Université de Montréal

Modélisation de l'interprétation des pianistes
&
Applications d'auto-encodeurs sur des modèles temporels

par

Stanislas Lauly

Département d'informatique et de recherche opérationnelle

Faculté des arts et des sciences

Mémoire présenté à la faculté des arts et des sciences en vue de l'obtention du grade de maître ès sciences (M.Sc) en informatique

Avril 2010

Université de Montréal
Faculté des arts et des sciences

ce mémoire intitulé:

Modélisation de l'interprétation des pianistes

&

Applications d'auto-encodeurs sur des modèles temporels

présenté par :
Stanislas Lauly

À été évalué par un jury composé des personnes suivantes :

Yoshua Bengio
président-rapporteur

Douglas Eck
directeur de recherche

Pascal Vincent
codirecteur de recherche

Pierre Mckenzie
Membre du jury

Résumé

Ce mémoire traite d'abord du problème de la modélisation de l'interprétation des pianistes à l'aide de l'apprentissage machine. Il s'occupe ensuite de présenter de nouveaux modèles temporels qui utilisent des auto-encodeurs pour améliorer l'apprentissage de séquences.

Dans un premier temps, nous présentons le travail préalablement fait dans le domaine de la modélisation de l'expressivité musicale, notamment les modèles statistiques du professeur Widmer. Nous parlons ensuite de notre ensemble de données, unique au monde, qu'il a été nécessaire de créer pour accomplir notre tâche. Cet ensemble est composé de 13 pianistes différents enregistrés sur le fameux piano Bösendorfer 290SE. Enfin, nous expliquons en détail les résultats de l'apprentissage de réseaux de neurones et de réseaux de neurones récurrents. Ceux-ci sont appliqués sur les données mentionnées pour apprendre les variations expressives propres à un style de musique.

Dans un deuxième temps, ce mémoire aborde la découverte de modèles statistiques expérimentaux qui impliquent l'utilisation d'auto-encodeurs sur des réseaux de neurones récurrents. Pour pouvoir tester la limite de leur capacité d'apprentissage, nous utilisons deux ensembles de données artificielles développées à l'Université de Toronto [19].

Mots clés: apprentissage machine, performance expressive, réseau de neurones récurrent, musique, auto-encodeur, modèle temporel.

Abstract

This thesis addresses the problem of modeling pianists' interpretations using machine learning, and presents new models that use temporal auto-encoders to improve their learning for sequences.

We present previous work in the field of modeling musical expression, including Professor Widmer's statistical models. We then discuss our unique dataset created specifically for our task. This dataset is composed of 13 different pianists recorded on the famous Bösendorfer 290SE piano. Finally, we present the learning results of neural networks and recurrent neural networks in detail. These algorithms are applied to the dataset to learn expressive variations specific to a style of music.

We also present novel statistical models involving the use of auto-encoders in recurrent neural networks. To test the limits of these algorithms' ability to learn, we use two artificial datasets developed at the University of Toronto [19].

Keywords: machine learning, expressive timing, expressive performance, recurrent neural networks, music, auto-encoders, temporal models.

Contents

List of Tables

List of Figures

Remerciements

J'aimerais particulièrement remercier mes parents, Christiane et Jean-Paul, qui m'ont aidé et soutenu tout au long de ce mémoire et sans qui tout ceci aurait été complètement impossible.

Merci à Sean Wood pour toutes les longues et intéressantes discussions qui m'ont aidé durant la maîtrise et aussi pour le reste.

Merci à Hugo Larochelle, Alexandre Lacoste, James Bergstra et Pascal Lamblin pour les longues explications qui m'ont été si utiles.

Merci à Mélissa De Cristofaro et Mathieu Kastlé pour le long parcours universitaire que nous avons vécu ensemble.

Merci à François Maillet Simon Lemieux, Philippe Hamel Thierry Bertin-Mahieux, Norman Casagrande, Jasmin Lapalme, Tarik Daouda et Lysiane Bouchard pour la vie au labo et les bons moments.

Finalement, je voudrais remercier mon directeur et mon co-directeur, Douglas Eck et Pascal Vincent, pour m'avoir donné la possibilité de réaliser un rêve.

1 Introduction

La perception musicale est un phénomène très complexe qui demande pourtant relativement peu d'effort à un être humain. Ce qui est plus étonnant, c'est que nous distinguons avec une grande facilité les genres (ou styles) musicaux comme le rock, le jazz ou le rap. Avec un peu d'entraînement, on peut même reconnaître le style d'un musicien en particulier. Les variations qui permettent à une musique d'appartenir à un genre en particulier peuvent être représentées en partie par la notation musicale, mais pas totalement. En effet, une grosse part des variations qui permettent de reconnaître le genre d'une musique et qui la rendent appréciable pour les individus ne sont apprises que par l'exemple et sont difficilement quantifiables.

Lorsqu'on fait jouer une partition de musique par un ordinateur, la performance obtenue est trop «parfaite». Chaque note est jouée exactement au moment où on l'attend et aucune erreur n'est commise. La performance est si «parfaite» qu'elle en devient ennuyante. À l'opposé, si on demande à un pianiste de jouer cette même pièce, la performance obtenue est imparfaite. Les notes sont déviées dans le temps et sont jouées avec des intensités différentes. Mais ces variations créent de l'attente et de l'anticipation dans l'auditoire, ce qui donne de l'émotion à la pièce et la rend beaucoup plus intéressante pour une oreille humaine.

Ce mémoire pose d'abord la question suivante: Pouvons-nous, à l'aide de l'apprentissage machine, apprendre la logique qui dicte les déviations de temps expressives propres à un style de musique en particulier, pour ensuite pouvoir appliquer cette logique sur une nouvelle partition? En d'autres mots, pouvons-nous créer un pianiste virtuel capable de comprendre un style musical pour ensuite l'appliquer sur n'importe quelle pièce?

Pour y répondre, nous avons décidé de nous concentrer sur un seul style de musique, c'est-à-dire les valses. Pour apprendre les déviations de temps propres aux valses, nous présenterons certains modèles en apprentissage machine.

Cette recherche sur l'interprétaton de la musique nous a poussés à étudier les modèles statistiques temporels. Étant donné que nous faisons partie du laboratoire LISA (Laboratoire d'Informatique des Systèmes Adaptatifs), nous nous sommes familiarisés avec une percée récente dans le monde de l'apprentissage machine. Il s'agit des réseaux profonds [10] (*«Deep belief nets»*) qui consistent en un apprentissage par niveau en se servant de

RBM (*Restricted Boltzmann Machine*) ou d'auto-encodeurs. Cette nouvelle approche a permis d'obtenir des résultats encore inégalés dans certains domaines comme la vision.

Ce mémoire pose ensuite une deuxième question: Est-il possible de créer de meilleurs modèles temporels en s'inspirant des réseaux profonds et en utilisant des auto-encodeurs?

Pour répondre à cette question, les données sur la musique ne suffisent pas. Nous utilisons également des données artificielles avec une complexité bien définie qui permet d'observer les limites de capacité d'apprentissage de différents modèles.

1.1 Motivation pour l'apprentissage des performances musicales

Le sujet abordé ici est délicat. En effet, l'être humain a de la difficulté à accepter l'idée de se faire remplacer par une machine. On s'est habitué à ce que des tâches physiques puissent être effectuées mécaniquement, mais on n'accepte encore mal l'intrusion de la machine dans le domaine artistique, considéré comme étant purement humain. La peur est de perdre la créativité et l'imagination. En ce qui nous concerne, nous pensons au contraire qu'une telle intrusion dans le monde de la musique peut permettre de créer des outils qui assisteront l'artiste dans sa démarche créative et qui lui permettront de dépasser ses limites.

Dans notre cas, l'outil est le pianiste virtuel qui pourrait servir, par exemple, d'accompagnateur pour le musicien solitaire ou d'interprète pour le compositeur. En effet, ce dernier n'a pas toujours facilement accès à un pianiste qui peut jouer et donner de la vie à ses compositions. De plus, cette recherche pourrait mener un jour à la création d'un orchestre virtuel, ce qui serait encore plus intéressant pour les musiciens.

1.2 Difficulté pour l'apprentissage des performances musicales

Cette tâche de grande envergure présente un certain nombre de difficultés à surmonter, notamment le problème de l'ensemble des données d'apprentissage.

Les performances musicales sont généralement sous format audio, ce qui rend presque impossible l'extraction des informations qui sont utiles. Le format MIDI est le plus simple et le plus adéquat, car il réussit à représenter toute une performance avec seulement quelques variables, comme la durée des notes jouées et leur force. Cependant, il existe peu de performances humaines en format MIDI pouvant permettre de créer un ensemble

de données d'apprentissage assez gros pour bien représenter un genre musical.

D'une part, il faut construire notre propre ensemble de performances, ce qui implique le choix d'un style de musique, ni trop compliqué, ni trop simple. Il faut ensuite identifier un certain nombre de pièces de musique et trouver plusieurs pianistes différents pour jouer ces pièces. Finalement, il faut enregistrer les performances et s'assurer qu'elles sont bien interprétées, ce qui exige beaucoup de temps.

D'autre part, il faut choisir les variables qui doivent être apprises par les modèles et qui permettent de représenter au mieux les variations générées par l'interprète d'une performance. Il faut également extraire ces variations en comparant la performance avec la partition. Ce travail n'est pas facile, car même s'il existe des algorithmes de comparaisons, un certain travail manuel est nécessaire pour pouvoir transformer les performances en données d'apprentissage pour les modèles statistiques.

1.3 Motivation pour la recherche de modèles temporels plus puissants

La motivation initiale qui nous a poussés à vouloir des modèles temporels plus puissants est la génération de musique. On parle ici de génération de pièces de musique, ce qu'il ne faut pas confondre avec la performance musicale d'un pianiste virtuel. En effet, la difficulté pour les modèles est de comprendre que la musique est composée de répétition mélodique (figure 1), ce qui est pourtant facile pour l'être humain. Le problème c'est que cette tâche est encore beaucoup trop difficile pour n'importe quel modèle statistique existant à ce jour. Les dépendances entre les notes d'une pièce de musique sont souvent à trop long terme (perte de l'influence du gradient dans le temps[29]) et la complexité est trop grande, car beaucoup de variables changent entre les répétitions des différentes mélodies.

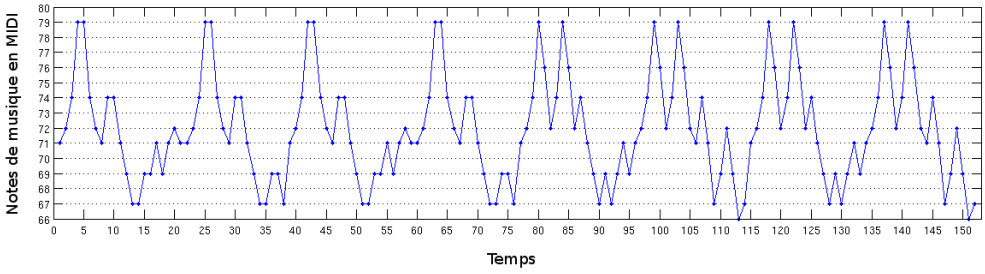

Figure 1: Contour d'une mélodie représentant l'aspect répétitif de la musique. L'axe des X représente les notes de musique en format MIDI et l'axe des Y indique le temps.

Ces données ne sont donc pas appropriées pour comparer et tester les capacités des modèles. Nous avons donc senti la nécessité de nous orienter vers des données plus simples mais semblables, qui demandent aux modèles d'apprendre à répéter.

Nous avons été inspirés par les bonnes performances des auto-encodeurs dans le pré-entraînement de réseaux de neurones. Cela nous a amenés à croire que l'on pouvait peut-être trouver une façon d'appliquer les auto-encodeurs sur des réseaux de neurones récurrents de façon à améliorer l'apprentissage.

1.4 Difficulté pour la recherche de modèles temporels plus puissants

La grande difficulté de cette recherche est, comme le lecteur s'en doute, de trouver la bonne façon d'utiliser les auto-encodeurs, si elle existe. Un très grand nombre de possibilités d'utilisation s'offrent à nous et il faut réussir à choisir celles qui sont les plus susceptibles d'améliorer l'apprentissage du modèle, ce qui peut s'avérer difficile.

1.5 Aperçu des chapitres

Dans ce mémoire, après ce premier chapitre d'introduction, le chapitre 2 permet de mieux comprendre où en est rendue la recherche dans la modélisation des performances musicales. À ce titre, nous parlons des travaux préalables qui ont été effectués dans le domaine.

On passe ensuite au chapitre 3. Celui-ci explique les modèles statistiques de base, les modèles utilisés pour l'apprentissage des performances expressives et les modèles temporels expérimentaux que nous avons créés.

Le chapitre 4 continue avec l'observation et l'analyse des capacités d'apprentissage de plusieurs modèles temporels expérimentaux qui utilisent des auto-encodeurs. Ces modèles sont appliqués sur deux ensembles de données artificielles différents.

Le chapitre 5 présente tout ce qu'il faut savoir sur le format des données de performances expressives et sur leurs créations. On observe et analyse ensuite les résultats de l'apprentissage des deux modèles utilisés pour cette tâche.

On finit naturellement par un chapitre de conclusion qui récapitule les faits et qui propose de nouvelles idées de recherche.

2 Travaux préalables sur les performances expressives musicales

L'interprétation d'une pièce de musique, réalisée par un pianiste professionnel ou par un amateur, est un phénomène humain très complexe, autant au niveau des mouvements physiques du musicien qu'au niveau de la structure du tempo, de la déviation de temps et de la variation de l'intensité. Le Pr. Widmer, un des chercheurs les plus respectés dans le domaine de la performance expressive, a travaillé sur sa modélisation. Cette section s'appuie en grande partie sur la recherche faite par Gerhard Widmer et Werner Goebl [26].

2.1 Différentes approches face au problème de la déviation expressive de temps

Beaucoup de recherches se sont faites au sujet de l'expressivité des performances depuis les 100 dernières années et l'article écrit par Gabrielsson [6] en fait un excellent survol. Plutôt que d'énumérer tous les modèles qui existent, nous allons nous concentrer sur les plus connus parmi ceux qui peuvent générer des performances à partir de nouvelles partitions. Le but de cette section est d'expliquer différentes approches sur la modélisation de la logique des performances expressives en musique. Nous commencerons par l'approche la plus populaire et la plus citée de toutes, c'est-à-dire le modèle KTH.

2.1.1 KTH (AI)

Ce modèle bien connu est basé sur un système de règles et a été développé, sur plus de vingt ans de recherche, par «*The Royal Institute of Technology*» à Stockholm en Suède. C'est un exemple parfait d'intelligence artificielle classique; on parle ici de système expert. Ce modèle est expliqué en profondeur dans l'article de Friberg [4].

Le modèle KTH est composé d'un ensemble de règles qui permet de prédire pour une pièce les déviations de temps, les intensités et les articulations qui doivent être appliquées, en se basant sur le contexte musical local.

La plupart des règles agissent sur un contexte très local affectant une note individuellement, mais il y a aussi des règles qui s'appliquent à toute une phrase. Chaque règle

est composée d'un nombre de paramètres qui varient. Le modèle a été développé en utilisant une approche qui s'appelle «*analysis-by-synthesis approach*» [18] et consiste à avoir un musicien professionnel qui évalue toutes les tentatives de valeurs, données par un chercheur, pour les paramètres d'une règle. Musicien et chercheur sont donc constamment en train d'évaluer la proposition de l'autre afin de trouver les meilleurs paramètres pour chacune des règles. Les auteurs de l'article [4] expliquent que l'avantage de cette approche réside dans le fait qu'elle permet de modéliser une sorte d'interaction entre l'interprète et l'auditeur. Cependant, le désavantage est qu'il y a trop d'attente envers le ou les musiciens experts et qu'il n'y a qu'un petit nombre de jugements externes qui sont proposés pour une règle.

Voici un exemple de règle locale du modèle KTH : diminuer le temps des notes qui sont relativement courtes et rallonger le temps des notes qui sont relativement longues. Voyons maintenant une règle qui s'applique à plusieurs notes : modifier l'intensité des notes proportionnellement à leur fréquence. Un des problèmes que l'on peut voir dans ce modèle est le fait que plusieurs règles peuvent s'appliquer sur les caractéristiques d'une même note. Cette interaction peut donner des effets qui n'étaient pas voulus à l'origine.

Le tableau qui suit est un survol des règles du modèle KTH.

Phrasing	
Phrase arch	Create arch-like tempo and sound level changes over phrases
Final ritardando	Apply a ritardando in the end of the piece
High loud	Increase sound level in proportion to pitch height
Micro-level timing	
Duration contrast	Shorten relatively short notes and lengthen relatively long notes
Faster uphill	Increase tempo in rising pitch sequences
Metrical patterns and grooves	
Double duration	Decrease duration ratio for two notes with a nominal value of 2:1
Inégales	Introduce long-short patterns for equal note values (swing)
Articulation	
Punctuation	Find short melodic fragments and mark them with a final micropause
Score legato/staccato	Articulate legato/staccato when marked in the score
Repetition articulation	Add articulation for repeated notes.
Overall articulation	Add articulation for all notes except very short ones
Tonal tension	
Melodic charge	Emphasize the melodic tension of notes relatively the current chord
Harmonic charge	Emphasize the harmonic tension of chords relatively the key
Chromatic charge	Emphasize regions of small pitch changes
Intonation	
High sharp	Stretch all intervals in proportion to size
Melodic intonation	Intonate according to melodic context
Harmonic intonation	Intonate according to harmonic context
Mixed intonation	Intonate using a combination of melodic and harmonic intonation
Ensemble timing	
Melodic sync	Synchronize using a new voice containing all relevant onsets
Ensemble swing	Introduce metrical timing patterns for the instruments in a jazz ensemble
Performance noise	
Noise control	Simulate inaccuracies in motor

Table 1: Survol des règles du modèle KTH [5].

Finalement le graphique qui suit représente, pour une pièce de musique, les variations temporelles générées par l'application des règles du modèle KTH.

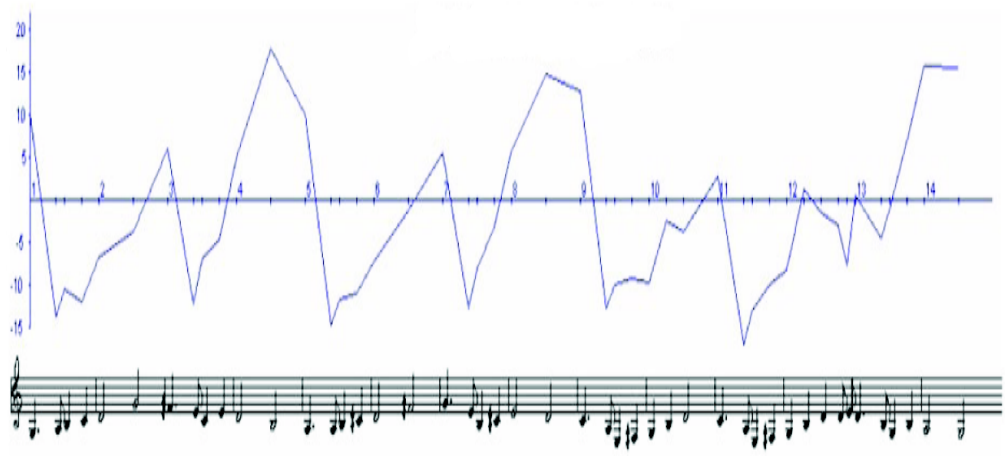

Figure 2: Variations temporelles générées par le modèle KTH [5] pour la pièce suèdoise «*Ekorr'n satt i granen*» avec l'axe des X pour les mesures et l'axe des Y pour le «*timing*».

2.1.2 CMERS (Computational Music Emotion Rule System)

Le modèle CMERS [14] permet de contrôler les émotions perçues dans un contexte musical en faisant des modifications au niveau de la partition et de la performance jouée. Des règles sont appliquées sur une partition pour modifier l'émotion ressentie. Ces règles sont obtenues par l'analyse statistique de la relation entre une émotion et une caractéristique musicale (ex: accord majeur≈heureux, accord mineur≈triste).

Le modèle CMERS peut être vu comme un système de filtres, où chaque filtre modifie un aspect différent de la musique. Un filtre représente une règle et les règles sont regroupées par émotion. Le système utilise des fichiers MIDI et fonctionne en temps réel. L'utilisateur peut donc modifier les émotions perçues dans une musique pendant qu'elle joue.

Plus de 150 interprétations émotionnellement différentes de trois pièces de musique ont été étudiées afin de créer les règles. L'auteur insiste sur la distinction à faire entre une émotion perçue et une émotion ressentie. L'émotion perçue correspond à celle que l'on croit que la musique exprime, alors que l'émotion ressentie correspond à celle que l'on éprouve après avoir écouté la musique. L'auteur précise que le modèle CMERS ne modifie que l'émotion perçue.

La représentation des émotions qui a été utilisée pour la création des règles est illustrée dans la figure qui suit. L'axe des X représente le fait que l'émotion perçue est positive ou négative (content, faché) et son intensité. L'axe des Y indique le niveau d'énergie de l'émotion (excité, calme).

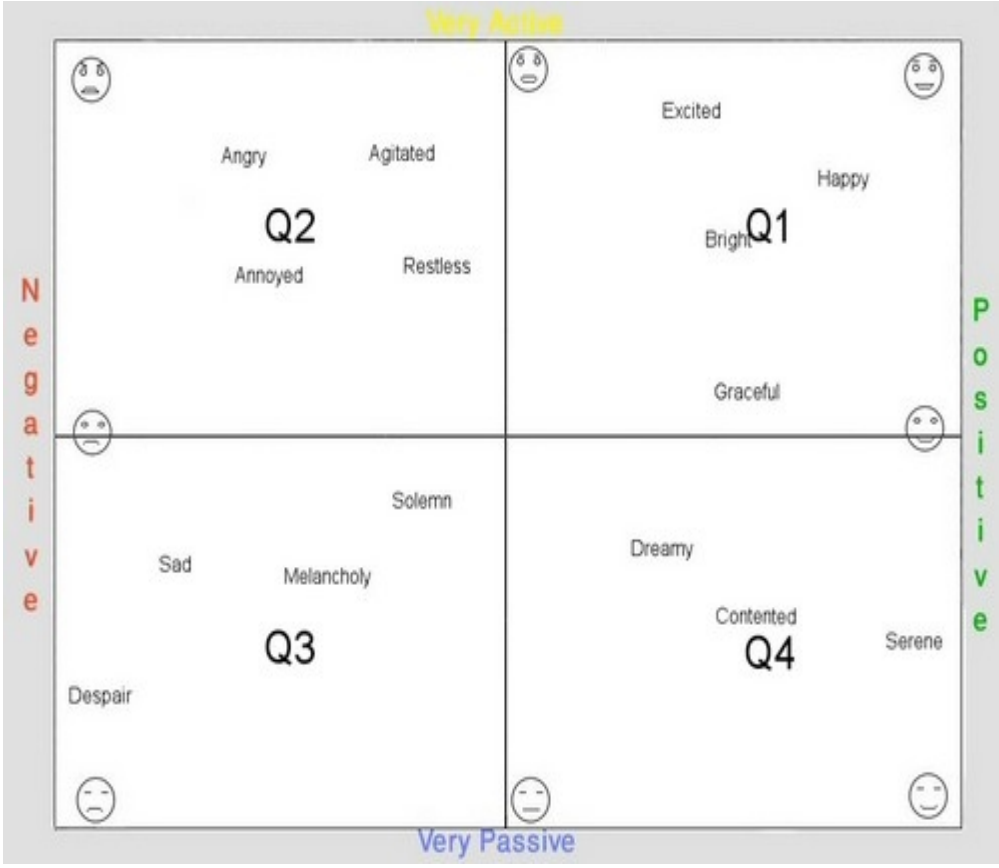

Figure 3: Représentation de l'émotion utilisée par le modèle CMERS [14]. L'axe des X représente l'état d'esprit de positif à négatif (content, faché) et l'axe des Y indique le niveau d'énergie (excité, calme).

Notons que ce modèle tente de modéliser les émotions musicales, ce qui ne correspond pas à la façon dont un pianiste joue. Une autre approche est donc nécessaire si l'on veut réussir à modéliser un style de musique.

2.1.3 Les modèles statistiques de WIDMER

L'approche de Widmer pour modéliser l'expressivité des performances [26] consiste à appliquer des modèles statistiques sur un ensemble de données composé de performances.

Ces données sont des représentations précises d'interprétation faite par des pianistes professionnels. C'est l'ordinateur, et non l'être humain, qui va automatiquement trouver des régularités significatives dans les données. Pour ce faire, on se sert de l'apprentissage machine, notamment les techniques de «*inductive machine learning*». Ces algorithmes d'apprentissage permettent de générer des règles qui peuvent être interprétées et directement utilisées pour la création de performances. La technique de Widmer se divise en deux modèles distincts. Le premier apprend les règles locales, c'est-à-dire comment un pianiste va jouer une note en particulier dans une pièce. Le deuxième modèle apprend les règles qui s'appliquent à des groupes de notes et qui ont des dépendances à plus long terme, comme le tempo.

Modèle local

Dans une première tentative pour trouver des règles générales aux performances humaines, les auteurs ont développé un nouvel algorithme d'apprentissage inductif [25] appelé PLCG (Partition+Learn+Cluster+Generalize). Le modèle PLCG est un méta-algorithme qui utilise plusieurs autres algorithmes heuristiques. L'idée de base est d'apprendre plusieurs modèles en parallèle, chacun d'eux entraîné sur un sous-ensemble différent de données, de trouver des groupes de règles similaires entre les modèles, de généraliser ces groupes en fusionnant les règles (semblable aux techniques employées pour les arbres de décisions), de sélectionner certaines de ces généralisations pour le modèle final qui ensuite optimise certains critères. Les auteurs prétendent que cette méthode a l'avantage d'améliorer un des problèmes majeurs dans l'apprentissage de modèles de règles: le danger de sous-optimiser la sélection de certaines règles, danger causé par la possibilité de maxima locaux dans une mesure discriminante.

Le modèle PLCG a besoin de trois autres algorithmes heuristiques pour fonctionner. Un algorithme L pour apprendre les règles, un algorithme H de classification hiérarchique et un critère S de sélection de règles. Le modèle PLCG est expliqué par le pseudo-code qui suit [25] (page 7).

