Université de Montréal

Composition automatique de musique à l'aide de réseaux de neurones récurrents et de la structure métrique

par

Jasmin Lapalme

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Mémoire présenté à la Faculté des études supérieures
en vue de l'obtention du grade de
Maître ès sciences (M.Sc.)
en informatique

novembre 2005

Université de Montréal

Faculté des études supérieures

Ce mémoire intitulé

Composition automatique de musique à l'aide de réseaux de neurones récurrents et de la structure métrique

présenté par

Jasmin Lapalme

a été évalué par un jury composé des personnes suivantes :

_____

(président-rapporteur)

Douglas Eck

_____

(directeur de recherche)

Mémoire accepté le:

_____

Résumé

Nous proposons un modèle qui génère de la musique dans un certain style. Nous utilisons les techniques d'apprentissage machine pour apprendre les séquences musicales. Notre modèle est un réseau de neurones récurrents Long Short-Term Memory (LSTM). Les pièces musicales sont fortement contraintes par des structures à long terme dont une appelée la structure métrique. Cette structure permet de garder une forme fixe et globale durant toute une pièce. Nous proposons une méthode pour encoder cette structure avec différentes représentations de notes, via des évèvements passées auparavant (time delays) relatif à la structure métrique, afin que les réseaux de neurones puissent en tenir compte. Notre modèle génératif utilise cette structure afin d'apprendre les dépendances temporelles entre les notes et il permet d'obtenir de meilleures prédictions qu'un modèle n'en tenant pas compte. Nous ajoutons une couche cachée standard à LSTM pour aider à la génération de mélodies. Cette couche permet de générer de meilleures mélodies avec un contour mélodique.

Mots-clés

Informatique, musique, réseau de neurones récurrents, Long Short-Term Memory, structure métrique

Abstract

In this work, we use machine-learning methods to develop a generative model of music, trained to play in a certain style. Our model is a type of recurrent neural network called Long Short-Term Memory (LSTM). While musical pieces are typically characterized by several types of long-timescale structure, the one we focus on is the metrical structure. We propose a method to encode this structure via time delays furnished to the LSTM network. Using this technique, we obtain better prediction results than a model that ignores metrical structure. We further improve our generative model by adding a standard feed-forward hidden layer to LSTM model ; this allows the network to generate better melodic contours.

Keywords

Computer science, music, recurrent neural networks, Long Short-Term Memory, metrical structure

Table des matières

Liste des tableaux

Table des figures

Chapitre 1

INTRODUCTION

La musique et l'informatique sont deux domaines qui convergent de plus en plus. L'écoute de musique sur son ordinateur fait maintenant partie courante de la vie de beaucoup de gens. La demande pour des outils qui gèrent, traitent et jouent de la musique ne cesse d'augmenter avec l'arrivée des lecteurs portatifs MP3, tels que les iPod en 2001. Ces lecteurs peuvent contenir jusqu'à 60 Go de données ce qui en fait une base de données intéressantes à traiter pour suggérer des chansons à acheter et/ou à jouer. On peut pousser plus loin en voulant composer de nouvelles chansons selon le style se trouvant dans le lecteur portatif. Pour faire ceci, nous avons besoin de modèles qui sont capables de traiter la musique de façon efficace.

La musique contient des séquences spéciales vues les structures fixes contenues dans une chanson. Comme l'ont dit Cemgil et al. [2003] dans leur conclusion, les musiciens ont des connaissances à priori très fortes sur les structures musicales, les harmonies, tempo et expressions. Nous utilisons la structure métrique qui se retrouve dans la musique. La structure métrique est une structure hiérarchique inférée par les temps forts et faibles survenant dans une chanson. Eck and Casagrande [2005] présentent un modèle qui permet de trouver cette structure métrique lorsque les moments où surviennent les notes sont fournis. Or, nous avons ces moments car nous utilisons des fichiers MIDI. Nous discutons de la structure métrique plus en détail dans la section 2.3.

Nous proposons un modèle qui génère de la musique dans un certain style. Notre modèle est basé sur les réseaux de neurones couramment utilisés dans le domaine de l'apprentissage machine. Pour générer des séquences qui correspondent à de la musique, nous devons avoir un modèle dynamique qui peut comprendre les dépendances temporelles dans les séquences. Les modèles génératifs de musique ont été étudiés dans la communauté scientifique. Nous faisons un survol des différentes méthodes utilisées dans le chapitre 3. Nous utilisons un réseau de neurones récurrents LSTM qui permet un apprentissage de séquences, initialement proposé par Hochreiter and Schmidhuber

[1997]. Ce type de réseau a des propriétés importantes pour la musique qui donnent des avantages par rapport aux réseaux de neurones standards. Nous expliquons ceci en détail dans la section 3.2.2.

L'apprentissage des réseaux de neurones consiste à minimiser une fonction d'erreur correspondant à la tâche que l'on veut réaliser. Nous utilisons la descente de gradient pour minimiser l'erreur de prédiction de la prochaine note. Ce type de tâche se nomme next-step prediction. Nous détaillons les calculs des passes avant et arrière pour les réseaux récurrents standard (Section 4.1) et LSTM (Section 4.2).

Les réseaux de neurones prennent des vecteurs de nombres réels, or nous devons représenter les notes de musique de façon vectorielle. Dans le chapitre 5, nous détaillons les trois types de représentations que nous avons utilisées : locale, fréquentielle et une motivé par les principes de psycho-acoustique (Shepard). Nous les avons comparées afin de comprendre comment elles diffèrent en générant de la musique. Nous n'avons pas pu les départager l'une par rapport à l'autre par les générations qu'elles produisaient. Par contre, les représentations fréquentielle et de Shepard diminuent le temps d'apprentissage car elles ont des dimensions plus petite que la représentation locale.

Nous utilisons la structure métrique dans notre modèle génératif. Nous avons développé une technique permettant à notre modèle d'avoir cette information à priori. Nous avons effectué des expériences pour évaluer l'importance de cette information. Nous calculons l'erreur de prédiction de la prochaine note étant donné les notes précédentes. Les chansons générées à l'aide du modèle entraîné nous permettent d'évaluer la performance de notre modèle à générer de nouvelles séquences dans le même style. Nous avons modifié le modèle LSTM afin qu'il puisse avoir des neurones cachées en parallèles aux blocs LSTM.

Lorsque nous utilisons notre méthode pour fournir la structure, nous devons être conscients de la dimension de la représentation d'une note. La dimension de la représentation ne doit pas être excessive car nous quadruplons la dimension du vecteur d'entrée lorsque nous utilisons la structure métrique (Section 6.2.2). Ce problème est résolu en utilisant des représentations qui n'augmentent pas de taille lorsque le nombre de notes possible à générer augmente.

Nous obtenons de meilleurs résultats en prédiction en tenant compte de la structure métrique par rapport à un modèle sans structure métrique. Ces résultats (Chapitre 7) sont obtenus avec de vraies chansons en format MIDI provenant de deux bases de données différentes. Les chansons générées ont une meilleure structure globale que

celles obtenues avec les modèles de génération note par note existants. Les neurones cachées ajoutées aux blocs LSTM permettent un meilleur contour mélodique.

Nous terminons ce mémoire en analysant le résultat de nos expériences et en présentant les travaux futurs intéressants à entreprendre.

Chapitre 2

MUSIQUE

Dans ce chapitre, il y aura une introduction à la théorie de la musique, une explication des différents formats digitaux de musique et une explication de la structure métrique dans la musique.

