les transitions dites *liens facteurs* obtenues par construction. Les flèches pointillés dénotent les transitions dites *liens suffixes*. Par exemple, à partir de l'état 6 nous regardons le plus grand motif possible aboutissant à cet état : *MiReRe*. Ce dernier est répété dans le mot et sa répétition se termine à l'état 3. Nous construisons un lien suffixe entre l'état 6 et l'état 3.

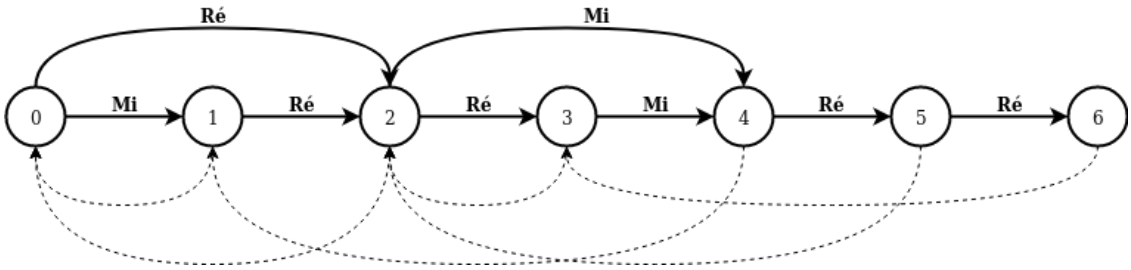

Figure 2.4 – *Exemple d'oracle des facteurs construit sur le mot MiReReMiReRe. Les flèches représentent les transitions; en traits pleins les liens facteurs et en pointillés les liens suffixes.*

Pour créer une nouvelle mélodie il suffit de parcourir l'automate en suivant les transitions. Plusieurs stratégies de parcours sont possibles comme le suggère [3].
Nous pouvons ajouter un facteur de continuité. Ce dernier limite le nombre de sauts effectués.
Une stratégie complémentaire, pour éviter de boucler sur la même mélodie (par exemple, lire Ré à partir de l'état 0 puis remonter le lien suffixe, de nouveau lire Ré, etc.) est d'introduire une liste tabou des états visités et d'empêcher de revenir dans ces états avant un certain temps.
Nous pouvons combiner plusieurs stratégies entre elles, afin d'améliorer la qualité de la mélodie créée.

2.2.3 Oracle des facteurs avec un modèle probabiliste

La problématique ne repose pas sur le modèle présenté ci-dessus, mais sur une amélioration due à Ken Déguernel [12]. L'idée est d'introduire des modèles probabilistes dans l'oracle des facteurs [4].

Le but est de créer un modèle probabiliste capable de prédire une mélodie à un instant t en connaissant un ensemble d'informations issues de diverses dimensions musicales (par exemple le rythme ou le style) de tous les instants précédents.
Le problème d'un tel modèle est l'espace à explorer : il est bien trop grand (trop d'informations musicales à retenir à tous les instants). Pour réduire cet espace, nous allons interpoler le modèle avec des sous-modèles plus petits. Ces sous-modèles sont réduits à une seule variable musicale. Un sous-modèle pourrait ne prendre en compte que les informations rythmiques, par exemple.

En pratique, deux sous-modèles ont été mis en place : le premier prédit la mélodie à l'instant t en connaissant la mélodie à l'instant $t-1$, le deuxième prédit la mélodie à l'instant t en connaissant l'harmonie [1] à ce même instant. Plus l'ensemble de sous-modèles est grand, meilleur sera l'interpolation et le rendu final n'en sera que meilleur.

1. Une harmonie est un ensemble de notes jouées simultanément.

Ce modèle probabiliste doit s'intégrer à l'oracle. L'objectif de cette intégration est de concevoir un système d'improvisation semblable à celui d'un improvisateur. L'idée provient d'une citation de Marilyn Crispell (pianiste américiaine de jazz). Dans un article, intitulé Eléments d'improvisation [13] (écrit pour Cecil taylor et Anthony Braxton), elle déclare :

> *« Le développement d'un motif doit être fait de manière logique, organique, [...] basé sur l'intuition enrichie par des connaissances (de tout ce qui a été étudié, joué, écouté, des différents styles musicaux auxquels on a été exposé etc. au cours de notre vie - y compris toutes les expériences personelles); le résultat étant un vocabulaire musical personnel. »*

Dans un premier temps, nous créons le modèle probabiliste qui sera entraîné soit à l'aide d'un corpus, soit directement avec le jeu d'un musicien. Dans la citation ci-dessus, cela correspond aux connaissances acquises au cours de la vie.
Dans un second temps, nous construisons l'oracle des facteurs juste pour la mélodie. Chaque état représente une note. Dans les dires de Crispell, c'est la manière logique avec laquelle le motif doit se développer. L'oracle est crée en simultané avec ce que joue le musicien (il peut aussi être crée à l'aide d'un corpus).
Maintenant, la navigation dans l'oracle se fera à l'aide de la probabilité de franchir une transition donnée (voir Figure 2.5). Aucune transition supplémentaire n'est créée (ces transitions dépendent des heuristiques expliquées dans l'article [3]).

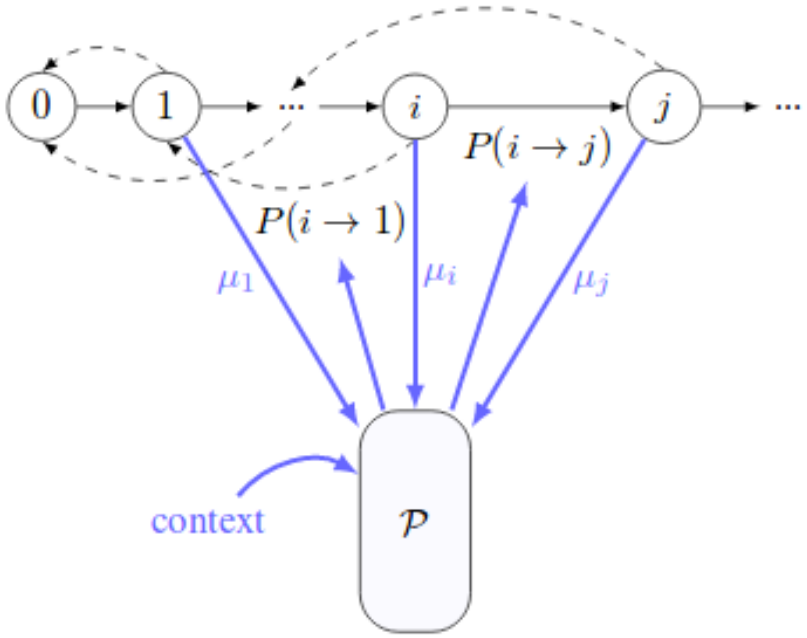

Figure 2.5 – *Utilisation d'un modèle probabiliste (multi-dimensionnel), noté* **P**, *avec l'oracle des facteurs. Soit i l'état courant. Soient les états j et 1 les états atteignables depuis i. En utilisant les contextes μ_i et μ_1,* **P** *calcule un score pour passer de l'état i à 1. De même pour le passage de i à j. Les scores sont normalisés. Nous obtenons la probabilité $P(i \to 1)$, resp. $P(i \to j)$, la probabilité d'aller de i à 1, resp. i à j.*

Pour mieux s'imprégner de l'ensemble de cette description, nous allons regarder le diagramme de classes associé à cette nouvelle structure. Ce diagramme va mettre en lumière les concepts jusqu'alors évoqués. Il sera également un point d'accroche pour la suite du mémoire. Le diagramme, présenté en Figure 2.6, montre que le projet est décomposé en trois modules :

- le module Proba.py retranscrit le modèle probabiliste ;
- le module Oracle.py implémente l'oracle des facteurs ;

- et le module Utils.py modélise les notes musicales et fait le lien entre musique et informatique.

Le module Proba.py est composé de la manière suivante. Il possède un modèle composé d'un ensemble de sous-modèles [4]. Dans la pratique, deux sous-modèles, cités plus haut, sont utilisés. À savoir, un premier sous-modèle pour la représentation de bigramme (couple de la note courante et de la note précédente) et un autre sous-modèle pour représenter l'harmonie courante et la note courante. L'oracle va se servir du modèle interpolé pour créer ses nouvelles transitions.

Justement, le module Oracle.py est la mise en œuvre de cet oracle des facteurs. Nous retrouvons l'oracle composé d'un ensemble d'états. Chaque état est composé d'une transition entrante et sortante, d'une liste de liens facteurs entrants et sortants, le lien suffixe sortant et la liste de liens suffixes entrants. Une transition, comme nous l'avons vu, va d'un état d'entrée vers un état de sortie, et possède un symbole de transition. Le symbole de transition est de type XmlNote, décrit ultérieurement.

C'est le module Utils.py qui va permettre la transition entre la musique et le modèle informatique. Il va donc représenter les composants musicaux nécessaires. Notamment, l'harmonie avec la classe XmlChord et les notes avec la classe XmlNote sont décrites dans ce module.

2.2.4 Problématique détaillée

Ce dernier modèle, malgré une meilleure qualité des mélodies créées, effectue encore des erreurs, que nous pouvons qualifier de fausses notes. Elles ne s'inscrivent pas dans le reste du contexte proposé. L'objectif est de doter le modèle d'une forme d'auto-écoute et d'auto-critique. Il va devoir évaluer dans un premier temps les séquences musicales qu'il produit. Pour, dans un second temps, éviter qu'il reproduise les mauvaises séquences.

Une des problématiques porte sur la capacité de créer une intelligence artificielle capable d'identifier une fausse note. De nombreuses questions se posent : comment pouvons définir la notion de fausse note ? Comment mettre en œuvre un agent numérique capable de déceler ces fausses notes ? Cette première problématique vient en écho à une autre problématique : est-il possible de faire apprendre à l'oracle à ne plus commettre ses erreurs ?

Les réponses aux premières questions ont constitué la première partie partie du travail ; représenter une note d'un point de vue informatique et d'identifier la notion fausse note. À la suite de cela, il a fallu mettre en place une intelligence qui détecte les fausses notes.
Le second travail se concentre sur la mise en place d'une méthode d'apprentissage, dite de renforcement, pour doter le modèle d'un outil lui permettant de ne plus refaire les mêmes erreurs. Cela vient apporter un fragment de réponse à la dernière question soulevée.

Le reste du mémoire s'articule autour de deux axes. Nous nous attacherons dans un premier temps à définir des notions importantes de musique et d'intelligence artificielle. Ces notions permettrons de mieux saisir les solutions proposées et mises en œuvre dans un second temps. Nous présenterons également les résultats obtenus, et discuterons des choix faits.

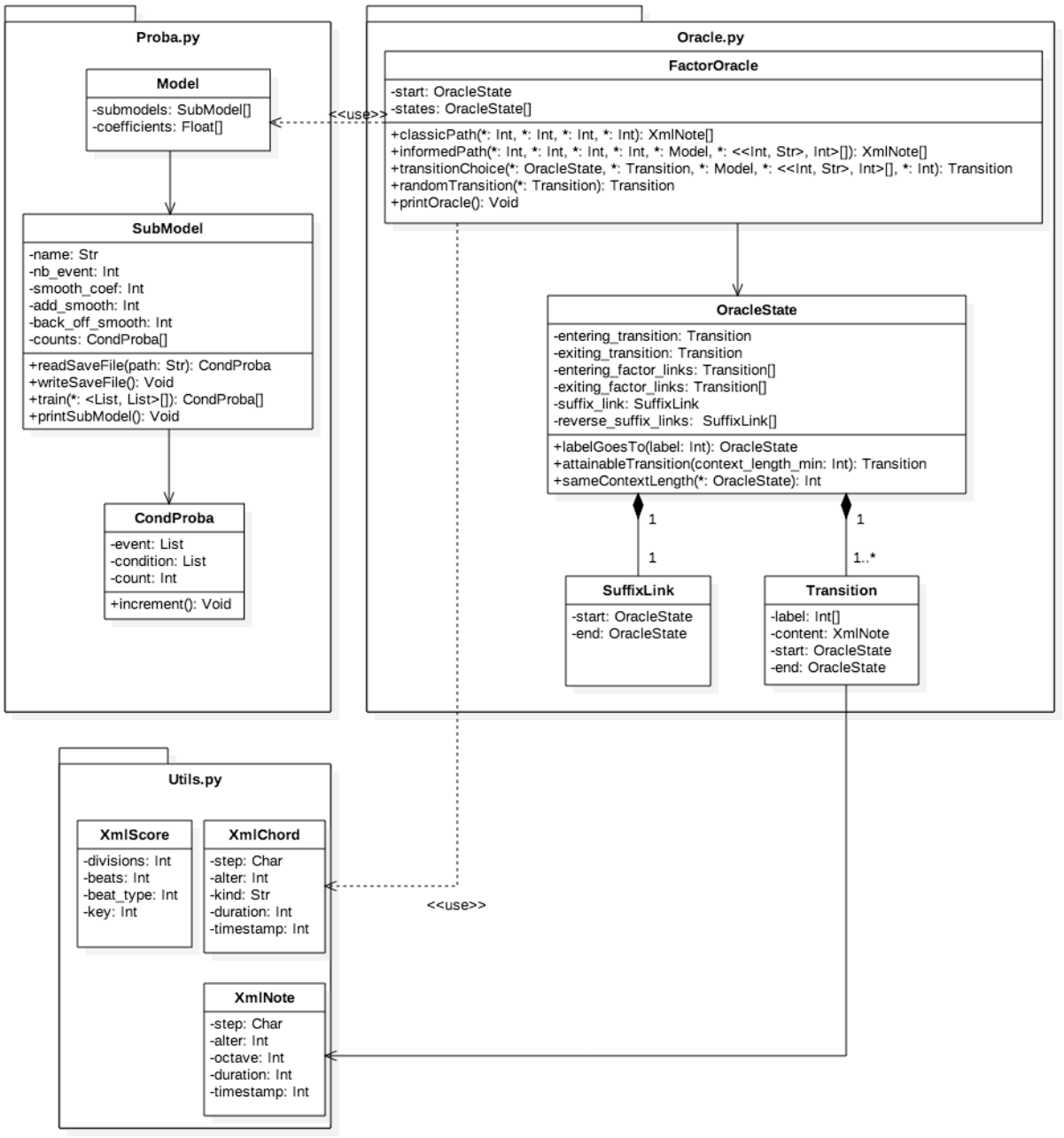

Figure 2.6 – *Diagramme de classes pour la version améliorée de l'oracle des facteurs.*

3 Introduction à la musique et à l'apprentissage automatique

Cette première partie se consacre à définir les notions musicales nécessaires à la compréhension du sujet, du problème et des choix qui ont été faits.
Nous nous intéresserons dans un premier temps aux notions en lien avec la musique. Ces notions sont directement liées au diagramme de classe (Figure 2.6) décrit en partie 2.2.3.
Cette partie va également définir la notion de fausse note dans le cadre que nous nous sommes fixés.

3.1 De la musique avant toute chose

Nous écoutons tous de la musique. Pourtant, tout le monde ne sait pas en jouer, ou ne connaît pas les notions de musique. Nous allons introduire quelques notions élémentaires de musique. Nous allons nous concentrer sur les notions suivantes :

- la notion de note, l'élément clef du problème ;
- la notion de rythme ;
- brièvement la notion d'harmonie ;
- et toutes ces notions vont nous permettre d'aborder notre notion de fausse note.