Algorithm 1 Modèle PLCG[25]

Ayant :

- un ensemble de données d'entraînement D
- un concept de cible c
- un algorithme d'apprentissage de règles L
- un algorithme de classification hiérarchique H
- un critère de sélection de règles S

Algorithme :

1. Séparer l'ensemble d'entraînement D en n sous-ensembles D_i, $i = 1, ..., n$.
2. Apprendre séparément les modèles partiels de règles $R_i = \{r_{ij}\}$ en fonction de la prédiction de la cible c pour chaque sous-ensemble d'entraînement D_i en utilisant l'algorithme d'apprentissage L.
3. Rassembler les ensembles de règles R_i en un gros ensemble de règles R, $R_i \subset R$.
4. Utiliser l'algorithme de classification hiérarchique H pour grouper les règles similaires de l'ensemble R en arbre de groupe C_i, où $i = 1, ..., k$.
5. Pour chaque groupe C_i, calculer la *généralisation la moins générale [15]* pour toutes les règles dans C_i, où $\hat{r}_i = glmg(\{r_{ij} | r_{ij} \epsilon C_i\})$. L'arbre résultant T des règles $\hat{r}_i$ représente la généralisation, de degré variable, des règles originales.
6. À partir de l'arbre de généralisation T, sélectionner les règles $\hat{r}_i$ qui optimisent le critère de sélection S.

L'algorithme d'apprentissage de règles L utilise de l'apprentissage machine connu sous le nom de «diviser pour régner» (*«divide-and-conquer»*). Les auteurs se servent de la méthode FOIL [16] pour apprendre les règles. Cet algorithme apprend les modèles une règle à la fois et ceux-ci sont optimisés en maximisant une fonction de coût (mesure discriminante). Un critère d'arrêt basé sur une certaine précision des résultats est utilisé pour l'entraînement. Le processus général s'arrête quand plus aucune nouvelle règle ne donne un gain d'information, ce qui permet de satisfaire le critère d'arrêt.

Pour la fonction de regroupement de règles H, les auteurs se sont servis d'un algorithme de partitionnement hiérarchique bas-haut («*bottom-up*») standard [7] qui construit un arbre binaire de groupe, où chaque règle représente une feuille de l'arbre.

Le critère de sélection de règles S est un algorithme qui commence avec un ensemble de règles vides. À chaque étape, il rajoute une règle qui fait partie de l'ensemble de règles déjà trouvé et qui est considérée comme étant la plus précise de toutes. L'estimé

de Laplace est utilisé pour calculer la précision. Encore une fois, la sélection des règles s'interrompt quand un critère d'arrêt, basé sur la précision, est satisfait.

L'ensemble d'entraînement utilisé dans les expériences est composé de 12 performances différentes. Chacune de ces performances représente une sonate différente de W.A. Mozart. Les interprétations ont été jouées par un pianiste professionnel et ont été enregistrées sur un piano Bösendorfer 290SE, contrôlé par ordinateur. Ceci permet de saisir tous les détails sur le tempo, la dynamique ainsi que l'articulation qui ont été utilisés pour chaque performance. Pour montrer au lecteur comment quelques règles simples peuvent prédire le comportement d'un pianiste dans certains cas précis, la figure 4 compare les variations de tempo prédites par les règles avec celles d'un vrai pianiste.

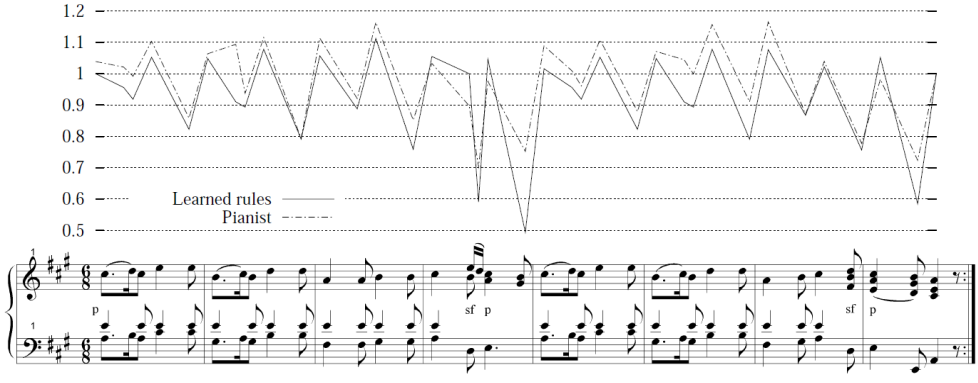

Figure 4: Représentation [25] de la variation du tempo jouée par un pianiste (ligne solide) et par un modèle [25] (ligne pointillée) pour le premier mouvement de la sonate de Mozart (K.331). Les courbes représentent le tempo relatif à chaque note. Les notes au-dessus de 1.0 sont ralenties, les notes en-dessous de 1.0 sont accélérées. Une performance parfaitement régulière sans variation temporelle correspond à une ligne droite de 1.0.

Pour l'entraînement des modèles de cette section, les auteurs n'ont utilisé que la mélodie des performances, ce qui correspond à un ensemble d'entraînement de 41116 notes. Chaque note est décrite par 29 attributs qui représentent des informations comme la position dans une mesure ou le contour mélodique local. A partir de ces données d'entraînement, 17 règles ont été repérées pour prédire l'expressivité locale des notes. Voici un exemple de règles trouvé par un modèle et interprété par les auteurs :

- Si on a deux notes de même longueur suivies d'une note plus longue, il faut rallonger la note qui précède la dernière longue note si elle se trouve dans le premier temps de la mesure.

Une description complète de toutes les règles locales trouvées figure dans l'article de Widmer [24]. Un fait intéressant est que certaines des règles trouvées par l'apprentissage machine présentent de grandes ressemblances avec certaines règles du modèle KTH.

Modèle long terme

Alors que le modèle décrit précédemment découvrait la structure locale, ce modèle [27] s'occupe de comprendre la logique du rythme et des variations expressives qui apparaissent tout au long de la pièce de musique. Les règles qui régissent ces variations ne représentent qu'une petite partie de l'expressivité d'une performance. Le but pour le modèle est d'apprendre à prédire le tempo et le contour dynamique (comme le crescendo et le decrescendo) pour former les sructures expressives qu'un pianiste applique à une partition.

L'entrée du modèle est composée par les annotations d'une partition de musique et par une représentation de la variation du tempo et de la dynamique appliquées par un pianiste. Le tempo et la dynamique, à un moment précis dans la pièce, sont représentés comme des facteurs multiplicatifs de leurs moyennes respectives pour cette interprétation. Par exemple, une dynamique de 1.5 veut dire que la note a été jouée 50% plus fort que la moyenne des dynamiques de la performance.

Les courbes originales de tempo et de dynamique ne sont pas directement données au modèle; un pré-traitement est d'abord appliqué. Le pré-traitement consiste à découper une courbe (tempo ou dynamique) de la totalité d'une pièce en sous-parties et à trouver à la main, pour chacune d'entre elles, une fonction quadratique qui représente au mieux les données. Quatre niveaux de découpage, qui se suivent, ont été utilisés. Au premier niveau, on trouve une fonction quadratique pour la totalité de la pièce, puis on soustrait cette fonction de la courbe originale, ce qui donne une nouvelle courbe à laquelle on a enlevé une variation globale. Le deuxième niveau utilise cette nouvelle courbe et la découpe en sous-parties (correspondant aux phrases). Une fonction quadratique est trouvée pour chacune de ces sous-parties. Une nouvelle courbe est calculée en soustrayant les fonctions qui viennent d'être trouvées à la courbe utilisée au début de ce deuxième niveau. La même technique est utilisée pour le troisième et le quatrième niveau, où chacun d'eux se base sur la courbe calculée par le niveau qui le précède et où les sous-parties sont

toujours plus raffinées (comme un découpage par mesure). Après le quatrième niveau, il reste une courbe résiduelle représentant l'expressivité qui dépend du contexte local et qui est apprise par le modèle PLCG.

L'entrée du modèle est donc composée d'une liste de facteurs multiplicatifs pour le tempo et d'une autre liste pour la dynamique. Chaque élément d'une liste correspond à un des quatre niveaux de découpage et donne la valeur de la fonction quadratique à un moment précis dans la sous-partie qu'elle représente. Pour reconstruire la variation complète du tempo ou de la dynamique, pour un moment spécifique dans la partition, il suffit d'additionner les différents éléments de la liste qui représente l'approximation quadratique de la courbe.

Pour apprendre ces variations de tempo et de dynamique, les auteurs utilisent la technique du plus proche voisin comme algorithme d'apprentissage. Un seul voisin est utilisé pour prédire la forme de la structure des phrases d'une nouvelle pièce de musique. Pour une nouvelle phrase donnée, l'algorithme cherche dans sa mémoire afin de trouver (au même niveau de découpage), parmi tous les exemples qu'il a vus, la phrase la plus similaire. Il prédit alors la fonction quadratique de la phrase en mémoire comme étant la bonne fonction pour la nouvelle phrase.

La similarité entre les courbes expressives des phrases est obtenue en calculant la distance euclidienne entre la cible (celle du modèle) de la nouvelle phrase et celle de la phrase gardée en mémoire.

Combinaison du modèle local et du modèle long terme

Comme il a été mentionné auparavant, les variables qui représentent une courbe expressive sont des facteurs multiplicatifs de la moyenne de cette courbe. Appliquer les prédictions à plusieurs niveaux, faites par le modèle long-terme et le modèle local, s'obtient simplement en appliquant successivement les différents niveaux de découpage de courbe. Pour une nouvelle pièce donnée, le système commence avec une courbe expressive initiale qui est droite (liste de valeurs égales à 1.0) et multiplie successivement la valeur courante par la prédiction des modèles en partant du niveau de découpage le plus raffiné (modèle local) et en finissant au niveau le plus général. La figure qui suit représente, avec la courbe rouge, la prédiction du modèle total, pour les variations dynamiques, sur une

nouvelle pièce de musique. La courbe noire correspond à la courbe expressive d'un vrai pianiste et les lignes bleues indiquent les différents niveaux de découpage.

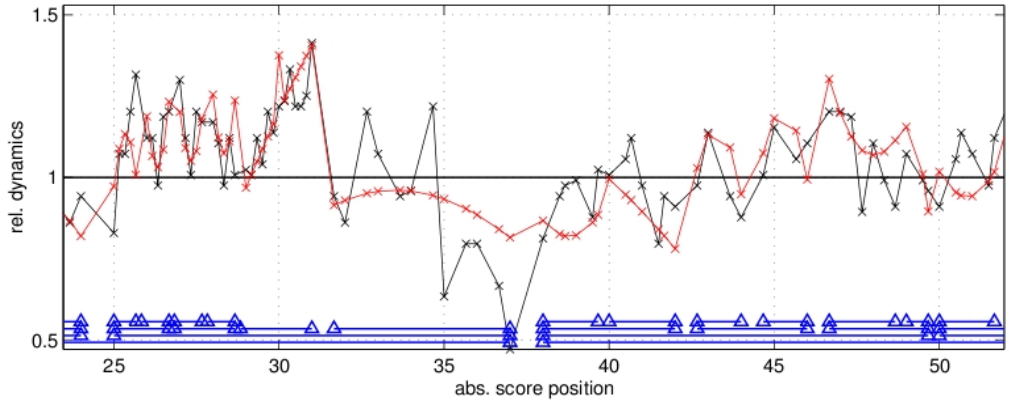

Figure 5: Résultat de Widmer dans son article [27] avec en rouge la prédiction du modèle pour les variations dynamiques d'une pièce, en noir la courbe expressive d'un vrai pianiste et en bleu les différents niveaux de découpage.

2.1.4 Apprentissage machine et IA

Si on considère le but de créer un bon modèle génératif pour l'expressivité des performances de pièces de musique, l'approche classique en intelligence artificielle - les systèmes experts - consiste à accumuler toutes les connaissances humaines possibles sous forme de règles pour ensuite les utiliser dans la prédiction des variations expressives. Le premier problème avec cette approche est d'optimiser les paramètres de chacune des règles. Cette tâche est souvent faite à la main par des experts dans le domaine et demande un travail colossal, très couteux en temps et en effort et qui n'est pas toujours couronné de succès. Le deuxième problème se trouve au niveau des règles. En effet, l'expressivité musicale est un phénomène très complexe: il possède des dépendances à court et à long terme et ne peut pas être entièrement expliqué par un petit nombre de règles. Même s'il existe certaines règles qui couvrent une bonne partie de l'expressivité des performances, il y a toujours des exceptions qui demandent de rajouter un nombre considérable de nouvelles règles. Sans compter le fait que le choix de celles-ci, déterminé par des humains, est plutôt subjectif. Ceci a pour effet de rendre le modèle lourd, difficile à utiliser et donne souvent des résultats peu satisfaisants. Le point fort de cette technique est de permettre la compréhension, en langage humain, de la logique de l'expressivité des interprétations.

L'apprentissage machine est une autre approche de l'intelligence artificielle. Elle n'a

pas pour but de trouver des règles interprétables, mais plutôt de trouver la logique des variations expressives sous forme de fonction. Selon nous, cette approche est plus appropriée pour créer un bon modèle génératif qui doit apprendre un phénomène aussi complexe que l'expressivité musicale. Dans cette approche, on laisse le modèle apprendre tout seul la fonction qui représente le mieux les données fournies en exemple, ce qui épargne beaucoup de travail à la main, travail qui serait nécessaire dans une approche comme celle des systèmes experts.

2.2 Analyse

Le modèle KTH est visiblement une bonne technique de compréhension qui permet d'expliquer en mots, sous forme de règles, la logique des variations expressives musicales. Pour transformer ces règles en modèle génératif, il faut trouver des valeurs fixes pour les nombreux paramètres qui les composent. Plusieurs tentatives ont été faites dans le but d'optimiser un ou plusieurs paramètres de ces règles sur des données différentes, comme dans Friberg [3], Kroiss [12] ou Zannon et De Poli [31, 30]. Les résultats obtenus par ces différentes expériences ont démontré la validité d'une bonne partie des règles. Il faut cependant en ajuster les paramètres pour chaque ensemble de données afin d'obtenir une représentation raisonnable de l'expressivité. Selon Werner et Goebel[26], la capacité à généraliser n'a pas été démontrée et reste encore une question à développer. Un autre problème avec ce modèle est la nécessité de la perception d'un expert. Avec le CMERS et Widmer, on passe à une autre étape où on est moins intéressé à donner un sens aux règles qui sont trouvées, ce qui ouvre la porte vers d'autres modèles.

Le modèle CMERS présente lui aussi une bonne approche pour comprendre l'expressivité musicale. La différence avec le modèle KTH est que celui-ci essaye de comprendre l'expressivité que les pianistes professionnels appliquent sur les différentes pièces de musique, alors que le CMERS veut apprendre les variations expressives en fonction des émotions. L'avantage de cette technique est que les règles et les paramètres sont choisis à partir d'un ensemble de données empiriques qui représentent la perception générale de l'auditeur, contrairement au modèle KTH qui se base sur la perception d'un petit nombre de musiciens experts et de chercheurs. Le fait de s'intéresser principalement aux émotions a permis aux auteurs de trouver des règles très simples facilement utilisables

pour la génération. Le problème pour nous est que la complexité du style musical n'a pas été apprise et que cela ne permet donc pas la reproduction expressive d'une pièce de musique comme un musicien le ferait.

Le modèle de Widmer est plus approprié pour trouver la logique de l'expressivité musicale. En effet, il ne se base pas principalement sur la perception subjective de l'être humain pour expliquer une variation temporelle ou dynamique, mais plutôt sur une analyse statistique appliquée à des données empiriques. Du point de vue d'un modèle génératif, le fait de vouloir utiliser l'apprentissage machine comme méthode pour trouver la logique expressive - sous forme de règles qui sont facilement interprétables - n'est peut-être pas la solution optimale. Ceci pousse à se demander si l'on peut améliorer l'apprentissage en utilisant d'autres sortes de modèles statistiques (comme les réseaux de neurones). Un des problèmes avec les modèles à règles comme FOIL est qu'ils sont souvent instables, c'est-à-dire que le fait de rajouter ou d'enlever un exemple d'apprentissage peut grandement modifier la structure du modèle ainsi que les prédictions qu'il génère, un peu comme les arbres de décisions. Cependant, le méta-algorithme PLCG que les auteurs utilisent pour filtrer et regrouper les règles qui se ressemblent permet d'avoir un modèle plus robuste. Les auteurs précisent que leur modèle donne de mauvais résultats pour ce qui est de l'apprentissage du tempo. Ils pensent que le problème vient du fait qu'ils utilisent des fonctions quadratiques pour représenter les courbes expressives et que ces fonctions sont insuffisantes pour le tempo. Finalement, les données employées pour l'apprentissage sont composées seulement par les notes de la mélodie dominante des performances, mais les pianistes ont créé l'expressivité en modifiant toutes les notes de la pièce. Il y a donc une perte d'information qui n'est peut-être pas négligeable.

Malgré toutes les recherches sur la modélisation des performances musicales, une question qui nous tient particulièrement à coeur subsiste: Peut-on apprendre, à partir d'exemples, les variations expressives propres à un style de musique? Une partie de la réponse se trouve peut-être dans les résultats de Bruno Repp [17]. En effet, cette recherche montre que les pianistes on tendance à jouer et à rejouer une séquence de musique avec le même rythme: ils accélèrent et ralentissent au même moment. Ceci laisse croire qu'il existe, pour une pièce, une logique expressive commune entre musiciens. Le graphique qui suit [17] illustre ce phénomène.

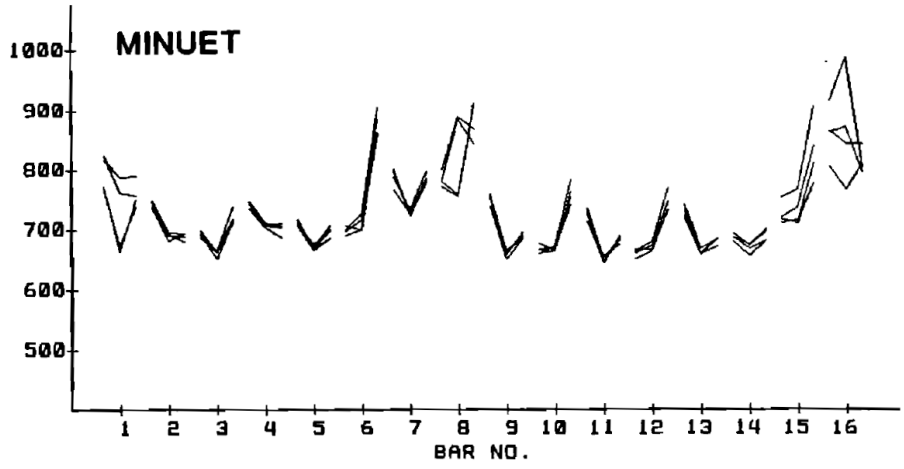

Figure 6: Moyenne totale de la variation rythmique des performances de 15 pianistes populaires pour un menuet de Beethoven [17]. Le menuet est séparé en quatre mêmes séquences qui sont représentées par quatre courbes superposées avec l'axe des X pour les mesures et l'axe des Y pour le «*timing*». Les mesures sont séparées par un espace.

Le fait de penser que l'on peut apprendre l'expressivité en fonction d'un style musical est une théorie qui, après une observation de nos données (chapitre 5), s'avère être justifiée par la ressemblance des variations expressives entre les pianistes. Trouver la fonction qui représente ce genre de variation semble être une tâche idéale pour l'apprentissage machine statistique quand on dispose d'un ensemble d'entraînement adéquat.

3 Apprentissage machine statistique

Le début de cette section est partiellement basé sur le chapitre 11 du livre d'introduction à l'apprentissage machine [1] et explique le fonctionnement de plusieurs modèles statistiques que nous avons appliqués sur des données séquentielles. Dans ce chapitre, nous essayons d'expliquer de façon exhaustive certains modèles de base, en allant du plus simple au plus complexe. Il s'agit du perceptron, du réseau de neurones ainsi que du réseau de neurones récurrent. Le but de cette démarche est de faciliter, par la suite, l'explication des modèles que nous avons développés. En effet, leurs présentations dépend de celle qui est faite pour les modèles de base.

La nomenclature suivante est utilisée tout au long de ce chapitre: L'ensemble d'entraînement D, utilisé par chacun des modèles, contient N séquences S^n, où $1 \leq n \leq N$. Une séquence S^n contient un nombre M^n d'éléments d'entrées x^t et de cibles c^t, où $1 \leq t \leq M^n$. Un vecteur d'entrée x^t possède J composantes x_j^t, où $1 \leq j \leq J$. Une cible c^t contient K composantes c_i^t, où $1 \leq i \leq K$.

Nous utiliserons par ailleurs les notation et conventions suivantes:

- La norme Euclidienne d'un vecteur x est notée $\|x\| = \sqrt{\sum_{j=1}^{J} x_j^2}$.
- La transposée d'une matrice ou d'un vecteur est indiqué par $'$, ainsi le vecteur $x = [x_1, \ldots, x_J]'$ est un vecteur colonne.
- Nous indiquerons par un $\tilde{}$ l'opération d'étendre un vecteur en lui ajoutant une première composante constante de valeur 1. Ainsi si $x = [x_1, \ldots, x_J]'$, alors $\tilde{x} = [1, x_1, \ldots, x_J]'$
- Soit A une matrice de réels de dimension $m \times n$, c.a.d. de m rangées par n colonnes. On notera A_{ij} l'élément à la rangée i et la colonne j. On notera A_i le *vecteur colonne* correspondant à la $i^{ème}$*rangée* de A, c.a.d. $A_{i\cdot} = [A_{i1}, \ldots, A_{in}]'$, et $A_{\cdot j}$ le *vecteur colonne* correspondant à la $j^{ième}$ colonne de A, c.a.d. $A_{\cdot j} = [A_{1j}, \ldots, A_{mj}]'$.

Le premier modèle que nous allons présenter est un simple réseau de neurones, mais pour bien expliquer son fonctionnement, il faut commencer par l'élément le plus simple qui le compose, c'est-à-dire le perceptron.

3.1 Perceptron

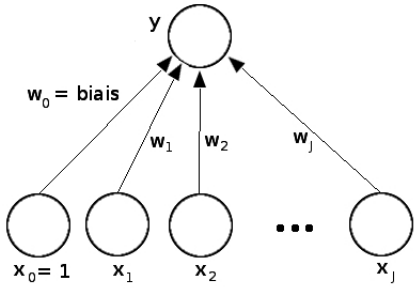

Figure 7: Perceptron

Le perceptron est composé de variables en entrée, qui forment une couche d'entrée, et d'une variable de sortie. L'entrée peut être vue comme étant de l'information qui vient directement de l'environnement, ou qui provient de la sortie d'un autre perceptron. Pour chaque entrée x, composée des éléments $x_j \epsilon \mathbb{R}$, où $1 \leq j \leq J$, il y a un poids de connexion associé, $w_j \epsilon \mathbb{R}$. La sortie y représente la somme des éléments d'une entrée multipliée par leur poids assigné (voir figure 7).

$$y = \sum_{j=1}^{J} w_j x_j + w_0 \tag{1}$$

w_0 représente le biais qui rend le modèle plus général et est associé à l'élément x_0 qui a toujours la valeur de 1, ce qui donne $w = [w_0, w_1, ..., w_J]'$ et $\tilde{x} = [1, x_1, ..., x_J]'$. Avec ces vecteurs augmentés, on peut représenter la sortie y du perceptron par un produit scalaire entre le vecteur d'entrée et le vecteur de poids correspondant.

$$y = w'\tilde{x} \tag{2}$$

Ce genre de perceptron permet de définir un hyperplan qui sépare l'espace des données x en deux, une partie positive ($y > 0$) et l'autre négative ($y < 0$). On peut donc séparer deux classes en assignant une classe au côté positif de l'hyperplan et une autre classe au côté négatif.

Pour obtenir pour la prédiction de la cible c une probabilité conditionnelle $y \simeq P(c = 1|x)$ à la sortie du perceptron, on applique la fonction sigmoïde au résultat obtenu, ce

qui donne :

$$y = sigmoïde(w'\tilde{x}) = \frac{1}{1 + e^{-w'\tilde{x}}} \tag{3}$$

3.2 Réseaux de neurones sans couche cachée

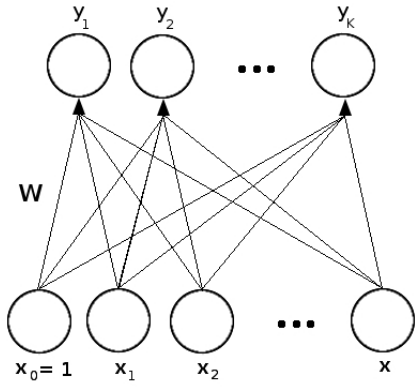

Figure 8: Réseaux de neurones sans couche cachée

Si K représente le nombre de classes et que $K > 2$, alors on utilise un perceptron par classe, ce qui correspond à un réseau de neurones sans couche cachée. Chaque perceptron utilise un vecteur de poids W_i (i=1 à K) et indique à quel point les données en entrée peuvent faire partie de la classe que i représente (voir figure 8). On regroupe donc tous les vecteurs de poids dans la matrice de connexions W.

$$Perceptron_i = o_i = \sum_{j=1}^{J} W_{ij} x_j + W_{i0} \tag{4}$$

Maintenant, pour calculer la probabilité conditionnelle $y_i \simeq P(classe = i|x)$ d'une classe dans ce modèle à plusieurs perceptrons, il suffit d'utiliser la fonction softmax.

$$y_i = \frac{e^{o_i}}{\sum_k e^{o_k}} \tag{5}$$

Pour classifier les données en entrée, on choisit la classe qui correspond au y_i qui a la valeur la plus élevée.

$$classe = \underset{i}{argmax}(y_i), 1 \leq i \leq K \tag{6}$$

Le réseau que l'on vient de voir offre beaucoup d'applications utiles possibles, mais il n'est pas toujours assez puissant. Le fait de n'avoir qu'une seule couche de poids a comme inconvénient de n'approximer que des fonctions linéaires. On estime une fonction non linéaire à l'aide d'une couche cachée. Il s'agit d'une couche de neurones supplémentaire que l'on place entre la couche d'entrée et celle de sortie. Le modèle que l'on vient de voir est une sorte de réseau de neurones qui n'a pas de couche cachée. Nous allons donc tout naturellement passer à l'étape suivante qui est celle des réseaux de neurones à une couche cachée.