La structure métrique est un élément majeur de la musique. Nous en tenons compte dans nos modèles car il permet de simplifier l'apprentissage de nos modèles et d'ajouter une structure globale à nos compositions. La section 6.2.2 explique comment cette structure métrique est utilisée dans nos modèles.

2.1. Théorie de la musique

Il y a quelques principes importants de théorie de la musique qu'il nous faut expliquer.

Dans la musique occidentale, le plus petit intervalle entre deux notes est appelé le demi-ton. Il y a sept noms de notes : Do, Ré, Mi, Fa, Sol, La et Si. Cette série contient 12 demi-tons. Une note a le même nom si elle est jouée à un intervalle de 12 demi-tons de différence. Cet intervalle s'appelle l'octave. Entre le Do et le Ré, le Ré et le Mi, le Fa et le Sol, Sol et le La il y a 2 demi-ton (ou un ton) d'intervalle. On peut désigner une note entre ces notes en ajoutant des altérations aux notes : le dièse (on ajoute un demi-ton) ou le bémol (on enlève un demi-ton). Alors le Do dièse (Do♯) est équivalent au Ré bémol (Ré♭). On nomme l'intervalle de 7 demi-tons la quinte. C'est un intervalle important car il fait partie la majorité des accords. La figure 2.1 montre ces intervalles.

On peut représenter ces 12 notes (les 7 notes correspondant à chaque nom et les 5 notes intermédiaires) à l'aide d'un cercle pour représenter leur proximité psycho-acoustique (Shepard [1982]). On divise un cercle en douze pointes et on y place chacune des notes. Le cercle chromatique (2.2) est celui où les notes adjacentes sont séparées d'un demi-ton. Le cercle des quintes (2.3) est celui où les notes adjacentes sont séparées

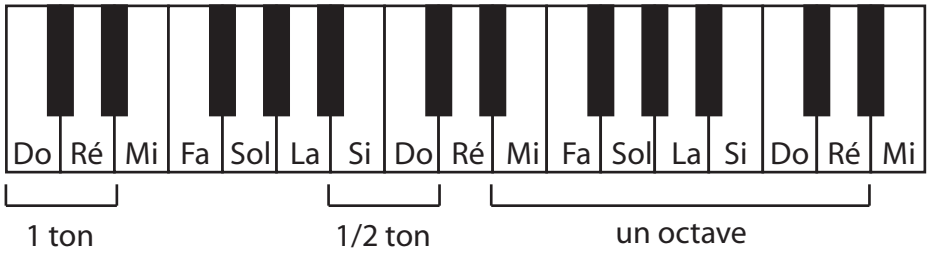

Fig. 2.1. Un clavier de piano. Les notes sur le piano sont séparées par un demi-ton. Toutes les notes blanches représentent des notes sans altérations. Les altérations permettent de nommer les notes noirs. Le dièse (♯) ajoute un demi-ton à la note et le bémol (♭) enlève un demi-ton. La note noire entre le Do et le Ré s'appelle soit Do♯ ou Ré♭.

d'une quinte. Il est intéressant de remarquer que dans les deux cercles, les paires de notes opposées dans les deux cercles sont identiques. Cet intervalle est appelé triton, pour trois tons ; il est connu pour la grande dissonance entre deux notes séparées par cet intervalle.

On appelle la tierce majeure un intervalle de 4 demi-tons et la tierce mineure un intervalle de 3 demi-tons. Dans la formation d'un accord majeur à trois notes, on prend une note que l'on appelle la tonique et dont l'accord prendra le nom de cette note. Les deux autres notes seront la tierce majeure et la quinte de la tonique. Pour un accord mineur, on prendra la tierce mineure au lieu de la tierce majeure. Franklin [2004] a utilisé les tierces majeures et mineures pour créer une représentation vectorielle de notes. Les cercles montrant sa représentation sont utilisées à la figure 2.4. Winold and Rehm [1979] fait une revue plus complète de la théorie de la musique.

On peut représenter les notes de façon fréquentielle. Il y a un standard où le La est un son de fréquence fondamentale 440Hz. Si on écrit la gamme de façon fréquentielle se sera une courbe exponentielle. Une note séparée d'un octave d'une autre aura une fréquence fondamentale deux fois plus grande et portera le même nom. Pour obtenir la fréquence de la prochaine note dans la gamme, on multiplie la fréquence fondamentale par $\sqrt[12]{2}$. Dans la table 2.1, un octave est représenté avec la fréquence de chaque note.

Lorsque l'on entend une note, on perçoit une fréquence fondamentale ainsi que des harmoniques à cette fréquence fondamentale à différentes amplitudes. On obtient ces amplitudes en calculant la transformée de Fourier du signal entendu. On peut donc écrire tout signal comme l'équation 2.1.

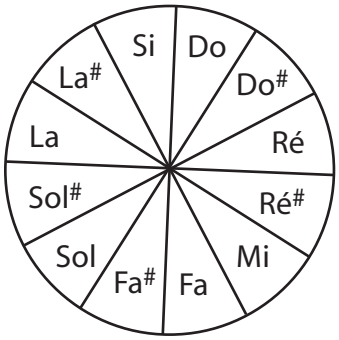

Fig. 2.2. Le cercle de la gamme chromatique. Chaque note adjacente à une autre a un intervalle d'un demi-ton.

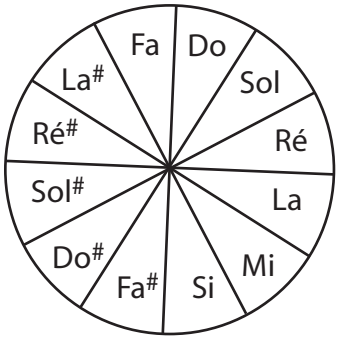

Fig. 2.3. Le cercle des quintes. Chaque note adjacente à une autre a un intervalle d'une quinte.

Note	Fréquence (Hz)
La	440
Si♭	466
Si	494
Do	523
Do♯	554
Ré	587
Ré♯	622
Mi	659
Fa	698
Fa♯	740
Sol	784
Sol♯	831

Tab. 2.1. Fréquence des 12 notes se retrouvant dans le même octave.

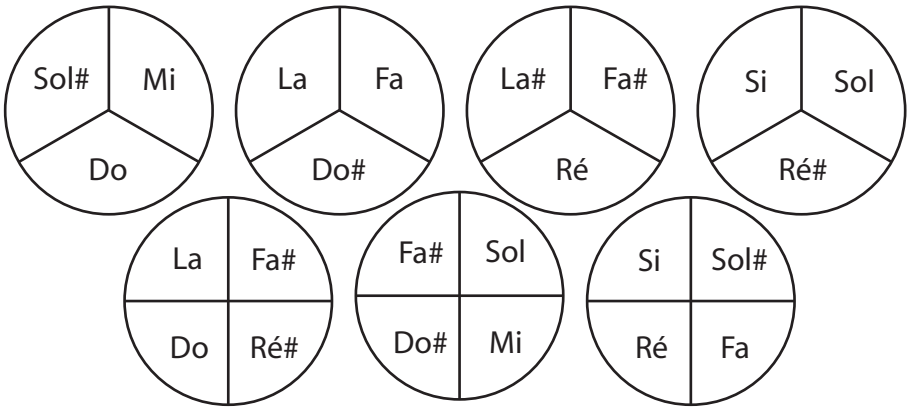

Fig. 2.4. Figure tirée de Franklin [2004]. Les cercles du haut représentent les tierces majeures (4 demi-tons) et les cercles du bas les tierces mineures (3 demi-tons). Sa représentation est un vecteur à 7 dimensions (1 pour chaque cercle) et un 1 est mis pour chaque cercle contenant la note que l'on veut représenter. Un La est représenté par le vecteur $[0100100]$.

$$s(t) = \sum_{i=1}^{\infty} a_i \sin(i f_0 t) \tag{2.1}$$

où a_i est l'amplitude de la i-ème harmonique de f_0.