3.1.1 Prendre de la hauteur

Rappelons rapidement le nom des sept notes usuellement utilisées dans la musique classique occidentale : Do, Ré, Mi, Fa, Sol, La et Si. Dans le jargon musical, le nom de la note s'appelle la *hauteur* (de la note). Pour nous conformer aux écritures anglaises, très présentes en informatique, les notes seront dénotées par des lettres. La lettre A dénotera un La, la lettre B dénotera un Si, et ainsi de suite (voir Tableau 3.1).

Nom Français	La	Si	Do	Ré	Mi	Fa	Sol
Equivalent Anglais	A	B	C	D	E	F	G

Table 3.1 – *Equivalence entre les noms français des notes et les noms anglais.*

3.1.2 Altérer les notes

En musique, nous ne nous contentons pas des sept notes évoquées précédemment. Nous pouvons les altérer. Les deux grandes altérations de la musique classique occidentale sont le bémol, noté ♭, et le dièse, noté ♯. Le bémol sert à diminuer d'un demi-ton, alors que le dièse permet d'augmenter d'un demi-ton. Qu'est-ce qu'un ton ?
Pour illustrer cette notion, nous allons regarder sur un piano. Un piano classique est composé de touches noires et de touches blanches. Chaque touche est séparée d'un demi-ton de sa voisine (voir Figure 3.1). Par soucis de simplification, nous allons accepter les relations suivantes : F♭ = E et E♯ = F. De même, B♯ = C et C♭ = B. Comme les notes C (Do) et D (Ré) sont séparées de deux touches, elles sont séparées d'un ton. Cette représentation est une version simplifiée de la notion de ton, mais nous suffira par la suite.

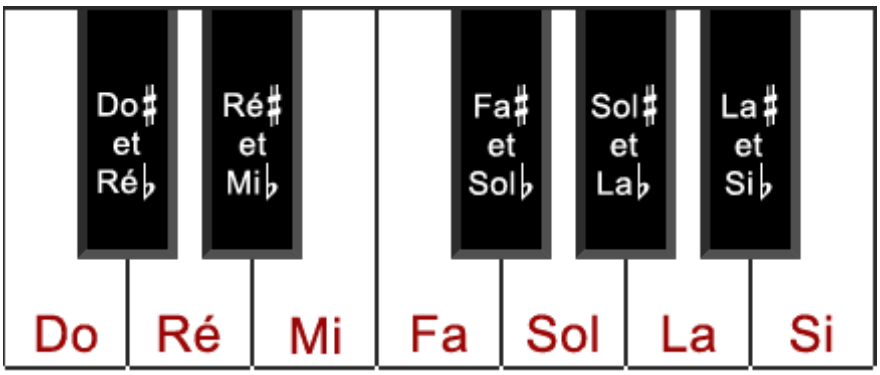

Figure 3.1 – *Illustration de la notion de ton sur un clavier piano (tempérament égal). Entre chaque touche il existe un demi-ton.*

En prenant en compte cette simplification, nous comptons 12 notes distinctes. Nous allons rajouter une treizième note à ces douze notes. Cette treizième note symbolisera le silence. Le silence fait partie intégrante de la musique. L'inclure en tant que note est justifié.

3.1.3 Durée des notes

Du temps binaire de la noire...

Connaître la hauteur des notes ne suffit pas pour faire de la musique. Il faut, pour donner vie à une note, définir une durée pendant laquelle elle sera jouée. C'est ce que nous appelons communément le rythme. Dans la musique dite rythmée un morceau possède une pulsation, c'est-à-dire un battement qui marque chaque temps. Et dans beaucoup de morceaux un temps est appelé une noire. Dans la suite, nous nous permettons de faire la confusion entre une noire et un temps. Nous nous placerons dans le cadre de morceaux rythmés avec une pulsation égale à la noire.
De cette pulsation de référence, nous pouvons introduire des rythmes. Un rythme sera un multiple de la noire. Dans la musique classique occidentale, les rythmes binaires sont très présents. Ces rythmes binaires sont des puissances de deux d'un temps. Le Tableau 3.2 montre quelques rythmes binaires de bases.

Nom	Symbole	Durée (en temps)
Ronde		4
Blanche		2
Noire		1
Croche		$\frac{1}{2}$
Double Croche		$\frac{1}{4}$
Triple Croche		$\frac{1}{8}$

Table 3.2 – *Tableau récapitulatif des rythmes binaires.*

Une blanche vaut deux noires et la ronde en vaut quatre, équivalent à deux blanches. La croche en vaut $\frac{1}{2}$, la double croche (dite double) vaut $\frac{1}{4}$ de temps. Il faut donc deux doubles pour faire une croche. La triple croche (dite triple) vaut $\frac{1}{8}$ de temps. Nous pouvons imaginer des temps non listés, par exemple la quadruple croche en vaudrait deux fois moins que la triple, c'est-à-dire $\frac{1}{16}$ de temps.
Nous pourrions diviser le temps non pas par deux mais par trois, par cinq, etc. . Une division par trois s'appelle un triolet et vaut $\frac{1}{3}$ de temps. Un quintolet sera la division par cinq, et ainsi de suite. Ces divisons sont des divisions dites irrégulières dans le cadre d'un système rythmique binaire.

...à un temps entier

Nous venons de voir comment attribuer un symbole indiquant le rythme d'une note. Ce rythme se base sur un temps de référence, la noire. Pouvons-nous imaginer un autre système rythmique ? Regardons une autre façon considérer le rythme.

Cette façon se base sur ce que l'on appelle le *tatum*, pour Time Quantum. Nous définissons le tatum comme suit : le tatum est la plus petite subdivision qui permette d'exprimer l'intégralité des durées présentes dans un morceau.
Nous attribuons alors à cette subdivision la valeur de 1. La valeur des autres notes en est déduite par la règle de trois.

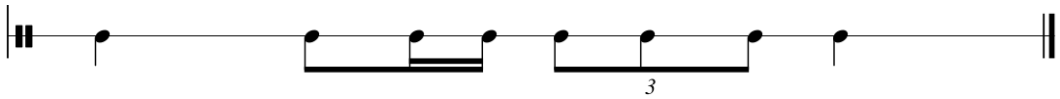

Figure 3.2 – *Exemple d'une ligne rythmique. Nous avons un cas de division irrégulière, indiquée par le chiffre 3. Cela indique qu'une croche, dans ce groupement, vaut $\frac{1}{3}$ de temps et non $\frac{1}{2}$.*

Pour la ligne rythmique proposée dans la figure Figure 3.2, nous avons des doubles croches de durée $\frac{1}{4}$ de noire, des triolets de durée $\frac{1}{3}$, une croche de durée de $\frac{1}{2}$ et des noires de durée 1. Le calcul du PGCD de 4, 3, 2 et 1 nous donne un tatum de 12. Autrement dit, le plus petit temps commun est un douzième de noire. Nous en déduisons la durée de la double exprimée en tatum : $12 \div 4 = 3$. Le triolet vaudra 4 ($12 \div 3 = 4$), la croche vaudra 6 ($12 \div 2 = 6$), la noire vaudra 12.

Rythme	Durée Tatum	Durée Temps
Noire	12	1
Croche	6	1/2
Triolet	4	1/3
Double croche	3	1/4

Table 3.3 – *Tableau des valeurs des rythmes pour la Figure 3.2.*

Cette vision rythmique est propre à chaque morceau. Elle varie, ce qui rend difficile pour comparer les morceaux en eux. Nous verrons dans le Chapitre 5 comment nous nous y sommes pris pour y remédier.

Elle permet cependant de traiter le rythme avec des valeurs entières et non des fractions. Cet manière va nous permettre d'identifier les durées à des indices d'un vecteur. Cela nous amène à modéliser avec cette vision du rythme.

Harmonie ou accord

En musique, il est courant de jouer plusieurs notes à la fois. La définition de l'harmonie se réduit à un ensemble de sons joués simultanément. Mais, dans la musique occidentale, l'harmonie est la théorie qui étudie les accords. Un accord désigne une superposition de notes (si les notes ne sont pas superposées, nous parlons d'arpège).

3.1.4 Notion de fausse note

Nous venons de voir trois éléments clefs en musique, la hauteur, le rythme et l'harmonie. Il nous faut nous poser la question suivante : pouvons-nous maintenant définir une fausse note ?

Quelques remarques importantes sont nécessaires. Le silence n'est pas faux. L'hypothèse n'est ni absurde, ni contraignante. Il faut mieux ne rien jouer que de jouer une note non appréciée, ou du moins ne collant pas avec le reste de la mélodie (succession temporelle de notes) ou de l'harmonie.
Justement le reste de la mélodie est au centre de la deuxième remarque : une note en elle-même ne peut être fausse. Il faut considérer la note dans son contexte. Par la suite, notre contexte se limitera à la mélodie.
Nous avions envisagé d'inclure l'harmonie dans le contexte, mais cela n'a pas été mis en œuvre dans la solution proposée. Étant donné que le problème est difficile, le simplifier dans un premier temps est raisonnable, quitte à le préciser dans un second temps.

Nous venons de voir la nécessité d'un contexte afin de définir la notion de fausse note. Pourtant, nous n'avons toujours pas défini cette notion. Notre réflexion se base sur un ensemble de données. Par ailleurs, rappelons le cadre dans lequel cette recherche s'inscrit : le modèle ne possède aucune connaissance en matière de musique. Il est donc proscrit d'utiliser la théorie de la musique afin de définir la notion de fausse note. Le modèle doit créer une théorie à partir des données observées. Une manière de définir la notion de fausse note est la suivante : une note est fausse si elle est éloignée de son contexte. Plus elle sera éloignée, plus elle sera fausse. Le contexte est constitué de l'harmonie, du rythme et de la mélodie. L'éloignement est l'éloignement par rapport à la distribution des données.
Pour préciser cette définition abstraite, nous l'avons interprété comme suit : une note est fausse dans un contexte donné si celle-ci est choisie aléatoirement. Ici, notre notion d'aléatoire va seulement modifier la hauteur (et l'octave). Nous excluons toute modification du rythme et de l'harmonie. Toute modification du rythme entraîne la nécessité de garder une cohérence musicale, et l'harmonie dépend des autres musiciens. Mais, est-ce que cette interprétation est valide ?

D'un point de vue des probabilités cela est correct. En moyenne, une note aléatoirement choisie sera éloignée de son contexte (éloignée de la distribution des données). En vous faisant écouter un thème musical, puis en écoutant derechef ce thème avec une note modifiée aléatoirement, il y a de fortes chances que la note en question ne sonne pas juste à votre oreille. De même, un morceau formé avec uniquement des notes aléatoires à peu de chance d'être apprécié par le commun des mortels.

Nous venons de voir les notions musicales nécessaires pour la suite. Nous avons d'abord vu la notion de hauteur d'une note, c'est son nom. Puis, nous avons vu l'équivalent anglais des noms des notes, c'est A pour La, B pour Si, ... Nous avons ensuite dénombré les notes, au nombre de 13. Nous avons également regardé deux façons de noter le rythme. La première est le rythme binaire avec comme unité de temps de référence la noire et la deuxième, plus adaptée à l'informatique, est le rythme exprimé en tatum. Nous avons en avons profité pour définir l'harmonie.
Grâce à ces trois notions, nous avons pu définir la notion de fausse note dans notre contexte. À savoir, une note éloignée de son contexte est considérée comme fausse. Pour s'approcher le plus de cette définition, Nous avons considéré que modifier la hauteur et l'octave d'une note suffit à faire que cette note soit éloigné de son contexte. Nous avons aussi limité le contexte à la mélodie, en excluant l'harmonie.

4 Introduction à l'apprentissage automatique

Nous avons défini quelques notions de musique et ce qui nous a permi de définir la notion de fausse note. Malheureusement, cela ne suffit pas à résoudre notre problématique. Quittons le monde de la musique, pour nous plonger dans le monde de l'intelligence artificielle. Ce domaine de recherche est très large. Nous allons nous intéresser qu'à deux types d'intelligence artificielle : celle des réseaux de neurones et celle de l'apprentissage par renforcement.

L'objectif principal de ces deux apprentissages est l'apprentissage par l'erreur. L'intelligence artificielle va apprendre des erreurs qu'elle va commettre. Le réseau de neurones est comme un enfant, nous allons lui montrer énormément de morceaux de musique avec les réponses. Il va apprendre à partir de ces données fournies.
L'apprentissage par renforcement est différent. L'intelligence artificielle va nous proposer de nouvelles mélodies, et nous la sanctionnerons ou la récompenserons en fonction de ce qu'elle propose. Elle va apprendre par un système de récompenses quels sont les meilleurs choix qu'elle doit faire et ceux à éviter.

Dans les deux cas, nous expliquerons le mécanisme et les étapes à travers un exemple simple. Nous définirons avant le vocabulaire spécifique à chaque apprentissage.

4.1 Les réseaux de neurones

4.1.1 Une définition

D'après le Larousse, un neurone est :

> « *une cellule de base du tissu nerveux, capable de recevoir, d'analyser et de produire des informations.* »

D'un point de vue informatique la définition est aussi courte, c'est une unité chargée de calculer. Cette unité prend en entrée des nombres, effectue un calcul, le résultat est donné en sortie.

4.1.2 Vocabulaire

Un neurone prend un certain nombre d'*entrées*. Ces entrées, que nous notons par un vecteur x, sont des nombres. Nous associons un *poids* à chaque entrée. Le poids est un nombre qui vient pondérer la valeur de l'entrée. Les poids seront notés par un vecteur w. Souvent, nous ajoutons un élément appelé *biais*, noté b.
Enfin, le dernier élément pour effectuer le calcul est la *fonction d'activation*, notée f. Nous pouvons mettre n'importe quelle fonction scalaire. Dans la pratique, nous utilisons des fonctions dont la dérivée est facile à calculer pour simplifier des calculs.

La *sortie y* du neurone est la valeur de la fonction d'activation appliquée à de la somme pondérée (avec les poids respectifs) des entrées, soit

$$y = f(xw^T + b)$$

Si la sortie n'est pas celle attendue, comment corriger l'erreur ? Comment lui faire apprendre ? Dans un neurone, la seule chose que nous sommes autorisés à corriger est la valeur des poids et la valeur du biais. La phase d'apprentissage consiste à corriger ces valeurs à l'aide de l'erreur que le neurone a commis. Pour calculer cette erreur, nous avons besoin d'une *fonction d'erreur*. Il existe de nombreuses façons de calculer l'erreur (erreur quadratique moyenne, la divergence de Kullback-Leibler, distance d'Hellinger, etc.). La fonction d'erreur sera notée E.
Avec cette fonction, nous mettons à jour la valeur des poids et du biais. La mise à jour est donnée par la formule suivante :

$$\Delta w_i = -\alpha \frac{\partial E}{\partial w_i}$$

Nous parlons de *rétropropagation* de l'erreur. C'est une version de base, de nombreuses amélioration ont été proposées. Dans cette formule, le *coefficient d'apprentissage*, noté α, apparaît. Il est lié à la vitesse à laquelle nous voulons que notre réseau apprenne de ses erreurs.
Le *gradient* de l'erreur $\frac{\partial E}{\partial w_i}$ est présent, c'est dans le calcul de ce dernier que la dérivée de la fonction d'activation est nécessaire. La simplicité de la fonction d'activation augmente la rapidité de calcul.