3.3 Réseaux de neurones

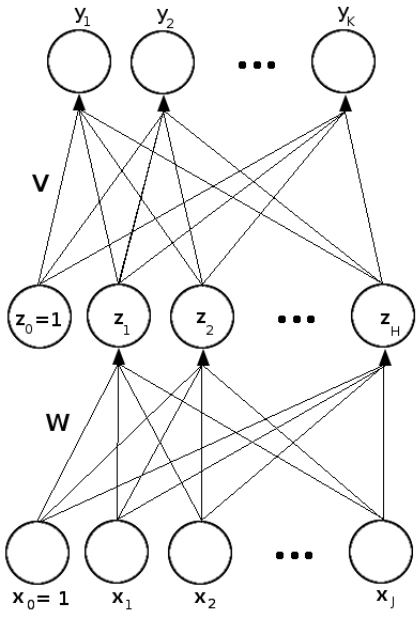

Figure 9: Réseau de neurones à deux couches

Il s'agit de rajouter, dans le modèle que l'on vient de voir (section 3.2), une couche cachée comportant H neurones entre la couche de neurones d'entrée (J neurones) et la couche de neurones de sortie (K neurones). C'est cette couche cachée qui permet d'estimer une fonction non linéaire de l'entrée. Dans cette section, pour un problème de régression, nous définissons un réseau de neurones en deux étapes: la propagation avant et la propagation arrière.

3.3.1 Propagation avant (*«forward propagation»*)

Une donnée x est fournie à la première couche du modèle, c'est-à-dire l'entrée. On propage x à travers la matrice de poids W de dimension $H \times J$ en y ajoutant un vecteur de biais w_0 de dimension H, ce qui donne un résultat qu'on appelle le vecteur d'«activation» a, de dimension H.

$$a = Wx + w_0 \tag{7}$$

En appliquant la fonction sigmoïde sur l'activation, on obtient la couche cachée z. z est un vecteur de dimension H et chacun de ses éléments z_h peut être vu comme le résultat d'un perceptron différent.

$$z_h = \frac{1}{1 + e^{-a_h}} \tag{8}$$

$h = 1, ..., H$ où H est le nombre d'éléments (de neurones) de la couche cachée.

Il faut maintenant calculer le vecteur de sortie y de notre modèle de dimension K. On peut voir un élément y_i de la sortie du réseau de neurones comme un perceptron qui prend en entrée la couche cachée. On propage le vecteur z, qui a été calculé auparavant, à travers la nouvelle matrice de poids V de dimension $K \times H$ qui fait la connexion entre la couche cachée et la couche de sortie, en y ajoutant un vecteur de biais v_0 de dimension K. On obtient donc le vecteur de prédiction y de dimension K pour un problème de régression.

$$y = Vz + v_0 \tag{9}$$

Le but ultime de notre modèle est d'estimer une fonction de régression qui prend en paramètre une entrée x et prédit la cible c qui lui correspond. On veut donc que la prédiction y, calculée par le réseau de neurones à partir de l'entrée x, soit le plus près possible de la cible c. Dans ce but on utilise, pour chaque prédiction y, la fonction de coût suivante :

$$Coût(y, c) = \|c - y\|^2 = \sum_{k=1}^{K} (c_k - y_k)^2 \tag{10}$$

Cette fonction de coût s'appelle la différence au carré (*mean squared error*) et nous permet d'évaluer à quel point la prédiction du modèle est proche de la bonne valeur: elle se nomme aussi la fonction d'erreur. D'autres fonctions peuvent être utilisées comme la cross-entropy pour la classification . En diminuant le coût que l'on vient de voir (10) on parviendra à apprendre la fonction de prédiction. Diminuer le coût du réseau de neurones pour une entrée x et une cible c s'appelle entraîner le modèle et est expliqué dans la section qui suit.

3.3.2 Propagation arrière (BP «*Backpropagation*» [8])

L'apprentissage avec la propagation arrière (technique de calcul du gradient) consiste à modifier les poids du réseau de neurones pour obtenir des prédictions plus proches des cibles. Pour savoir quelle est la valeur de la modification d'un poids, il faut calculer le gradient de la fonction de coût du modèle par rapport à ce poids. Le gradient représente la pente de la fonction de coût pour le poids. Il faut donc soustraire la valeur du gradient au poids pour diminuer l'erreur. Le gradient de chacune des connexions V_{kh} qui font le lien entre la couche cachée et la couche de sortie se calcule en faisant la dérivée de la fonction de coût par rapport à chaque poids, ce qui donne:

$$(\nabla Coût)_{V_{kh}} = -2(c_k - y_k)z_h \tag{11}$$

pour tous les k et les h, où $1 \leq k \leq K$ et $1 \leq h \leq H$.

Ce qui peut aussi s'écrire de façon matricielle:

$$(\nabla Coût)_V = -2(c - y)z' \tag{12}$$

Le calcul des gradients des biais est le suivant:

$$(\nabla Coût)_{v_0} = -2(c - y) \tag{13}$$

Une fois que l'on a calculé le gradient de chaque poids de la matrice V et des biais v_0, il faut s'occuper de la matrice de connexions W qui fait le lien entre la couche cachée et la couche d'entrée. On utilise la technique de la dérivation en chaîne pour calculer le gradient de chacun des poids W_{hj}, ce qui donne:

$$(\nabla Coût)_{W_{hj}} = \underbrace{-2(c-y)'}_{\partial \mathrm{Coût}/\partial \mathrm{y}} \underbrace{V_{\cdot h}}_{\partial \mathrm{y}/\partial \mathrm{z_h}} \underbrace{z_h(1-z_h)x_j}_{\partial \mathrm{z_h}/\partial \mathrm{W_{hj}}} \tag{14}$$

pour tous les h et les j, où $1 \leq h \leq H$ et $1 \leq j \leq J$. Ici nous avons utilisé le fait que $\frac{\partial z_h}{\partial W_{hj}} = \frac{\partial z_h}{\partial o_h}\frac{\partial o_h}{\partial W_{hj}} = sigmoid'(z_h)x_j = z_h(1-z_h)x_j$, $sigmoid'$ désignant la dérivée de la fonction sigmoide.

Le calcul des gradients pour le vecteur de biais w_0 de la première couche est le suivant:

$$(\nabla Coût)_{w_{0h}} = -2(c-y)'V_{\cdot h}z_h(1-z_h) \tag{15}$$

La mise à jour de tous les poids que l'on vient de voir se calcule en multipliant le gradient qui lui correspond avec le taux d'apprentissage η. On soustrait ensuite ce résultat à son poids initial pour diminuer l'erreur.

$$W \leftarrow W - \eta(\nabla Coût)_W \tag{16}$$

$$V \leftarrow V - \eta(\nabla Coût)_V \tag{17}$$

Le modèle que l'on vient de voir peut modéliser les dépendances locales d'un ensemble d'entraînement composé de séquences d'éléments, avec un seul élément par entrée et le prochain élément qui suit comme cible. Cependant, un tel modèle ne peut pas représenter les dépendances dans le temps entre les différents éléments d'une de ces séquences. On peut toutefois appliquer une fenêtre de temps sur la séquence et donner cette fenêtre comme entrée pour le réseau de neurones, ce qui permet de modéliser une partie des dépendances dans le temps. Le prochain modèle que nous allons voir est une version de réseau de neurones qui prend en considération les dépendances dans le temps.

3.3.3 Réseau de neurones à décalage temporel (TDNN «*Time delay neural networks*» [21])

Le réseau de neurones peut être utilisé pour apprendre à partir de données temporelles. Il suffit de découper une séquence d'éléments x en une série de fenêtres temporelles qui se chevauchent. Chacune de ces fenêtres est composée d'une sous-séquence f_t ($[x_1, x_{2,}..., x_t]$) de longueur t. Pour un réseau de neurones, une fenêtre f_t représente l'entrée du modèle pour un élément de la couche cachée, une fenêtre f_{t+1} pour l'élément suivant et ainsi de suite pour toute la couche cachée. On fait ce découpage en fenêtre à chaque couche. La plus grande restriction de cette technique est que la longueur de la fenêtre temporelle doit être fixée à priori [1].

3.4 Réseaux de neurones récurrents (RNN «*Recurrent Neural Net*» [28])

Dans cette section, nous allons définir le modèle des réseaux de neurones récurrents en expliquant la technique de la propagation avant pour les dépendances dans le temps. Nous allons ensuite montrer comment entraîner un tel modèle avec la méthode de propagation arrière.

3.4.1 Propagation avant («*forward propagation*»)

La propagation avant consiste à dérouler les prédictions dans le temps pour une séquence en particulier, à générer une sorte d'arbre de prédiction qui servira ensuite à la propagation arrière. La figure 10 représente la propagation avant pour une séquence S de longueur M.

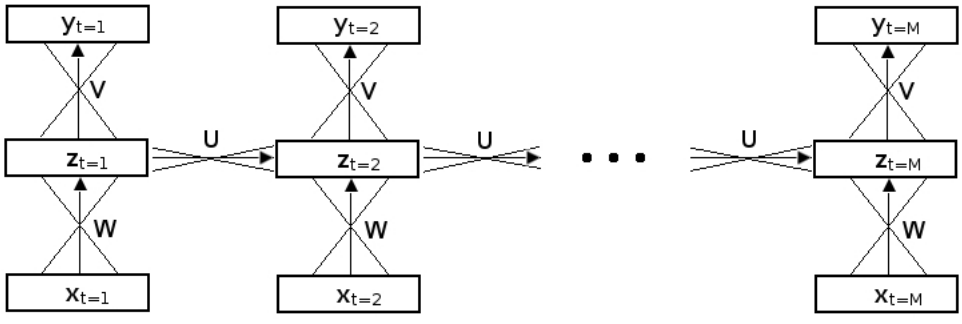

Figure 10: Réseau de neurones récurrent

On peut voir le réseau de neurones récurrent comme plusieurs réseaux de neurones simples partagent leurs paramètres, un par entrée de séquence, avec la matrice de connexions U en plus, faisant le lien entre les différentes couches cachées qui se suivent.

Pour chaque vecteur d'entrée qui fait partie de la séquence d'observation, on applique, dans un ordre croissant, la propagation de l'entrée dans le modèle. Ce calcul est expliqué en deux étapes.

Première étape

Chaque entrée correspond à un moment t. La cible, la couche cachée et l'entrée de ce moment t peuvent s'écrire respectivement c^t, z^t et x^t. Parallèlement, la cible, la couche cachée et l'entrée précédant le moment t s'écrivent c^{t-1}, z^{t-1} et x^{t-1}.

On commence par multiplier le vecteur d'entrée x^t avec la matrice de poids W pour créer un vecteur d'activation, mais ce n'est pas suffisant, l'information doit aussi passer à travers le temps. Pour ce faire, on propage le vecteur z^{t-1} de la couche cachée du moment précédent à travers la nouvelle matrice dynamique U de dimension $H \times H$, ce qui donne une deuxième activation, porteuse d'information temporelle. A ceci s'ajoute comme précédemment un vecteur de biais w_0. La couche cachée est donc calculée en appliquant la sigmoïde sur le résultat de l'addition entre l'activation provenant de l'entrée et l'activation provenant de la couche cachée précédente et le biais.

$$z^t = sigmoïde(\underbrace{Uz^{t-1} + Wx^t + w_0}_{a^t}) \tag{18}$$

Nous notons a^t le vecteur d'activation des neurones avant la non-linéarité dans la couche cachée au temps t. Dans le cas de la première couche cachée de la séquence (t=1), la contribution provenant de la connexion dynamique est fixée à zéro ($z^0 = 0$).

Deuxième étape

Il faut maintenant calculer le vecteur de sortie y^t du modèle. Comme pour le réseau de neurones, on propage la couche cachée z^t, qui a été calculée auparavant, à travers la matrice de poids V. On obtient donc y^t, le vecteur de prédiction au moment t.

$$y^t = Vz^t + v_0 \tag{19}$$

On répète les deux étapes que l'on vient de voir pour chacune des entrées de la séquence S.

Le déroulement dans le temps de toutes les prédictions de cibles et de couches cachées est gardé en mémoire dans des listes pour être utilisé dans l'étape suivante qui est simplement une mise à jour des différents poids du modèle selon la technique de la propagation arrière utilisée pour les réseaux de neurones dynamiques.

3.4.2 Propagation arrière dans le temps (BPTT «*Backprop through time*» [23])

Le but du modèle est d'estimer une fonction qui calcule la vraisemblance des cibles c^t d'une séquence. Puisque les données sont dépendantes dans le temps, la prédiction du modèle pour c^t dépend non seulement de l'entrée x^t, mais aussi de son passé, c'est-à-dire $x^1, x^2, ..., x^{t-1}$. On calcule le coût pour chaque prédiction y^t de la façon suivante :

$$Coût(y^t) = \|c^t - y^t\|^2 \tag{20}$$

Encore une fois d'autres fonctions peuvent être utilisées pour l'apprentissage. On peut aussi calculer le coût de la prédiction de la séquence au complet, ce qui donne:

$$C = Coût(y^1, y^2, ..., y^M) = \sum_{t=1}^{M} \|c^t - y^t\|^2 \tag{21}$$

Cette fonction de coût ne peut cependant pas être utilisée comme point de comparaison entre les séquences des données, car elles n'ont pas toutes la même longueur. On utilise donc la moyenne de ce coût pour pouvoir comparer le résultat des prédictions.

On veut que la prédiction y^t se rapproche de la cible c^t. Pour ce faire, on utilise la propagation arrière que l'on explique en trois étapes.

Calcul du gradient sur les poids et biais de la couche de sortie

Plaçons-nous à la fin du déroulement dans le temps généré par le réseau de neurones dynamique. On calcule l'erreur de prédiction du dernier élément, y^t, de la séquence où $t = M$. On obtient le gradient de chaque connexion v_{ih} de la matrice V, qui fait le lien entre la sortie et la couche cachée, en multipliant l'erreur de y_i^t par le neurone de

la couche cachée z_h^t. On fait ce calcul pour chacun des éléments t de la séquence S de longueur M.

$$(\nabla C)_{V_{kh}} = \sum_{t=1}^{M} -2(c_k^t - y_k^t) z_h^t \tag{22}$$

où z^t représente le vecteur de la couche cachée et h l'index du neurone dans cette couche.

Ceci peut aussi s'écrire de façon matricielle par

$$(\nabla C)_V = \sum_{t=1}^{M} -2(c^t - y^t) \left(z^t\right)' \tag{23}$$

De même le gradient pour le biais de sortie v_0 est donné par

$$(\nabla C)_{v_0} = \sum_{t=1}^{M} -2(c^t - y^t)$$

Les mises à jour sur ces biais et sur les poids de la matrice de connexion V sont accumulées et gardées en mémoire. Il est important de ne pas modifier la valeur des poids de toutes les matrices de connexions avant d'avoir obtenu tous les gradients de la séquence, car le calcul des différents gradients nécessite la valeur originale des connections qui ont été utilisées pour la propagation avant.

Calcul récursif du gradient sur les activations des couches cachées

On peut calculer le gradient sur la couche cachée au temps M et sur son activation avant non-linéarité avec les formules:

$$\begin{aligned} \frac{\partial C}{\partial z^M} &= -2V'(c^M - y^M) \\ \frac{\partial C}{\partial a^M} &= \frac{\partial C}{\partial z^M} \star z^M \star (1 - z^M) \end{aligned}$$

où $\star$ indique un produit terme à terme entre deux vecteurs de même dimension.

Puis on peut récursivement calculer ces graidents pour la couche cachée au temps t à partir de ceux calculés au temps suivant $t+1$ avec les formules récursives suivantes:

$$\begin{aligned}\frac{\partial C}{\partial z^t} &= -2V'(c^t - y^t) + U'\frac{\partial C}{\partial a^{t+1}} \\ \frac{\partial C}{\partial a^t} &= \frac{\partial C}{\partial z^t} \star z^t \star (1 - z^t)\end{aligned}$$

Gradient sur les connexions menant à la couche cachée

Une fois que l'on sait calculer $\frac{\partial C}{\partial a^t}$ pour toute la séquence, il est aisé d'obtenir les gradients sur les paramètres du calcul de la couche cachée, à savoir w_0, W et U:

$$\begin{aligned}(\nabla C)_W &= \sum_{t=1}^{M} \frac{\partial C}{\partial a^t} \left(x^t\right)' \\ (\nabla C)_{w_0} &= \sum_{t=1}^{M} \frac{\partial C}{\partial a^t} \\ (\nabla C)_U &= \sum_{t=1}^{M} \frac{\partial C}{\partial a^t} \left(z^{t-1}\right)'\end{aligned}$$

Mise à jour des paramètres par descente de gradient

La mise à jour de tous les poids des matrice V, W et U ainsi que des vecteurs de biais w_0 et v_0 se calcule en multipliant le gradient qui lui correspond avec le taux d'apprentissage η. On soustrait ce résultat à la valeur initiale des paramètres pour diminuer l'erreur.

Pseudo-code

Le pseudo-code qui suit (algorithme 2) est équivalent à la propagation arrière que l'on vient de voir pour la mise à jour des différentes matrices de connexions et biais. Par souci de concision, on a choisi d'utiliser ici la notation ∇C_z pour $\frac{\partial C}{\partial z^t}$ et ∇C_a pour $\frac{\partial C}{\partial a^t}$.

3.5 Nouveau modèle expérimental: Auto-encodeur récurrent (RAE «*Récurrent Auto-Encodeur*»)

Notre technique d'apprentissage peut se voir comme une application de deux modèles différents sur une même séquence. C'est-à-dire un réseau de neurones récurrent suivi d'auto-encodeurs récurrents que nous allons voir plus loin dans cette section.

Algorithm 2 Pseudo-code de la propagation arrière pour un réseau de neurones récurrent

Initialisation des poids des matrices V, W et U avec des valeurs entre -1 et 1
Initialisation du vecteur gradTemp à 0 (avec le même nombre d'élément que la couche cachée)
Pour $n = 1$ jusqu'à N (Chaque séquence)
    $gradTemp \leftarrow 0$
    $\nabla C_V \leftarrow 0$, $\nabla C_{v_0} \leftarrow 0$
    $\nabla C_W \leftarrow 0$, $\nabla C_{w_0} \leftarrow 0$
    $\nabla C_U \leftarrow 0$
    Pour $t = M^n$ jusqu'à 1 (chaque entrée t de la séquence n en ordre décroissant)
        $c \leftarrow c^{nt}$
        $z \leftarrow z^{nt}$
        $z^{t-1} \leftarrow z^{n,t-1}$
        $x \leftarrow x^{nt}$
        $y \leftarrow y^{nt}$
        # vecteur de gradient pour y :
        $\nabla C_y = -2(c - y)$
        $\nabla C_V \leftarrow \nabla C_V + \nabla C_y \, z'$
        $\nabla C_{v_0} \leftarrow \nabla C_{v_0} + \nabla C_y$
        # vecteur de gradient pour z :
        $\nabla C_z = V' \nabla C_y + gradTemp$
        # vecteur de gradient sur l'activation avant non-linéarité de la couche cachée :
        $\nabla C_a = \nabla C_z * z * (1 - z)$
        $\nabla C_W \leftarrow \nabla C_W + \nabla C_a \, x'$
        $\nabla C_{w_0} \leftarrow \nabla C_{w_0} + \nabla C_a$
        $\nabla C_U \leftarrow \nabla C_U + \nabla C_a \, \left(z^{t-1}\right)'$
        $gradTemp \leftarrow U' \nabla C_a$
    FIN Pour t
    $V \leftarrow V - \eta \, \nabla C_V$
    $v_0 \leftarrow v_0 - \eta \, \nabla C_{v_0}$
    $W \leftarrow W - \eta \, \nabla C_W$
    $w_0 \leftarrow w_0 - \eta \, \nabla C_{w_0}$
    $U \leftarrow U - \eta \, \nabla C_U$
FIN Pour n

Ceci correspond à optimiser un coût qui est la somme du coût C du réseau récurrent ordinaire, cherchant à prédire la cible, et des coûts de reconstruction d'un autoencodeur récurrent qui tente de prédire l'état caché au temps $t-1$ à partir de l'état caché au temps t. Le gradient sur les paramètres du réseau devrait donc être la somme (possiblement pondérée) des gradients liés à ces deux types de coûts. Mais plutôt que de faire une mise à jour des poids avec un seul pas de gradient combinant les gradients des deux types de coûts, nous avons plutôt opté pour une mise à jour alternée des poids en utilisant en alternance le gradient d'un type de coût puis de l'autre, avec chacun leur taux d'apprentissage différent. Pour des taux d'apprentissage petits les deux approches reviennent au même déplacement en moyenne, notre approche en alternance pouvant être vue comme une variante davantage stochastique de la mise à jour unique.

Pour une séquence donnée de l'ensemble d'entraînement, on commence par appliquer le modèle que l'on vient de voir dans la section précédente (3.4), modifiant ainsi les poids en fonction de l'erreur de prédiction de la cible. Avec les nouveaux poids de cette première partie, on utilise des auto-encodeurs sur la même séquence, modifiant cette fois les poids en fonction de l'erreur de reconstruction de la couche cachée au temps t-1 à partir de la couche cachée au temps t.

Notre modèle fait du «*online learning*», c'est-à-dire que les paramètres du modèle sont ajustés après l'observation complète de chaque séquence S de façon à diminuer l'erreur d'entraînement. Le fonctionnement du modèle ressemble beaucoup à celui du réseau de neurones récurrent. Il faut débuter par l'application de la technique de propagation avant avec des auto-encodeurs.

3.5.1 Propagation avant et auto-encodeurs

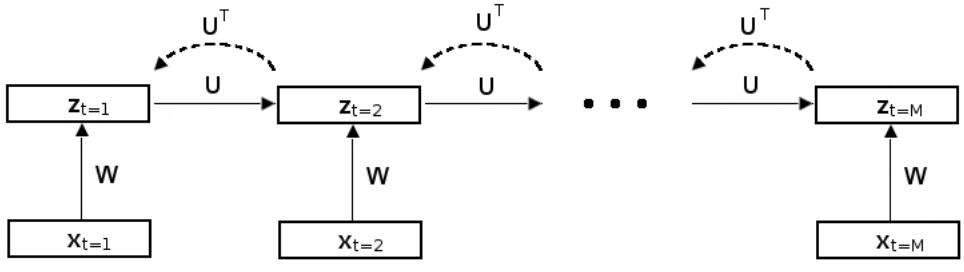

Figure 11: Auto-encodeur récurrent

On commence par faire la propagation expliquée dans la section 3.4.1, qui permet d'obtenir le déroulement dans le temps de toutes les prédictions de couches cachées. Cependant, à chaque moment t, on ne s'intéresse pas à la prédiction de la cible c^t à l'aide de la couche cachée z^t, mais plutôt à la reconstruction de la couche cachée précédente z^{t-1} qui est la partie de l'auto-encodeur. Pour ce faire, on propage le vecteur z^t, qui a déjà été calculé auparavant, à travers la transposée de la matrice dynamique U. La reconstruction est donc calculée en appliquant la sigmoïde sur le produit de U' avec z^t auquel on a ajouté un vecteur de biais u_0.

$$\hat{z}^{t-1} = sigmoïde(U'z^t + u_0) \tag{24}$$

où $\hat{z}$ représente la reconstruction de la couche cachée. Le déroulement dans le temps de toutes les prédictions et de toutes les reconstructions de couches cachées est gardé en mémoire dans des listes pour être utilisé dans l'étape suivante qui permet la mise à jour des différents poids du modèle. Il faut noter que l'on pourrait également vouloir reconstruire l'entrée à partir de la couche cachée (voir variante #4 plus bas).

3.5.2 Propagation arrière dans le temps («*Backprop through time*»)

La fonction d'erreur utilisée dans cette partie du modèle correspond à l'erreur de reconstruction de la couche cachée précédente pour chaque élément de la séquence observée. On utilise les auto-encodeurs pour forcer notre connection dynamique à être capable de reconstruire son entrée, ce qui pourrait permettre d'avoir plus d'informations utiles qui passent dans le temps. La fonction d'erreur utilisée pour la reconstruction est simplement la cross-entropy.

$$CoûtRec(\hat{z}^t) = -\sum_h z_h^t \log \hat{z}_h^t + (1 - z_h^t) \log(1 - \hat{z}_h^t) \tag{25}$$

$$CR = CoûtRec(\hat{z}^1, \hat{z}^2, ..., \hat{z}^{M-1}) \tag{26}$$

La propagation arrière du gradient de la reconstruction de la couche cachée est expliquée en trois étapes.

Étape 1

Contrairement à la propagation arrière vue dans la sous-section 3.4.2, l'erreur ne provient pas des différentes prédictions y de cibles générées par le modèle. La matrice de poids V n'est pas utilisée et elle ne subira donc pas de mise à jour. Pour tous les moments t de la séquence, on utilise la prédiction $\hat{z}^t$ pour calculer le gradient de chaque connection de la matrice transposée U' à l'étape de la reconstruction seulement.

$$(\nabla CR)_{U'_{k,h}} = \sum_{t=2}^{M} (\hat{z}_k^{t-1} - z_k^{t-1}) z_h^t \tag{27}$$

Ces gradients sont gardés en mémoire et seront utilisés pour mettre U' à jour seulement quand tous les gradients du modèle déroulé dans le temps seront calculés.

Étapes 2 et 3

Ces étapes sont les mêmes que dans la sous-section 3.4.2 vue précédemment mais avec la fonction de coût CR et la matrice U'. En fait, on peut considérer l'auto-encodeur récurrent comme un réseau de neurones dynamique avec la prédiction y^t et la cible c^t remplacée par la couche cachée z^{t-1} et la reconstruction $\hat{z}^{t-1}$ du moment précédent. La matrice V est remplacée par la transposée de la matrice dynamique U'. Le graphique suivant illustre cette vision et permet de voir pourquoi ce sont les mêmes dernières étapes que dans la sous-section précédente.