Lorsque l'on entend deux notes qui partagent des harmoniques, ces deux notes tendent à se ressembler. C'est ainsi que la quinte est un intervalle où les deux notes tendent à se ressembler car on obtient la quinte en multipliant la fréquence fondamentale par un facteur tout près de $\frac{3}{2}$. Les suites d'harmoniques ($\{x, 2x, 3x, 4x, ...\}$ et $\{\frac{3}{2}x, 3x, \frac{9}{2}x, 6x, ...\}$) des deux notes séparées d'une quinte partagent plusieurs harmoniques. La figure 2.5 présente des transformées de Fourier de certaines notes jouées au piano.

Ceci est important lorsque nous utilisons des représentations vectorielles à présenter à nos réseaux de neurones. Nous utilisons des représentations où la distance euclidienne entre deux vecteurs représentent la distance psycho-acoustique. Les sections 5.2 et 5.3 décrivent deux représentations qui tiennent compte des harmoniques des notes.

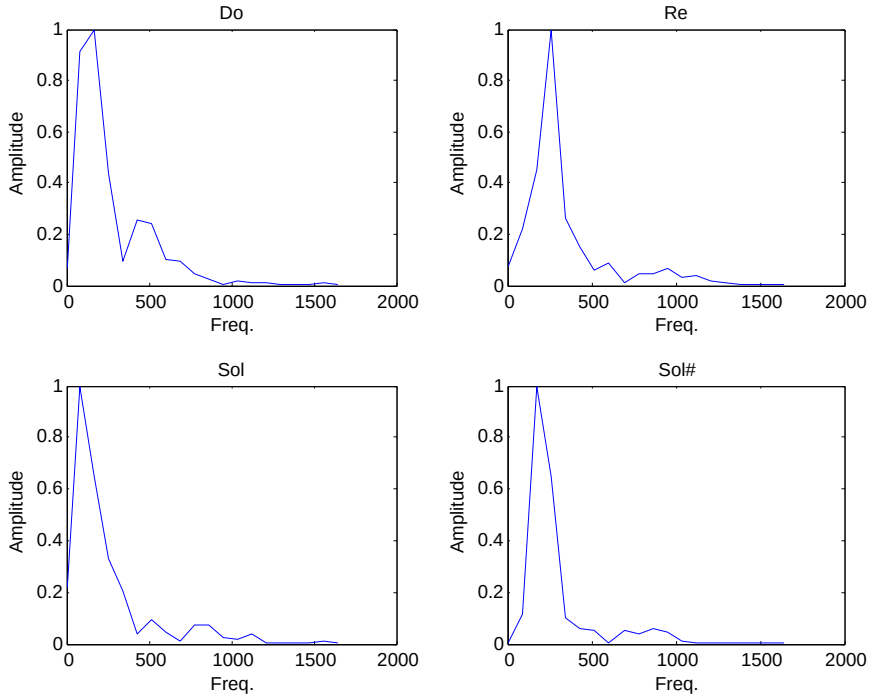

Fig. 2.5. Transformées de Fourier de différente notes jouées sur un piano. L'axe des x représente les fréquences et l'axe de y représente l'amplitude (la force perçu auditivement) d'une fréquence. On peut remarquer que le Sol (En bas à gauche) et le Do (en haut à gauche) ont une transformée semblable : Les piques dans les graphes sont alignées sur l'axe des x, on dit d'elles qu'elles partagent des harmoniques.Par contre, on remarque que le Do (en haut à gauche) et le Sol♯ (en bas à droite), les transformées sont différentes. Le Sol♯ n'a pas de deuxième pique comme l'ont le Do et le Sol. La norme de la différence des deux vecteurs des transformée entre le Do et Sol est de 0,2650 et le Do et Sol♯ est de 0,7686. La distance est plus grande entre le Do et Sol♯. Le Do et Sol♯ ont un triton d'intervalle (Voir le début de la section 2.1).

2.2. Formats de musique

La musique numérique se divise en deux grands formats : Les fichiers MIDI (Musical Instrument Digital Interface) et PCM (Pulse code modulation).

2.2.1. MIDI (Musical Instrument Digital Interface)

Le format MIDI est un format très compact mais peu réaliste. Seuls les moments où une note est jouée sont inscrits dans le fichier. Pour chaque note, il y a une indication

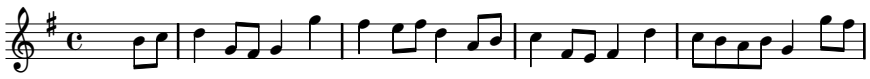

Fig. 2.6. Ceci est la partition d'une bourrée de J.S Bach générée à partir d'un fichier MIDI (le même que la figure 2.7).

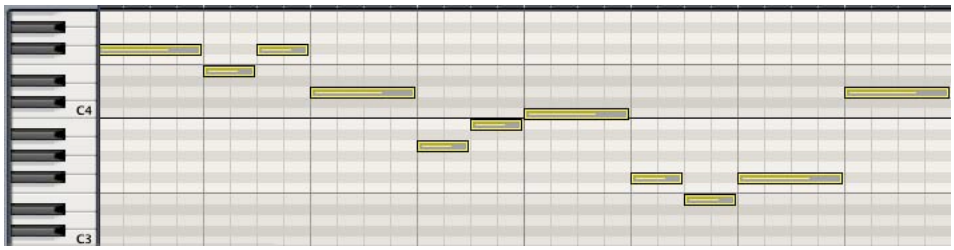

Fig. 2.7. Ceci est le piano roll d'une bourée de J.S Bach généré à partir d'un fichier MIDI (le même que la figure 2.6). L'axe horizontal représente le temps et l'axe vertical représente la hauteur des notes (Les notes de piano sont réprésentées). Les notes jouées à un instant (sur l'axe des x) sont les notes dont où un rectangle est vis-à-vis cet instant. Dans cet exemple, chque note est jouée seule.

de la vélocité, de la hauteur, de la durée et du type d'instrument. Pour faire jouer ces fichiers, nous devons avoir des échantillons de note pour que ces échantillons soit joués aux moments indiqués dans le fichier. Ce format de fichier a plusieurs avantages, en particulier la partition exacte (figure 2.6) peut être induite du fichier. Ainsi, nous pouvons obtenir la position et hauteur exactes de chaque note. On ne peut obtenir ceci d'un fichier PCM, du moins sans faire de traitement de signal complexe.

Une deuxième représentation graphique, autre que la partition, est utilisée pour un fichier MIDI. On l'appelle le piano roll, représenté à figure 2.7.

2.2.2. PCM (Pulse code modulation)

Le format PCM est un format brut de musique. C'est une série de valeurs comprises entre un intervalle défini auparavant (entre 0 et 2^n où n est le nombre de bits sur lesquelles on encode chaque échantillon). Cette valeur représente la position de la forme d'onde de la musique à un instant. Pour obtenir ce format à l'aide d'un son, on échantillonne le son à une vitesse définie auparavant (44,1 kHz pour les DC audio) et on mesure la position de la forme d'onde à chaque temps. Ce format est très lourd en mémoire, mais

il permet une reproduction très précise du son si on échantillonne à vitesse relativement élevée.