Il doit y avoir convergence de la fonction d'erreur au fur et à mesure des étapes d'apprentissage. Cette convergence assure que le modèle approche le résultat attendu.

4.1.3 Un exemple

L'exemple de neurone que nous allons prendre est illustré par la Figure 4.1. Un tel neurone est appelé *perceptron*. Notre objectif est de montrer une étape d'apprentissage d'un neurone et d'illustrer le vocabulaire spécifique aux réseaux de neurones. Notre neurone va devoir apprendre à calculer le nombre opposé de la somme de deux nombres donnés.

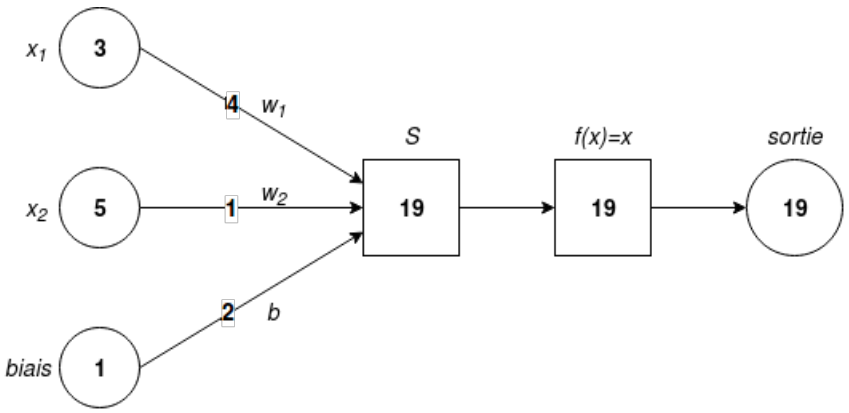

Figure 4.1 – *Schéma d'un neurone formel. Ce neurone possède 2 entrées, un biais, 3 poids, une fonction d'activation définie par $f(x) = x$, et une sortie.*

Détaillons le processus de calcul du neurone. Dans notre exemple nous avons 2 entrées, que l'on note x_1 et x_2, ayant pour valeur respective 3 et 5. La sortie attendue est l'opposé de la somme,

cela équivaut à $-(3+5) = -8$. Nous avons les poids, w_1, resp. w_2, associé à x_1, resp. x_2. Chaque poids a pour valeur respective 4 et 1. Le biais est noté b, avec comme valeur 2. Nous posons $f(x) = x$, la fonction d'activation. Calculons la sortie du neurone :

$$\begin{aligned} y &= f(xW^T + b) \\ &= xW^T + b \\ &= x_1 w_1 + x_2 w_2 + b \\ &= 3 \times 4 + 5 \times 1 + 2 \\ &= 19 \end{aligned}$$

La sortie vaut alors 19. Ce n'est pas le résultat souhaité. Nous allons rétropropager l'erreur. Notre fonction d'erreur sera l'erreur moyenne quadratique, donnée par la formule suivante :

$$E = \frac{1}{2}(y_{att} - y)^2$$

où y_{att} dénote le résultat attendu. Nous posons $\alpha = 0.05$. Le gradient vaut $-(19 - (-8)) = -27$ (pour les détails du calcul voir Appendice B), la formule de mise à jour est donnée par :

$$w_i \leftarrow w_i + 0.05 \times (-27) x_i, \text{ pour } i = 1, 2$$

En appliquant cette formule à nos poids et à notre biais, nous obtenons alors les nouveaux poids :

$$\begin{cases} w_1 &= 4 + 0.05 * (-27) * 3 &= -0.05 \\ w_2 &= 1 + 0.05 * (-27) * 5 &= -5.75 \\ b &= 2 + 0.05 * (-27) * 1 &= 0.65 \end{cases}$$

Remarquons que nos nouveaux poids sont plus petits que les précédents, nous avons perdu du poids. Le nouveau calcul, avec les mêmes entrées, donne une sortie de -28.25. L'erreur commise vaut $\partial E = -20.25 > -27$ (voir Figure 4.2). Ce gradient est plus proche de zéro que le précédent, c'est signe que nous approchons un optima . Notre neurone progresse. Si l'erreur vaut 0, alors la sortie calculée vaut la sortie souhaitée, dans notre cas très précis. C'est le résultat auquel nous voulons aboutir [1].
L'objectif principal de l'apprentissage est de minimiser la fonction d'erreur. L'erreur étant minimale, le résultat sera lui optimal (optimalité locale).

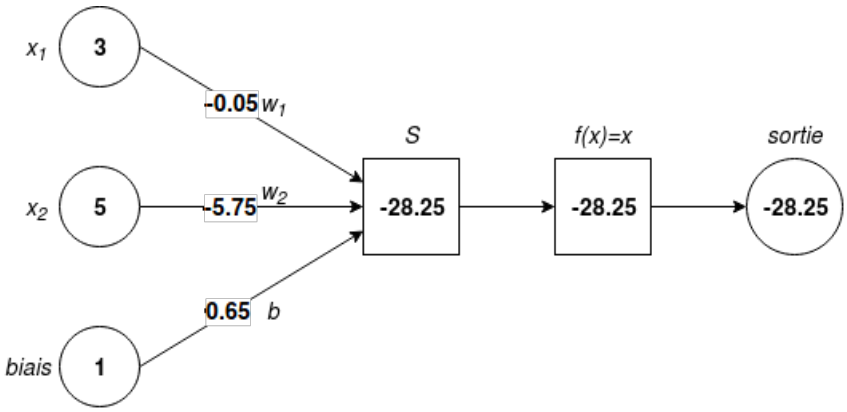

Figure 4.2 – *Schéma du neurone présente à la Figure 4.1 après une étape d'apprentissage.*

1. Le résultat final doit être $w_1 = w_2 = -1$ et $b = 0$.

4.1.4 Réseau de neurones multicouches

Faire la somme de deux nombres puis calculer l'opposé n'a rien de spectaculaire. Comment augmenter les capacités des neurones ?

La solution consiste à ne pas prendre qu'un neurone, mais plusieurs et créer des *couches* dites cachées. Une couche est un ensemble de neurones qui prendront tous le même vecteur d'entrées. Il est possible de créer plusieurs couches et de les lier entre elles. Un *réseau* de neurones est ainsi créé. La Figure 4.3 illustre un réseau de neurones. À gauche de la figure, nous avons trois entrées. Elles sont toutes connectées à une couche composée de 4 neurones. Cette même couche est complètement connectée à une autre couche composée de trois neurones. Enfin, cette dernière couche est connectée à deux neurones en sortie.

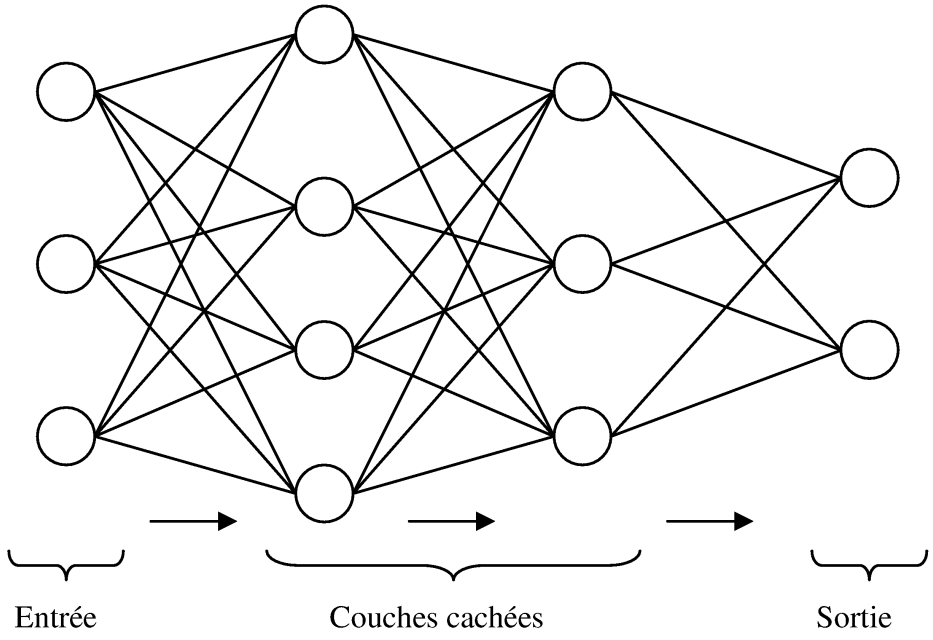

Figure 4.3 – *Schéma général d'un réseau de neurones. Chaque cercle représente un neurone. Ses caractéristiques sont : 3 entrées, 2 sorties et 2 couches cachées.*

4.1.5 Un réseau pour la musique ?

Dans le traitement automatique des langues naturelles, beaucoup de réseaux de neurones ont su montrer leur efficacité [6]. En existe-t-il d'aussi d'efficaces, ou du moins adaptés, à la musique ?

Nous avons défini la notion de fausse note à l'aide d'un contexte, à savoir une mélodie. Une mélodie est une succession temporelle de notes. Nous pourrions définir un réseau, comme ci-dessus, dont chaque entrée représente une note. La notion de temps est difficile à exprimer avec un tel réseau. En effet, nous pourrions lui donner en entrée cinq notes. Puis, à l'instant suivant, nous lui donnerions la même séquence en ayant enlevé la première note, décalé les autres d'un cran et ajouté la nouvelle note. Mais, cela ne lui dira pas que cette nouvelle séquence vient juste après la première séquence. Il va analyser les séquences de façon statique. Ces réseaux de neurones n'ont pas de notion mémoire, n'ont aucune persistance [14].

Pour permettre à l'information de devenir persistante, des réseaux de neurones particuliers ont été inventés : les réseaux de neurones récurrents (RNN). Le schéma ci-dessous montre l'idée

générale des RNNs. L'idée est de faire une boucle d'un réseau de neurones quelconque, noté A. À un instant t, il prend une entrée x_t et donne la sortie h_t. La boucle fait transiter l'information au fur et à mesure du temps. Les informations deviennent persistantes. Ce fonctionnement peut être vu comme une succession du même réseau (voir Figure 4.4).

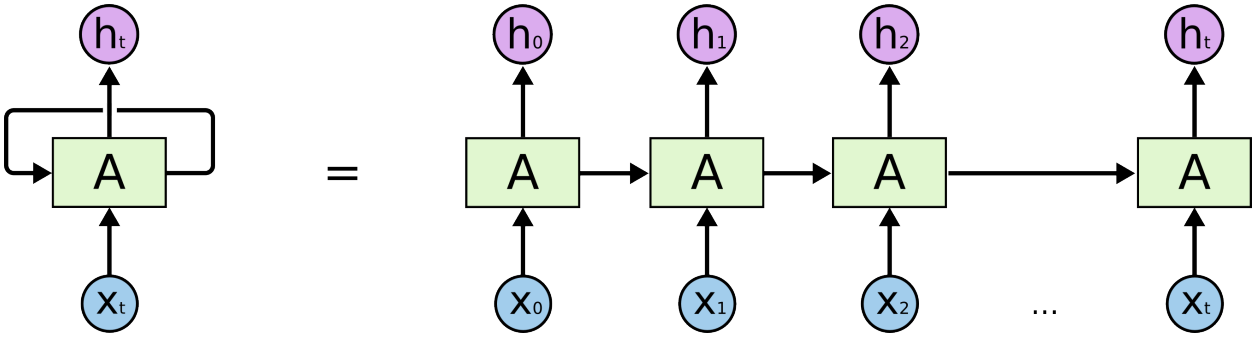

Figure 4.4 – *Schéma abstrait et pratique d'un réseau de neurones récurrent [14].*

En théorie les RNNs répondent aux problèmes de temporalité, mais en pratique ils sont confrontés à la dépendance à long terme. Ils ne peuvent corréler l'information que de sur des courts intervalles de temps, de l'ordre de 10 notes dans notre cas. Pour corréler l'information, il est nécessaire de propager l'erreur dans le sens inverse du temps.
Par exemple, pour détecter si un morceau est dans le style du blues, il est important de détecter les changements d'accords. Pourtant, ces changements peuvent être séparés par maintes notes [15].

Pour remédier à ce problème de dépendance à long terme beaucoup de stratégies ont été proposées. Comme Williams et Zipser en 1995 avec l'algorithme *Back-Propagation Through Time*, ou Robins et Fallside en 1987 avec leur stratégie *Real-Time Recurrent Learning*. Mais tous ont eu le défaut que la rétropropagation a de moins en moins d'effet lorsqu'elle remonte dans le temps ou bien l'effet inverse, c'est-à-dire de faire exploser le gradient (divergence).

En 1997, Hochreiter et Schmidhuber introduisent le réseau *Long-Short Term Memory* (LSTM). C'est un RNN particulier qui évite le problème lié au gradient. Nous n'expliquerons pas en détail les raisons pour lesquelles les LSTMs fonctionnent, l'analyse est faite par [16] et [17]. Regardons succinctement leur construction. Les LSTMs tirent leur construction d'un principe appelé *porte*. Une porte est un composant qui va gérer l'information. Les LSTMs sont composés de trois portes. La première est la *porte d'oubli*. Elle permet d'oublier de l'information issue des instants précédents. La *porte d'entrée* permet elle d'ajouter de l'information issue de l'instant courant. Enfin, la *porte de sortie* indique l'information à transmettre aux instants suivants.

La théorie de la musique pour détecter une fausse note nous est proscrite ; les réseaux de neurones semblent être une solution pour parer à cette restriction. En effet, les réseaux de neurones sont capables de représenter des idées abstraites [6]. Nonobstant, il nous faut introduire une notion de temps pour traiter la musique. Les LSTMs, dont l'efficacité a été démontrée, vont nous permettre de mettre en place cette nécessité.

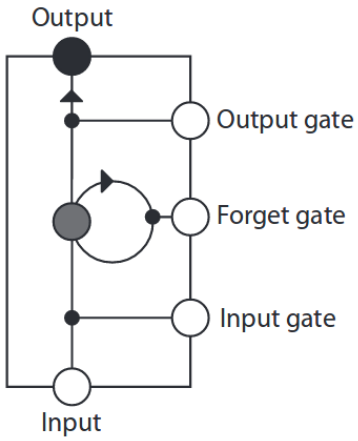

Figure 4.5 – *Schéma d'un LSTM. Illustration issue de [15].*

4.2 Apprentissage par renforcement

Nous avons, a priori, une architecture pour la détection de fausses notes. Nous allons maintenant présenter une méthode pour apprendre à un modèle à ne plus refaire la même erreur.

L'idée derrière l'apprentissage par renforcement est simple. Tout comme pour les réseaux de neurones, nous allons faire appendre le système grâce à ses erreurs. Contrairement à la méthode précédente, nous n'allons pas lui présenter les valeurs d'entrées et le résultat, puis calculer l'erreur qu'il commet. L'apprentissage va se faire à l'aide de la méthode du bâton et de la carotte. Nous lui fournissons les entrées. S'il fait une erreur, nous sanctionnons le système, sinon nous le récompensons. Cette partie s'intéresse à détailler comment récompenser le système, tout en introduisant le vocabulaire de base.