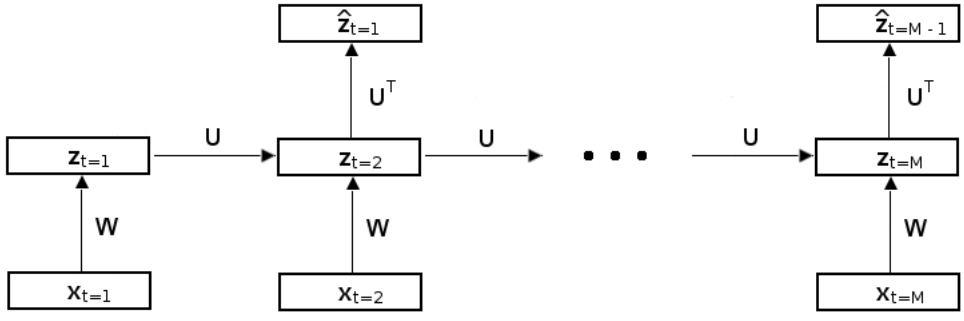

Figure 12: Auto-encodeur récurrent visualisé comme un réseau de neurones dynamique

Nous allons maintenant présenter brièvement quelques modèles qui sont des variantes de ces auto-encodeurs récurrents. Ces différents modèles seront numérotés afin de simplifier la présentation de leurs performances dans la section réservée aux résultats.

3.6 Bruit de la couche cachée #1 (NRAE «*Noisy Recurrent Neural Net*»)

La prédiction de la couche cachée $\hat{z}^t$ est calculée avec $\tilde{z}^t$ qui correspond à la couche cachée z^t que l'on a bruitée [20] . Ceci est fait en appliquant une certaine probabilité de corruption pour chaque élément de z^t. Le fait de corrompre ou non un élément est décidé par un tirage selon une loi de Bernoulli de moyenne π. Les éléments corrompus sont simplement mis à zéro. Il faut noter que la cible z^t n'est pas modifiée.

$$z^{t+1} = sigmoïde(\underbrace{U\tilde{z}^t + Wx^{t+1} + w_0}_{a^t}) \tag{28}$$

$$\hat{z}^t = sigmoïde(U'z^{t+1} + u_0) \tag{29}$$

3.7 Matrice de reconstruction non liée #2 (RAE-untied)

La reconstruction de la couche cachée $\hat{z}^t$ n'est pas calculée en utilisant la transposée de la matrice de poids dynamique U. On se sert d'une autre matrice de connection B qui est indépendante de U et qui a les mêmes dimensions.

$$\hat{z}^t = sigmoïde(Bz^{t+1} + u_0) \tag{30}$$

3.8 Reconstruction locale #3 (RAE-local)

L'aspect temporel de la section 3.5.2 n'est pas calculé pour la reconstruction de la couche cachée. Ceci a pour conséquence de couper l'influence temporelle de chacune des reconstructions et de n'avoir que des apprentissages locaux. Il n'y a donc que l'étape du réseau de neurones récurrent qui s'occupe de l'aspect temporel.

3.9 Reconstruction de l'entrée #4 (RAE-input)

Ici on ne veut pas reconstruire la couche cachée précédente z^{t-1} à partir de z^t, on veut plutôt reconstruire l'entrée x^t. Il s'agit d'auto-encodeurs appliqués sur les entrées. Dans cette variante, on fait exactement la même chose que dans la section 3.5.2 des auto-encodeurs récurrents. Cependant, on remplace chaque cible z^{t-1} par l'entrée x^t. On

remplace aussi la matrice de connection U' par W' et on inclut un vecteur de biais a_0. Ceci donne comme erreur de reconstruction la cross-entropy suivante:

$$\hat{x}^t = softmax(W' z^t + a_0) \tag{31}$$

$$erreurReconstruction = -\sum_j x_j^t \log \hat{x}_j^t \tag{32}$$

$\hat{x}^t$correspond à la reconstruction de l'entrée x au moment t. Il faut noter qu'on peut aussi combiner les objectifs de reconstruction de l'entrée et de reconstruction de la couche cachée précédente, comme indiqué ci-dessous.

3.10 Utilisation des auto-encodeurs comme préentraînement #5 (RAE-RNN)

On veut utiliser les reconstructions des couches cachées et des entrées pour initialiser les poids du modèle qui seront ensuite employés par des réseaux de neurones récurrents pour finaliser l'apprentissage. Expliquons cette technique en deux étapes.

Premièrement, on applique la technique de la reconstruction de la couche cachée (3.5) et celle de la reconstruction de l'entrée (3.9). On fait donc un certain nombre d'époques en fonction des erreurs de reconstruction. Les réseaux de neurones récurrents ne sont pas utilisés dans cette partie. On aurait pu sommer les gradients des deux coûts, mais cela donnait de l'instabilité numérique.

Deuxièmement, on finalise l'apprentissage en entraînant seulement les réseaux de neurones récurrents pour la prédiction des cibles des données. Le réseau commence l'entraînement en utilisant les matrices de poids apprises dans la première étape.

Cette technique en 2 étapes contraste avec toutes les autres présentées où les coûts de reconstruction et de prédiction de cible sont optimisés simultanément. Nous verrons à la section 4 que la technique en 2 étapes fonctionne moins bien.

3.11 Reconstruction de la couche cachée $t-2$ #6 (RAE-t-2)

Comme pour l'auto-encodeur récurrent, on reconstruit la couche cachée z^{t-1} à l'aide de z^t. Cependant, on utilise la prédiction $\hat{z}^{t-1}$ pour reconstruire la couche cachée z^{t-2}.

$$\hat{z}^{t-2} = sigmoïde(U' \, sigmoïde(U' z^t + u_0) + u_0) \tag{33}$$

C'est l'erreur (cross-entropy) de la reconstruction $\hat{z}^{t-2}$ qui est utilisée pour l'apprentissage. On veut donc savoir si on peut améliorer le résultat en forçant le modèle à se souvenir d'un élément de la séquence qui se trouve plus loin dans le temps.

4 Expérience sur les auto-encodeurs récurrents

Dans cette section, nous allons tester les performances des modèles expérimentaux présentés dans la section 3. Nous parlons ici du modèle d'auto-encodeur récurrent et de ses variantes (3.5 à 3.11). Le réseau de neurones récurrent est utilisé comme modèle de comparaison (3.4). Deux différents ensembles de données artificielles sont utilisés pour tester les algorithmes d'apprentissage et aucune fenêtre temporelle n'est appliquée pour faciliter l'entraînement.

4.1 Données artificielles

Deux ensembles de données artificielles ont été utilisés et sont expliqués dans cette section. Pour les générer, nous nous sommes basés sur la procédure décrite dans le rapport technique de Sutskever et Hinton [19]. Ces données servent à savoir si les modèles peuvent apprendre à répéter une séquence ou une partie de séquence sur une période de temps relativement longue.

Le premier ensemble de données est considéré comme étant plus facile à apprendre que le deuxième. Il est composé d'un ensemble d'entraînement, de validation et de test constitués respectivement de 1000, 200 et 200 séquences. Une séquence est de longueur 60 avec l'alphabet {0, ..., 5}. Chaque séquence est de forme $x \bullet 0^{30} \bullet x$, où x est une sous-séquence de longueur 15 générée aléatoirement avec les symboles 1 à 5 (équiprobables). Les 30 zéros servent à séparer les répétitions de x - on suppose ce x unique pour chaque séquence (la probabilité de générer deux fois le même x est extrêmement faible).

Le deuxième ensemble de données, utilisé pour tester les capacités des différents modèles, est plus difficile à apprendre que le premier parce qu'il est composé de séquences où le nombre de zéros qui séparent les deux occurrences de x varie et est beaucoup plus grand. Les données sont, elles aussi, composées d'un ensemble d'entraînement, de validation et de test constitués respectivement de 1000, 200 et 200 séquences. La longueur d'une séquence varie entre 61 et 125 avec l'alphabet {-1,0, ..., 5}. Chaque séquence est de forme $x \bullet 0^{20+r} \bullet (-1) \bullet 0^{10} \bullet x$, où x est encore une sous-séquence de longueur 15 générée aléatoirement avec les symboles 1 à 5 et où r est une variable aléatoire discrète calculée à partir d'une distribution géométrique ayant une moyenne de 50. Le symbole (-1) sert à informer le modèle qu'il doit répéter x sous peu. On a donc ici des séquences

où le nombre d'éléments, qui séparent les deux x, varie entre 31 et 95.

Le but de tout ceci est de voir si les modèles peuvent apprendre à compter et surtout s'ils peuvent se souvenir, à long terme, de ce qu'ils doivent répéter.

4.2 Expériences pour le premier ensemble de données artificielles

Nous allons observer les comportements et les résultats des différents modèles qui ont eu pour tâche d'apprendre la logique des données artificielles du premier ensemble. Chaque élément des séquences est transformé en un vecteur binaire «*one hot*» de longueur six qui représente une des six classes possibles (0 à 5). Un tel vecteur correspond à une entrée au temps t pour les modèles.

Commençons par les résultats empiriques de tous les modèles pour l'ensemble d'entraînement, de validation et de test en utilisant la cross-entropie comme mesure de comparaison. On ne parle pas ici de la cross-entropie de toute une séquence, mais plutôt de la moyenne de la cross-entropie par élément de séquence. Le tableau qui suit représente les meilleurs résultats de validation obtenus pour chacun des modèles testés.

Modèle	Entraînement	Validation	Test
Réseau de neurones récurrent	0.7661	0.7758	0.7661
Auto-encodeur (RAE)	0.3830	0.3935	0.3950
NRAE (#1)	0.3882	0.3950	0.3961
RAE-untied (#2)	0.7134	0.7241	0.7277
RAE-local (#3)	0.3855	0.4141	0.4159
RAE-input (#4)	0.8183	0.8194	0.8207
RAE-RNN (#5)	0.7642	0.7789	0.7791
RAE-t-2 (#6)	0.3834	0.3945	0.3968

Table 2: Meilleur résultat de cross-entropie pour chaque modèle.

Dans ce tableau de résultats (tableau 2), plus la cross-entropie est petite, moins il y a d'erreurs dans les prédictions du modèle et meilleur est son apprentissage. Cependant, bien que cette mesure d'erreur permette de comparer les modèles entre eux, elle ne donne pas beaucoup d'indication sur la performance des prédictions des sous-séquences x.

Nous sommes principalement intéressés par la capacité qu'ont les modèles à prédire la deuxième occurrence du signal x (de longueur 15) pour une séquence artificielle. Le tableau qui suit représente le pourcentage de bonnes prédictions seulement pour la sous-partie x des séquences.

Modèle	Entraînement	Validation	Test
Réseau de neurones récurrent	26.70%	24.60%	23.03%
Auto-encodeur (RAE)	100%	100%	99.93%
NRAE (#1)	99.99%	100%	100%
RAE-untied (#2)	35.29%	32.50%	33.07%
RAE-local (#3)	99.95%	99.73%	99.73%
RAE-input (#4)	19.02%	18.57%	17.53%
RAE-RNN (#5)	28.71%	24.80%	23.70%
RAE-t-2 (#6)	99.99%	99.93%	99.80%

Table 3: Pourcentages de bonnes prédictions des sous-séquences x pour chaque modèle.

Afin d'être sûr d'obtenir des résultats de comparaison fiables, nous avons testé un grand nombre d'hyperparamètres pour le modèle de réseau de neurones récurrent. Entre 20 et 400 neurones ont été essayés pour la couche cachée ainsi que plusieurs taux d'apprentissage qui se situent entre 1e-2 et 1e-6. Pour chaque combinaison possible du nombre de neurones cachés et du taux d'apprentissage, cinq initialisations de poids différentes ont été appliquées. Un taux d'apprentissage d'environ 1e-3 et des couches cachées entre 40 et 180 neurones sont les hyperparamètres qui ont donné les meilleurs résultats pour le modèle de réseau de neurones récurrent.

L'auto-encodeur récurrent (RAE) présente trois hyperparamètres à optimiser: le nombre de couches cachées ainsi que deux taux d'apprentissage, le premier pour l'erreur en fonction de la prédiction de la cible et le second pour l'erreur de reconstruction de la couche cachée précédente. Un taux d'apprentissage d'environ 1e-3 pour le premier et d'environ 1e-4 pour le deuxième ainsi que des couches cachées entre 160 et 400 neurones sont les hyperparamètres qui ont donné les meilleurs résultats.

Le modèle d'auto-encodeur récurrent avec du bruit (NRAE #1) a un hyperparamètre de plus qui est le pourcentage de bruit appliqué aux couches cachées avant leur reconstruction. Un taux d'apprentissage d'environ 1e-3 pour le premier et d'environ 1e-4 pour le deuxième, des couches cachées entre 300 et 400 neurones ainsi que des pourcentages de bruit entre 0.1% et 10% sont les hyperparamètres qui ont fourni les meilleurs résultats.

La variante avec la matrice de reconstruction non liée (RAE-untied #2) a les mêmes hyperparamètres que l'auto-encodeur récurrent. Un taux d'apprentissage d'à peu près 1e-3 pour le premier et d'à peu près 1e-4 pour le deuxième ainsi que des couches cachées entre 60 et 300 neurones sont les hyperparamètres qui ont donné les meilleurs résultats.

La variante avec les auto-encodeurs locaux (RAE-local #3) a elle aussi les mêmes

hyperparamètres que l'auto-encodeur récurrent. Un taux d'apprentissage d'environs 1e-3 pour le premier et un autre entre 4e-4 et 1e-4 pour le deuxième ainsi que des couches cachées entre 200 et 400 neurones sont les hyperparamètres qui ont donné les meilleurs résultats.

Pour la variante qui applique les auto-encodeurs sur les entrées (RAE-input #4), le deuxième taux d'apprentissage est utilisé pour l'erreur de reconstruction de l'entrée. Un taux d'apprentissage d'environ 1e-4 pour le premier et un autre entre 2e-5 et 3e-6 pour le deuxième ainsi que des couches cachées entre 40 et 400 neurones sont les hyperparamètres qui ont donné les meilleurs résultats.

La variante qui utilise les reconstructions des couches cachées et des entrées pour initialiser les poids des matrices de connection (RAE-RNN #5) est composée de cinq hyperparamètres. Il s'agit du nombre de couches cachées, du nombre d'époques pour la première étape (expliquée dans la sous-section 3.10) qui sert à initialiser les matrices de connection, ainsi que de trois taux d'apprentissage: le premier pour l'erreur en fonction de la prédiction de la cible, le second pour l'erreur de reconstruction de la couche cachée précédente et le troisième pour l'erreur de reconstruction de l'entrée. Les hyperparamètres qui ont donné les meilleurs résultats sont un premier taux d'apprentissage d'environ 1e-3 , un deuxième autour de 1e-5 et un troisième d'environ 1e-5 ainsi que des couches cachées entre 50 et 200 neurones. Un fait important à observer est que tous les meilleurs résultats ont été obtenus avec les plus petits nombres d'époques testés, c'est-à-dire entre 5 et 25.

La dernière variante qui reconstruit la couche cachée au temps $t-2$ (RAE-t-2 #6) possède les mêmes hyperparamètres que l'auto-encodeur récurrent. Un taux d'apprentissage autour de 1e-3 pour le premier et autour de 1e-4 pour le deuxième ainsi que des couches cachées entre 150 et 400 neurones sont les hyperparamètres qui ont donné les meilleurs résultats.

Le critère d'arrêt pour tous les modèles correspond à un arrêt forcé de l'apprentissage si le modèle n'améliore pas l'erreur de cross-entropie pour l'ensemble de validation avant 200 époques.

Maintenant que nous avons constaté les performances des 8 modèles mentionnés dans les tableaux de résultats 2 et 3, il pourrait être intéressant d'observer les courbes

d'apprentissage ainsi que les prédictions de chacun d'entre eux. Cependant, pour ne pas rendre cette section trop lourde pour le lecteur, nous allons nous concentrer seulement sur les modèles que nous jugeons être les plus pertinents.

La figure qui suit représente les courbes d'apprentissage de l'ensemble de test pour le réseau de neurones récurrent, pour l'auto-encodeur récurrent et pour les variantes #1 (NRAE) et #6 (RAE-t-2) qui ont donné les meilleurs résultats pour l'ensemble de validation (tableaux 2 et 3).

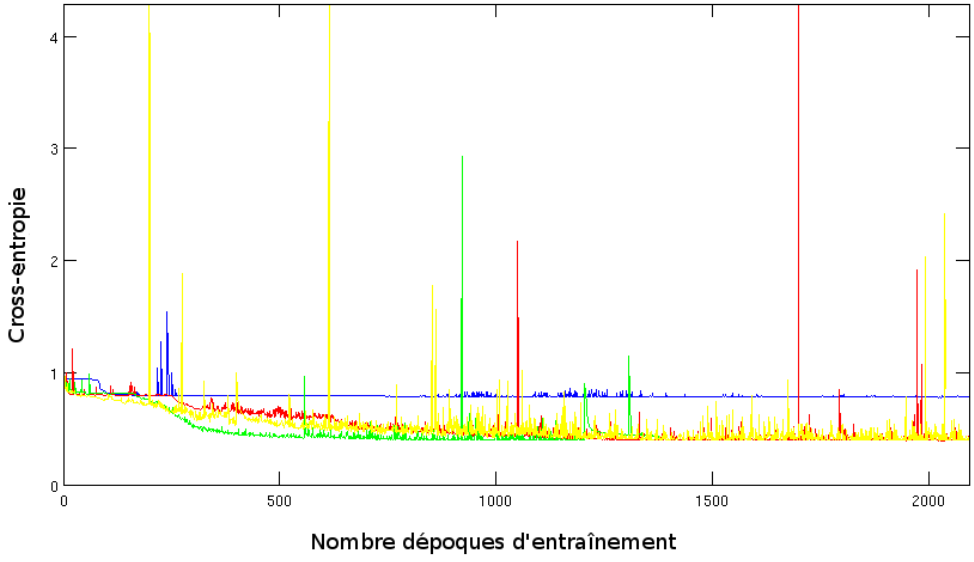

Figure 13: Courbes d'apprentissage pour l'ensemble de test avec le réseau de neurones récurrent en bleu, l'auto-encodeur récurrent (RAE) en rouge, la variante #1 (NRAE) en vert et la variante #6 (RAE-t-2) en jaune. L'axe des X représente le nombre d'époques et l'axe des Y représente l'erreur de la cross-entropie.

Cette illustration de l'erreur en fonction du temps d'apprentissage montre la grande variation de l'erreur pour les modèles expérimentaux par rapport à celle du modèle de comparaison.

Tous ces résultats ne nous informent cependant pas sur les capacités de prédiction des réseaux récurrents. Pour ce faire, nous allons observer les estimations des modèles avec une séquence de l'ensemble de test en entrée. Nous ne le ferons cependant pas pour les variantes #1 (NRAE) et #6 (RAE-t-2), car elles ont le même comportement que l'auto-encodeur récurrent (RAE). Commençons par le réseau de neurones récurrent.

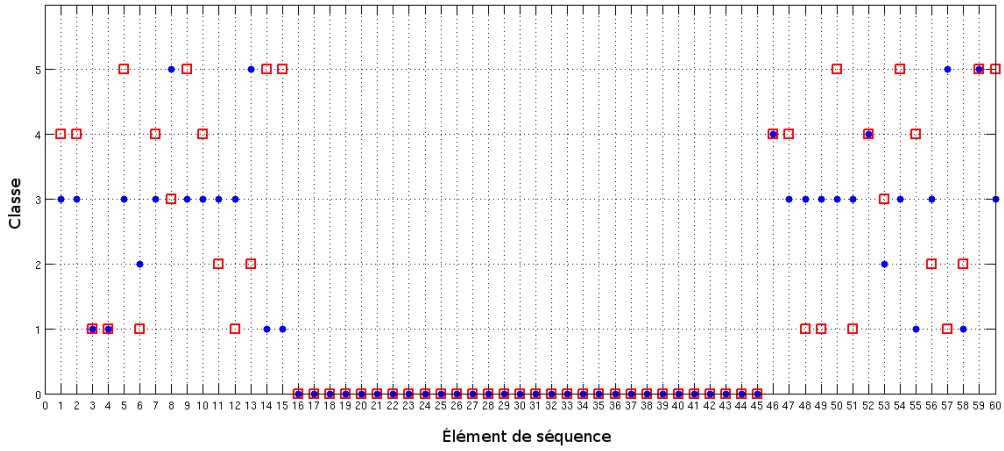

Figure 14: Prédictions du réseau de neurones récurrent, avec la cible en carrés rouges et la prédiction en points bleus

Dans la figure 14, la cible en rouge (carrés) correspond à une séquence du type $x \bullet 0^{30} \bullet x$. La prédiction du modèle est en bleu (points). On peut voir que le réseau récurrent a appris à compter. Il faut d'abord faire abstraction des erreurs du début, car le modèle ne peut pas prédire un signal qu'il n'a pas encore vu. Nous disons que le modèle a appris à compter parce qu'il sait quand le signal et le silence commencent et finissent. Cependant, le modèle est incapable de répéter le signal après les 30 zéros et sa prédiction pour celui-ci est même très mauvaise. Observons maintenant les capacités de prédiction de l'auto-encodeur récurrent.

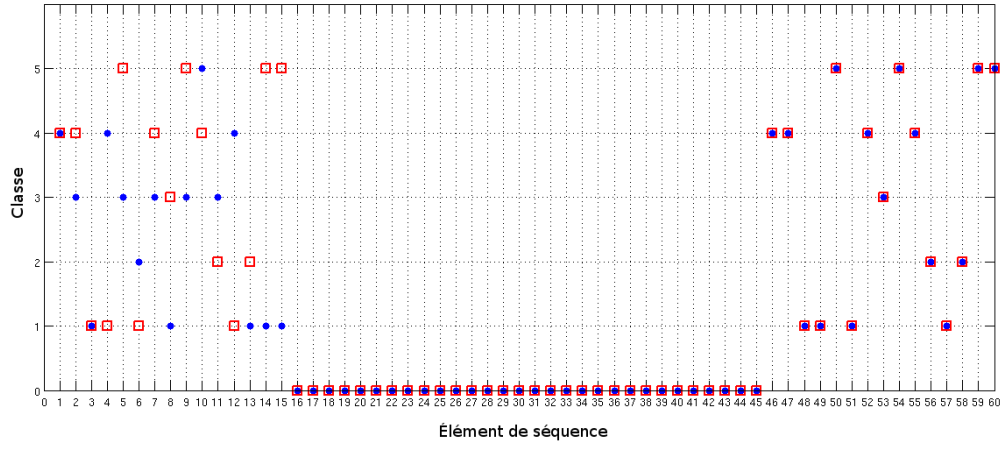

Figure 15: Prédictions de l'auto-encodeur récurrent, avec la cible en carrés rouges et la prédiction en points bleus

Les résultats de la figure 15 nous permettent de constater, avec joie, que l'auto-encodeur récurrent obtient une bien meilleure prédiction du signal que le modèle de comparaison. Pour mieux comprendre où se trouve la limite de la capacité d'apprentissage des modèles, nous allons maintenant oberver leurs comportements sur le deuxième ensemble de données artificielles qui est plus complexe.

4.3 Expériences pour le deuxième ensemble de données artificielles

Nous allons encore une fois observer les comportements et les résultats des différents modèles, mais cette fois-ci ce sera pour apprendre la logique un peu plus complexe du deuxième ensemble de données artificielles. Chaque élément des séquences est transformé en un vecteur binaire «*one hot*» de longueur sept qui représente une des sept classes possibles (-1, 0, 1, 2,..., 5). Un tel vecteur correspond à une entrée au temps t pour les modèles.

Comme pour la section 4.2, nous allons observer les résultats empiriques de tous les modèles pour l'ensemble d'entraînement, de validation et de test en utilisant la cross-entropie comme mesure de comparaison. Voici les meilleurs résultats de validation obtenus pour chacun des modèles testés.

Modèle	Entraînement	Validation	Test
Réseau de neurones récurrent	0.5241	0.5304	0.5361
Auto-encodeur (RAE)	0.5050	0.5146	0.5212
NRAE (#1)	0.4995	0.5202	0.5258
RAE-untied (#2)	0.5238	0.5301	0.5357
RAE-local (#3)	0.5074	0.5274	0.5302
RAE-input (#4)	0.6054	0.6111	0.6186
RAE-RNN (#5)	0.5232	0.5305	0.5359
RAE-t-2 (#6)	0.4578	0.4832	0.4868

Table 4: Meilleur résultat de cross-entropie pour chaque modèle

Dans le tableau de résultats (tableau 4), on peut voir que le modèle #6 (RAE-t-2) a obtenu le meilleur résultat avec la plus petite cross-entropie. Cependant, cette mesure d'erreur ne donne toujours pas beaucoup d'indication sur la performance des prédictions des signaux x (de longueur 15) pour les séquences artificielles. Observons maintenant le pourcentage de bonnes prédictions seulement pour la sous-partie x des séquences.

Modèle	Entraînement	Validation	Test
Réseau de neurones récurrent	20.98%	19.70%	19.47%
Auto-encodeur (RAE)	32.97%	31.17%	30.96%
NRAE (#1)	34.03%	29.10%	29.67%
RAE-untied (#2)	21.73%	19.93%	20.70%
RAE-local (#3)	32.22%	28.87%	28.73%
RAE-input (#4)	18.63%	17.20%	18.83%
RAE-RNN (#5)	21.91%	18.00%	20.63%
RAE-t-2 (#6)	44.75%	37.37%	38.70%

Table 5: Pourcentages de bonnes prédictions des sous-séquences x pour chaque modèle.

Un taux d'apprentissage entre 1e-3 et 1e-4 et des couches cachées entre 40 et 400 neurones sont les hyperparamètres qui ont donné les meilleurs résultats pour le modèle de réseau de neurones récurrent.

L'auto-encodeur récurrent (RAE) a trois hyperparamètres à optimiser. Les meilleurs résultats sont obtenus avec un premier taux d'apprentissage autour de 1e-3 pour l'erreur en fonction de la prédiction de la cible et un deuxième taux entre 3e-5 et 5e-6 pour l'erreur de reconstruction de la couche cachée précédente ainsi que des couches cachées entre 140 et 400 neurones.