Tous les formats de compression tel que MP3, M4A, OGG, etc... prennent comme source le format PCM. Lors du décodage, on obtient à nouveau un PCM le plus près possible de l'original qui est retransmis à une sortie tel qu'un haut-parleur.

Nous utilisons les fichiers MIDI avec tous nos modèles, car nous nous devons disposer de la position et de la hauteur exactes de chaque note. La qualité sonore n'affecte pas nos modèles.

2.3. Structure métrique

Pour analyser la musique, nous devons utiliser différents outils.

Lerdahl and Jackendoff [1983–1984] mentionnent deux types de structure dans la musique :

- Structure par groupe (Grouping structure)
- Structure métrique (Metrical structure)

et deux types de réductions pour analyser la musique.

- Réduction par étirement du temps (Time-span reduction)
- Réduction par prolongation (Prolongational reduction)

Ces deux types de structure et deux types de réductions permettent de mieux analyser la musique. Les musicologues peuvent s'en servir pour différencier certains styles de différents compositeurs.

2.3.1. Structure par groupe

La structure par groupe est une structure basée sur la hauteur et la longueur des notes uniquement. On veut regrouper des notes qui sont des formes (une forme en musique est l'équivalent d'une phrase dans le langage) dans la chanson. Avec ces groupes, on peut former une hiérarchie qui nous permettra d'identifier les formes générales d'une chanson jusqu'à la plus spécifique. Tous les groupes sur un même niveau de hiérarchie ne doivent jamais être à cheval les uns sur les autres. La hiérarchie des groupes ne suit pas toujours la structure métrique mais elle est souvent corrélée. La figure 2.8 montre un exemple de la structure par groupe.

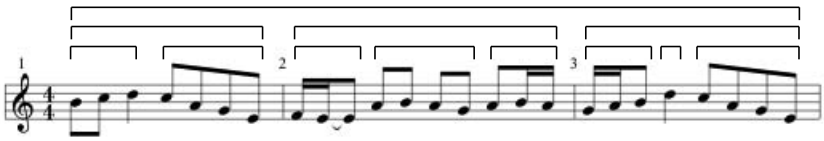

Fig. 2.8. Représentation de la structure par groupe. Ce regroupement est basé sur les notes sans se fier à la métrique de la chanson. On remarque par contre qu'il y a une corrélation entre les deux.

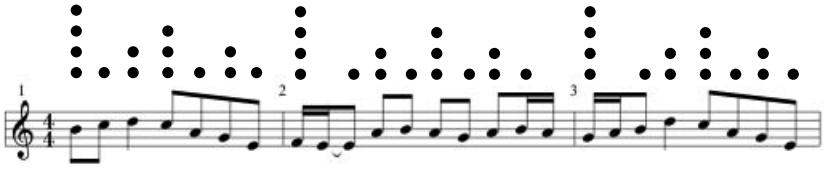

Fig. 2.9. Une partition avec les niveaux de hiérarchie (niveaux des points) de la structure métrique.

2.3.2. Structure métrique

La structure métrique est inférée par les temps forts et faibles perçus dans la musique. Ces temps forts et faibles sont reconnus à l'aide des niveaux hiérarchiques de séquences périodiques dans une chanson.

Ces niveaux de hiérarchies sont représentés dans la notation moderne de la musique à l'aide des rondes, blanches, noires, croches, etc...(Handel [1993]) . Une ronde dure 2 fois plus longtemps qu'une blanche, une blanche dure 2 fois plus longtemps qu'une noire et ainsi de suite. Les mesures représentent un nombre de temps fixe tout au long de la chanson. La plupart des chansons pop contemporaine ont 4 temps dans une mesure. Les temps forts sont sur le premier et troisième temps de chaque mesure. Lorsque que l'on écoute une chanson et nous tapons du pied, nous tapons sur des temps forts de la chanson. Cette hiérarchie est représentée dans la figure 2.9.

Avec cette structure métrique, on peut prévoir plus facilement les changements d'accords et les répétitions qui surviennent dans une chanson (Cooper and Meyer [1960]). Dans la plupart des chansons pop moderne, une série d'accord est répétée plusieurs fois et cette série d'accord est alignée avec les mesures. Les changements d'accord dans cette série le sont aussi.

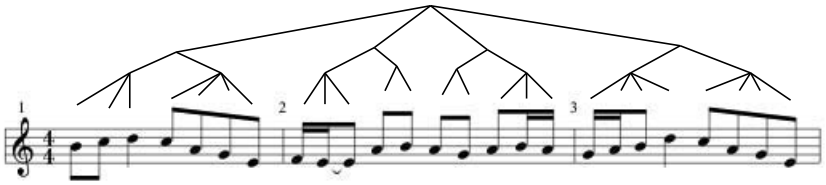

Fig. 2.10. Réduction par étirement du temps. L'arbre est bâtit à l'aide des deux strutures, métrique et de groupe, et à chaque noeud on peut identifier une note. Cette note est souvent déterminé par la structure métrique.

2.3.3. Réduction par étirement du temps

Avec ces deux types de regroupement, ils est maintenant possible de faire des réductions : c'est à dire, enlever des notes dans la chanson originale et conserver l'essentiel de la chanson.

Le réduction par étirement du temps consiste à créer un arbre basé sur la structure par groupe et la structure métrique. À chaque noeud de l'arbre, on doit identifier la note la plus importante parmi ces fils. La note sera souvent déterminée par la structure métrique puisqu'elle identifie les temps forts et faibles. La longueur de la note est la somme de toutes les longueurs des notes filles. Pour faire la réduction, on choisit un niveau de l'arbre et on prend chacune des notes. La figure 2.10 représente un arbre bâti à l'aide la structure métrique et la structure par groupe.

2.3.4. Réduction par prolongation

L'autre de type de réduction consiste à exprimer les moments relax et intenses d'une chanson. En résumé, on identifie ces moments en analysant deux événements qui ont le même parent dans la hiérarchie. Si l'événement arrivant en premier dans la chanson est plus stable que le premier, on identifiera ce moment comme "intense". Si le contraire survient, le moment sera identifié comme "relax". Un évènement stable est un évènement qui est aligné avec la structure métrique. Une appogiature n'est pas un évènement stable car elle n'est jamais alignée à la structure métrique.

Dans Lerdahl and Jackendoff [1983–1984], il indique cette réduction sur le même arbre que la réduction par étirement du temps en mettant des ronds vides ou pleins aux noeuds de l'arbre.

2.4. Conclusion

Nous avons fait un survol très bref de la théorie de la musique, des formats digitaux de musique et des différentes structures dans la musique.

Nous accordons beaucoup d'importance à cette structure métrique, car dans la prochaine section nous verrons que les modèles actuels de génération de musique souffrent d'un problème majeur. Ils ont de la difficulté à apprendre des corrélations entre des événements séparées par de longs intervalles. Cette structure métrique nous permet de simplifier le problème en identifiant aux modèles les intervalles importants, et qui sont fixes en musique. La section 6.2.2 explique comment nous tenons compte de cette structure métrique dans nos modèles. Elle ajoute une structure globale à nos compositions et simplifie l'apprentissage de nos modèles.

Chapitre 3

COMPOSITION AUTOMATIQUE

La composition automatique de musique n'est pas une idée nouvelle. Il y a des systèmes et logiciels commerciaux qui composent de la musique mais sans l'aide de techniques d'apprentissage machine tel que Band in a Box (www.pgmusic.com/bandbox.htm) et Wolfram tone (tones.wolfram.com). Ces systèmes déterminent un grand ensemble de règles sur la structure globale, les répétitions, le contour mélodique, etc.. et insèrent un élément aléatoire en choisissant au hasard parmi les règles.