4.2.1 Vocabulaire

Cet apprentissage nécessite un peu de vocabulaire spécifique. Tout d'abord, l'objet de l'apprentissage est appelé *agent*. Un agent est mis dans un *environnement*. L'environnement est tout ce qui ne concerne pas l'agent. L'agent va évoluer et interagir avec cet environnement.

Processus décisionnel markovien

L'agent est à tout moment t dans un *état* s_t. Ce sont les caractéristiques sur lesquels nous voulons qu'il apprenne. L'agent va passer, au fur et à mesure de l'apprentissage, d'état en état. Pour changer d'état, il effectue une *action* a_t. Et à chaque action faite, nous lui donnons une *récompense* r_t (bonus ou malus).
Ce cheminement est mis en évidence ci-contre, dans la Figure 4.6.

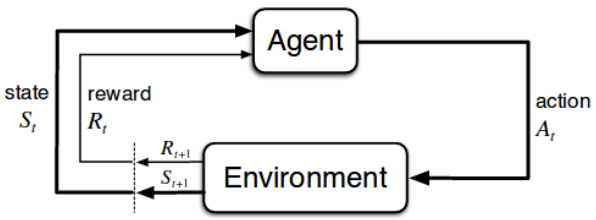

Figure 4.6 – *Schéma d'interaction entre l'agent et l'environnement. Image issue de [18]*

Un *processus décisionnel markovien* (MDP) repose sur le fait que la probabilité d'être dans l'état s_{t+1} ne dépend que de l'état actuel s_t et de l'action a_t qui est faite.

Anticipation des récompenses futures

Pour être plus proche du monde réel, nous devons prendre en considération non simplement la récompense obtenue immédiatement, mais celles qui viendront ultérieurement. Comment allons nous nous y prendre ?

Avec un MDP, nous pouvons rapidement estimer la récompense totale obtenue au bout de n actions :

$$R = r_1 + r_2 + \cdots + r_n$$

La récompense totale future à l'instant t est notée R_t et vaut alors $r_t + r_{t+1} + \cdots + r_n$ (récompense de l'instant t jusqu'à l'instant n). Le problème est que nous supposons qu'en effectuant la même action dans le futur nous obtiendrons la même récompense. Or nous ne pouvons en être sûrs, nous ne contrôlons pas l'environnement. Pour ne pas faire cette supposition, nous introduisons le facteur d'actualisation que l'on note usuellement γ. La récompense totale future vaut alors :

$$R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^{n-t} r_n$$

Le facteur d'actualisation est compris entre 0 et 1 ; plus nous nous projetons dans le futur, moins nous considérons les récompenses à venir. Par simplification, nous pouvons aussi écrire :

$$R_t = r_t + \gamma(r_{t+1} + \gamma(r_{t+2} + \dots)) = r_t + \gamma R_{t+1}$$

Pour $\gamma = 0$, notre stratégie consiste à prendre immédiatement la récompense. Si l'environnement est déterministe alors nous pouvons poser $\gamma = 1$.

Q-fonction

La *fonction de valeur* estime la qualité d'une action donnée dans un état donné. La notion de qualité se fonde sur les récompenses qui peuvent être perçues. Bien entendu, les récompenses qu'un agent peut espérer recevoir dépendent de l'action qu'il choisit. Une *politique* π représente, pour chaque état s et chaque action a, la probabilité d'effectuer l'action a dans l'état s. Cette probabilité est usuellement notée $\pi(s, a)$.

La *fonction d'action-valeur pour la politique* π, noté usuellement noté $Q^\pi(s, a)$ et appelé *Q-fonction*, représente la récompense obtenue en faisant l'action a dans l'état s selon la politique π. Cette fonction peut être être définie à l'instant t par :

$$Q^\pi(s_t, a_t) = \max_\pi R_{t+1}$$

L'objectif est d'estimer la fonction de valeur optimale, notée Q^*, pour obtenir la politique optimale [18]. La politique est alors déterminée par :

$$\pi(s) = \operatorname*{argmax}_a Q(s, a)$$

Pourtant nous ne pouvons toujours pas calculer cette Q-fonction. Pour ce faire, nous allons procéder comme pour les récompenses. Nous allons exprimer la fonction d'un état s et d'une action a en fonction de sa valeur dans l'état suivant s' et de la récompense r.

$$Q(s, a) \quad = \quad r + \gamma \max_{a'} Q(s', a') \tag{4.1}$$

Cette équation est appelée équation de Bellman. De manière logique, la meilleure récompense pour cette action et cet état est la somme de la récompense immédiate et du maximum possible pour l'état suivant.

Q-learning

Le Q-learning est une façon d'approximer itérativement la Q-fonction et d'obtenir l'un de ses optimum à l'aide de l'équation de Bellman (Équation 4.1). En 1989, Watkins l'introduit et il le définit par la formule suivante :

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right]$$

Dans les cas simples, la Q-fonction est représentée à l'aide d'un tableau. L'algorithme dit Q-learning est décrit dans l'Algorithme 1) :

Algorithm 1 Pseudo-code de l'algorithme dit Q-learning [18]

Initialize $Q(s, a)$ arbitrarily
for each episode **do**
    Initialize s
    repeat
        Choose a from s using policy derived from Q.
        Take action a, observe r and s'
        $Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right]$
        $s \leftarrow s'$
    until s is a terminal state

D'autres algorithmes ont été développés. Il existe deux catégories d'apprentissage : *en ligne* et *hors ligne*. L'apprentissage en ligne apprend à partir de la politique menée et suppose qu'elle sera poursuivie par la suite. L'apprentissage hors ligne apprend en ignorant la politique menée et va opter pour la première politique efficace trouvée.
Pour le Q-learning, nous approximons Q^* indépendamment de la politique suivie. C'est donc un algorithme hors ligne.

4.2.2 Un exemple

Pour illustrer ce vocabulaire, regardons un exemple.

La grille représente l'environnement, voir ci-contre (Figure 4.7). Notre agent sera une souris. Les cases grisées représentent un fromage et sont des état terminaux (l'expérience est finie une fois l'état atteint). Les états sont : $S = \{1, 2, ..., 14\}$. Quatre actions sont possibles : $A = \{Haut, Bas, Droite, Gauche\}$. Chaque transition faite est récompensée à hauteur de -1.

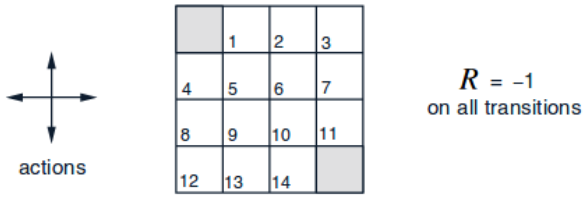

Figure 4.7 – *Définition de l'environnement dans lequel l'agent, une souris, va évoluer. Schéma issu de [18]*

Au départ, nous supposons que notre souris est dans l'un des états non terminaux. La politique initiale est de faire une action au hasard, avec équiprobabilité pour chaque état. La Figure 4.8 montre la séquence d'améliorations de la politique à l'aide d'un algorithme de renforcement : l'algorithme d'itération de la politique (voir [18]).

Au départ, la souris peut faire n'importe quelles actions dans n'importe quels états ; elle ne connaît pas où les fromages se situent. À chaque étape d'itération, nous pouvons observer que les choix des actions à faire s'affinent. Le nombre d'actions par états se réduit. Les récompenses valent -1 et agissent comme une punition. La souris n'aime pas les punitions, elle va définir ses choix afin de minimiser celles-ci.

L'apprentissage par renforcement repose sur une hypothèse markovienne : la récompense et l'état suivant dépendent de l'état actuel et de l'action faite. Le Q-learning est un algorithme qui semble pouvoir répondre à notre problème ; faire apprendre à notre modèle à ne plus refaire les mêmes erreurs. Cependant, nous allons voir dans la partie suivante que nous allons devoir penser autrement pour notre cas particulier.

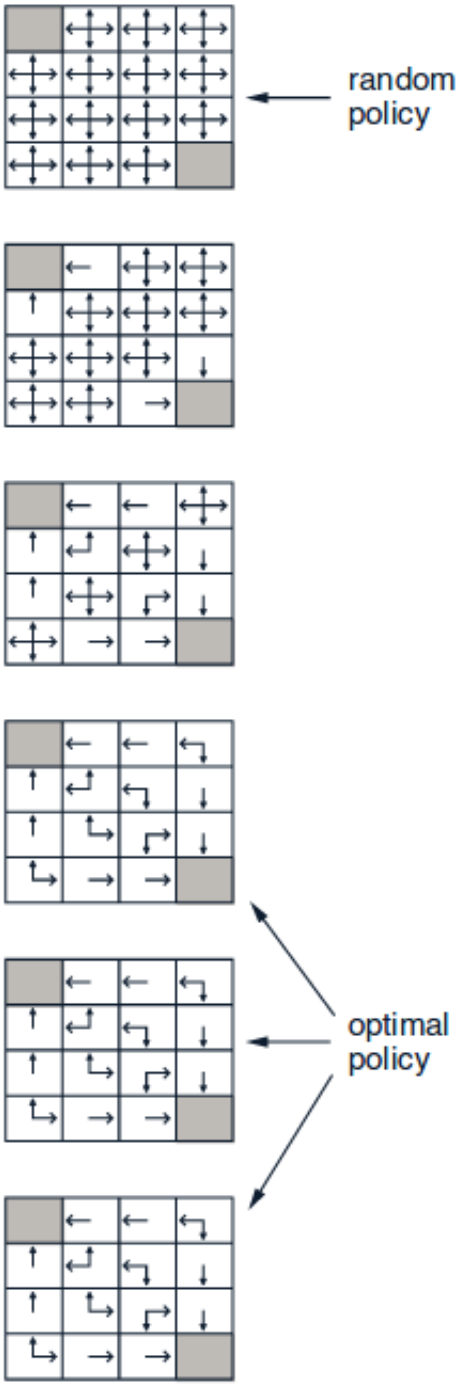

Figure 4.8 – *Aperçu de l'apprentissage par renforcement sur l'exemple. La politique initiale consiste à choisir une action au hasard. À chaque itération, nous observons que le choix des actions pour un état donné se précise, se réduisant à une ou deux actions. La politique optimale détermine l'action à faire selon l'état afin de minimiser les malus. Schéma issu de [18].*

5 Modèle et solution proposée

Nous venons de voir un ensemble de briques nous permettant de répondre à nos différentes questions : qu'est-ce qu'une fausse note ? Est-il possible de créer un modèle capable de détecter de telles notes ? Enfin, pouvons apprendre au modèle à ne plus proposer de fausses notes ? Il nous reste à les assembler afin de proposer une solution à ces questionnements.

Cette partie va se consacrer aux modèles qui ont été proposés pour accomplir les objectifs. Le premier modèle répondra à la question de la détection de fausse note. Le second modèle viendra apporter une solution afin d'éviter la répétition des erreurs faites par l'oracle des facteurs lors de la génération d'improvisations.
Nous regarderons, dans un premier temps, la modélisation d'une note et le format des jeux de données, ce qui nous amènera à l'architecture du premier modèle. Dans un dernier temps, nous verrons la mise en place d'une forme d'apprentissage par renforcement pour l'oracle. Finalement, nous présenterons les outils choisis pour la mise en œuvre des modèles.

5.1 Encodage des notes

Nous l'avons vu, un réseau de neurones ne peut manipuler que des nombres. Or, nous avons affaire à des notes musicales. Il nous faut donc trouver une représentation numérique pour les notes. Avant de détailler le choix fait, nous allons brièvement regarder le format des jeux de données.

5.1.1 Format des données d'entrée

Dans le diagramme de classes, présenté dans la partie 2.2.3, le module Utils.py comporte une classe nommée XmlNote. Cette classe tire son nom du format de nos données, à savoir le format MusicXML. Ce format a été créé afin de pouvoir permettre aux logiciels de Composition Assistée par Ordinateur (CAO), ou bien de Musique Assistée par Ordinateur (MAO) de pouvoir échanger entre eux dans un format standard de données [1].
Ce format tire ses origines d'un autre format appelé XML. XML est un format d'échange de données entre différents ordinateurs/logiciels. XML est un format qui se veut à la fois lisible par l'Homme et la machine [2].

1. http ://www.musicxml.com/
2. https ://www.w3schools.com/Xml/

```
<measure>
  <note>
    <pitch>
      <step>G</step>
      <octave>4</octave>
    </pitch>
    <duration>2</duration>
  </note>
  <note>
    <rest/>
    <duration>8</duration>
  </note>
  <note>
    <pitch>
      <step>C</step>
      <octave>3</octave>
    </pitch>
    <duration>4</duration>
  </note>
</measure>
```

Ci-contre nous avons un extrait d'un fichier au format MusicXML. Une mesure, marquée par la balise *measure*, est composée de notes. Chaque note possède plusieurs fils.
Le premier, marqué par la balise *pitch*, possède deux fils (voir première note). Le premier, marqué par la balise *step*, dénote la hauteur et le deuxième dénote l'octave.
Le deuxième fils de la note a pour balise *duration*; c'est la durée de la note exprimé en tatum.
La deuxième note de la mesure ne possède pas de fils *pitch*, il est remplacé par un fils qui a pour balise *rest*. Cela signifie que nous affaire à un silence. Ce silence a également une durée.

Figure 5.1 – *Extrait d'un fichier MusicXML.*

5.1.2 Modélisation d'une note

Les données au format MusicXML sont mises sous forme de classe (XmlNote). C'est de cette dernière que nous effectuons l'encodage des notes. Précisons la manière dont nous allons encoder une note afin de correspondre à un format de données utilisé par un réseau de neurones.

Nous avons décidé de ne nous intéresser qu'à la hauteur et au rythme des notes. Rappelons que nous avons distingué 13 hauteurs de notes, à savoir douze demi-tons et le silence. Pour encoder ces hauteurs nous allons utiliser l'encodage dit *one-hot* [19]. Cet encodage consiste à représenter chaque hauteur à l'aide d'une représentation *single*. C'est-à-dire que le vecteur d'entrée contiendra un indice unique ayant pour valeur un et le reste des indices seront des zéros (voir Tableau 5.1).

L'avantage majeur est qu'il n'introduit pas de notion de distance entre les différentes notes. Chaque note sera à distance de un les unes des autres. Nous avons fait ce choix car il n'y a aucune justification musicale à cette distanciation. Chaque note doit être à égale distance de toutes les autres. Pour passer d'un état à un autre, il faut à chaque fois modifier deux bits. En contrepartie, la taille de représentation augmente : 13 bits sont nécessaires pour représenter les 13 *notes* alors que nous pourrions nous contenter de 4 bits (le calcul de l'entropie donne : $\log_2(13) \approx 3.70$).