Le modèle d'auto-encodeur récurrent avec du bruit (NRAE #1) présente quatre hyperparamètres à optimiser. Les meilleurs résultats sont obtenus avec un premier taux d'apprentissage autour de 1e-3 et un deuxième taux autour de 1e-5, des couches cachées entre 150 et 400 neurones ainsi que des pourcentages de bruit entre 0.1% et 4% appliqués aux couches cachées avant leur reconstruction.

La variante avec la matrice de reconstruction non liée (RAE-untied #2) offre trois hyperparamètres à optimiser. Les meilleurs résultats sont obtenus avec un premier taux d'apprentissage d'environ 1e-4 et un deuxième taux d'environ 1e-7 ainsi que des couches cachées entre 40 et 200 neurones.

La variante avec les auto-encodeurs locaux (RAE-local #3) a elle aussi trois hyperparamètres à optimiser. Les meilleurs résultats sont obtenus avec un premier taux d'apprentissage autour de 1e-3 et un deuxième taux autour de 1e-6 ainsi que des couches cachées entre 250 et 400 neurones.

Pour la variante qui applique les auto-encodeurs sur les entrées (RAE-input #4), trois hyperparamètres doivent être optimisés. Les meilleurs résultats sont obtenus avec un premier taux d'apprentissage autour de 1e-4 et un deuxième taux entre 1e-5 et 1e-7, utilisé cette fois-ci pour l'erreur de reconstruction de l'entrée, ainsi que des couches cachées entre 60 et 300 neurones.

La variante qui utilise les reconstructions des couches cachées et des entrées pour initialiser les poids des matrices de connection (RAE-RNN #5) est composée de cinq hyperparamètres qui sont déjà détaillés dans la section 4.2 . Les meilleurs résultats sont obtenus avec un premier taux d'apprentissage autour de 1e-4 , un deuxième entre 1e-4 et 1e-5 et un troisième entre 1e-4 et 1e-5 ainsi que des couches cachées entre 40 et 200 neurones. Le plus important est d'observer le fait que les meilleurs résultats ont été encore une fois obtenus avec les plus petits nombres d'époque testés, c'est-à-dire entre 1 et 10.

La dernière variante qui reconstruit la couche cachée au temps $t-2$ (RAE-t-2 #6) a les trois mêmes hyperparamètres que l'auto-encodeur récurrent. Les meilleurs résultats sont obtenus avec un premier taux d'apprentissage autour de 1e-4 pour le premier et autour de 1e-6 pour le deuxième ainsi que des couches cachées entre 200 et 400 neurones.

Observons maintenant les courbes d'apprentissage et les prédictions de certains modèles. La figure qui suit représente la courbe d'apprentissage de l'ensemble de test pour le réseau de neurones récurrent, pour l'auto-encodeur récurrent (RAE) et pour les variantes #1 (NRAE) et #6 (RAE-t-2) qui ont donné les meilleurs résultats pour l'ensemble de validation (tableaux 4 et 5).

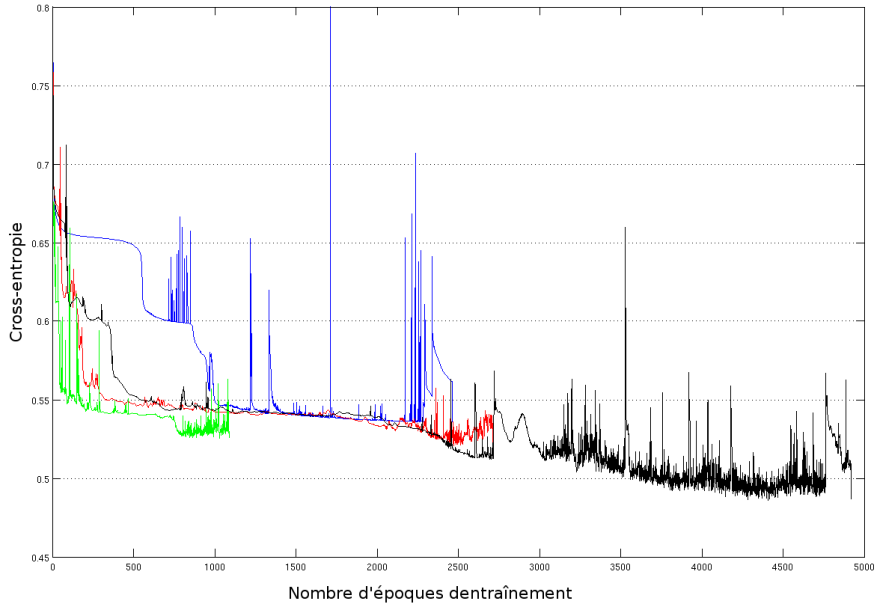

Figure 16: Courbes d'apprentissage pour l'ensemble de test avec le réseau de neurones récurrent en bleu, l'auto-encodeur récurrent (RAE) en rouge, la variante #1 (NRAE) en vert et la #6 (RAE-t-2) en noir. L'axe des X représente le nombre d'époques et l'axe des Y représente l'erreur de la cross-entropie.

Poursuivons avec les prédictions, pour une séquence de l'ensemble de test en entrée, d'un réseau de neurones récurrent ainsi que celles de la variante #6 (RAE-t-2) qui est le modèle ayant donné les meilleurs résultats dans le tableau 5.

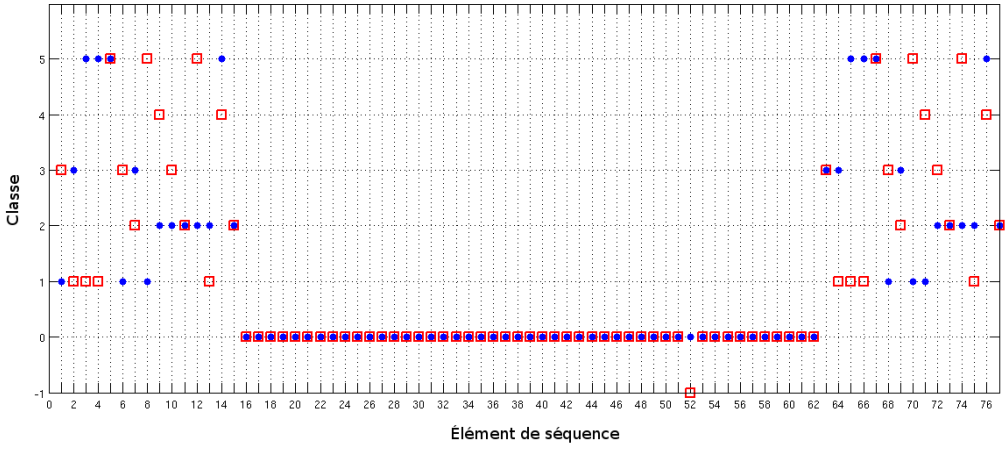

Figure 17: Prédictions du réseau de neurones récurrent, avec la cible en carrés rouges et la prédiction en points bleus.

Dans la figure 17, avec la prédiction en points bleus et la cible en carrés rouges, on

peut voir que le modèle récurrent a appris quand le silence se termine, mais ne sait pas comment répéter le signal x. Observons maintenant le second modèle.

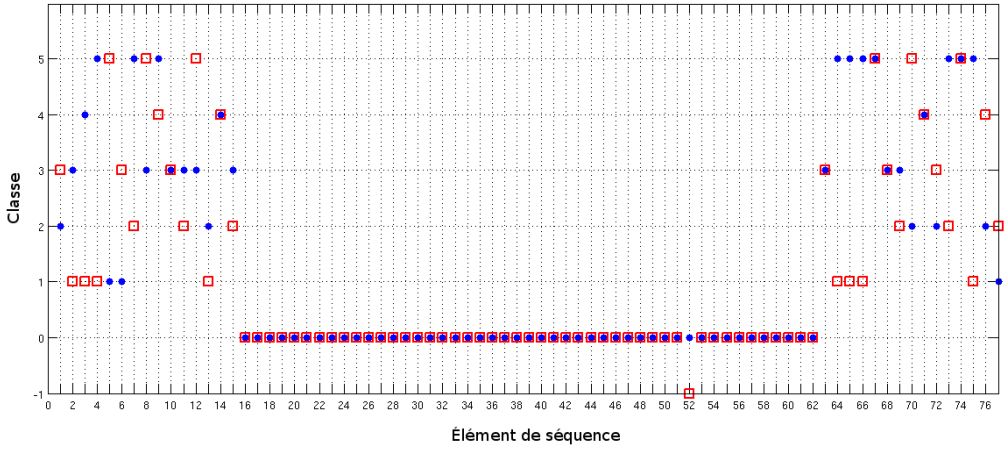

Figure 18: Prédictions de l'auto-encodeur récurrent, avec la cible en carrés rouges et la prédiction en points bleus.

La figure 18 ne permet pas de constater une meilleure prédiction du signal x pour l'utilisation d'auto-encodeurs. Cependant, les résultats du tableau 5 démontrent une amélioration significative de certains modèles expérimentaux par rapport au modèle de référence (RNN), surtout pour la variante #6 (RAE-t-2).

Le deuxième ensemble de données artificielles a juste la bonne complexité pour tester les limites de l'apprentissage des plus puissants modèles temporels dont certains ont été testés à l'université de Toronto.

4.4 Résultats d'autres modèles

Sutskever et Hinton ont appliqué leur modèle TKRNN *«Temporal-Kernel Recurrent Neural Networks»* [19] sur les mêmes deux ensembles de données artificielles. Les modèles LSTM *«Long-Short Term Memory»* [11] et TDNN *«Time Delay Neural Network»* [22] ont été utilisés à des fins de comparaison.

Le LSTM n'a pas réussi à comprendre la logique de répétition pour les deux ensembles de données artificielles. Cependant, pour le premier ensemble, le modèle a réussi à apprendre que le nombre de 0 qui sépare les deux instances de x est de 30.

Le TDNN a parfaitement résolu le problème du premier ensemble, mais n'a pas réussi

pour le deuxième.

Le TKRNN a lui aussi résolu le premier problème parfaitement. Pour le deuxième problème, le modèle a réussi à prédire correctement 76% de la deuxième instance du signal x.

Il faut toutefois noter qu'il y a des différences entre la méthodologie employée par Sutskever [19] et la nôtre. Premièrement, les chercheurs de Toronto ne propagent pas le gradient pour l'erreur de prédiction de la première instance de x, ce qui diminue la variance des mises-à-jour et facilite possiblement l'apprentissage. Deuxièmement, nous avons utilisé un ensemble de données de taille fixe alors que dans leurs expériences chaque séquence donnée au modèle pour l'entraînement est une nouvelle séquence générée. Dans leur cas, il ne peut donc pas y avoir de surentraînement. Cette différence dans la méthodologie s'explique par le fait que nous voulons utiliser nos modèles de la même façon que pour des données réelles, même si cela rend la tâche plus difficile.

4.5 Discussion

Le réseau de neurones récurrent nous a surpris par ses capacités. En effet, les résultats du premier problème (fig.14) montrent que ce modèle peut compter jusqu'à 30 étapes de temps. De plus, pour le deuxième problème, il comprend aussi que la répétition de la classe 0 se termine dix étapes après l'apparition de la classe -1. Cependant, le réseau ne parvient pas à saisir la dépendance entre les deux occurences de x pour les ensembles de données, ce qui est sûrement lié au problème de perte de l'influence du gradient dans le temps [29].

Les résultats de l'auto-encodeur récurrent prouvent que l'utilisation d'auto-encodeurs dans un modèle temporel peut améliorer l'apprentissage de façon significative. Le fait de pousser la matrice dynamique du réseau à reconstruire son entrée force celle-ci à se souvenir du passé, ce qui semble avantager la propagation de l'information temporelle. Toutefois, le poucentage de bonne prédiction pour la seconde occurence de x dans le deuxième ensemble de données n'est que de 31% environ, ce qui n'est que 11% de plus que le hasard. Une petite partie de la logique de répétition paraît donc avoir été comprise, mais le nombre d'étapes qui sépare les deux x de certaines séquences semble juste trop grand pour les capacités du modèle. Il est aussi intéressant de constater que le

taux d'apprentissage pour la reconstruction de la couche cachée représente toujours une fraction entre 10% et 0.5% du taux d'apprentissage de l'erreur de la cible.

La courbe verte de la figure 13 et surtout celle de la figure 16 nous pousse à croire que l'utilisation de bruit (#1 NREA) dans la reconstruction de la couche cachée accélère l'apprentissage. On parle ici d'une diminution significative du nombre d'époques nécessaires pour minimiser l'erreur de prédiction de la cible. Cependant, le bruit semble ne pas permettre au réseau d'atteindre une performance aussi optimale que lorsqu'il n'y en a pas, mais il ne s'agit là que d'une petite différence. Il faut noter que le pourcentage de bruit optimal pour la reconstruction est toujours relativement faible, entre 0.1% et 4%.

Le fait d'utiliser une deuxième matrice (#2 RAE-untied), indépendante de la matrice dynamique, pour reconstruire l'entrée de la connection temporelle ne donne définitivement pas d'aussi bons résultats que l'auto-encodeur récurrent de base. Malgré tout, la légère amélioration dans le premier problème, par rapport au modèle de comparaison, montre que le fait de forcer le modèle à prédire le passé améliore la propagation de l'information temporelle.

Dans la variante #3 (RAE-local), le gradient de la reconstruction de la couche cachée n'est pas propagé dans le temps, ce qui ne l'empêche pas d'améliorer considérablement ses prédictions, même si les résultats sont toujours un peu moins bons que l'auto-encodeur récurrent de base. Ce modèle nous permet de constater que la majorité de l'apprentissage se fait au début de la propagation du gradient.

La façon dont on a utilisé la reconstruction de l'entrée (#4 RAE-input) n'a pas donné de bonnes performances. Il semble que cette technique favorise trop l'information locale au détriment de l'information temporelle à court et à long terme. Ceci pourrait expliquer le fait que les résultats soient plus faibles que ceux du modèle de comparaison. Cependant, il ne faut pas écarter la possibilité qu'il puisse exister d'autres façons d'utiliser la reconstruction de l'entrée pour améliorer l'apprentissage.

Le DBN «Deep Belief Net» [9], qui est la base de notre inspiration, fonctionne par niveau d'apprentissage. Ces niveaux sont entraînés l'un après l'autre afin d'initialiser les poids des matrices de connection pour ensuite faire séparément la mise-à-jour globale du réseau en fonction de la cible. Nous avons donc voulu faire de même pour les modèles temporels et initialiser les poids des matrices de cette variante (#5 RAE-RNN) en

utilisant les reconstructions des couches cachées et des entrées (préentraînement) pour ensuite faire séparémemt la mise-à-jour globale du réseau récurrent en fonction de la cible. Malheureusement, cette technique a échoué, car les meilleurs résultats, qui ne surpassent pas ceux du modèle de comparaison, ont été obtenus avec peu de préentraînement.

Le fait de reconstruire une couche cachée plus loin dans le passé (#6 RAE-t-2) semble améliorer les performances de prédictions du signal x comparativement à celles de l'auto-encodeur récurrent de base. Ces résultats plutôt étonnants nous poussent à croire qu'il est peut-être possible d'améliorer encore l'apprentissage du modèle en recontruisant des couches cachées qui se situent plus loin dans le passé.

La variabilité dans l'apprentissage des modèles est peut-être partiellement causée par la propagation du gradient de l'erreur de la prédiction de la première occurence de x. En effet, cette occurence ne peut pas être prédite, car le modèle ne l'a encore jamais vue. Il est cependant important de pouvoir surmonter ce genre de difficulté qui est inévitable dans des données réelles comme la musique.

Une des découvertes majeures de cette recherche (obtenue en comparant les résultats du modèle RAE-RNN, #5, avec ceux du RAE) est le fait que l'utilisation de la reconstruction des couches cachées pour la mise à jour des poids des connections doivent se faire en même temps que la mise à jour des poids du modèle en fonction de la cible. Cette découverte est plutôt surprenante si on considère le fait que le DBN, qui nous a inspirés, fonctionne différemment en séparant complètement les deux sortes de mise-à-jour. Cependant, il y a peut-être d'autres liens à faire: les niveaux d'apprentissage du DBN pourraient être comparés, pour le modèle temporel, à une utilisation par étape de différentes reconstructions dans le temps. Par exemple, on commence par entraîner le modèle de la même façon que l'auto-encodeur récurrent (on reconstruit la couche cachée au moment $t-1$), après un certain nombre d'époques pour la première étape. On passe ensuite à la suivante qui doit reconstruire la couche cachée au moment $t-2$ (variante #6) et, là encore, on fait un certain nombre d'époques. On continue en reconstruisant toujours plus loin dans le passé pour un nombre d'étapes prédéterminées. Le fait que cela puisse améliorer l'apprentissage n'est qu'une hypothèse et devra être vérifié dans de futures recherches.

Il serait aussi intéressant de voir si l'apprentissage du premier ensemble de données

artificielles peut aider un modèle à comprendre la logique plus complexe du deuxième ensemble. En effet, les deux ensembles sont parfaits pour ce genre de test, car la logique de répétition est la même. De cette façon, un modèle n'aurait pas à apprendre tout d'un seul coup, mais un petit peu à chaque ensemble. On pourrait ainsi rajouter d'autres ensembles avec une logique toujours un peu plus complexe pour se rendre finalement jusqu'à la complexité de données réelles comme celles pour la génération de partition de musique. Encore là, il ne s'agit que d'une hypothèse à vérifier [2].

Finalement, les figures 19 et 20, qui suivent, permettent de visualiser l'évolution dans le temps de la couche cachée du réseau de neurones récurrent (fig.19) et de l'auto-encodeur récurrent (fig.20). Les deux sont appris pour le premier ensemble de données artificielles et appliqués sur une séquence test. Chaque image montre la séquence artificielle dans les 6 premières lignes, la prédiction du modèle dans les 6 suivantes et l'état de la couche cachée dans le reste des lignes. La comparaison des figures montre que l'utilisation des auto-encodeurs produit une profonde différence entre les deux représentations internes apprises. Notons que la différence entre la grandeur de couche cachée des deux modèles est due à une question d'optimisation des hyperparamètres.

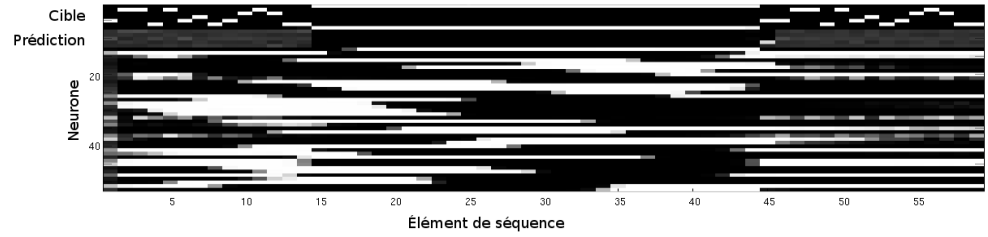

Figure 19: Visualisation de l'évolution dans le temps de la couche cachée du réseau de neurones récurrent. L'image montre la séquence artificielle dans les 6 premières lignes, la prédiction du modèle dans les 6 suivantes et l'état de la couche cachée dans le reste des lignes.

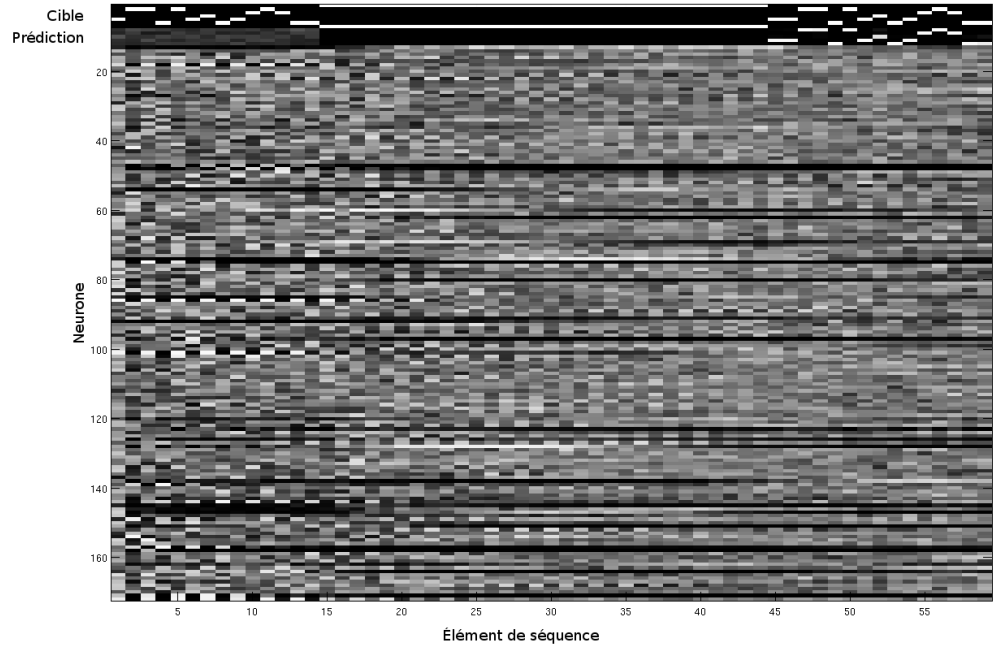

Figure 20: Visualisation de l'évolution dans le temps de la couche cachée de l'auto-encodeur récurrent. L'image montre la séquence artificielle dans les 6 premières lignes, la prédiction du modèle dans les 6 suivantes et l'état de la couche cachée dans le reste des lignes.

5 Apprentissage machine appliqué aux performances expressives

Dans cette section, nous présentons d'abord les données représentant les performances humaines de cinq pièces de musique. Nous exposons ensuite les résultats obtenus par des modèles statistiques pour l'apprentissage des variations expressives qui composent ces données.

Les modèles expérimentaux que nous avons vus dans le chapitre 4 ne sont pas présentés dans cette section sauf pour le RAE (3.5). En effet, les données expressives de ce chapitre ne sont pas adéquates pour comparer les capacités des modèles expérimentaux qui ne donnent pas de meilleurs résultats que les NN et les RNN utilisés plus bas et introduits dans le chapitre 3.

5.1 Les données

5.1.1 Information générale

Afin d'obtenir nos données, nous avons enregistré les performances de treize (13) pianistes interprétant chacun cinq (5) valses de Schubert. Pour ce faire, nous avons utilisé le piano impérial Bösendorfer 290SE contrôlé par ordinateur à l'aide de son système «CEUS». Presque toutes les pièces ont été enregistrées deux (2) fois par pianiste, ce qui donne un ensemble de données de cent vingt quatre (124) performances représentant une heure quarante de temps et plus de quarante-cinq mille (45 000) notes de musique jouées. Nous avons aussi créé pour chaque pièce la partition de référence en fichier MIDI à fin de comparaison. En utilisant un programme MATLAB de comparaison midi [13], nous avons extrait toute l'information nécessaire pour chaque note jouée dans les performances de l'ensemble de données. La partition qui suit est une des cinq pièces de Schubert utilisées pour les enregistrements et que l'on peut retrouver dans l'annexe du document.

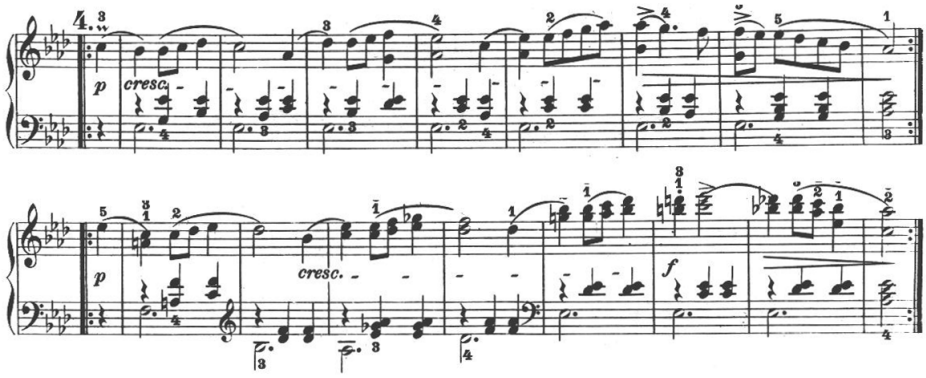

Figure 21: Valse de Schubert Op 9a no 4. Cette partition est une des cinq partitions jouées par les pianistes.

5.1.2 Perspective sur les données, point de vue, onset

Pour commencer, il faut voir les données comme étant des séries séquentielles. Dans notre cas, on a des pièces de musique qui peuvent se découper en «onset». Un onset est un terme anglais dont le sens est le suivant: un moment où une note ou encore un groupe de notes de musique est joué, sachant qu'une note doit appartenir à un seul onset à la fois. La pièce de musique est donc découpée en une multitude de regroupements de notes, que l'on nommera «onsets» et qui se suivront les uns après les autres. Cette façon de procéder permet d'analyser plus facilement la variation des dimensions entre les onsets, c'est-à-dire la période des onsets, la vélocité des notes jouées et la déviation de temps local à l'intérieur d'un onset. Ces dimensions sont responsables de l'expressivité de la pièce.

Figure 22: Mesure découpée par des rectangles où chaque rectangle représente un «onset».

5.1.3 Période

Nous allons utiliser trois sortes de temps de note: la croche, la noire et la blanche qui valent respectivement un demi temps, un temps et deux temps. L'information principale que l'on désire apprendre est ce que nous appellerons la période (*«inter beat intervals»*, *seconde/temps*), qui est exactement l'inverse du tempo (*temps/seconde*). On peut voir la période comme étant la durée (mesurée avec une précision en millisecondes) d'une note noire à un moment précis dans la performance d'une pièce musicale. Logiquement, dans une partition, la durée d'une noire devrait toujours être la même tout au long de la pièce. Mais en réalité, un interprète ne donne pas toujours la même valeur en millisecondes à une noire. En fait, il la fait varier pendant qu'il joue la pièce. C'est cette variation du temps qui contribue principalement à l'expressivité de la performance. On l'appelle en anglais «*expressive timing*» et nous proposons de traduire ce terme par «déviation expressive du temps». Pour nos données, on calcule la période à chaque onset, ce qui veut dire qu'on détermine la durée d'une noire pour chaque onset. Soit $t = 1, 2, ..., N$ les différents onsets (en ordre) d'une pièce de musique. La période du onset t correspondra au quotient de la durée de la performance (différence en millisecondes entre le commencement des onsets t et t+1 joués) avec la durée de la partition (différence en temps entre les onsets t et t+1 annotés - en utilisant la valeur de la note, e.g. croche=0.5, noire=1, blanche=2).