Dans les prochaines sections, nous expliquons différentes approches existantes en composition automatique de musique.

3.1. Systèmes à règles

En utilisant ces principes de règles, Henz et al. [1996] ont fait un système qui permet à l'usager de choisir certaines contraintes pour contrôler le style de la musique. Ils appellent leur système un "intention-based system". Plus précisément, leur système compose des chansons à quatre voix (Basse, Ténor, Alto, Soprano). Ils commencent par choisir une tonalité qui sera la même pour toute la chanson. Ils ont toujours une contrainte que la Basse doit jouer des notes plus basses que le Ténor, le Ténor doit jouer des notes plus basses que l'Alto, l'Alto doit jouer des notes plus basses que le Soprano. Ils vont construire la chanson en appliquant successivement, ce qu'ils appellent, des fonctions harmoniques, aux notes. Ils vont choisir parmi ces fonctions celles qui vont respecter les contraintes. Les fonctions sont basées entre autres sur les modes mineurs, modes majeurs, les tierces et les quintes. Ils ajoutent une contrainte qui empêche les notes d'une même voix de sauter d'un trop grand écart. L'utilisateur doit choisir parmi des fonctions harmoniques et ainsi générer une composition.

Dans la même lignée des systèmes à règles, il y a des compositeurs qui utilisent des algorithmes génétiques pour générer de la musique. Ils traitent une séquence de notes comme étant une population vivante. Ils commencent avec une séquence d'une seule

note et la font évoluer pour obtenir une séquence de plusieurs notes pour avoir une chanson. La section “Evolving Composition Systems” de Todd and Werner [1999] explique le principe commun de cette méthode. Les algorithmes qui modélisent l'évolution de populations vivantes vont tenter d'avoir beaucoup de nouveauté dans les individus qui sont créés. Chaque individu est unique dans une population. Dans le cas de la musique, les individus d'une population correspondent aux notes dans une séquence musicale. En musique, il a des notes qui sont identiques au cours d'une même chanson ce qui n'est pas le cas dans une population vivante. Ainsi pour éviter qu'il y ait trop de nouvelles notes, ils ajoutent des règles qui permettent de garder une structure. Cette structure correspond souvent à la structure métrique et tente de répéter des sous-séquences pour obtenir structure globale tout au long de la chanson. L'algorithme général consiste donc à générer plsuieurs séquences à partir de la population de notes existantes. Ensuite, avec toutes ses nouvelles séquences, on élimine toutes les séquences générées qui ne respectent pas les règles. Avec ces nouvelles séquences, on regénère d'autre séquences. On répète ces étapes jusqu'à ce que nous obtenions une séquence assez longue.

Pour l'implanter cet algorithme, il faut énumérer une série de règles pour modifier les séquences existantes. Ces règles doivent respecter le plus possible le contour mélodique et la structure métrique. Il faut aussi énumérer une série de règles pour éléminer les séquences qui ne les respectent pas. Il peut être fastidieux d'énumérer ces règles.

Un des meilleurs modèle génératif de musique par règle est fait par Cope [1991]. Son modèle identifie différentes structures parfois de façon manuelle ou automatique dans des pièces classiques. Il identifie les formes qu'il a appelle “signature” avec des techniques de pattern matching. Ces signatures permettent selon lui d'identifier le style musicale. Avec ces signatures, il trouve les circonstances dans lesquelles les signatures peuvent être accolées. Ceci est qu'une infime partie des traitements qu'il impose aux chansons dans sa base de données. Suite à ces traitements, qui sont justifiés par des arguments de théorie musicale, il obtient une série de règles qui restreint le modèle a composé de la musique dans le même style. Les chansons produites sont très intéressantes, mais on se demande si vraiment l'ordinateur à composer car l'usager doit fournir beaucoup d'information pour créer les règles.

3.2. Système par apprentissage

Le domaine auquel nous nous intéressons est plus au niveau des techniques d'apprentissage machine applicables à la génération de musique. Nous nous y intéressons

car nous tentons d'imiter ce qu'un humain fait quand il apprend à composer de la musique dans un certain style en écoutant d'autres chansons du même style. Par la suite, il généralise ce qu'il a entendu et en compose de la nouvelle. Les musiciens font ceci inconsciemment mais les méthodes d'apprentissage permettent de faire ceci avec les machines.

Nous ciblons des techniques telles que les tables de transitions, modèles graphiques et réseaux de neurones. Nous tentons d'apprendre le style d'un groupe de chansons et d'être capable de générer une nouvelle chanson dans le même style.

3.2.1. Tables de transitions

La technique intuitive est celle de la table de transitions. Nous tentons de modéliser un style en apprenant ou en fixant la probabilité de générer une note sachant les notes passées. Nous voulons apprendre :

$$P(n(t)|n(t-1), n(t-2), ...)$$

où $n(t)$ est la note au temps t.

Nous pourrions dire qu'un style correspond à un ensemble de probabilités de transition. Il est possible d'apprendre ces probabilités à l'aide d'un modèle n-gram et d'une technique de lissage (Kantor [2001]). Par contre, comme mentionné dans Mozer [1994], cette technique souffre de quelques inconvénients. Par exemple, elle ne peut se servir de la note au temps t pour prédire la note au temps $t+n$ sans avoir à gérer les notes au temps $t+n-1$, $t+n-2$,... . Il est parfois plus utile de connaître la note au temps t que la note au temps $t+n-1$ pour prédire la note au temps $t+n$. Ceci est dû à la structure métrique qui se retrouve dans la musique. Nous accordons une grande importance à ce fait et nous en tenons compte dans notre approche. La musique n'est pas locale, les événements qui se sont passé il y a plusieurs temps sont très importants. Nous avons donc besoin d'un grand n et alors le problème devient "sparse". Le probème devient "sparse" car le nombre de séquences possibles de n notes grandit exponentiellement avec n pour prédire la prochaine note. Si nous voulons prédire une note dans un ensemble de M notes, le nombre de possibiblité de séquences de n notes sera de M^n. On appelle ce problème la malédiction de la dimensionnalité car pour apprendre ce genre de problème, le nombre d'exemple d'entraînement devra augmenter exponentiellement aussi.

3.2.2. Réseaux de neurones

Une autre technique est celle des réseaux de neurones sur laquelle nous allons nous concentrer. Nous devons utiliser un réseau de neurones qui se rappelle du passé. En musique, les séquences de notes sont très fortement corrélées par le temps. Nous ne pouvons donc pas utiliser un réseau "feed-forward" standard. Nous allons plutôt utiliser les réseaux de neurones récurrents pour ajouter cette notion de mémoire. Un réseau récurrent utilise les valeurs du passé pour calculer les nouvelles valeurs de neurones. Mozer [1994] a fait de la composition avec des réseaux de neurones sans avoir des résultats convaincants avec des corpus d'entraînement de musique composée par un humain. Il dit de son modèle qu'il fait de la musique que seule sa mère peut aimer. Il a montré que les réseaux de neurones étaient cependant plus puissants que les n-gram.