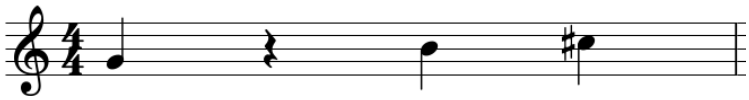

Figure 5.2 – *Fragment de musique utilisée pour illustrer l'encodage des notes.*

La hauteur ne suffit pas pour représenter une note, il nous faut aussi son octave. La majorité des mélodies se jouent entre la première et la cinquième octave (inclue). Nous encoderons ces cinq octaves sur cinq bits, en utilisant l'encodage one-hot, car, là encore, il n'y a aucune raison

N° notes	Hauteurs de notes													Octave				
	C	C♯	D	D♯	E	F	F♯	G	G♯	A	A♯	B	Silence	1	2	3	4	5
1 (Sol)								1								1		
2 (Silence)													1					
3 (Si)												1				1		
4 (Do dièse)		1															1	

Table 5.1 – *Encodage one-hot de la hauteur de la note et son octave pour le fragment musical visible à la Figure 5.2. Afin de faciliter la lecture les 0 sont représentés par des cases vides.*

d'introduire des distances entre chaque octave.

Dans le Tableau 5.1, nous observons que deux notes différentes sur un même octave partage un bit en commun. Cela induit une notion de distance évoquée précédemment. Cependant, encoder chaque note en différenciant les octaves nous amènerait à un vecteur d'encodage de taille 65 (13×5, trois fois plus grand que l'actuel encodage), ce qui augmenterait amplement le temps d'apprentissage [19]. En somme, un tel encodage a déjà été utilisé pour la génération musicale [19].

Nous avons treize bits pour la hauteur et cinq pour l'octave. Ce qui nous donne 18 bits d'encodage. Enfin, nous allons encoder le rythme. Plus précisément, nous allons encoder l'attaque de chaque note, c'est-à-dire à quelle moment dans un temps celle-ci est jouée.

Pour le rythme, nous utilisons la notion de tatum que nous avons vu. Cependant, nous avons vu que la valeur du tatum dépend de chaque morceau. Pour contourner ce problème, nous allons nous fixer un tatum. Ce tatum sera le même quelque soit le morceau. La taille du vecteur pour encoder le rythme sera de la valeur de la noire exprimée en tatum.

Reste à savoir la valeur que nous allons fixer à ce tatum. En combien de parties devons-nous segmenter une noire afin de pouvoir effectuer une liste de rythmes donnée ? Pour ce faire regardons le Tableau 5.2 :

Nom	Rythme/Valeur					
noire	binaire	1				
croche	binaire	2	triolet	3		
double	binaire	4	triolet	6	quintolet	5
triple	binaire	8	triolet	12	quintolet	10

Table 5.2 – *Liste du nombre de temps nécessaire pour faire une noire complète. Exemple : pour encoder une croche en triolet il faut décomposer la noire en trois. Ainsi, la décomposition d'une noire qui permette d'encoder une double croche (4), un triolet de doubles (6) et un quintolet de doubles (5). Il faut alors un vecteur de taille $PPCM(4, 5, 6) = 60$.*

Pour modéliser la double croche, le triolet de doubles et le quintolet de doubles, il faut alors découper en 60 parties la noire. Ces trois rythmes (et tous ceux qui en découlent, à savoir croche, triolet de croches, etc.) permettent de pouvoir encoder une grande majorité des rythmes usuels, et donc d'apprendre sur un large jeu de morceaux. Nous avons donc choisi de fixer 60 tatums par temps.

Un problème supplémentaire survient : le problème des notes liées. Ce sont les notes qui démarrent hors d'un temps et se prolongent sur les temps suivants. Autrement dit, que faire des notes dont la durée exprimée en tatums dépasse le seuil fixé ? C'est le cas pour une blanche, qui

vaut deux noires, sa durée en tatums vaut deux fois le seuil.

Nous allons quantifier les morceaux pour parer à ce problème. La quantification va décomposer les temps supérieurs à la noire en temps inférieurs ou égales à une noire. La Figure 5.3 illustre le problème des attaques des notes liées et leur quantification qui facilite la représentation (voir Tableau 5.3).
Dans la figure, la deuxième note est une noire pointée qui vaut dans notre cadre 90 tatum. Cette valeur est supérieure à 60. Nous allons la quantifier par deux notes, une noire pointée est équivalente à une noire suivie d'une croche liée (90 = 60 + 30). Seul la noire est attaquée, la croche ne l'est pas.

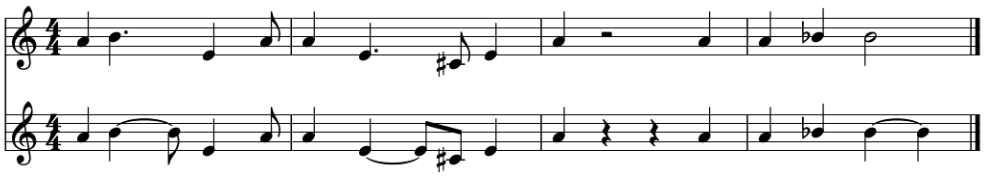

Figure 5.3 – *Sur la portée supérieure, un fragment original de morceau. Sur la portée inférieure, la version quantifiée.*

	Indice du tableau d'encodage rythmique						
N°Notes	0	...	...	30	...	...	59
1	1						
2	1						
3							
4				1			
5				1			

Table 5.3 – *Exemple d'encodage rythmique pour les cinq premières notes de la deuxième portée de la Figure 5.3. Afin de faciliter la lecture des 0 sont représentés par des cases vides.*

Finalement, le vecteur d'entrées est la concaténation de l'encodage one-hot de la hauteur, de l'octave et du rythme. La taille du vecteur, avec les valeurs fixées, est de 78. Nous n'avons pas qu'une note à encoder, mais une mélodie entière. La taille de cette mélodie est fixée par un paramètre. Pour les tests, nous avons décidé de fixer la taille de cette mélodie à 9 notes. L'algorithme d'encodage complet est décrit ci-dessous (voir Algorithme 2).

Algorithm 2 Pseudo-code d'encodage des notes

```
1:  indice = 0                                  ▷ Indice du tableau rythme à mettre à 1
2:  for chaque note do
3:      hauteur ← Encodage la hauteur
4:      octave ← Encodage de l'octave
5:      while indice > tatum do
6:          Créer une note liée à partir de cette note et l'encoder
7:          indice ← indice − tatum
8:      if note ≠ silence then
9:          rythme[indice] ← 1
10:     indice ← indice+ durée de la note
11:     Concaténer hauteur, octave, rythme
```

5.1.3 Contexte passé et anticipé

Afin d'affiner notre représentation, nous allons donner un peu plus de sens à notre mélodie. Pour cela, nous avons choisi de découper ces N notes en deux parties de même longueur :

- la première représentera la séquence musicale jouée (notion de passé) ;
- la deuxième représentera la séquence musicale à jouer (notion d'anticipation).

La Figure 5.4 illustre cette notion. Une partie hachurée symbolise les notes jouées, l'autre partie symbolise les notes anticipées. Une note entre les deux parties symbolise la première note qui va être jouée. En découpant la mélodie de cette manière, nous avons deux cas à traiter. Le premier cas est lorsque la fausse note se situe avant la note courante. La fausse note a donc été jouée. L'oracle ne peut pas, a posteriori, modifier la séquence musicale. En revanche, dans le cas où la fausse note se situe dans les notes anticipées, l'oracle peut modifier son comportement, car, les notes anticipées sont du ressort de l'oracle. C'est à partir des notes jouées qu'il devra estimer la meilleure séquence musicale à anticiper. Dans cette séquence future, l'objectif est d'avoir le moins de fausses notes.

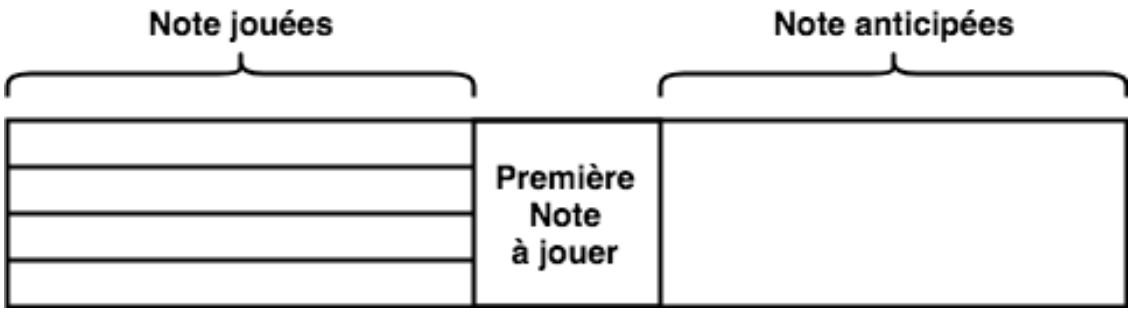

Figure 5.4 – *Schéma montrant le découpage de la mélodie en deux parties. La partie hachurée modélisera les notes jouées, l'autre partie modélisera les notes anticipées. Au centre, la note courante.*

5.2 Modèle de détection de fausse note

Maintenant que nous avons vu l'encodage des notes, attachons nous à la présentation du modèle, des choix qui ont été faits et leur raison. Ici, nous proposons un modèle unique capable de réaliser séparément les deux tâches décrites précédemment : la fausse note a déjà été jouée et la fausse note est anticipée.

5.2.1 Architecture

Nous avons montré la nécessité d'inscrire une note dans un contexte, à savoir la mélodie, pour définir la notion de fausse note. Nous avons également vu les réseaux de neurones récurrents et les LSTMs qui répondent le mieux au problème de la dépendance à long terme et aux problèmes de gradient. Les LSTMs permettent de faire transiter l'information des instants t, $t-1$, ... à un instant $t+1$. Musicalement, nous pouvons imaginer cela comme l'impact des notes jouées par le musicien sur la note actuelle.

Mais, l'improvisateur ne fait pas qu'improviser note à note. Il s'imagine une séquence musicale à venir, une mélodie à jouer. Cette mélodie imaginée a un impact direct sur le jeu du musicien.

Il nous faut alors penser à un réseau capable à la fois de transmettre l'information de l'instant t vers l'instant $t+1$, mais également dans l'autre sens, de l'instant t vers l'instant $t-1$. Le *Bidirectionnal RNN* (BiRNN) va permettre de modéliser cette double transmission [20].

BiRNN

L'idée sous-jacente est de dupliquer un RNN. Un RNN va de l'instant t_0 à l'instant t_n, l'autre va aller en sens inverse. Un BiRNN est montré en Figure 5.5.

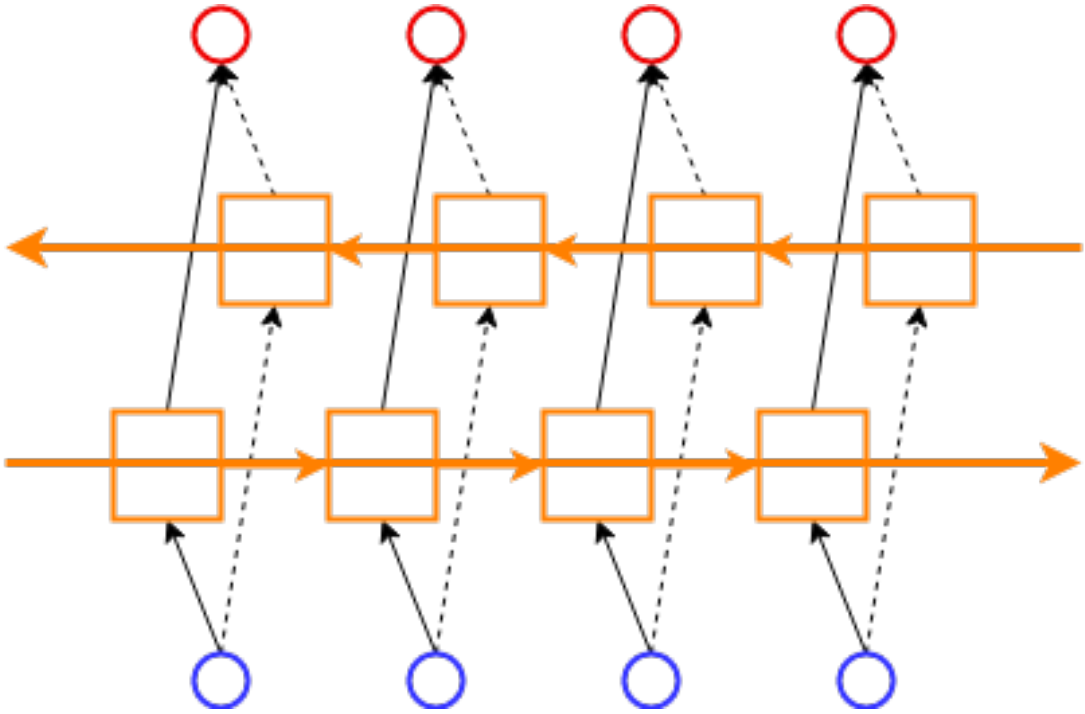

Figure 5.5 – *Représentation d'un BiRNN. En bleu (en bas) sont représentés les entrées. En orange (au centre) nous observons deux RNNs allant dans deux directions : la première suivant le temps, l'autre "remontant" le temps. En rouge (en haut) nous avons les sorties.*

Dans la Figure 5.5, chaque entrée, illustrée par un cercle en bas de la figure, représente une note. Le nombre de notes sera fixé à l'aide d'un paramètre. Le taille du vecteur de sortie d'un BiRNN correspond à la même taille que le vecteur d'entrées, à savoir le nombre de notes que nous fixerons. Dans un soucis de simplification du problème, nous ne voulons pas savoir si la n-ième note est fausse, mais simplement s'il existe une fausse note dans la séquence musicale donnée en entrée. C'est un point-clef de la modélisation.

Pour ce faire, nous avons besoin que de deux neurones en sortie. Le premier neurone indiquera s'il y a une fausse note, l'autre indiquera s'il n'y pas de fausse note. Il s'agit là d'une représentation one-hot pour une sortie binaire (fausse ou non). Pour unifier chaque sortie du BiRNN, nous avons ajouté une couche supplémentaire. Elle connecte l'ensemble des sorties du BiRNN et permet de *mélanger* chaque sortie afin que le réseau soit capable de plus d'abstraction (c'est la force des réseaux de neurones). La fonction d'activation de cette sortie est la fonction *softmax*. Cette fonction permet à la somme des sorties d'être égales à un. Ainsi, nous pouvons donner une signification probabiliste à chaque sortie. Une probabilité que la mélodie possède une fausse note pour la première sortie, et l'évènement contraire pour la second sortie.
Grâce au dropout, une technique de régulation qui vise à accélerer la convergence du gradient de l'erreur [21] à chaque itération un certain nombre de neurones sera désactivé. Cette technique

permet d'accélérer la convergence du réseau [21]. La Figure 5.6 schématise le réseau final choisi. Nous avons comme première couche le BiRNN, puis une couche supplémentaire afin de n'avoir que deux neurones en sortie. En sortie, nous avons deux neurones, un pour dire s'il y a une fausse note, l'autre pour dire si la mélodie est bonne (aucune fausse note).