On écrit ce calcul comme suit:

$$periode_t = \frac{timeOnsetPerf_{t+1} - timeOnsetPerf_t}{timeOnsetScore_{t+1} - timeOnsetScore_t} \tag{34}$$

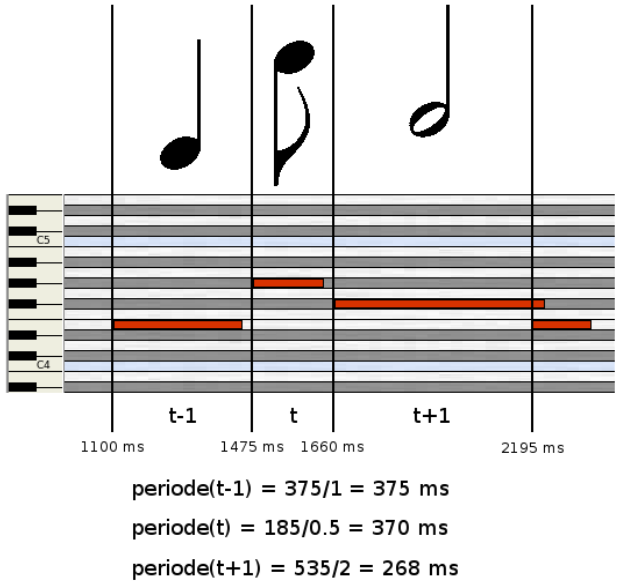

Figure 23: Exemple de calcul de la période avec la partition en haut et la représentation d'une performance en MIDI au milieu.

On obtient donc la durée en millisecondes pour une noire à chaque nouvel onset. Le résultat de la période varie tout au long de la pièce et c'est cette variation que l'on veut principalement apprendre, car nous croyons qu'elle suit une logique propre à un style de musique (dans notre cas la valse). Mais ce n'est pas tout, car la déviation expressive du temps n'est pas seulement causée par la période, mais aussi par une déviation de temps local.

5.1.4 Déviation de temps local

Pour expliquer la déviation de temps local, il y a une distinction à faire entre les onsets d'une partition et ceux d'une performance. Dans un fichier MIDI, qui représente une partition, les temps sont «parfaits». On observe donc que lorsqu'il y a plusieurs notes dans un onset, chacune de celles-ci est jouée exactement au même moment. C'est ce phénomène qui cause en partie l'effet robotique et ennuyeux. Dans le MIDI d'une performance, on prend les mêmes regroupements de notes que dans la partition, mais le calcul du onset de ces regroupements de notes correspond à la moyenne de la coordonnée de temps du début de chacune des notes du regroupement. En effet, contrairement à

la partition, les pianistes ne jouent jamais les notes d'un accord exactement au même moment. On remarque habituellement un effet de roulement, où les notes plus graves sont jouées avant les notes plus aiguës. Ce que l'on veut donc apprendre dans ce cas-ci, c'est le temps en milliseconde qui sépare une note de son onset. Si la note est jouée un peu avant la moyenne de son onset, elle aura une valeur négative, sinon elle aura une valeur positive.

La période et la déviation de temps local sont responsables de la déviation expressive du temps. Mais pour créer une performance expressive, il faut aussi apprendre la logique derrière la force (vélocité) avec laquelle un interprète joue les notes.

5.1.5 Vélocité des notes

Apprendre la vélocité que les pianistes appliquent sur les notes d'une pièce de musique est une tâche qui se sépare en deux parties. Sa représentation sera vue plus loin dans ce chapitre. La première partie est la variation de la force de la note en fonction de sa position dans la pièce. La deuxième partie est la variation de cette même force en fonction de la position de la note dans son accord. En effet, toutes les notes d'un accord n'ont pas la même vélocité et les notes plus aiguës sont souvent jouées avec plus de force que les notes plus graves. La période, la déviation de temps local et la vélocité sont les trois informations que l'on apprend pour créer de l'expressivité dans une pièce musicale.

5.1.6 Normalisation

On veut isoler les variations qui nous intéressent, communes entre les pianistes, des autres variations. La normalisation permet de faire ressortir les ressemblances entre les performances et de faciliter leurs comparaisons.

La période et la vélocité sont normalisées de la façon suivante:

Prenons $x_{1,}x_2, ..., x_M$ représentant l'ensemble des périodes ou des vélocités qui compose une séquence de longueur M. Pour normaliser un élément x_m, on applique le logarithme à ce x_m et on lui soustrait la moyenne du logarithme de tous les x_m de la même séquence.

$$NormalisationLog(x_k) = log_2(x_k) - \frac{\sum_{m=1}^{M} log_2(x_m)}{M} \tag{35}$$

Ceci nous permet d'obtenir un rapport logarithmique entre les x_m. En effet, on veut qu'une valeur deux fois plus petite qu'un certain x_m ait la même distance qu'une autre valeur qui est deux fois plus grande que ce x_m.

La normalisation des déviations de temps local est effectuée différemment. On divise l'élément x_m que l'on veut normaliser par celui de la même séquence qui a la plus grande valeur. Dans ce cas-ci, x_m représente une variation autour de la moyenne.

$$Normalisation(x_k) = \frac{x_k}{\max_{m=1}^{M}(|x_m|)} \tag{36}$$

5.1.7 Représentation des données

Nous allons expliquer la représentation de la partition, de la cible du modèle ainsi que de son entrée. Celles-ci sont utilisées pour chaque onset des séquences. Nous avons choisi une représentation de la partition pour un onset et cette représentation est composée de seize variables binaires, var_1 à var_{16}, expliquées plus bas.

Information provenant de la partition seulement

Dans une partition, la notation utilisée pour indiquer la force des notes jouées se sépare en deux grandes familles, fort et doux. Pour simplifier l'apprentissage et également parce qu'il y en a peu d'exemples, nous avons regroupé les notations qui représentent la force dans un groupe et celles qui représentent la douceur dans un autre. La variable $var_1 = 1$ quand la partition indique que le onset doit être joué piano (p) ou mezzo-piano (mp) ou encore pianissimo (pp) (ce qui veut dire avec douceur (p), avec douceur modérée (mp) et très doucement (pp)). Si, au contraire, il est marqué de jouer forte (f) ou mezzo-forte (mf) ou bien fortissimo (ff) (c'est-à-dire avec force (f), avec force modérée (mf) et très fort (ff)) alors $var_2 = 1$. Quand il n'y a pas d'indication sur la force avec laquelle il faut jouer le onset, on met var_1 et var_2 à zéro. Ces deux variables ne peuvent pas avoir la valeur 1 en même temps, on ne peut donc pas avoir var_1 et var_2 égales à 1 pour le même onset.

Les variables var_3 et var_4sont utilisées pour indiquer s'il y a une progression dans la force avec laquelle il faut jouer les notes d'un onset. Quand il est marqué cresc, alors $var_3 = 1$, ce qui signifie que le onset doit être joué plus fort que le onset précédent. À

l'opposé, quand il est marqué decresc, alors $var_4 = 1$, ce qui veut dire que le onset doit être joué moins fort que le onset précédent. Ces deux variables ne peuvent pas avoir la valeur 1 en même temps pour le même onset, mais les deux peuvent avoir la valeur 0.

Le cinquième élément de la représentation de la partition pour un onset est l'accent ($<$). Ce signe placé au-dessus d'une note ou d'un accord indique qu'ils doivent être joués avec intensité pour faire en sorte d'être plus remarqués que les autres notes de musique qui les entourent. Donc, quand il y a un accent, $var_5 = 1$ et quand il n'y en a pas, $var_5 = 0$.

La variable $var_6 = 1$ est utilisée pour un onset marqué par un staccato, sinon la valeur est zéro (0). Le staccato est représenté dans la partition par un point au-dessus du onset et signifie que les notes qui composent ce onset doivent être jouées moitié moins longtemps que ce qui est écrit. Il s'agit ici de la définition la plus répandue, car il n'y a pas de consensus, entre les pianistes que nous avons rencontrés, sur la longueur exacte de temps qui doit être imputée à une note staccato. Il y a, par exemple, des pianistes qui plaquent une note staccato juste un bref instant et laissent un silence pour le reste de la durée de cette note. D'autres, par contre, pense qu'une note staccato doit être jouée la moitié du temps qui lui est accordée.

Les trois prochaines variables var_7 , var_8 et var_9 sont liées les unes aux autres. Elles représentent la position du onset dans sa mesure. Puisque nos données d'entraînement ne sont composées que de performances de valses, toutes les séquences que nous observons sont constituées de mesures à trois temps. Quand le onset est joué à l'intérieur du premier temps d'une mesure ,var_7=1. Les variables var_8 et var_9 correspondent respectivement aux temps deux et trois. On utilise le concept «*one-hot*» qui dit que une seule de ces trois variables est égale à 1 à la fois (pour chaque onset), les autres sont égales à 0.

Comme celles que l'on vient juste de voir, les trois variables qui suivent, var_{10} , var_{11} et var_{12} sont liées entre elles et utilisent le concept «*one-hot*». Elles représentent la durée du onset. On ne parle pas de la durée de chacune des notes qui composent le onset, mais de la longueur en temps qui le sépare de la prochaine apparition de notes. Ce sont les temps fixes de la partition que l'on utilise, plus particulièrement la croche, la noire et la blanche. La noire est une unité de base qui équivaut à un temps, la croche vaut la moitié de la noire donc un demi-temps et la blanche vaut deux noires donc deux temps. Il existe

plusieurs autres temps en musique en plus des trois qui viennent d'être nommés, mais pour ce qui est de la durée des onsets, les cinq valses qui ont servi à établir les données ne sont pas composées par d'autres sortes de temps. Comme précédemment, une seule de ces trois variables est égale à 1 pour chaque onset.

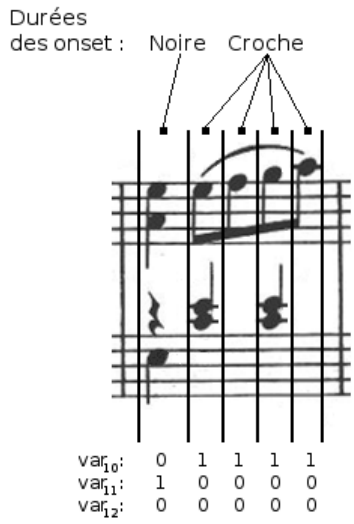

Figure 24: Représentation «*one-hot*» de la durée des onsets pour une mesure, avec var_{10} pour les croches, var_{11} pour les noires et var_{12} pour les blanches.

Les quatre dernières variables de la représentation de la partition, var_{13} , var_{14}, var_{15} et var_{16}, sont utilisées pour indiquer les fins de phrase ainsi que certaines positions relatives (en pourcentage) à la pièce de musique . Dans les partitions, il n'y a que les barres de fin de phrase comme information, mais un pianiste regarde une partition de musique dans son ensemble et sait où il se situe dans une phrase. Puisque le tempo change beaucoup au début et à la fin d'une phrase, nous avons voulu donner cette information au modèle à l'aide de deux variables, var_{13} et var_{14}. Elles représentent respectivement l'avant dernière et la dernière mesure d'une phrase. Quand un onset se trouve dans une de ces mesures, la variable correspondante est égale à 1 et l'autre garde la valeur zéro. On utilise var_{15} pour indiquer la position du onset dans la phrase et var_{16} pour la position dans la pièce. Ces deux dernières variables ont des valeurs continues entre 0 et 1 inclusivement et correspondent au pourcentage de temps joué en fonction du temps total de la phrase et de la pièce.

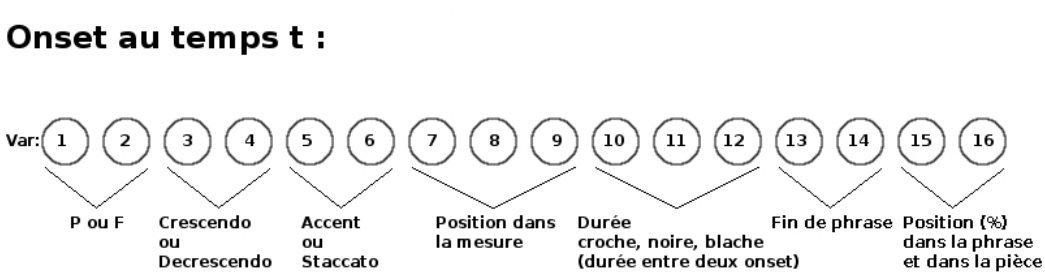

Figure 25: Représentation de la partition pour un onset.

La cible

La cible correspond à ce que notre modèle doit prédire pour un onset en particulier. La cible est un vecteur de seize variables réelles dont la valeur se situe entre -1 et 1. Ce vecteur se sépare en trois groupes.

Le premier groupe est constitué des sept premières variables et représente la force avec laquelle les notes d'un onset sont jouées (cet aspect est expliqué dans la sous-section 5.1.5). La première variable est associée à la note la plus grave du onset et chacune des variables suivantes est associée à une note plus aiguë que la précédente. On utilise seulement sept variables parce qu'il n'y a pas de onset dans l'ensemble de données qui soit composé de plus de sept notes de musique. Quand il y en a moins, les premières variables sont associées aux notes présentes dans le onset. Les variables de la fin, qui ne sont associées à aucune note, sont mises à zéro et ne sont pas incluses dans le calcul du coût (exemple d'un onset composé de quatre notes: $var_1 = -0.5$, $var_2 = -0.23$, $var_3 = 0.05$, $var_4 = 0.84$, $var_5 = 0$, $var_6 = 0$, $var_7 = 0$,).

Les sept variables suivantes correspondent au deuxième groupe qui est utilisé pour représenter la déviation locale de temps pour les notes d'un onset (ce concept est expliqué dans la sous-section 5.1.4). La première variable est associée à la note la plus grave du onset. Chacune des variables suivantes est associée à une note plus aiguë que la précédente, exactement comme dans le premier groupe, mais avec les variables var_8 à var_{14}. Encore une fois, on utilise seulement sept variables parce qu'il n'y a pas de onset qui soit composé de plus de sept notes.

Les deux dernières variables de la cible correspondent au troisième groupe qui sert à représenter la période d'un onset (ceci est expliqué dans la sous-section 5.1.3). La période

est ce qu'il y a de plus important à apprendre, car c'est là que se trouve la majorité de l'expressivité d'une performance musicale. La première variable (var_{15}) représente la durée que doit avoir une noire pour un onset en particulier. La deuxième variable est un complément de la première et correspond à la différence de période entre un onset et celui qui le précède ($var_{16}(temp\ t) = var_{15}(t) - var_{15}(t-1)$). De cette façon, on représente le signal (la variation temporelle) que l'on veut apprendre avec var_{15} et on souligne la transition entre les moments d'accélération et les moments de ralentissement d'une performance avec var_{16}.

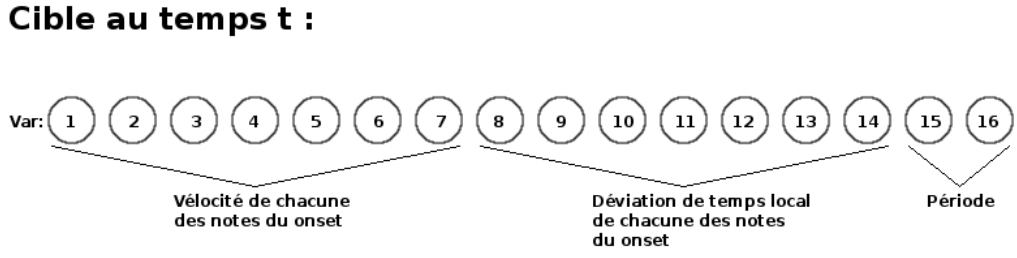

Figure 26: Cible du modèle

L'entrée du modèle

Les séquences qui font partie de nos données sont composées d'une suite de vecteurs d'entrées. Chacun d'eux représente une petite fenêtre qui entoure le onset auquel il est associé. Une entrée est composée des vecteurs de la partition des deux onsets qui le précèdent, de celui qu'il représente et des deux qui le suivent. L'entrée est aussi constituée des deux vecteurs de cibles qui correspondent aux deux onsets précédents. Avec cette fenêtre temporelle, on désire faciliter l'apprentissage des dépendances dans le temps.

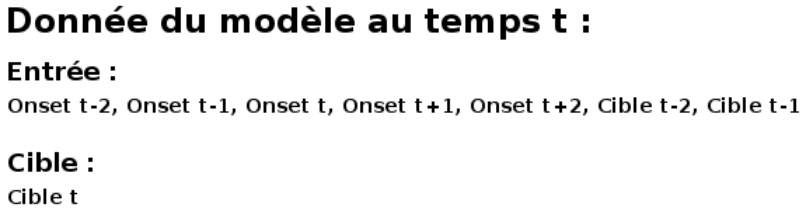

Figure 27: Entrée et cible du modèle

5.1.8 Comparaison en MIDI entre la partition et la performance

Pour créer les données, il a d'abord fallu extraire les informations dont on avait besoin dans les performances jouées par les pianistes et enregistrées sur le piano impérial Bösendorfer 290SE. Le format des enregistrements a été créé par la compagnie Bösendorfer et contient beaucoup trop d'informations pour nos besoins. En effet, toutes les deux millisecondes, l'angle de chacune des 96 touches du piano est enregistré en mémoire ainsi que toutes les informations sur le mouvement des pédales et sur la vélocité des marteaux. Une performance en format Bösendorfer peut faire jusqu'à six mega-octets, ce qui rend l'analyse très lente comparée à celle du format MIDI qui pour la même performance présente une taille de cent quarante kilo-octets. La perte d'information qui nous intéresse est minime, voire négligeable. On utilise donc un programme qui a été fait par Jasmin Lapalme, un ancien élève à la maîtrise dans le laboratoire du LISA, pour convertir les fichiers Bösendorfer en fichiers MIDI. Ce programme analyse un fichier Bösendorfer pour en extraire toutes les informations pertinentes, c'est-à-dire trouver chaque note qui est jouée ainsi que la durée et le moment exact où elle apparaît. Les informations sur le mouvement des pédales sont aussi analysées.

Une fois que toutes les performances enregistrées sur le piano ont été converties en fichier MIDI, on passe à l'étape de la comparaison avec la partition. C'est cette étape cruciale qui permet d'extraire, dans une performance, les informations qui composent les données utilisées par notre modèle. Pour ce faire, nous avons employé un programme [13], en Matlab, spécialement conçu pour comparer une performance avec sa partition, les deux devant être en format MIDI. Nous avons donc dû retranscrire les cinq partitions de valse dans ce format.

Malheureusement cet outil de comparaison fait souvent des erreurs ce qui nécessite une intervention humaine pour les corriger, processus qui peut s'avérer coûteux en temps. En effet, un pianiste ne joue jamais parfaitement une pièce de musique et apporte des modifications qui rendent la comparaison avec la partition très difficile.

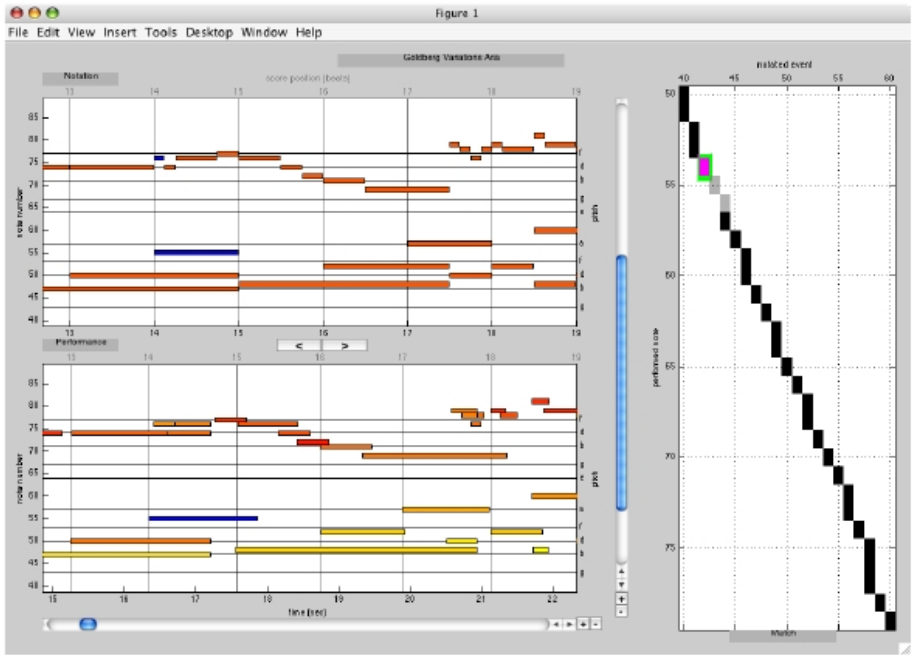

Figure 28: Interface graphique qui permet de comparer et visualiser deux fichiers MIDI. Elle est utilisée pour corriger manuellement les erreurs.

5.2 Expérience sur les performances expressives des pièces de Schubert

Dans cette section, en utilisant les données que l'on vient de voir dans la section 5.1, nous allons montrer les résultats obtenus par un réseau de neurones et un réseau de neurones récurrent pour la prédiction de la période (inverse du tempo), de la dynamique et de la déviation de temps local. Le but ici n'est pas d'analyser la différence entre les deux modèles, mais d'obtenir les meilleurs résultats possibles. C'est pour cette raison que l'on applique des fenêtres de temps sur des séquences de notes de musique afin de créer les entrées utilisées par les modèles. Ceci permet d'avoir des liens plus directs pour représenter les dépendances à court terme. Nous allons commencer par observer les résultats obtenus pour la période.

5.2.1 Apprentissage de la période

La période est une des variations les plus intéressantes à apprendre, car elle est celle qui est la plus responsable de l'expressivité d'une performance [17]. Elle est influencée à la

fois par des dépendances à court et à long terme. Cette dépendance à long terme est en lien avec la structure générale des courbes expressives.

Dans une expérience faite par Bruno Repp [17], dont nous avons parlé dans la section 2.2, plusieurs pianistes ont joué une même pièce de musique classique afin de créer une liste d'interprétations. Une nouvelle interprétation, générée artificiellement et qui représente la moyenne des variations expressives (calculées à partir des performances enregistrées) est rajoutée à la liste. Il a été demandé ensuite à un auditoire de noter les différentes interprétations en fonction de leurs préférences. Les résultats ont démontré que la performance des moyennes expressives faisait partie des interprétations les mieux notées. Ceci nous permet d'utiliser la moyenne des variations expressives de nos pianistes enregistrés sur la même pièce comme point de comparaison avec la génération de nos modèles. Le graphique qui suit est une superposition des variations de la période effectuée par les pianistes que nous avons enregistrés sur la pièce (annexe 2) de Schubert. Les performances de cette partition ne font pas partie de l'ensemble d'entraînement.

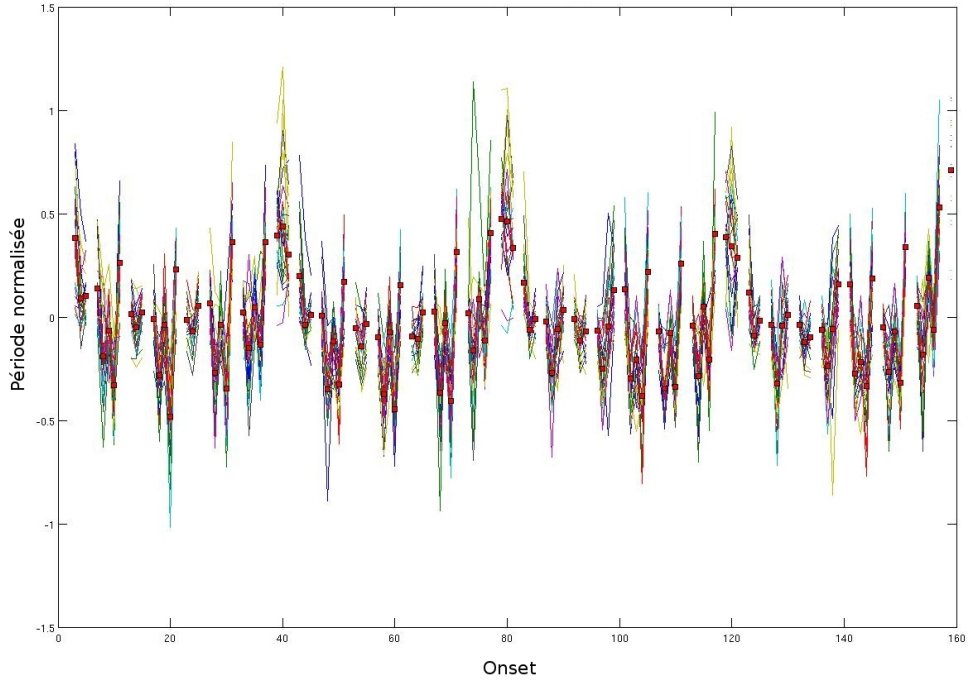

Figure 29: Variation normalisée (pour la pièce de l'annexe 2) de la période pour l'interprétation de tous les pianistes avec les carrés rouges représentant la moyenne. La période ralentit quand la courbe augmente et accélère quand elle diminue. Un espace blanc entre deux regroupements de notes représente un changement de mesure.

La courbe rouge avec les carrés correspond à la moyenne des variations expressives générées par l'ensemble des pianistes enregistrés. L'axe horizontal indique le déplacement dans le temps par onset; l'axe vertical représente la variation de la durée pour une note noire. La valeur zéro signifie que la durée correspond à la moyenne des durées de la pièce, la période ralentit quand la courbe augmente et accélère quand elle diminue. Les espaces blancs sont utilisés pour séparer les mesures entre elles. Les gros ralentissements autour des onsets 0, 40, 80, 120 et 160 sont dûs aux fins de phrases. En observant la figure 29, on peut facilement se rendre compte que presque tous les musiciens accélèrent et ralentissent au même moment, à l'exception des fins de phrases qui varient beaucoup d'une personne à l'autre, même si majoritairement les pianistes ralentissent. Ces similarités entre les performances indiquent la présence d'une logique commune qui, à première vue, nous permet de croire qu'elle pourra être apprise par un modèle statistique.