Un problème majeur des réseaux de neurones récurrents, tel que défini dans Williams and Zipser [1989], est que ce modèle ne peut apprendre des événements ayant des dépendances à long terme (Bengio et al. [1994]). Ceci est causé par le "vanishing gradient", c-à-d que le gradient diminue à mesure que l'on remonte loin dans le temps. Ceci nous convainc qu'il sera très difficile d'apprendre de la musique avec un réseau de neurones récurrents. La musique a des dépendances précises à long terme. Par exemple, les changements d'accord dans une chanson surviennent à des moments très précis et ne peuvent être décalés d'un temps sans que l'auditeur ne s'en aperçoive.

Nous utilisons un modèle qui souffre moins de ce problème qui se nomme LSTM (Section 3.2.2.1). Nous utilisons aussi différentes techniques pour aider à contrer ce problème en encodant la structure métrique (Section 6.2.2).

3.2.2.1. Long Short-Term Memory

Un type de réseau de neurones récurrents qui peut ne pas souffrir du vanishing gradient est LSTM (Long short-term memory) présenté par Hochreiter and Schmidhuber [1997]. Gers [2001] dans sa thèse de doctorat a ajouté le concept des forget gate et peephole connection. Le modèle est expliqué en détail dans la section 4.2. Un résultat très intéressant à propos de LSTM est qu'il est possible de faire apprendre des langages tel que $a^n b^n$ avec un n très grand (Schmidhuber et al. [2002]). LSTM est capable de compter, ce qui est intéressant au niveau de la musique. Il peut ne pas souffrir du vanishing gradient car il possède des neurones dont la valeur n'est pas limitée par une fonction d'activation, mais plutôt par des multiplications, appelés portes, aux alentours

de ceux-ci. Ces multiplications sont apprises pendant l'apprentissage. Dans la section 4.2.1, l'équation 4.16 explique le calcul de ces neurones.

LSTM a eu du succès dans la génération de musique (Eck and Schmidhuber [2002]) pour appendre des structures de chansons blues et générer des accords d'accompagnement ainsi que des solos dans la gamme blues. Il utilise la technique du next-step prediction qui consiste à prévoir la prochaine note dans la séquence. Le modèle apprend très bien la structure des accords, mais les solos générés ne sont pas très satisfaisants. LSTM a été utilisé pour apprendre d'autres corpus de musique à l'aide de différentes représentations de notes (Franklin [2004]) dont celle qui utilise les tierces majeures et mineures (figure 2.4).

On remarque quand même que LSTM ne suit pas une structure globale pour la musique. Il peut ainsi capter des corrélations entre événements décalés par plusieurs temps, mais ne reste jamais dans une structure fixe. Cette structure est définie par la structure métrique définie dans la section 2.3. Il est possible de déterminer la structure métrique d'une chanson si on connaît le temps exact où les notes sont jouées (Comme c'est le cas dans un fichier MIDI). Eck and Casagrande [2005] montre un modèle efficace et exact qui permet de la déterminer. Il utilise la technique de l'auto-correlation pour construire la hiérarchie de la structure métrique. Le modèle se trompe très rarement avec des fichiers MIDI et lorsqu'il est effectué avec plusieurs fichiers du même style on peut trouver les erreurs automatiquement en supposant qu'ils ont tous la même structure métrique.

3.3. Conclusion

Nous avons vu un ensemble de modèles existants pour la génération de musique automatique. Nous allons nous concentrer sur LSTM car nous avons vu qu'il souffre moins du problème de "vanishing gradient" et qu'il a déjà été utilisé avec succès dans la génération de musique.

Chapitre 4

RÉSEAUX DE NEURONES RÉCURRENTS

Dans cette section, nous expliquons le calcul et l'implantation des deux réseaux récurrents que nous utilisons. Dans les deux cas, il y a un ensemble d'entraînement D qui contient N séquences, d_i, où $1 \leq i \leq N$. L'ensemble d'entraînement aura des entrées de m_{in} dimensions et des cibles de m_t dimensions. Chaque séquence, d_i, contient n_i tranches de temps. Pour chaque tranche de temps, il y a une série d'entrées ordonnée $I_j^i(t)$ où $1 \leq j \leq m_{\text{in}}$ et une série de cibles ordonnée $T_j^i(t)$ où $1 \leq j \leq m_t$ et $1 \leq t \leq n_i$. Dans notre cas, avec la musique, nous ferons abstraction du rythme : chaques notes auront la même longueur. La section 6.2 explique en détail l'encodage des notes pour les réseaux de neurones.

4.1. Rétropropagation à travers le temps

Cette section est inspirée de Williams and Zipser [1995].

4.1.1. Structure du réseau

Soit un réseau récurrent où il n'y a pas de connexion récurrente entre les unités cachées ou de sortie vers les unités d'entrée. Nous aurons alors un ensemble U avec des unités u_i à l'intérieur. Il y a trois sous-ensembles de U, u_{in} qui contient les unités d'entrée, u_{hid} qui contient les unités cachées et O qui contient les unités de sortie[1]. L'intersection de ces trois ensembles est l'ensemble vide et l'union est U. Nous avons aussi un ensemble L qui contient l'information sur les arcs du réseau. Chaque élément de L est un tuple à quatre éléments $(u_s, u_d, d_{sd}, w_{sd})$ dont le premier est l'unité source de l'arc (u_s), le deuxième est l'unité de destination(u_d) de l'arc, le troisième est un entier qui représente le délai avant que la valeur de u_s se propage à u_d et le quatrième est le poids de l'arc w_{sd}

[1]Nous utilisons O pour utiliser la même notation que le modèle LSTM.

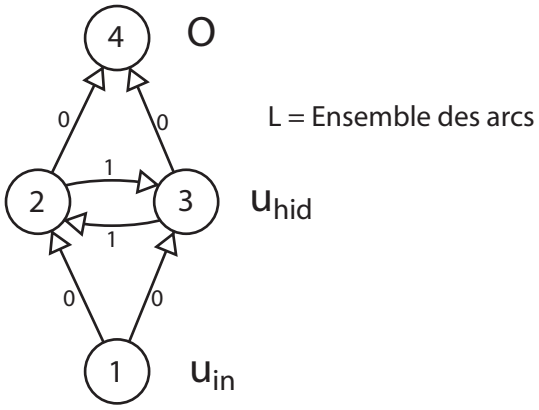

Fig. 4.1. Réseau de neurones récurrents avec les trois ensembles de neurones : les unités d'entrée (u_{in}), cachées (u_{hid}) et de sortie (O). Sur chaque arc, un nombre indique le délai, en nombre d'étapes, avant que la valeur du neurone source (s) influence le neurone de destination(d). Le poids de chaque arc est de w_{sd}. Pour avoir un réseau complètement connecté, il faut que chaque neurone d'entrée soit connecté à chaque neurone caché sans délai, chaque neurone caché connecté à chaque neurone caché avec un délai d'un pas et chaque neurone caché à chaque neurone de sortie sans délai.

Dans un tel réseau, nous allons appliquer le même principe que dans un réseau de neurones multi-couches en propageant le gradient dans tous les neurones. Cette technique s'appelle la rétropropagation à travers le temps. Nous devons dérouler le réseau dans le temps comme la figure 4.2 l'illustre.