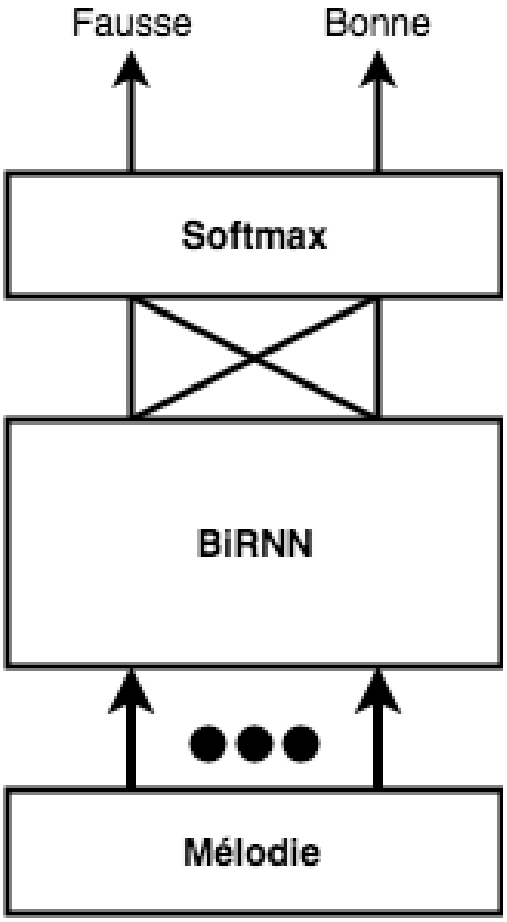

Figure 5.6 – *Schéma du réseau de neurones pour la détection de fausse note. En entrée nous avons la mélodie qui va passer à travers un BiRNN. Puis, toutes les sorties du BiRNN sont connectées à une couche avec comme fonction d'activation la tangente hyperbolique. Cette couche est totalement connectée à une couche appliquant le mécanisme de dropout. En sortie de cette dernière, nous récupérons deux valeurs : une pour indiquer s'il y a fausse note, l'autre si la mélodie est juste.*

Pour les mesures de performances, après l'apprentissage, nous voulons avoir des valeurs égales à 0 ou 1. Pour ce faire, nous prenons la sortie ayant la valeur la plus élévée (proche de un). Par exemple, nous pouvons avoir en sortie $(0.45, 0.55)$, alors nous dirons que la deuxième sortie est activée. Le modèle aura donc détecté aucune fausse note.

5.2.2 La fonction d'erreur

Pour compléter le modèle, il nous reste à choisir la fonction d'erreur. Nous avons déjà vu l'erreur quadratique moyenne. Cette erreur n'est pas adaptée à notre cas. En effet, cette erreur est plus adaptée lorsque nous avons affaire à une régression.
Ce n'est pas le cas ici. Nous voulons faire une classification, une mélodie contient une fausse note ou non. La fonction adaptée est la fonction d'entropie croisée pour le cas binaire. La formule est

la suivante :

$$BCE = -\sum_{y} \left[y_{att} \log(y) + (1 - y_{att}) \log(1 - y) \right]$$

avec y_{att} la sortie attendue et y la sortie du réseau de neurones. C'est la fonction d'erreur que nous avons choisi.

5.3 Modèle pour l'apprentissage par renforcement

Maintenant que nous avons vu le modèle qui va détecter les fausses notes, prenons le temps d'étudier le modèle pour l'apprentissage par renforcement.

5.3.1 Le problème et sa modélisation

L'oracle des facteurs est notre agent. Les états sont constitués des états de l'oracle des facteurs. Malheureusement, nous ne sommes pas dans un cas où nous définissons les récompenses et les actions à partir d'un seul état s_t donné (MDP d'ordre 1). La solution évoquée dans le Chapitre 4 doit être adaptée à notre cas. Ici, il nous faut prendre en compte une séquence mélodique. Autrement dit, il nous faut prendre en compte les actions $a_{t:t+k-1}$, avec k une valeur fixant l'horizon jusqu'à laquelle nous voulons estimer la qualité de la séquence.
Or, avec l'oracle, nous pouvons imaginer l'ensemble des k actions possibles à faire depuis un état donné pour construire un ensemble de séquences de longueur k possibles. Il nous est alors possible d'estimer l'espérance de la récompense de toutes ces séquences finies. Une fois chaque espérance calculée, il nous suffit de prendre la séquence musicale maximisant cette espérance. L'espérance des récompenses estimée est faite par le réseau de neurones que nous avons construit précédemment (voir partie 5.2).
La Figure 5.7 décrit l'idée exposée. Nous avons un état donné, l'état de départ. Depuis cet état, nous pouvons explorer l'ensemble des séquences accessibles. Elle sont au nombre de trois. Chaque séquence possède des fausses notes représentées en point de couleur rouge. Nous observons que la séquence numéro une possède le plus de fausses notes, alors que la séquence numéro trois n'en possède qu'une. Nous supposons que notre modèle juge que la séquence numéro trois est de meilleure qualité, l'espérance des récompenses est la maximum. Alors, il suffit à l'oracle de choisir la séquence d'actions menant à cet séquence musicale.

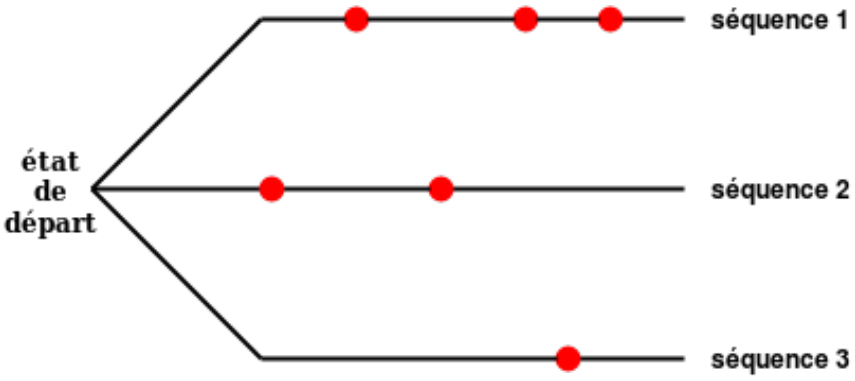

Figure 5.7 – *Schéma illustrant la maximisation de l'espérance des récompenses sur un ensemble de séquences musicales possibles depuis un état donné. Les points rouges représentent des fausses notes.*

5.4 Présentation des outils choisis

Afin de mettre en place toute cette architecture et de tester le modèle proposé, nous avons choisi les outils présentés dans cette partie.

5.4.1 Présentation de Grid5000

L'équipe Multispeech bénéficie au sein de l'Inria d'un outil dénommé Grid5000 (g5k) permettant l'exécution d'expériences. Grid500 [22] se veut être un banc d'essai numérique à grande échelle, offrant des capacités de calculs importantes. C'est une grille d'ordinateurs, composée de 1000 nœuds [3] et 8000 cœurs [4]. Cet outil est très adapté aux calculs parallèles, aux systèmes distribués et aux traitements de données en masse. Cet outil s'adapte donc parfaitement à l'intelligence artificielle, nécessitant beaucoup de calculs et de ressources.

Grid5000 permet un contrôle précis et complet des dépendances logicielles utilisées et de l'infrastructure matérielle, tout en offrant une isolation du réseau. Enfin, d'autres outils permettent la collecte de données autour de la consommation d'énergie et de l'utilisation du réseau. Beaucoup d'efforts sont faits pour permettre la reproduction d'expérience. Une communauté de scientifiques et d'ingénieurs œuvre en ce sens. Les outils logiciels proposés sont, notamment :

- la réservation de ressources physiques ;
- le déploiement et l'installation des ressources logicielles ;
- la virtualisation et la paramétrisation d'un environnement expérimental.

C'est pourquoi l'utilisation de Grid5000 semble réunir toutes les caractéristiques nécessaires pour mener à bien nos expériences.

5.4.2 Présentation de Python et *Tensorflow*

Afin de mettre en place l'intelligence artificielle, il nous faut choisir un langage de programmation. Parmi la diversité des langages existants, seuls quelques-uns sont plus adaptés pour l'apprentissage automatiques. Et parmi ceux-ci, Python en a les capacités. Python est un langage de programmation dit de haut niveau, donc il se veut plus proche dans langage humain. De nombreuses bibliothèques ont été écrites, facilitant la création de réseau de neurones. De plus, l'oracle des facteurs écrit par Ken Déguernel utilise Python. Cela renforce le choix de Python comme langage de programmation.

Tensorflow est l'une des bibliothèques destinées à la création de réseaux de neurones. À l'origine, Tensorflow a été developpé par des chercheurs et des ingénieurs de l'équipe *Google Brain* en collaboration avec l'organisation *Google's Machine Intelligence* [5]. L'objectif premier se focalisait sur l'apprentissage profond et l'apprentissage automatique, mais depuis la bibliothèque offre de nombreux services (calculs scientifiques, application en temps réel, ...). Tensorflow est libre de droits et sans contrainte de paiement.
Par ailleurs, Tensorflow est très utilisé par beaucoup de chercheurs en intelligence artificielle,

3. Un nœud désignant généralement un ordinateur, ou une entité connecté au réseau.
4. Processeur, unité centrale de traitement.
5. https://www.tensorflow.org/

rendant l'accès à une documentation et à une aide plus aisé. Tensorflow est toujours maintenu à jour (la version 1.2 date du 30 Juin 2017) ; la disponibilité d'aide et de documentation en est fortement renforcée.
De plus, Tensorflow a développé un autre outil, *Tensorboard* qui est directement intégré. Tensorboard permet la visualisation d'une multitude de données, comme la convergence du réseau de neurones, sa précision, etc. Un outil qui permet de ne pas attendre la fin de l'apprentissage, si nous nous rendons compte que le choix des paramètres n'est pas bon.
En parallèle, Tensorflow est capable de faire une sauvegarde d'un modèle. Cette fonctionnalité nous est nécessaire, et pour cause, avant de renforcer l'oracle nous devons faire apprendre à notre réseau de neurones à détecter les fausses notes. Il faut, dans un premier temps, faire apprendre le réseau, puis le sauvegarder, pour enfin le charger lorsque nous voulons générer une mélodie.
Ce sont les raisons qui nous ont amené à choisir Tensorflow.

5.4.3 Gestion de versions : *Git*

En vue de modification des paramètres du réseau de neurones, pour affiner les performances, il nous a semblé pertinent d'utiliser un outil de gestion de versions. Git est le choix que nous avons retenu. Cet outil permet de créer plusieurs versions de notre code, et de sauvegarder ces versions. Git est un logiciel libre et gratuit. Il se veut simple et performant. De plus, son utilisation est très répandue (dizaine de millions d'utilisateurs). Sa première version a été créée en 2005 et il est encore développé à l'heure actuelle (la version 2.14 date du 4 Août 2017) [6].

5.4.4 Visualisation d'une partition

Afin de pouvoir d'une part voir les données d'entraînement et d'autre part écouter les mélodies générées, nous avons besoin d'un éditeur de partition. L'outil choisi s'appelle MuseScore [7].
Son développement débute en 2008.
Il a été choisi pour deux raisons. La première est sa qualité d'être un logiciel libre et d'être gratuit. La deuxième est qu'il est encore maintenu actuellement. La version 2.1 a été publiée le 2 Mai 2017.
En somme, son interface est simple d'utilisation et rapide à prendre en main.

6. https://en.wikipedia.org/wiki/Git
7. https://musescore.org/en

6 Résultats et discussion

Une fois le modèle choisi et la mise en œuvre de ce dernier, il nous faut regarder les résultats obtenus. Ce chapitre va expliquer les différents critères d'évaluations. Nous présenterons les jeux de données sur lesquels le modèle a fait son apprentissage. Puis, nous exposerons les résultats obtenus en fonction des jeux de données et des critères choisis. Par la suite, nous discuterons des résultats et de l'ensemble des choix que nous avons faits.

6.1 Les résultats

6.1.1 Critères d'évaluation

Pour valider un modèle, il faut des critères d'évaluations. Nous avons, dans notre choix, deux évaluations à faire :

- la détection de fausse note ;
- l'apprentissage par renforcement et l'amélioration globale de l'oracle.

Pour la deuxième évaluation, nous devons écouter les mélodies produites avec et sans apprentissage. Dans le cas où la distinction ne pourrait être faite par une personne lambda (par exemple nous-même), nous devrions prendre un panel de musiciens professionnels. Ils jugerons alors de l'amélioration ou non.
Pour le premier critère, nous allons appliquer les critères propres aux réseaux de neurones. Ces critères vont être décrits dans la sous-partie qui suit.

Matrice de confusion

Nous sommes dans un cas de classification binaire, une mélodie est bonne ou mauvaise. Naïvement, nous pourrions utiliser le taux de bonne classification pour évaluer le modèle. Cela ne suffit pas. Les données réelles ne sont pas nécessairement équiprobablement réparties. Imaginons que sur 100 mélodies nous ayons 20 de mauvaises et 80 bonnes. Un réseau qui dirait que toutes les mélodies sont bonnes aurait alors un taux de bonne classification de 80%. Pourtant, il ne prédit que des bonnes mélodies.
Pour préciser les mesures, nous allons devoir regarder la différence entre ses prédictions et les valeurs réelles et ce pour toutes les classes possibles (ici 2). La matrice de confusion va permettre de faire ce calcul. Chaque colonne de la matrice représente le nombre d'occurrences d'une classe estimée (fausse note ou non), tandis que chaque ligne représente le nombre d'occurrences d'une classe réelle. Le Tableau 6.1 montre un exemple de matrice de confusion. Nous avons deux classes. La classe *Faux* dénote les mélodies qui ont au moins une fausse note. La classe `Juste` dénote

		Classe prédite	
		Faux	Juste
Classe Réelle	Faux	2	4
	Juste	8	6

Table 6.1 – *Exemple de matrice de confusion. Le modèle a prédit au total dix fausses mélodies, seules deux sont réellement fausses, les huit autres étaient bonnes. Le modèle a prédit 10 bonnes mélodies et parmi ces dix quatre étaient fausses et les six autres sont bonnes.*

les mélodies qui n'ont aucune fausse note. Dans la matrice de confusion, nous avons dix fausses notes prédites et dix de bonnes prédites. Parmi les dix fausses notes, seulement deux sont réellement fausses. Et parmi les dix bonnes notes, seulement 6 sont réellement justes. Nous avons un taux d'erreur, ou taux de bonne classification, de valeur : $(2 + 6)/(2 + 8 + 4 + 6) = 0.4$. Une classification sera d'autant meilleure que sa matrice de confusion se rapprochera d'une matrice diagonale (valeurs uniquement présentes sur la diagonale).