Nous allons commencer par donner les résultats empiriques pour l'ensemble d'entraînement, de validation et de test en utilisant comme mesure la moyenne des différences au carré (MSE) entre la prédiction et la cible. Il faut noter qu'il ne s'agit pas de l'erreur moyenne pour une séquence (pièce de musique au complet), mais de l'erreur moyenne pour toutes les séquences. On utilise la fonction de coût d'un onset comme comparaison et non celle de la séquence parce que les séquences n'ont pas toujours la même longueur.

Pour les deux modèles, nous avons utilisé, afin d'optimiser les hyperparamètres, des taux d'apprentissage entre 1e-2 et 1e-6 et un grand nombres de neurones cachées différents (entre 20 et 400). Pour chaque combinaison d'hyperparamètre possible entre le taux d'apprentissage et le nombre de neurones de la couche cachée, plusieurs apprentissages avec une initialisation des poids différente (cinq seed) sont exécutés par modèle.

Les deux tableaux qui suivent représentent les meilleurs résultats de MSE obtenus pour les entraînements de réseaux de neurones et de réseaux de neurones récurrents, qui ont été vus dans la section 3, avec une couche cachée.

Réseau de neurones				
Validation (MSE)	Entraînement (MSE)	Test (MSE)	Nombre de neurones cachées	Taux d'apprentissage
0.04351	0.02973	0.03991	160	1e-4
0.04368	0.02981	0.04007	160	1e-5
0.04372	0.02840	0.03962	30	1e-3
0.04381	0.02881	0.03985	30	1e-3
0.04383	0.02880	0.04030	40	1e-3

Table 6: Les cinq meilleurs résultats de MSE pour l'ensemble de validation en utilisant des réseaux de neurones pour apprendre la période.

Réseau de neurones récurrent				
Validation (MSE)	Entraînement (MSE)	Test (MSE)	Nombre de neurones cachées	Taux d'apprentissage
0.04352	0.02964	0.04013	180	1e-4
0.04354	0.02956	0.04012	180	1e-5
0.04356	0.02942	0.03989	90	1e-4
0.04365	0.02942	0.04039	200	1e-5
0.04366	0.02937	0.03996	90	1e-5

Table 7: Les cinq meilleurs résultats de MSE pour l'ensemble de validation en utilisant des réseaux de neurones récurrent pour apprendre la période.

On peut voir en observant les deux tableaux qu'on obtient à peu près les mêmes résultats pour les deux modèles. Ce fait peut paraître surprenant. Nous avons fait ces expériences parce que nous nous attendions à ce que le réseau de neurones récurrent ait une meilleure MSE que le réseau de neurones. Le RNN possède une matrice de connection dynamique qui permet, en principe, d'apprendre les dépendances temporelles, ce que le NN n'a pas. Cependant, la façon dont les données sont encodées pour l'entrée des modèles peut expliquer ce comportement. En effet, beaucoup d'informations temporelles sont déjà incluses dans les entrées, comme le contexte local (information sur les onsets qui précèdent et qui suivent le onset à prédire), la position dans la mesure, dans la phrase et dans la pièce au total, ainsi que les indications spécifiques sur les débuts et les fins de phrase. La majorité des dépendances temporelles sont donc contenues localement dans une entrée ce qui permet au réseau de neurones de s'en servir. Une explication possible du fait que les réseaux de neurones récurrents ne donnent pas de meilleurs résultats est que justement la majorité des dépendances se trouve directement dans l'entrée immédiate et que le reste des dépendances à apprendre l'est à trop long terme pour les capacités du modèle. Il a été prouvé que ce genre de modèles souffre d'une perte exponentielle de

l'influence des dépendances en fonction du temps [29].

Observons d'abord les prédictions de la période pour le réseau de neurones.

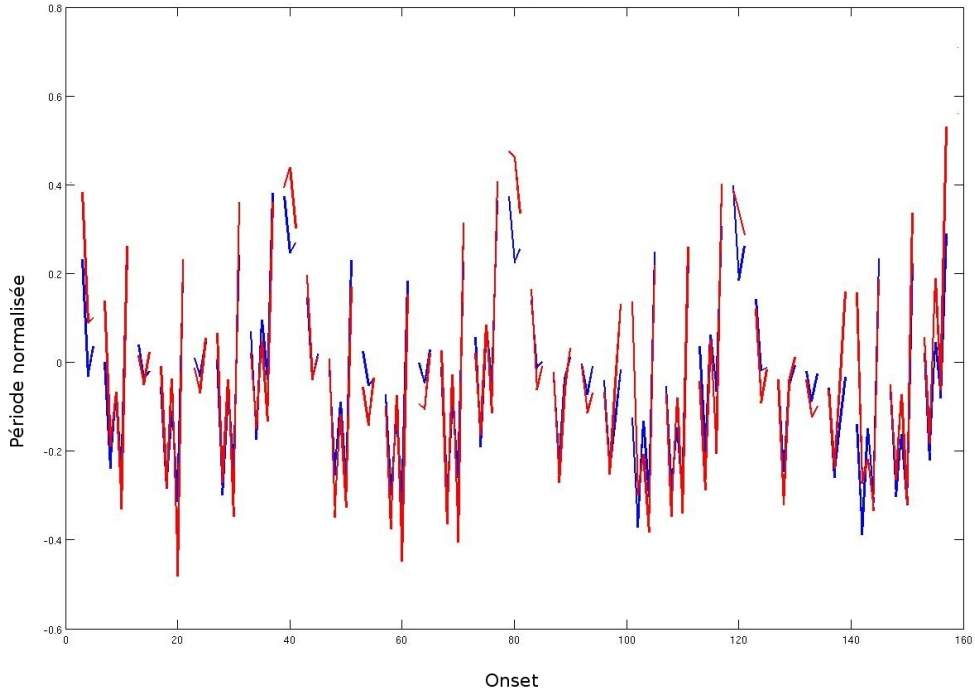

Figure 30: Génération (pour la pièce de l'annexe 2) de la période par un réseau de neurones en bleu et moyenne des variations des pianistes en rouge. La période ralentit quand la courbe augmente et accélère quand elle diminue. Un espace blanc entre deux regroupements de notes représente un changement de mesure.

Et maintenant regardons les prédictions de la période pour le réseau de neurones récurrent.

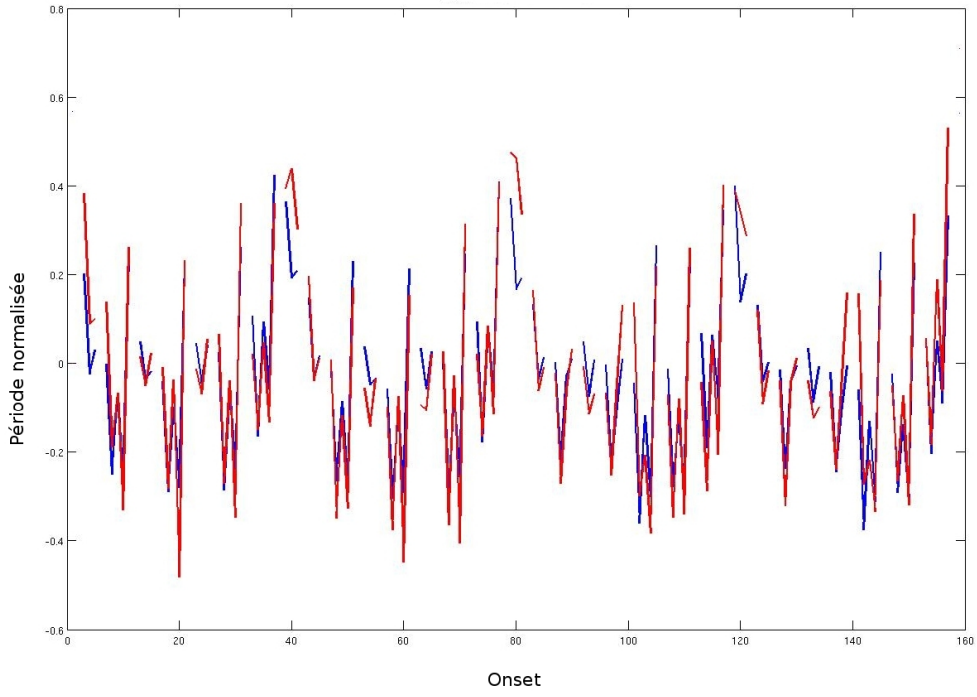

Figure 31: Génération (pour la pièce de l'annexe 2) de la période par un réseau de neurones récurrent en bleu et moyenne des variations des pianistes en rouge. La période ralentit quand la courbe augmente et accélère quand elle diminue. Un espace blanc entre deux regroupements de notes représente un changement de mesure.

Puisque les deux modèles donnent des résultats similaires, nous allons dorénavant observer uniquement les générations des réseaux de neurones récurrents. Les deux dernières figures que l'on vient de voir superposent la courbe expressive de la moyenne des performances (en rouge) avec celle qui a été généré par le modèle (en bleu). Il faut noter que la pièce (annexe 2) que nous sommes en train d'analyser ne fait pas partie de l'ensemble d'entraînement. Il s'agit donc de généralisation sur une nouvelle pièce de musique. Cependant, une partie des performances de cette partition compose une section de l'ensemble de validation, ce qui veut dire que les résultats sont légèrement biaisés. En comparant les deux courbes, on peut tout de même dire que le modèle statistique a réussi à comprendre, en grande partie, la logique des variations expressives de la période propre au style de musique qu'est la valse et qui est présente dans chacune des performances des pianistes. Le concept de ralentissement en fin de phrase a aussi été appris, mais la comparaison avec la courbe de la moyenne n'est peut-être pas appropriée, car même entre pianistes, il ne semble pas y avoir de consensus. On pourrait à la place com-

parer l'erreur avec les écarts entre pianistes. Il faut mentionner aussi que cette pièce de musique fait partie du même opus que les pièces de l'ensemble d'entraînement. Celles-ci sont donc relativement semblables, ce qui explique les si bons résultats.

Apprendre la logique de la variation de la période est un premier pas vers l'apprentissage de l'expressivité musicale. Il reste cependant encore beaucoup de choses à apprendre avant d'y parvenir, comme par exemple la variation de la vélocité des notes.

5.2.2 Apprentissage de la vélocité

La force avec laquelle les pianistes jouent une note de musique dépend d'abord de sa position dans l'accord dont elle fait partie. En effet, on a remarqué que les notes plus aiguës sont presque systématiquement jouées avec plus de force que les notes plus graves du même accord. La variation de la vélocité dépend aussi du contexte local, comme la position dans la mesure (temps fort, temps faible), et du contexte temporel à long terme, comme les montées et les descentes du contour mélodique ainsi que les débuts et les fins de phrases. Comme pour la période, nous allons commencer par donner les résultats empiriques pour l'ensemble d'entraînement, de validation et de test en utilisant la moyenne des différences au carré (MSE) comme mesure. Les deux tableaux qui suivent représentent les meilleurs résultats de MSE obtenus pour l'ensemble de validation par les entraînements de réseaux de neurones et de réseaux de neurones récurrents.

Réseau de neurones				
Validation (MSE)	Entraînement (MSE)	Test (MSE)	Nombre de neurones cachées	Taux d'apprentissage
0.04605	0.04782	0.05100	170	1e-5
0.04607	0.04803	0.05090	20	1e-4
0.04612	0.04799	0.05090	160	1e-5
0.04614	0.04803	0.05112	200	1e-5
0.04616	0.04779	0.05098	190	1e-5

Table 8: Les cinq meilleurs résultats de MSE pour l'ensemble de validation en utilisant des réseaux de neurones pour apprendre la vélocité.

Réseau de neurones récurrent				
Validation (MSE)	Entraînement (MSE)	Test (MSE)	Nombre de neurones cachées	Taux d'apprentissage
0.04604	0.04773	0.05067	130	1e-5
0.04619	0.04807	0.05124	350	1e-5
0.04620	0.04807	0.05138	240	1e-5
0.04623	0.04790	0.05112	190	1e-5
0.04626	0.04802	0.05116	150	1e-5

Table 9: Les cinq meilleurs résultats de MSE pour l'ensemble de validation en utilisant des réseaux de neurones récurrent pour apprendre la vélocité.

Ces expériences ont été exécutées avec les mêmes hyperparamètres que ceux vus pour la période. Encore une fois, les résultats entre les réseaux de neurones et les réseaux de neurones récurrents sont très semblables pour les raisons déjà expliquées un peu plus haut. Nous allons donc observer la génération de la vélocité pour le modèle récurrent seulement.

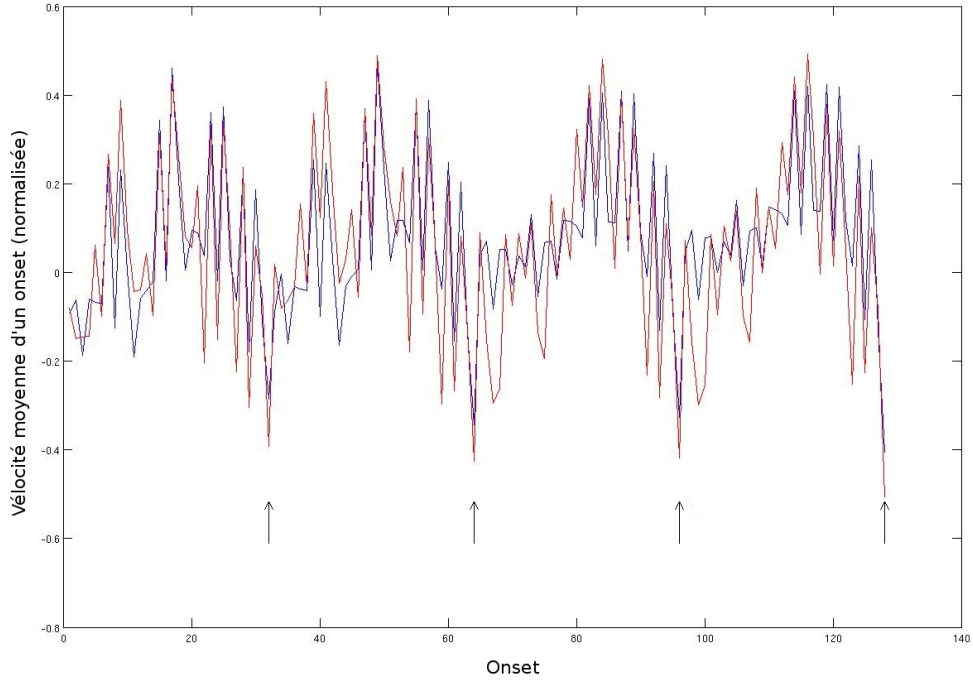

Figure 32: Génération (pour la pièce de l'annexe 2) de la vélocité moyenne des onset par un réseau de neurones récurrent en bleu et moyenne des variations normalisées des pianistes en rouge. Les notes de musique sont jouées plus fort quand la courbe augmente et moins fort quand elle diminue. Les flèches marquent les fins de phrases de la pièce.

Les courbes que l'on observe dans la figure 32 sont les résultats de la moyenne des

vélocités des notes de chaque onset. La courbe bleue représente la génération de la vélocité du réseau de neurones récurrent. La courbe rouge représente la moyenne des vélocités appliquées par les pianistes pour cette pièce de musique (annexe 2). Les quatre flèches marquent les fins de phrases. Malgré un comportement du modèle majoritairement semblable à celui des pianistes (c'est-à-dire que la prédiction de la force des notes jouées augmente et diminue généralement aux mêmes endroits que dans les performances des pianistes à des intensités semblables), on peut noter un certain nombre d'erreurs qui ne devraient pas avoir lieu. Ces erreurs s'expliquent par le fait que ce n'est pas exactement ces valeurs que l'on tente d'apprendre. En effet, on ne prédit pas la moyenne de l'intensité pour un onset, mais plutôt l'intensité pour chacune des notes du onset. Observons donc la génération de notre modèle pour chaque note de la première phrase de la même pièce de musique.

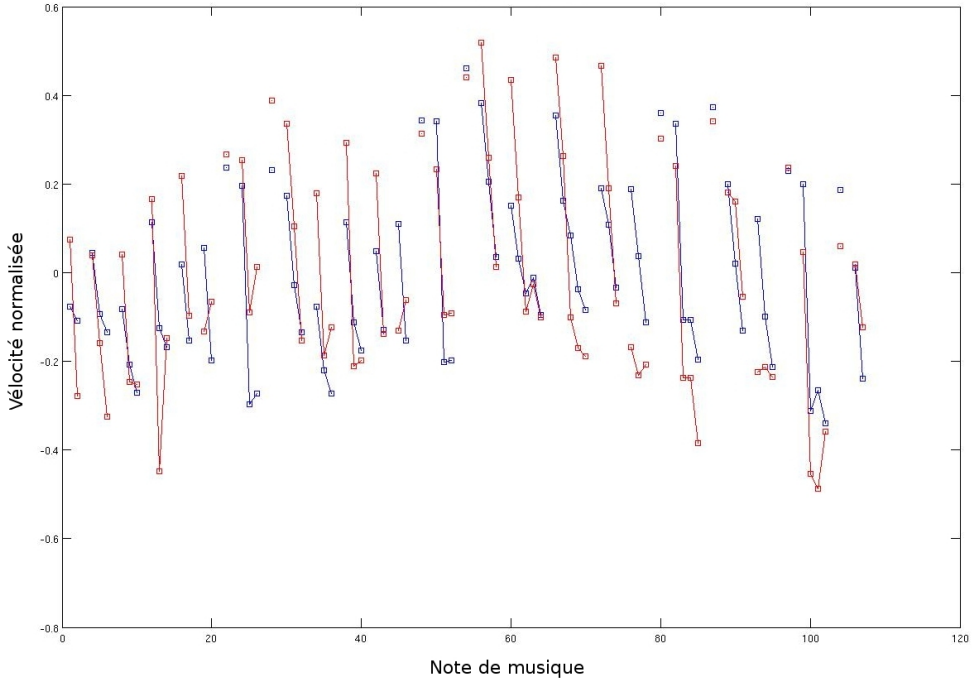

Figure 33: Génération (pour la première phrase de la pièce dans l'annexe 2) de la vélocité pour chaque note par un réseau de neurones récurrent en bleu et moyenne des variations des pianistes en rouge. Les notes d'un même onset sont liées entre elles par une ligne. Les notes de musique sont jouées plus fort quand la vélocité augmente et moins fort quand elle diminue.

Dans la figure 33, seules les notes d'un même onset sont liées entre elles. Toutes les

notes de la première phrase de la pièce jouée (annexe 2) sont représentées en double dans le graphique, une fois par les carrés bleus qui ont été générés par le modèle, une deuxième fois par les carrés rouges qui ont été calculés en faisant la moyenne des vélocités appliquées par les pianistes dans les interprétations. Les carrés bleus et rouges qui ne sont connectés à aucune autre note par une ligne sont simplement des onsets à une note. Cette figure nous permet de visualiser non seulement la variation de la dynamique tout au long de la phrase, mais surtout la grande variabilité au sein même des accords. Sur l'axe vertical, la valeur zéro indique la force moyenne avec laquelle les notes sont jouées pour toute la pièce; quand la valeur augmente ou diminue, la vélocité fait de même. Dans la figure 33, les points d'un même onset sont placés en ordre décroissant de fréquence, c'est-à-dire que les notes les plus aiguës sont à gauche et que les plus graves sont à droite. On peut donc voir que les notes aiguës sont majoritairement jouées plus fort que les notes graves, ce qui est plutôt logique puisque la mélodie dominante d'une pièce se trouve généralement dans la partie aiguë des notes de musique.

Comme pour la dynamique, la variation du temps des notes dépend du contexte à court et à long terme ainsi que du contexte local immédiat (position de la note dans un accord). Il nous reste à apprendre la déviation de temps d'une note par rapport au temps moyen du onset.

5.2.3 Apprentissage des déviations locales de temps

Jouer toutes les notes d'un accord exactement en même temps n'est pas naturel et donne l'impression que la mélodie est jouée par un robot. Il y a donc un aspect important de l'expressivité humaine qui est enlevé quand il n'y a pas de déviations locales de temps dans une performance. Cette variation que notre modèle doit apprendre pour une note de musique dépend principalement de sa position dans un accord, position établie en fonction des fréquences des notes (plus aiguës ou plus graves) qui composent le onset. Encore une fois, nous allons observer les meilleurs résultats empiriques des entraînements de réseaux de neurones et de réseaux de neurones récurrents pour l'ensemble de validation en utilisant la moyenne des différences au carré (MSE) comme mesure.

Réseau de neurones				
Validation (MSE)	Entraînement (MSE)	Test (MSE)	Nombre de neurones cachées	Taux d'apprentissage
0.09527	0.07635	0.09228	20	1e-5
0.09521	0.07676	0.09222	90	1e-5
0.09524	0.07601	0.09196	20	1e-5
0.09527	0.07656	0.09216	40	1e-5
0.09539	0.07742	0.09261	130	1e-4

Table 10: Les cinq meilleurs résultats de MSE pour l'ensemble de validation en utilisant des réseaux de neurones pour apprendre la déviation locale de temps.

Réseau de neurones récurrent				
Validation (MSE)	Entraînement (MSE)	Test (MSE)	Nombre de neurones cachées	Taux d'apprentissage
0.09496	0.07653	0.09192	30	1e-5
0.09498	0.07651	0.09194	40	1e-5
0.09499	0.07610	0.09204	70	1e-5
0.09517	0.07628	0.09232	60	1e-5
0.09523	0.07638	0.09221	60	1e-5

Table 11: Les cinq meilleurs résultats de MSE pour l'ensemble de validation en utilisant des réseaux de neurones récurrent pour apprendre la déviation locale de temps.

On peut voir en observant les tableaux que l'erreur moyenne d'une déviation locale de temps est significativement plus grande que celle de la vélocité et de la période. Ce comportement s'explique par le fait qu'il n'y a pas qu'une seule bonne façon d'appliquer une déviation de temps locale, ce qui se traduit par une grande variabilité dans les données entre les pianistes. Regardons maintenant la génération du réseau de neurones

récurrent.

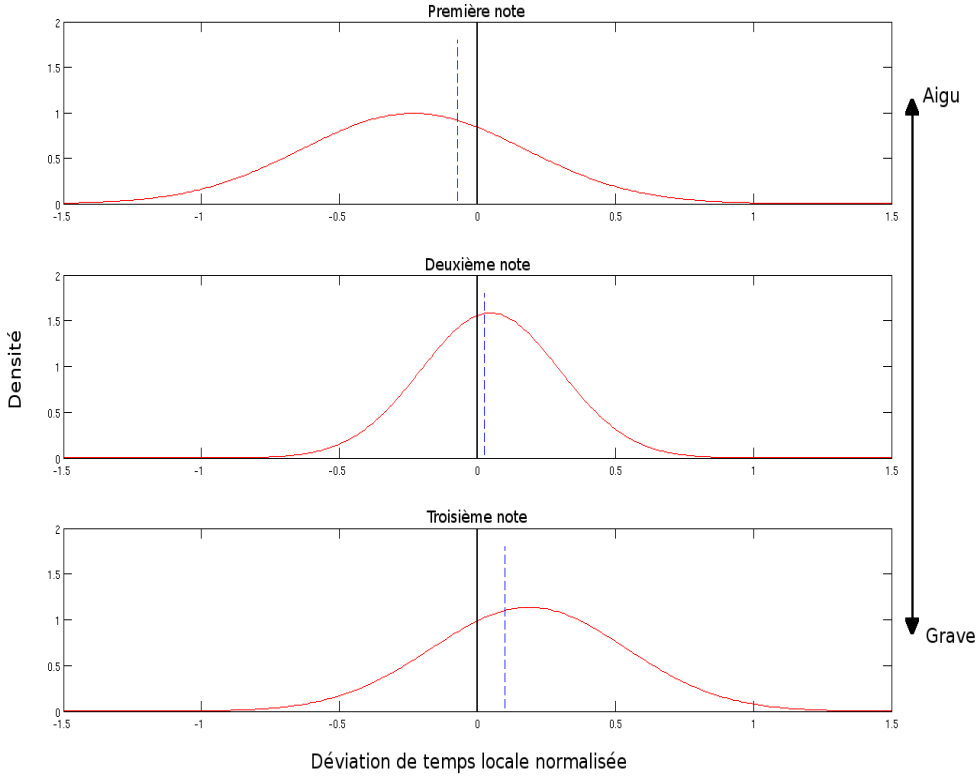

Figure 34: Génération de la déviation de temps locale pour chaque note d'un onset par un réseau de neurones récurrent avec les lignes bleu pointillées. Densité de probabilité des variations des pianistes avec les courbes en rouge. Chaque graphique représente une des trois notes du premier onset de la première mesure de la pièce de Schubert en annexe 2.

La figure 34 représente, avec les lignes bleu pointillées, la génération du modèle (RNN) de la déviation de temps locale pour chacune des trois notes du premier onset de la première mesure de la pièce de Schubert (qui se trouve en annexe 2). Chaque courbe en rouge correspond à la densité de probabilité des variations des pianistes pour une note. La valeur centrale zéro sur l'axe des X représente le moment où le onset est joué (moyenne des notes qui le compose). On peut donc voir que les notes aiguës sont presque toujours jouées avant les notes graves. Ce comportement est logique si l'on considère que la mélodie dominante d'une pièce se trouve généralement dans la partie aiguë des notes de musique. Le fait de jouer ces notes avant celles qui les accompagnent permet de mieux les entendre et d'accentuer leur importance.

5.2.4 Résultat du modèle expérimental RAE appliqué aux performances expressives

Voici les résultats obtenus de l'apprentissage d'un modèle RAE pour la période, la vélocité des notes ainsi que la déviation de temps local. On peut remarquer que les performances du RAE sont équivalent à ceux du NN et du RNN.

Apprentissages de la période					
Validation (MSE)	Entraînement (MSE)	Test (MSE)	Nombre de neurones cachées	Taux d'apprentissage RNN	Taux d'apprentissage RAE
0.04313	0.03014	0.03988	50	1e-4	1e-5
0.04313	0.03011	0.03996	50	1e-4	1e-5
0.04314	0.03017	0.04003	80	1e-4	1e-5
0.04321	0.03008	0.03984	50	1e-4	1e-5
0.04337	0.03010	0.04004	150	1e-4	1e-5

Table 12: Les cinq meilleurs résultats de MSE pour l'ensemble de validation en utilisant le RAE pour apprendre la période.