4.1.2. Passe avant

Dans la passe avant, nous voulons calculer la valeur des unités de sortie à chaque temps. Ainsi, nous pourrons calculer une valeur d'erreur en comparant ces sorties avec les cibles dans notre ensemble d'entraînement. Dans la séquence i, la valeur de chaque neurone j au temps t sera

$$x_j^i(t) = \begin{cases} 0 & \text{si } t < 0 \\ I_j^i(t) & \text{si } u_j \in u_{in} \\ f\left(y_j^i(t)\right) & \text{sinon} \end{cases} \tag{4.1}$$

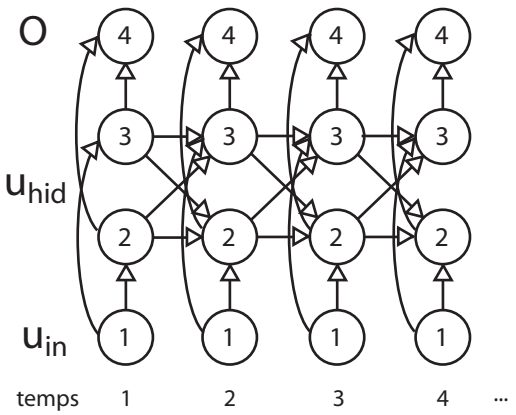

Fig. 4.2. Réseau de neurones récurrents de la figure 4.1 déplié dans le temps . Pour ce graphe, on doit avoir avoir une matrice de neurone de dimension t par n. Où t représente le nombre d'étapes sur lequel on calcule le gradient et n le nombre de neurones dans le graphe. À chaque temps, on ajoute tous les arcs du réseau en respectant les délais. Ainsi, on obtient un réseau de neurones multi-couche où l'on peut calculer le gradient.

où

$$y_j^i(t) = \sum_{(u_s,u_j,d_{sj},w_{sj})\in L} w_{sj} x_s^i(t - d_{sj}) \tag{4.2}$$

et $f\,(x)$ est la fonction d'activation. Dans nos expériences, nous avons choisi la sigmoïde.

$$f\,(x) = \frac{1}{1+e^{-x}} \tag{4.3}$$

Notre calcul d'erreur sera défini comme ceci

$$E_{total} = \sum_{d=1}^{N} \sum_{t=1}^{n_d} \sum_{u_j \in O} e_{ct}(d,t,j) \text{ où } ct \in \{mse, crent\} \tag{4.4}$$

Nous minimisons soit l'entropie croisée ($e_{crent}(d,t,j)$), soit la différence de l'erreur au carré ($e_{mse}(d,t,j)$). Nous aurons alors les fonctions d'erreur suivantes :

$$e_{mse}(d,t,j) = \left(x_j^d(t) - T_j^d(t)\right)^2 \tag{4.5}$$

$$e_{crent}(d,t,j) = -T_j^d(t) \ln\left(\frac{a_j^d(t)}{T_j^d(t)}\right) \tag{4.6}$$

où

$$a_j^d(t) = \frac{\exp(x_j^d(t))}{\sum_{u_k \in O} \exp(x_k^d(t))} \tag{4.7}$$

4.1.3. Calcul du gradient

Nous devons calculer le gradient de l'erreur totale par rapport aux poids des arcs du réseau. Nous utiliserons le principe de la propagation du gradient à travers le réseau déplié.

Définissons quelques termes :

$$f'(x) = \frac{\partial f(x)}{\partial x} \tag{4.8}$$

Le gradient de l'erreur totale par rapport aux neurones à chaque temps est :

$$g_j^d(t) = \frac{\partial e_{ct}(d,t,j)}{\partial x_j^d(t)} = \begin{cases} 0 & \text{si } t > n_d \\ x_j^d(t) - T_j^d(t) & \text{si } u_j \in O \\ f'\left(y_j^d(t)\right)\left(\sum_{(u_i,u_j,d_{sj},w_{sj}) \in L} w_{sj} g_j^d(t + d_{sj})\right) & \text{sinon} \end{cases}$$
$$\text{où } ct \in \{mse, crent\} \tag{4.9}$$

Nous avons simplifié l'équation 4.9 en supposant que la fonction d'activation pour les neurones de sortie est linéaire pour le cas de moyenne de l'erreur au carré et la sigmoïde logistique pour le cas de l'entropie croisée. Bishop [1996] justifie ce résultat.

Le gradient de l'erreur totale sur chacun des poids du réseau est donc :

$$\frac{\partial E_{total}}{\partial w_{sj}} = \sum_{d=1}^{N} \sum_{t=1}^{n_d} g_j^d(t) x_s^d(t - d_{sj}) \tag{4.10}$$

4.1.4. Implantation

Les algorithmes 4.1 (La passe avant) et 4.2 (La passe arrière) présentent une implantation possible d'un réseau réseau récurrent et de l'apprentissage à l'aide de BPTT.

Algorithme 4.1 Algorithme de la passe avant pour un réseau récurrent standard. E représente l'erreur totale à travers toute les séquences.

$Z \leftarrow$ tri topologique du graphe seulement avec les arcs de délai 0
neuron $\leftarrow 0$
$E \leftarrow 0$
pour $d = 1$ jusqu'à N {Chaque séquence} faire
  pour $t = 1$ jusquà n_d {Chaque temps dans la séquence} faire
    pour tout $u_j \in Z$ dans l'ordre topologique faire
      si $u_j \in u_{in}$ alors
        neuron[d][t][j] $\leftarrow I_j^d(t)$
      pour tout $(u_s, u_j, d_{sj}, w_{sj}) \in L$ faire
        si $t - d_{sj} > 0$ alors
          neuron[d][t][j] $\leftarrow$ neuron[d][t][j] + w_{sj} * neuron[d][$t - d_{sj}$][s]
      neuron[d][t][j] $\leftarrow f$ (neuron[d][t][j])
      si $u_j \in O$ alors
        $E \leftarrow E +$ erreur entre neuron[d][t][j] et $T_j^d(t)$

Algorithme 4.2 Algorithme de la passe arrière pour un réseau récurrent standard. Il faut abolument exécuter l'algorithme de la passe avant (4.1) pour obtenir le tableau neuron. Le gradient pour chacun des poids sera dans W.

$Z \leftarrow$ tri topologique du graphe seulement avec les arcs de délai 0
$W \leftarrow 0$
neugrad $\leftarrow 0$
pour $s = 1$ jusqu'à N {Chaque séquence} faire
  pour $t = n_s$ jusqu'à 1 {Chaque temps dans la séquence en ordre inverse} faire
    pour tout $u_j \in Z$ dans l'ordre topologique inverse faire
      si $u_j \in O$ alors
        neugrad[s][t][j] = gradient de l'erreur entre neuron[s][t][j] et $T_j^s(t)$
      pour tout $(u_j, u_d, d_{jd}, w_{jd}) \in L$ faire
        si $t + d_{jd} < n_s$ alors
          neugrad[s][t][j] $\leftarrow$ neugrad[s][t][j] + w_{jd} * neugrad[s][$t + d_{jd}$][j]
          W[s][j] $\leftarrow$ W[s][j] + neuron[s][t][j] * neugrad[s][$t + d_{jd}$][j]
      neugrad[d][t][j] $\leftarrow f'$ (neugrad[d][t][j])
pour tout $w_{sd} \in$ Ensemble de tous les arcs faire
  {Mise à jour des poids}
  $w_{sd} \leftarrow w_{sd} - \alpha *$ W[s][j]

4.2. Long Short Term Memory

Le modèle de réseau de neurone Long Short Term Memory (LSTM) est un autre réseau de neurone récurrent un peu plus complexe. Il a la particularité qu'il peut apprendre des évènements à des temps éloignés et précis. Un réseau de neurones récurrents