Précision et rappel

Avec la matrice de confusion, nous allons pouvoir définir deux critères de mesures de performances : la précision et le rappel.
La précision décrit la capacité du modèle à ne détecter que des mauvaises mélodies. La précision se traduit par le rapport entre le nombre réel de fausses mélodies parmi les prédites sur le nombre total de fausses mélodies prédites. Avec les valeurs du Tableau 6.1 nous avons,

$$\begin{aligned} precision &= \frac{\text{Fausses mélodies réelles parmi les prédites}}{\text{Total de fausses mélodies prédites}} \\ &= \frac{2}{2+8} \\ &= 0.2 \end{aligned}$$

Le rappel décrit la capacité du modèle à détecter toutes les fausses mélodies. Cela se traduit par le rapport entre le nombre réel de fausses mélodies parmi les prédites sur le nombre total de fausses mélodies réelles. Avec les valeurs du Tableau 6.1 nous avons,

$$\begin{aligned} rappel &= \frac{\text{Fausses mélodies réelles parmi les prédites}}{\text{Total de fausses mélodies réelles}} \\ &= \frac{2}{4+2} \\ &= 0.33 \end{aligned}$$

Nous n'avons défini la précision et le rappel que pour la classe *Faux* (il y a une fausse note dans la mélodie). Nous pourrions également définir ces deux notions pour la classe *Juste* (il n'y a pas de fausses notes dans la mélodie). La précision serait alors la capacité du modèle à ne détecter que de bonnes mélodies. Quant au rappel se serait la capacité du modèle à détecter toutes les bonnes mélodies. Les formules seraient alors les suivantes :

$$precision = \frac{\text{Bonnes mélodies réelles parmi les prédites}}{\text{Total de bonnes mélodies prédites}}$$

et,

$$rappel = \frac{\text{Bonnes mélodies réelles parmi les prédites}}{\text{Total de bonnes mélodies réelles}}$$

Enfin, pour n'avoir qu'un critère, il existe le *F-score* donné par la formule :

$$F = 2 \cdot \frac{precision \cdot rappel}{precision + rappel}$$

Il n'a malheureusement aucune signification. Si deux paramétrages donnent sensiblement le même taux d'erreur et le même *F-score*, comment les départager ? La précision et le rappel vont les départager. Mais comment ? Il nous faut choisir. Voulons-nous une meilleure précision ou un meilleur rappel ? Autrement dit, voulons-nous être sûr que la prédiction de mauvaises mélodies soit garantie, ou bien voulons-nous détecter toutes les mauvaises mélodies ?
Il nous semble que nous préférions que le modèle détecte toutes les fausses notes, quitte à ce qu'il fasse quelques erreurs parmi ses prédictions. Donc à deux paramétrages équivalent, nous choisirions celui avec un rappel plus fort. Bien entendu, la précision doit aussi atteindre une valeur acceptable.

Convergence

Ces critères de performances ne peuvent suffire pour s'assurer de la validité du modèle. Dans l'introduction sur les réseaux de neurones, nous avons écrit que l'aboutissement au résultat souhaité requiert la convergence de la fonction d'erreur. Nous allons donc également mesurer cette convergence.

6.1.2 Jeux de données

Avant d'évaluer le modèle, nous allons le faire apprendre, il faut donc un ensemble de données d'apprentissage. Les deux jeux de données d'apprentissage que nous avons utilisées sont les suivants :

- Un corpus d'improvisations de Charlie Parker (Omnibook) composé de cinquante fichiers au format MusicXML. Il y a qu'un style musical présent dans ce corpus ;
- Un corpus composé de plus de 1600 morceaux au format MusicXML avec des styles divers et variés. Ce corpus a été crée à l'aide de [23] permettant l'extraction de mélodie à partir un fichier au format MIDI.

Chaque corpus de données a été décomposé en trois catégories. Une première catégorie constitue le jeu d'apprentissage. C'est sur ces données que le modèle apprendra. La deuxième constitue le jeu de validation. Il permet de s'assurer qu'il n'y a pas sur-apprentissage. C'est-à-dire que le modèle n'apprend pas par cœur les données d'entraînement. Enfin, la troisième catégorie forme le jeu de test. Elle représente les données réelles sur lesquelles le modèle devra faire ses prédictions. Un fichier musical ne peut appartenir qu'à une seul catégorie. Le pourcentage de composition de chaque catégorie par rapport aux corpus initial est le suivant, 60% pour le jeu d'apprentissage, 25% pour le jeu de validation et 25% pour le jeu de test.

6.1.3 Résultats

Les résultats qui sont présentés dans cette section sont issus de plusieurs tests afin d'obtenir une paramétrage idéal. Les résultats montrés sont ceux ayant obtenus les meilleurs résultats aux vus des critères cités précédement.

Omnibook de Charlie Parker

Pour le premier jeux de données, le paramétrage et les résultats obtenus pour un tel paramétrage sont les suivants.

Configurations	**Valeurs**
Coef. apprentissage	4e-5
Batch	2000
Itérations	30000
Nombre de notes	9
Tatum	60
Optimiseur	Descente de gradient
Machine de calcul	1 CPU Intel Xeon X3440, 4 cores/CPU, 16GB RAM, 298GB HDD

Table 6.2 – *Liste de la configuration faite pour le premier jeu de données.*

		Classe prédite	
		Faux	Juste
Classe Réelle	Faux	4669	1475
	Juste	727	5417

Table 6.3 – *Matrice de confusion obtenu sur un ensemble de test pour le deuxième jeu de données.*

	Classe Faux	Classe Juste
Précision	0,8653	0,7860
Rappel	0,7599	0,8817
F-score	0,8092	0,8311
Taux d'erreur	0,8208	

Table 6.4 – *Critères d'évaluations pour la matrice de confusion présente dans le Tableau 6.3*

La Figure 6.1 montre l'erreur moyenne commise au cours des itérations. Nous remarquons que l'erreur converge et diminue. Il y a bien apprentissage.
Pour les critères d'évaluation, nous obtenons de bons résultats. Le taux d'erreur est autour des 82%. Cette valeur est supérieur au hasard (comme nous avons deux classes, le hasard aurait fait un taux d'erreur de 0,5). Le rappel est inférieur à la précision pour la classe *Faux* (présence d'une fausse note). Nous avions dit que nous aurions préféré un rappel plus fort qu'une précision. Néanmoins, au vu des valeurs obtenues, nous sommes satisfaits de tels résultats. Nous notons que le modèle détecte légèrement mieux les bonnes mélodies que celle contenant une fausse note.

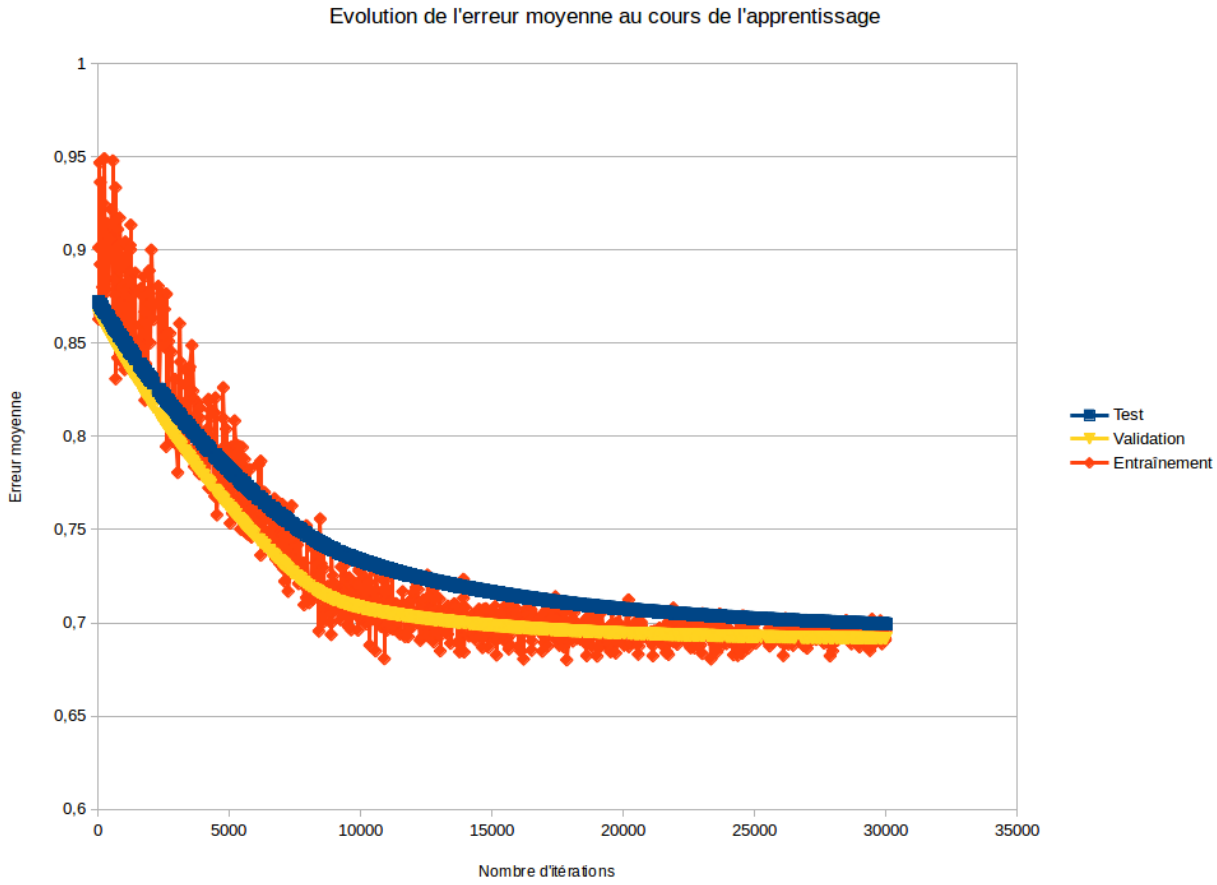

Figure 6.1 – *Graphique montrant l'évolution de la convergence de la fonction d'erreur pour le premier jeu de données.*

Jeux de données hétérogènes

Pour le deuxième jeu de données, le paramétrage et les résultats obtenus pour un tel paramétrage sont les suivants.

Configurations	**Valeurs**
Coef. apprentissage	5e-5
Batch	50
Itérations	100000
Nombre de notes	9
Tatum	60
Optimiseur	Descente de gradient
Machine de calcul	1 CPU Intel Xeon X3440, 4 cores/CPU, 16GB RAM, 298GB HDD

Table 6.5 – *Liste de la configuration faite pour le deuxième jeu de données.*

		Classe prédite	
		Faux	Juste
Classe Réelle	Faux	943	961
	Juste	974	1122

Table 6.6 – *Matrice de confusion obtenu sur un ensemble de test pour le deuxième jeu de données.*

	Classe Faux	Classe Juste
Précision	0,4919	0,5386
Rappel	0,4952	0,5353
F-score	0,4935	0,5369
Taux d'erreur	0,5162	

Table 6.7 – *Critères d'évaluations pour la matrice de confusion présente dans le Tableau 6.6*

La Figure 6.2 montre l'erreur moyenne commise au cours des itérations. Nous constatons que l'erreur moyenne diminue et qu'il y a convergence au cours de l'apprentissage. C'est signe d'apprentissage.
En revanche, pour les critères présentés dans le Tableau 6.7, les résultats ne sont pas ceux souhaités. Notre modèle, avec un tel jeu d'apprentissage, détecte légèrement mieux les mélodies sans fausses notes. En effet la précision, le rappel et F-score pour la classe *Juste* sont supérieurs à ceux de la classe *Faux*. Tous les critères pour la détection de fausses notes sont en dessous de 0,5. Malgré tout, le taux d'erreur est très légèrement mieux que le hasard.

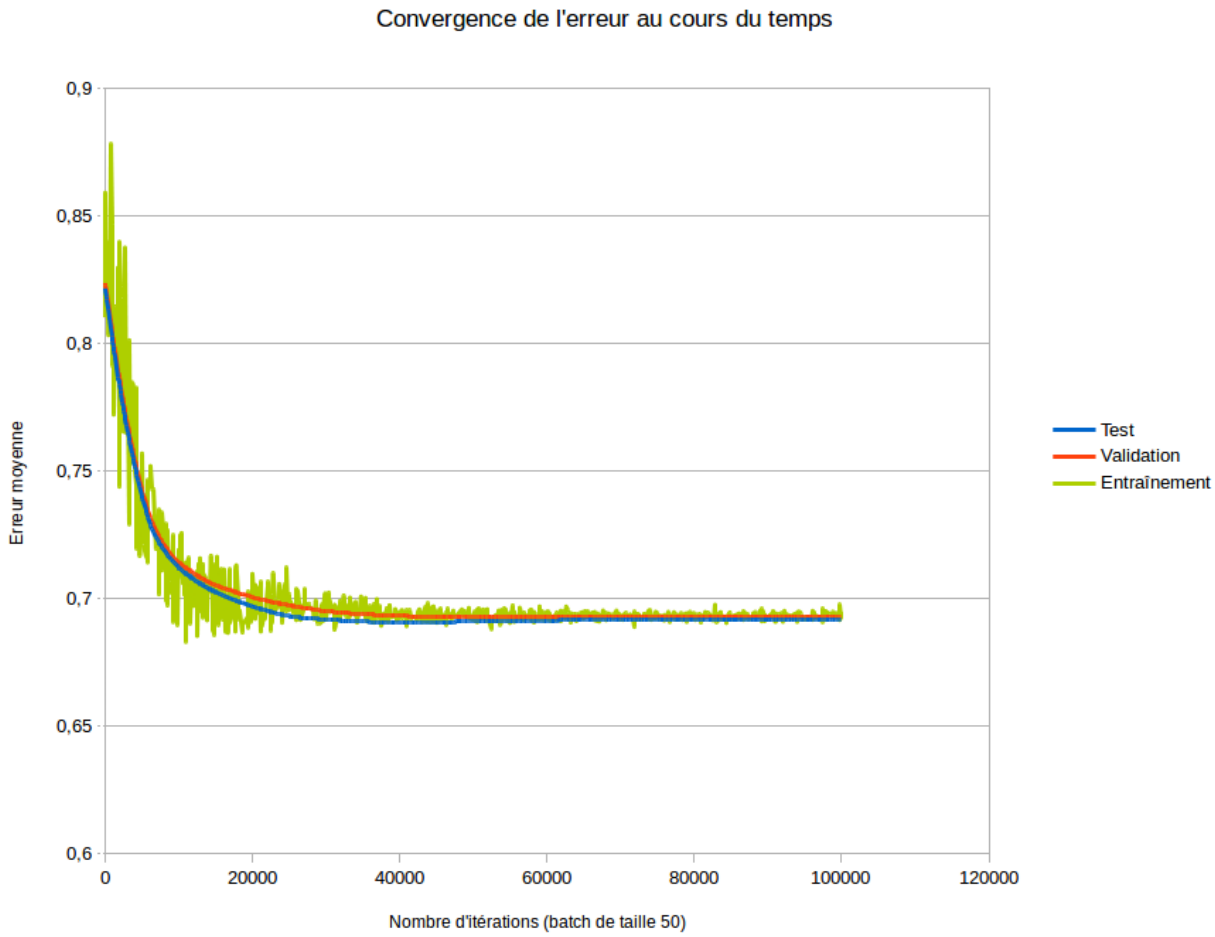

Figure 6.2 – *Graphique montrant l'évolution de la convergence de la fonction d'erreur pour le deuxième jeu de données.*

Apprentissage par renforcement

Nous n'avons pour le moment que modélisé l'apprentissage par renforcement. Nous n'avons donc pas de résultats à présenter.

6.1.4 Discussion et perspective

Discussion

Nous notons un écart très important dans les critères d'évaluation entre les deux jeux de données. Cet écart peut être dû à plusieurs raisons. En voici quelques unes :

- le modèle n'est capable d'apprendre qu'un style musical très précis (malheureusement ce n'est pas notre objectif) ;
- réduire le contexte à la seule mélodie ne permet pas au modèle d'apprendre sur un corpus diversifié ;
- le deuxième jeu de données est peut être de mauvaise qualité malgré un bref tri automatique. En effet, nous avons enlevé tous les morceaux n'ayant pas au moins cinq notes distinctes et/ou dont les notes ne sont pas dans les cinq octaves voulues ;
- Dû à l'hétérogénéité du deuxième corpus, l'optimisation du paramétrage demande plus de temps ;
- le choix de modélisation des notes ne permet pas d'aller plus loin dans la détection. Il est possible que la distance introduite en découpant octave et hauteur mène à de tels résultats.