Apprentissages de la vélocité					
Validation (MSE)	Entraînement (MSE)	Test (MSE)	Nombre de neurones cachées	Taux d'apprentissage RNN	Taux d'apprentissage RAE
0.04610	0.04820	0.05086	60	1e-4	1e-5
0.04618	0.04850	0.05123	200	1e-4	1e-5
0.04620	0.04839	0.05116	60	1e-4	1e-5
0.04621	0.04805	0.05070	50	1e-4	1e-5
0.04632	0.04813	0.05111	120	1e-3	1e-4

Table 13: Les cinq meilleurs résultats de MSE pour l'ensemble de validation en utilisant le RAE pour apprendre la vélocité.

Apprentissages de la déviation locale de temps					
Validation (MSE)	Entraînement (MSE)	Test (MSE)	Nombre de neurones cachées	Taux d'apprentissage RNN	Taux d'apprentissage RAE
0.09508	0.07649	0.09188	50	1e-5	1e-6
0.09512	0.07627	0.09201	50	1e-4	1e-5
0.09515	0.07702	0.09174	60	1e-5	1e-6
0.09523	0.07655	0.09197	40	1e-4	1e-5
0.09529	0.07658	0.09213	50	1e-4	1e-5

Table 14: Les cinq meilleurs résultats de MSE pour l'ensemble de validation en utilisant le RAE pour apprendre la déviation locale de temps.

5.2.5 Discussion

Les résultats que nous avons obtenus pour la prédiction de la période montrent que la logique de sa variation pour les valses de Schubert a été généralement bien apprise. Le fait que le réseau de neurones, le réseau de neurones récurrent ainsi que le RAE obtiennent des résultats aussi semblables, bien qu'une grosse partie de l'information temporelle se trouve dans chaque entrée, est légèrement décevant, car il reste, selon nous, une dépendance importante à apprendre. Nous parlons ici d'une dépendance à long terme qui correspond à l'intensité de la variation de la période. On a pu voir avec la figure 29 que presque tous les pianistes accélèrent et ralentissent aux mêmes endroits. La différence majeure entre eux, pour cette variation, est l'intensité qu'ils appliquent au ralentissement ou à l'accélération. L'intensité qui doit être appliquée à la période d'un onset dépend de celle qui a été appliquée dans le passé. La figure qui suit permet de visualiser ce concept et représente la variation de la période, pour la première phrase, d'une performance humaine suivie de la génération du modèle récurrent pour les trois autres phrases de la pièce.

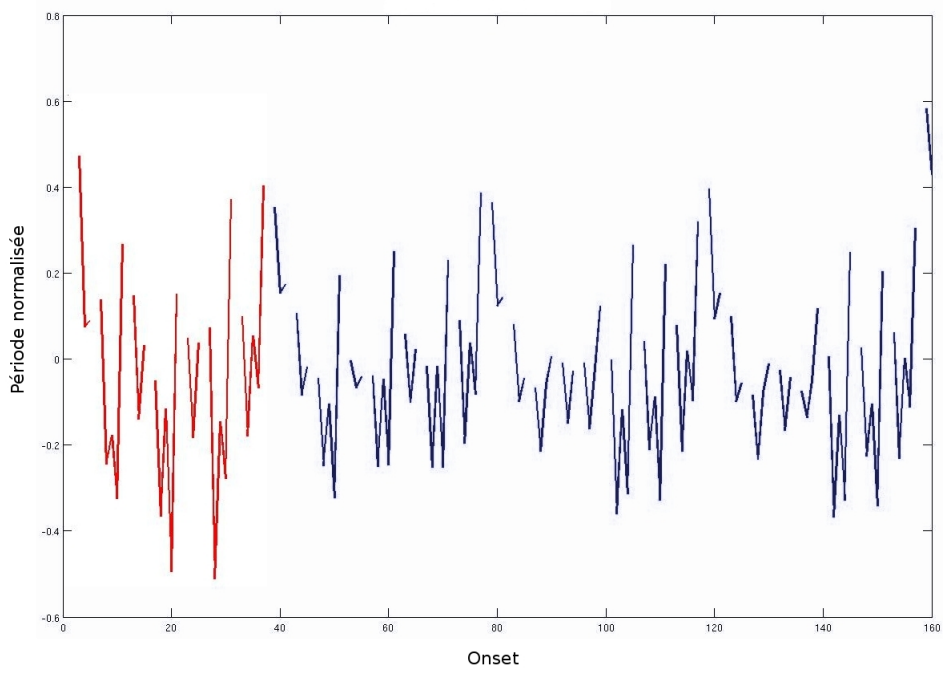

Figure 35: Performance d'une pianiste en rouge, pour la première phrase de la pièce en annexe 2, suivie par la génération du modèle récurrent en bleu pour la prédiction de la période du reste de la pièce. La période ralentit quand la courbe augmente et accélère quand elle diminue. Un espace blanc entre deux regroupements de notes représente un changement de mesure.

La courbe rouge, tirée de l'interprétation d'une pianiste, est plus extrême dans la variation de la période que la courbe bleue qui est générée par le modèle, c'est-à-dire que l'écart est plus grand entre les ralentissements et les accélérations. Pourtant, les données de la pianiste ont été utilisées par le réseau de neurones récurrent pour prédire la suite de la pièce, mais les résultats de la figure 35 montrent que le modèle n'a pas su tirer avantage de ces informations. Un modèle statistique plus puissant pourrait peut-être résoudre ce problème. Une autre solution plausible serait de mettre d'autres informations dans les données d'entrées comme la moyenne des variations dans une fenêtre de temps.

Les résultats obtenus pour les déviations locales de temps semblent satisfaisants pour un problème comme celui-ci où les données d'entraînement ne sont pas toujours très consistantes dans les variations. Par exemple, il n'y a pas de consensus qui dit que dans un accord à cinq notes, la troisième note la plus aiguë doit être jouée plus tôt ou plus tard que la quatrième. Ceci donne place a beaucoup d'interprétations possibles, toutes aussi bonnes les unes que les autres. Cependant, il existe tout de même une logique qui peut être perçue comme un roulement des notes depuis les plus aiguës aux plus graves, avec une petite variation au milieu, qui est partiellement apprise par le modèle récurrent. On pourrait peut-être améliorer les résultats en séparant l'entraînement par regroupements de onsets qui ont le même nombre de notes, c'est-à-dire qu'il y aurait un neurone d'entrée dédié juste aux onsets à une note et deux autres neurones d'entrée pour les onsets à deux notes et ainsi de suite pour les autres nombres de notes. Cette solution pourrait cependant avoir besoin de plus de données d'entraînement et risque de souffrir un peu du problème de la malédiction de la dimentionalité.

Finalement, nous pensons que notre approche pour le problème de la variation dynamique (vélocité) peut être amélioré. En effet, nous mélangeons les dépendances à long terme qui s'appliquent à un regroupement de notes (accord) avec les dépendances locales qui concernent la position d'une note dans un onset. De la même façon que nous avons séparé le problème de variation temporelle en deux avec la période et la déviation locale de temps, on peut séparer l'apprentissage de la force appliquée aux notes. On aura une représentation pour la variation moyenne de la dynamique entre onsets et une autre pour la variation locale entre les notes d'un même onset.

5.2.6 Exemple video

Nous invitons maintenant le lecteur à écouter le résultat final. Le CD inclus dans ce mémoire permet d'écouter 3 vidéos où l'on peut observer le fameux piano Bösendorfer 290SE jouer mécaniquement.

Nous recommandons de commencer par écouter le vidéo de la performance d'un vrai pianiste, puis celui de la partition sans variation expressive pour ensuite pouvoir apprécier l'interprétation générée par notre pianiste virtuel. Cette façon de procéder permet à l'auditeur de comparer et de bien entendre les variations apprises par le modèle statistique.

6 Conclusion

Dans ce mémoire, nous avons montré qu'il était possible d'apprendre, à l'aide des modèles statistiques, la logique des variations expressives propres à un style musical. Cette découverte suggère que certains domaines, plus particulièrement ceux en rapport avec l'art et considérés il n'y a pas si longtemps comme étant exclusifs aux capacités humaines, vont connaître de plus en plus d'intrusion technologique. La réalisation de ce genre de tâche est cependant fortement limitée par les capacités des modèles qui ne sont souvent pas assez puissants pour comprendre la logique trop complexe de certaines données, comme celles concernant la génération de partition de musique. C'est ce qui nous a poussés ensuite à vouloir découvrir de nouveaux modèles plus puissants qui prouvent que l'utilisation des auto-encodeurs sur un réseau de neurones récurrent peut permettre d'améliorer l'influence de l'information temporelle.

Chaque partie de ce mémoire rend compte d'un aspect spécifique. Au chapitre 2, nous présentons certains travaux connus sur la modélisation des performances expressives musicales. Notamment, nous parlons des modèles KTH et CMERS qui se basent sur les systèmes experts ainsi que du modèle de Widmer qui est un des premiers à avoir appliqué l'apprentissage machine sur les performances expressives.

Le chapitre 3 explique les modèles statistiques de base, du perceptron au réseau de neurones. Il passe ensuite à la présentation des modèles utilisés pour l'apprentissage des performances expressives, comme le réseau de neurones récurrent. Il explique aussi les modèles temporels expérimentaux que nous avons créés, comme l'auto-encodeur récurrent et ses différentes variantes.

Le chapitre 4 permet d'observer et d'analyser les capacités d'apprentissage des modèles temporels expérimentaux que nous avons créés et qui sont appliqués à deux différents ensembles de données artificielles.

Finalement, le chapitre 5 présente tout le travail effectué pour pouvoir apprendre un style musical à partir de performances expressives, en commençant par la création des données nécessaires pour se rendre jusqu'aux résultats de l'apprentissage des modèles.

6.1 Travaux futurs

Pour l'apprentissage de l'expressivité de nouveaux styles musicaux, la prochaine étape consiste à automatiser complètement l'extraction des informations nécessaires d'une performance musicale sans utiliser la partition qui a servi à sa création. Cela permettrait de pallier au plus gros problème de cette tâche, c'est-à-dire le manque de données d'apprentissage. Pour y parvenir, le défi à surmonter est de réussir à obtenir un identificateur de rythme fiable qui permettrait de mesurer les variations temporelles.

En ce qui concerne la recherche sur les auto-encodeurs appliqués à des modèles temporels, les résultats de la reconstruction de la couche cachée au moment $t-2$ nous poussent à vouloir explorer les possibilités de la reconstruction à plus long terme, comme $t-3$ ou plus. Nous voulons voir aussi si l'utilisation du bruit peut encore contribuer à accélérer l'apprentissage. Il serait finalement logique d'essayer d'appliquer les modèles sur les données de performances expressives, mais cette fois-ci sans utiliser de fenêtre temporelle pour représenter les données.

References

[1] Ethem Alpaydin. *Introduction to Machine Learning*. The MIT Press, 2004.

[2] Y. Bengio, J. Louradour, R. Collobert, and J. Weston. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 41–48. ACM, 2009.

[3] A. Friberg. Matching the rule parameters of phrase arch to performances of "traumerei:" a preliminary study. *In: A. Friberg, and J. Sundberg (Eds.), Proceedings of the KTH Symposion on Grammars for Music Performance*, pages 37–44, 1995.

[4] A Friberg. *A Quantitative Rule System for Musical Performance*. PhD thesis, Royal Institute of Technology, Stockholm, 1995.

[5] A. Friberg, R. Bresin, and J. Sundberg. Overview of the KTH rule system for musical performance. *Advances in Cognitive Psychology*, 2(2):145–161, 2006.

[6] A Gabrielsson. Music performance research at the millenium. *Psychology of Music*, 31:221–272, 2003.

[7] J. Hartigan. *Clustering algorithme*. John Wiley and Sons, 1975.

[8] R. Hecht-Nielsen. Theory of the backpropagation neural network. *Neural Networks*, 1:445, 1988.

[9] G. Hinton and R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313:504–507, 2006.

[10] G. E. Hinton, S. Osindero, and Y. Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 2006.

[11] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[12] W. Kroiss. Parameteroptimierung fur ein modell des musikalischen ausdrucks mittels genetischer algorithmen. Master's thesis, University of Vienna, 2000.

[13] Edward Large. Midi matcher. Technical report, Center for Complex Systems and Brains Sciences.

[14] Steven R. Livingstone. *Changing Musical Emotion through Score and Performance with a Computational Rule System.* PhD thesis, The University Of Queensland, Australia, 2008.

[15] Gordon D. Plotkin. A note on inductive generalization. *Machine Intelligence*, 5:153–163, 1970.

[16] J.R. Quinlan. Learning logical definitions from relations. *Machine Learning*, 5:239–266, 1990.

[17] B. H. Repp. The aesthetic quality of a quantitatively average music performance: Two preliminary experiments. *Music Perception*, 14:419–444, 1997.

[18] Friberg A. Sundberg, J. and L. Fryden. Common secrets of musicians and listeners - an analysis-bysynthesis study of musical performance. *Representing Musical Structure*, pages 161–197, 1991.

[19] Ilya Sutskever and Geoffrey Hinton. Temporal kernel recurrent neural networks. Technical report, University of Toronto, 2008.

[20] P. Vincent, H. Larochelle, Y. Bengio, and P.A. Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103. ACM, 2008.

[21] A. Waibel. Modular construction of time-delay neural networks for speech recognition. *Neural Computation*, 1(1):39–46, 1989.

[22] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K.J. Lang. Phoneme recognition using time-delay neural networks. *Readings in Speech Recognition*, pages 393–404, 1989.

[23] P.J. Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

[24] G. Widmer. Machine discoveries: A few simple, robust local expression principles. *Journal of New Music Research*, 31:37–50, 2002.

[25] G Widmer. Discovering simple rules in complex data: A meat-learning algorithm and some surprising musical discoveries. *Artificial Intelligence*, 32:259–268, 2003.

[26] G. Widmer and Goebl. Computational models of expressive music performance: The state of the art. *Journal of New Music Research*, 33(3):203–216, 2004.

[27] G. Widmer and A. Tobudic. Playing mozart by analogy: Learning multi-level timing and dynamics strategies. *Journal of New Music Research*, 32:259–268, 2003.

[28] R.J. Williams and D. Zipser. Gradient-based learning algorithms for recurrent networks and their computational complexity. *Backpropagation: Theory, architectures, and applications*, pages 433–486, 1995.

[29] P. Simard Y. Bengio and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994.

[30] P. Zanon and G. De Poli. Estimation of parameters in rule systems for expressive rendering in musical performance. *Computer Music Journal*, 27:29–46, 2003.

[31] P. Zanon and G. De Poli. Time-varying estimation of parameters in rule systems for music performance. *Journal of New Music Research*, 32:295–315, 2003.

Annexe I

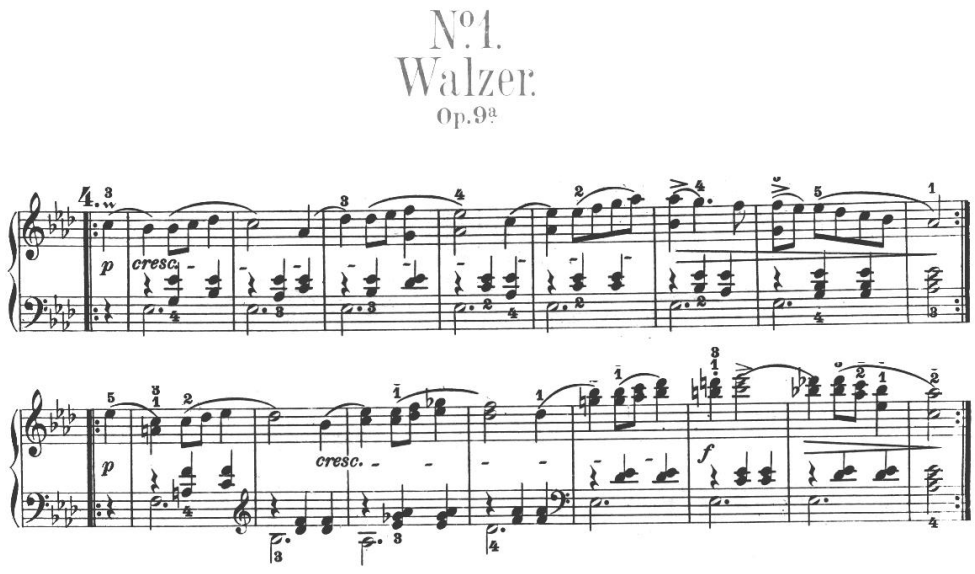

Figure 36: Valse de Schubert Op 9a no 4. Cette partition est une des cinq partitions jouées par les pianistes pour créer les données présenter dans le chapitre 5.

Annexe II

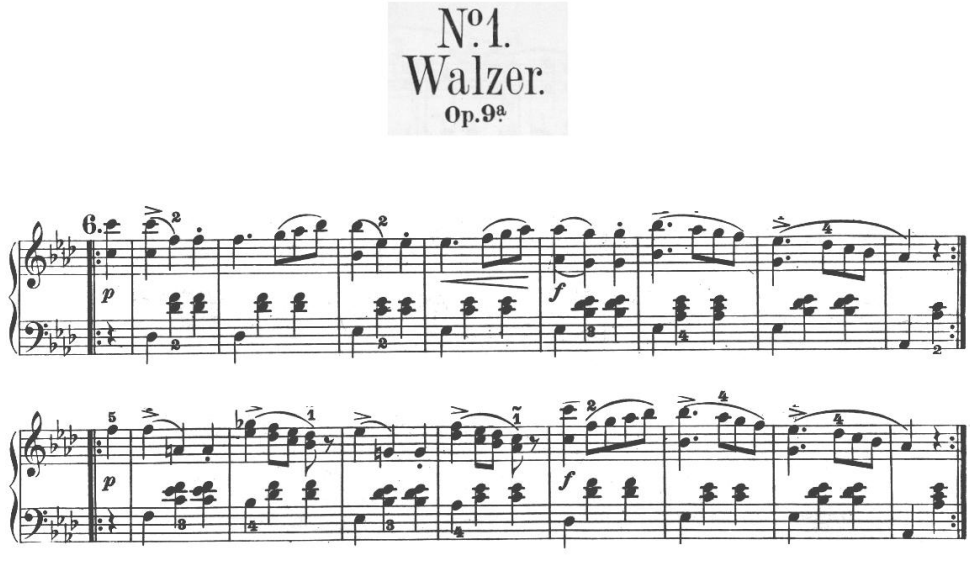

Figure 37: Valse de Schubert Op 9a no 6. Cette partition est une des cinq partitions jouées par les pianistes pour créer les données présenter dans le chapitre 5.

Annexe III

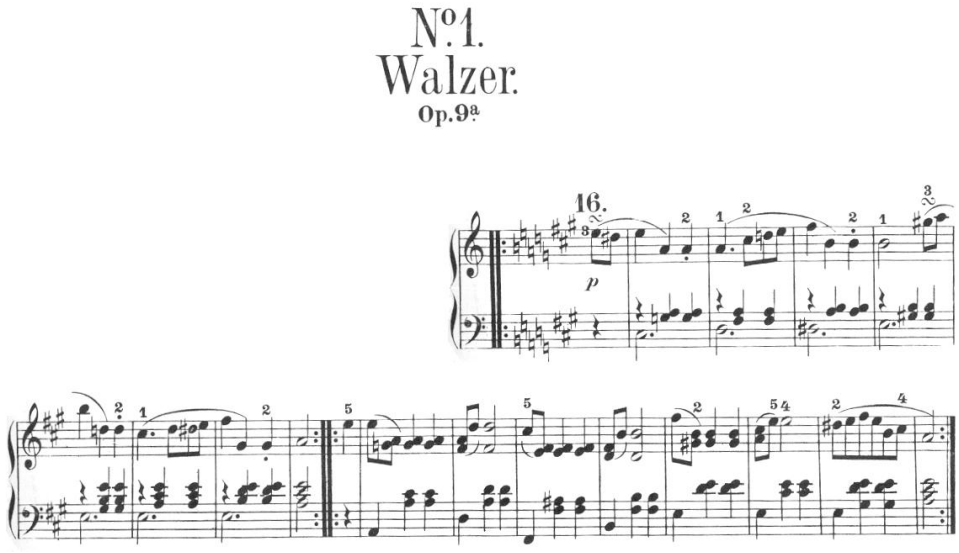

Figure 38: Valse de Schubert Op 9a no 16. Cette partition est une des cinq partitions jouées par les pianistes pour créer les données présenter dans le chapitre 5.

Annexe IV

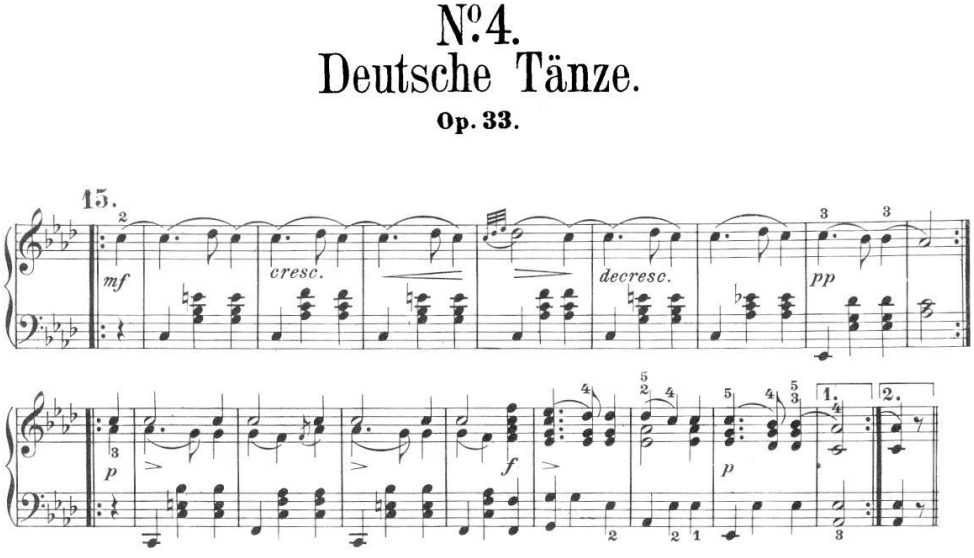

Figure 39: Valse de Schubert Op 33 no 15. Cette partition est une des cinq partitions jouées par les pianistes pour créer les données présenter dans le chapitre 5.

Annexe V

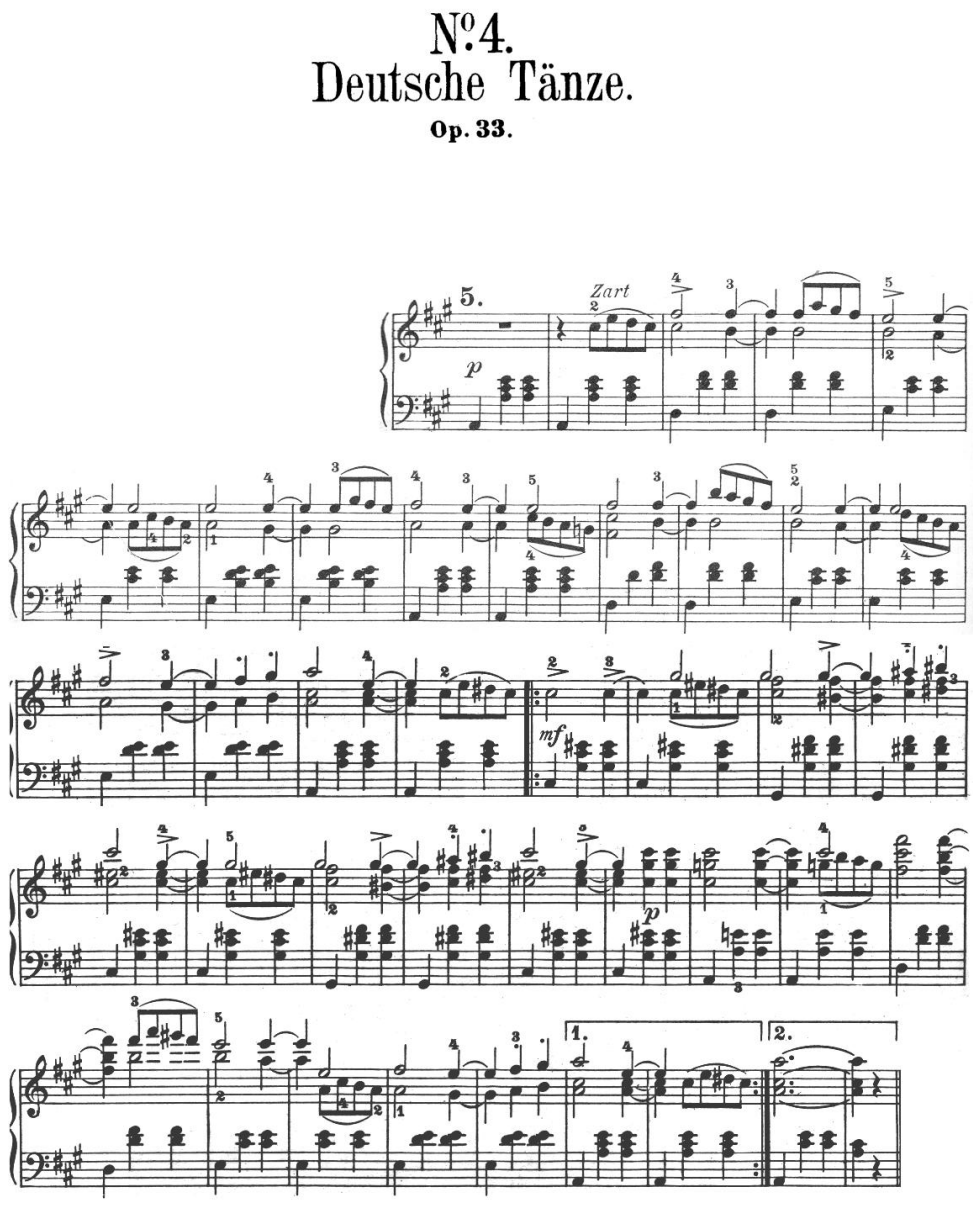

Figure 40: Valse de Schubert Op 33 no 5. Cette partition est une des cinq partitions jouées par les pianistes pour créer les données présenter dans le chapitre 5.