Perspective

Suite à cette liste de raisons possibles, nous allons soumettre quelques piste de recherche afin d'améliorer le modèle. La première est tout simplement de changer l'intégralité de l'architecture et d'en tester une autre, afin de pouvoir comparer différents modèle de détection de fausses notes. Nous n'avons testé le modèle que sur un contexte réduit à la mélodie. Il est possible d'étendre le contexte à l'harmonie et à d'autres dimensions musicales (dynamique du jeu, ...). Le modèle pourrait alors apprendre à distinguer les styles musicaux et ainsi cela augmenterait ses performances. Nous pouvons proposer un corpus aussi diversifié que le deuxième corpus, mais avec la garantie que celui-là sera jugé de bonne qualité auprès de musiciens professionnels.
Essayer plus de paramétrages pour le deuxième corpus. La quantité de données étant plus grande, il est possible que sur l'ensemble de tests faits, nous n'ayons toujours trouvé d'un optima local. Ce qui expliquerait la convergence et les résultats pour le second corpus. En somme, nous pourrions changer la méthode d'apprentissage. C'est-à-dire, qu'au lieu de présenter une mélodie avec une fausse note nous pourrions modifier l'ensemble des notes aléatoirement pour réduire de nombre à une seule note modifiée.
Par ailleurs, nous pourrions essayer de changer la représentation des notes. Par exemple, nous pourrions essayer avec un encodage one-hot des 88 notes usuelles. Nous pourrions aussi augmenter ou diminuer la valeurs du tatum.
Nous pourrions également essayer avec non plus un seul indice d'attaque, mais une forme décroissance de l'attaque. Par exemple, nous aurions un tableau dont le premier indice est l'attaque comme nous l'avons décrite et ses voisins (indice précédent et suivant) aurait des valeurs, non plus à zéros, mais à 0,33 par exemple.

Pour l'apprentissage par renforcement, nous avons proposé un modèle déterministe. Notre

modèle actuel ne prend pas du tout en compte ce que jouent les musiciens. Il n'y a donc aucune incertitude : nous connaissons parfaitement ce que nous allons jouer (dans la mesure où on est capable de lister toutes les séquences de n notes suivantes possibles) et l'espérance est égale à la récompense elle-même. Il serait plus intéressant si notre modèle de récompense prenait en compte ce que jouent les autres et nous ne savons pas ce que les autres musiciens vont jouer. Dans ce cas, il faut prendre l'espérance sur les séquences d'accords ou de rythmes que les autres musiciens vont potentiellement jouer. Ces séquences peuvent être représentées par un oracle avec des probabilités. Un lissage additif peut être effectué pour prendre en compte le fait que les autres musiciens peuvent jouer des séquences qui sortent de l'oracle. Ces probabilités ne dépendent pas explicitement de nos choix de transitions entre états, mais elles en dépendent implicitement car nos choix vont influencer le jeu des autres (influence mutuelle). Pour faire véritablement de l'apprentissage par renforcement (non un calcul de l'ensemble des possibilités), il faut à la fois :

- avoir un oracle harmonique/rythmique sur ce que jouent les autres musiciens ;
- modifier le BiLSTM pour prendre en compte ce que jouent les autres musiciens ;
- calculer l'espérance de la sortie du BiLSTM pour toutes les séquences possibles de l'oracle mélodique par rapport aux séquences possibles de l'oracle harmonique/rythmique afin de naviguer dans l'oracle mélodique

7 Conclusion

Mon travail en collaboration directe avec une laboratoire de recherche m'a permis d'aborder les problématiques liées au domaine de la recherche dans un contexte artistique.

Dans premier temps, j'ai dû appréhender le modèle actuel et le contexte dans lequel s'inscrit cette recherche. J'ai évoqué les limites d'un tel modèle, à savoir des limites rythmiques. À la suite de cela, j'ai constaté le besoin de définir la notion de fausse note sur laquelle la recherche devra s'inscrire. Cela m'a également permis de montrer l'existence de modèles d'apprentissage automatique adaptés au contexte musical.

Toute ma réflexion s'appuie sur les écrits scientifiques, ce qui m'a permis de définir un modèle afin de répondre aux problématiques. L'absence de recherches dans un tel contexte m'a laissé bon nombre de choix quant à la représentation du modèle. Par la suite, j'ai été confronté à la réalisation technique. Notamment, j'ai dû faire face à la recherche d'un paramétrage idéal. J'ai eu la chance de pouvoir être en complète autonomie durant tout ce projet avec le soutien de mes encadrants.

Ce projet a abouti à un premier modèle pour la détection de fausse. L'apprentissage par renforcement n'est qu'au stade de la modélisation, aucune réalisation n'a encore été faite. Un ensemble de tests a permis d'optimiser les modèles pour ainsi obtenir de meilleurs résultats. J'ai adopté une démarche scientifique pour analyser les résultats. J'ai ensuite soulevé des pistes de recherches pour découvrir les causes de ces résultats. Grâce à la mise en œuvre des outils d'apprentissage, nous pourrions apprendre au système à ne plus reproduire ses erreurs et constater leurs impacts dans la génération d'improvisation musicale.

Plusieurs propositions d'amélioration sont envisageables. Un autre modèle ou une autre représentation en sont chacun une. Il est en effet important de comparer notre modèle à d'autre modèles.

Cette contribution au projet ANR DYCI2 permettra de renforcer les travaux de recherche déjà effectués. Elle offrira également une plus grande visibilité quant à l'apprentissage automatique dans le domaine musicale et notamment dans la création d'improvisation.

Ce projet m'a fait découvrir le monde de la recherche, auquel j'affectionne un grand intérêt depuis. J'ai donc décidé à la suite de ce projet de poursuivre dans une thèse.

Bibliographie / Webographie

[1] F. Liang, "Bachbot : Automatic composition in the style of bach chorales," Août 2016.

[2] J. Nika and et al., "Omax." http://omax.ircam.fr/, 2015.

[3] G. Assayag and G. Bloch, "Navigating the Oracle : a Heuristic Approach," in *Proceedings of the International Computer Music Conference*, pp. 405–412, 2007.

[4] K. Déguernel, E. Vincent, and G. Assayag, "Using multidimensional sequances for improvisation in the OMax paradigm," 2016.

[5] J. Nika and M. Chemillier, "Improvisation musicale homme-machine guidée par un scénario temporel," *Technique et science informatiques*, vol. 7, pp. 651–684, Août 2014.

[6] L. Deng and D. Yu, *Deep Learning, Methods and Applications*. NOW Publishers, May 2014.

[7] G. Assayag, G. Bloch, and M. Chemillier, "Improvisation et réinjection stylistiques," in *Le feed-back dans la création musicale contemporaine - Rencontres musicales pluri-disciplinaires*, (Lyon, France), pp. 1–1, Mar. 2006. cote interne IRCAM : Assayag06c.

[8] B. Levy, "Omax, the software improviser," *IRCAM*, 2004-2012.

[9] A. de Cheveigné and H. Kawahara, "Yin, a fundamental frequency estimator for speech and music," *Acoustical Society of America*, 2002.

[10] C. Allauzen, M. Crochemore, and M. Raffinot, "Factor Oracle : A New structure for pattern matching," in *SOFSEM'99, Theory and Practice of Informatics*, pp. 291–306, 1999.

[11] "Apprentissage du style musical et interaction sur deux échelles temporelles," 2003.

[12] K. Déguernel, E. Vincent, and G. Assayag, "Improvisation musicale multidimensionnelle dans le paradigme OMax." Journées Jeunes Chercheurs en Acoustique, Audition et Signal, Nov. 2016. Poster.

[13] M. Crispell, "Elements of improvisation," in *Arcana : Musicians on Music* (J. Zorn, ed.), pp. 190–192, 2000.

[14] C. Colah, "Understanding LSTM Networks." http://colah.github.io/posts/2015-08-Understanding-LSTMs/, August 27, 2015.

[15] D. Eck and J. Lapalme, "Learning musical structure directly from sequences of music," Tech. Rep. 1300, Université de Montréal DIRO, http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.94.6654&rep=rep1&type=pdf, 2008.

[16] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget : Continual prediction with lstm.," *Neural Computation*, 2000.

[17] J. Schmidhuber, F. A. Gers, and D. Eck, "Learning nonregular languages : A comparison of simple recurrent networks and lstm," *Neural Computation*, 2002.

[18] R. S. Sutton and A. G. Barto, *Reinforcement Learning : An introduction. Second Edition*. MIT Press, 2012.

[19] G. Bickerman, S. Bosley, P. Swire, and R. M. Keller, "Learning to Create Jazz Melodies Using Deep Belief Nets," in *Proceedings Of The International Conference on Computational Creativity*, 2010.

[20] M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks," *Signal Processing, IEEE Transactions*, 1997.

[21] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout : A Simple Way to Prevent Neural Networks from Overfitting," in *Journal of Machine Learning Research 15*, June 2014.

[22] R. Bolze and et al., "Grid'5000 : A large scale and highly reconfigurable experimental grid testbed," in *International Journal of High Performance Computing Applications*, November 2006.

[23] D. Rizo, P. J. P. de León, C. Pérez-Sancho, A. Pertusa, and J. M. Iñesta, "A pattern recognition approach for melody track selection in midi files," in *Proc. of the 7th Int. Symp. on Music Information Retrieval ISMIR 2006* (T. A. Dannenberg R., Lemström K., ed.), (Victoria, Canada), pp. 61–66, 2006.

[24] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative Adversial Nets," 10 Juin 2014.

[25] W. Zaremba, I. Sutskever, and O. Vinyals, "Recurrent Neural Network Regularization," in *International Conference on Learning Representations*, 19 Feb 2015.

[26] J. Ho and S. Ermon, "Generative adversial imitation learning," *CoRR*, vol. abs/1606.03476, 10 June 2016.

[27] W. Chan, N. Jaitly, Q. V. Le, and O. Vinyals, "Listen, attend and spell," *CoRR*, vol. abs/1508.01211, 2015.

[28] C. Olah and S. Carter, "Attention and augmented recurrent neural networks," *Distill*, 2016.

[29] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *NIPS Deep Learning Workshop*, December 2013.

[30] T. Simonite, "Ibm builds biggest data drive ever," *Technology Review*, August 2011.

Liste des illustrations

Liste des tableaux

Liste des Algorithmes

Annexes

A Démonstration du langage reconnu par l'automate présenté en Figure 2.3

Rappel : Soit un langage L donné, A et b des expressions rationnels. Nous avons

$$L = AL + b \Leftrightarrow L = A^*b$$

Nous posons $\Sigma = (Do + Re + Mi + Fa + Sol + La + Si)$, et ϵ est le mot vide. En définissant les expressions reconnues par chaque état de l'automate, nous avons :

$$\begin{cases} L_0 &= DoL_1 \\ L_1 &= \Sigma L_1 + DoL_2 \\ L_2 &= \epsilon \end{cases}$$

$$\Leftrightarrow \begin{cases} L_0 &= DoL_1 \\ L_1 &= \Sigma L_1 + Do \\ L_2 &= \epsilon \end{cases}$$

$$\Leftrightarrow \begin{cases} L_0 &= DoL_1 \\ L_1 &= \Sigma^* Do \\ L_2 &= \epsilon \end{cases}$$

$$\Leftrightarrow \begin{cases} L_0 &= Do\Sigma^* Do \\ L_1 &= \Sigma^* Do \\ L_2 &= \epsilon \end{cases}$$

Finalement le langage reconnu est :

$$L_0 = Do\Sigma^* Do$$

C'est bien les mélodies commençant par Do et finissant par Do.

B Démonstration de la formule de rétro-propagation du gradient dans le cas du neurone présente en partie 4.1.3

Nous avons $\forall i \in \{1, 2\}$:

$$\frac{\partial E}{\partial w_i}(x_i) = \frac{\partial E}{\partial y} \frac{\partial y}{\partial w_i}(x_i)$$

Avec $E = \frac{1}{2}(y_t - y)^2$ l'erreur quadratique moyenne, et $y = f(xW^T + b) = x_1 w_1 + x_2 w_2 + b$. La fonction d'activation est $f(x) = x$. Nous avons :

$$\begin{aligned} \frac{\partial E}{\partial w_i}(x_i) &= \frac{\partial}{\partial y}[\frac{1}{2}(y_t - y)^2] \frac{\partial y}{\partial w_i}(x_i) \\ &= -(y_t - y)x_i \end{aligned}$$

CQFD.

NB : Le terme $\frac{\partial y}{\partial w_i}$ fait bien apparaître la dérivé de la fonction d'activation. C'est pourquoi la fonction d'activation doit notamment être de dérivée facile à exprimer. Par exemple,

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \text{ a pour dérivée } 1 - tanh^2(x)$$

De même,

$$sigm(x) = \frac{1}{1 + e^{-x}} \text{ a pour dérivée } sigm(x)(1 - sigm(x))$$

Résumé

Le projet ANR DYCI2 vise à explorer les interactions entre l'homme et des agents artificiels dans le domaine de l'improvisation musicale. L'apprentissage interactif, l'un des domaines de recherche du projet, veut mettre en avant des modèles capables de capturer les structures musicales. Toutes les recherches se basent sur le paradigme OMax, un système automatique d'improvisation. Malheureusement, ce système génère encore des fausses notes. Est-il possible d'apprendre au système à ne plus commettre de telles erreurs ?

Pour répondre à cette problématique, ce mémoire propose une méthode pour classifier les mélodies en deux catégories : celles qui ont au moins une fausse note et celles qui n'en ont aucune. Une telle classification devra permettre d'améliorer le système actuel à l'aide de l'apprentissage par renforcement. Après avoir défini quelques notions musicales et expliqué ce que sont les LSTMs, nous proposons un modèle de réseau de neurones pour la classification. Nous proposons également un procédé pour encoder les notes musicales. Nous utilisons le Deep Q-Learning en tant qu'algorithme d'apprentissage par renforcement. L'évaluation du réseau neuronal repose sur les critères classiques. L'évaluation de l'amélioration du système est basée sur l'écoute des mélodies générées. Enfin, nous discutons des méthodes utilisées.

Mots-clés : Réseau de neurones, Apprentissage par renforcement, Musique et Improvisation

Abstract

The DYCI2 ANR project aims to explore interactions between humans and artificial agents in the field of music improvisation. Interactive learning, one of the main's project research area, proposes models able to detect musical structures. All of these research are based on the OMax paradigm, an automatic music improvisation system. Unfortunately, such a system still creates false notes. Is it possible to have the system learned not to do those mistakes again ?

To answer that question, we propose a melodic classification into two classes : the ones which has at least one wrong note, and the others which have no ones. This classification allows us to enhance the system by using an reinforcement learning algorithm. After introducing some musical words and explaining what are LSTMs, we present our neural network model which is going to classify melodies. We also propose a musical encoding scheme. We use Deep Q-Learning as reinforcement learning algorithm to improve the current system. We evaluate our neural network model with classical criteria. The final enhancement will be evaluate by listening to the melodies. At last, we discuss about our strategies.

Keywords : Neural Networks, Reinforcement Learning, Music and Improvisation