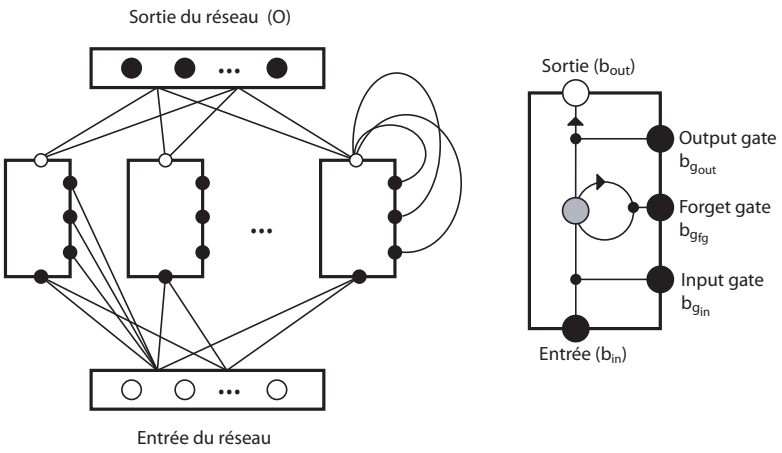

Fig. 4.3. Ceci est un réseau LSTM. Les arcs ne sont pas tous inscrits, il y a un arc entre chaque point blanc et noir. Dans un bloc LSTM, à gauche, les portes multiplient les valeurs. Ce n'est pas seulement une somme. Il peut y avoir plusieurs cellules qui partagent les mêmes portes. Ces portes servent à limiter la valeur de chacune des cellules. Il n'y a pas de fonction d'activation qui permet de limiter la valeur des cellules.

standard a beaucoup de peine à apprendre ce genre de tâche. Ce modèle a été présenté par Hochreiter and Schmidhuber [1997]. Gers [2001] dans sa thèse de doctorat a ajouté le concept des forget gate et peephole connection. Nous avons ajouté des neurones cachées parallèle aux blocs LSTM. Ceci aide à la génération de musique.

Pour accomplir ce genre de tâche, le modèle a des neurones qui ne sont pas contraints à un intervalle de valeur en appliquant une "squashing function". En plus, il y a le concept des portes (gates) qui permet au réseau d'apprendre à cacher l'état de certains neurones à d'autre neurones.

La figure 4.3 représente un réseau LSTM où les points noirs indiquent les noeuds du graphe qui recoivent de l'information. Les points blancs sont des noeuds qui produisent de l'information. Tous les arcs sortant de l'entrée du réseau et entrant dans la sortie du réseau n'ont pas de délai dans le temps. L'information est propagée dans une seule passe avant. Tous les autres arcs ont un délai un pas dans le temps. Par exemple, la sortie d'un bloc LSTM affectera la valeur des portes des blocs LSTM un pas plus tard dans le temps.

Chaque porte a une fonctionnalité particulière :

- Les portes d'entrée (input gate) permettent de montrer ou de cacher les valeurs d'entrée aux cellules à l'intérieur du bloc.
- Les portes d'oubli (forget gate) permettent de montrer ou de cacher la valeur des cellules du bloc.
- Les portes de sortie (output gate) permettent de montrer ou de cacher la valeur des cellules à la sortie du bloc.

Les portes ont aussi beaucoup d'importance dans la passe arrière. Ils permettent de contrôler le flux du gradient. Si les portes sont fermées le gradient ne sera pas pris en compte dans les étapes précédentes.

Les sections qui suivent expliquent le calcul de la passe avant et du gradient pour l'apprentissage des poids du réseau. Nous supposons un réseau LSTM complètement connecté avec N_b blocs où chaque bloc b a N_c^b cellules. Nous utiliserons la notation suivante pour représenter les différentes composantes du réseau LSTM.

- $B^i_{in_j}(t)$: La valeur de l'entrée de la cellule j du bloc i au temps t
- $B^i_{g_{in}}(t)$: La valeur de la porte d'entrée (input gate) du bloc i au temps t
- $B^i_{g_{fg}}(t)$: La valeur de la porte d'oubli (forget gate) du bloc i au temps t
- $B^i_{g_{out}}(t)$: La valeur de la porte de sortie (output gate) du bloc i au temps t
- $B^i_{out_j}(t)$: La valeur de la sortie de la cellule j du bloc i au temps t
- $B^i_{cell_j}(t)$: La valeur de la cellule j du bloc i au temps t

4.2.1. Passe avant

Dans la passe avant, nous procédons en trois étapes :

- Calcul des valeurs d'entrée de chacun des blocs soit les valeurs d'entrée et les valeurs des trois portes de chaque cellule pour chaque bloc
- Calcul des valeurs des cellules à l'intérieur de chaque bloc et calcul des sorties des blocs
- Calcul de la couche de sortie et du coût que l'on veut minimiser.

Afin d'alléger la notation dans les calculs de ce modèle, nous n'allons traiter qu'une séquence. Pour effectuer les calculs avec plusieurs séquences, il faut passer à travers les séquences en coupant les connexions récurrentes lorsque l'on change de séquences.

Dans les équations, nous utilisons les lettres grecques pour les sorties et les lettres romaines pour les activations. Les activations sont les sorties après l'application d'une "squashing function".

Voici le détail des calculs pour la première étape $\forall b \in \{1, .., N_b\}$:

$$\begin{aligned} \beta^b_{in_c}(t) &= \sum_{i=1}^{m_{\text{in}}} w_{in_i, B^b_c} I_i(t) + \sum_{i=1}^{N_b} \sum_{j=1}^{N^i_c} w_{B^i_j, B^b_c} B^i_{out_j}(t-1) \\ & \quad \forall c \in \{1, ..., N^b_c\} \end{aligned} \tag{4.11}$$

$$\beta^b_{g_{in}}(t) = \sum_{i=1}^{m_{\text{in}}} w_{in_i, B^b_{g_{in}}} I_i(t) + \sum_{i=1}^{N_b} \sum_{j=1}^{N^i_c} w_{B^i_{out_j}, B^b_{g_{in}}} B^i_{out_j}(t-1) \tag{4.12}$$

$$\beta^b_{g_{fg}}(t) = \sum_{i=1}^{m_{\text{in}}} w_{in_i, B^b_{g_{fg}}} I_i(t) + \sum_{i=1}^{N_b} \sum_{j=1}^{N^i_c} w_{B^i_{out_j}, B^b_{g_{fg}}} B^i_{out_j}(t-1) \tag{4.13}$$

$$\beta^b_{g_{out}}(t) = \sum_{i=1}^{m_{\text{in}}} w_{in_i, B^b_{g_{out}}} I_i(t) + \sum_{i=1}^{N_b} \sum_{j=1}^{N^i_c} w_{B^i_{out_j}, B^b_{g_{out}}} B^i_{out_j}(t-1) \tag{4.14}$$

$$B^b_a(t) = f\left(\beta^b_a(t)\right) \forall a \in \{in_1, ..., in_{N^b_c}, g_{in}, g_{fg}, g_{out}\} \tag{4.15}$$

Maintenant que les valeurs d'entrée pour chacun des blocs sont calculées, nous pouvons calculer les valeurs des cellules à l'intérieur des blocs ainsi que les valeurs de sortie des blocs.

$$\begin{aligned} B^b_{cell_c}(t) &= B^b_{in_c}(t) B^b_{g_{in}}(t) + B^b_{cell_c}(t-1) B^b_{g_{fg}}(t) \\ & \quad \forall b \in \{1, .., N_b\}, \forall c \in \{1, ..., N^b_c\} \end{aligned} \tag{4.16}$$

$$\beta^b_{out_c}(t) = B^b_{cell_c}(t) B^b_{g_{out}}(t) \tag{4.17}$$

$$B^b_{out_c}(t) = f\left(\beta^b_{out_c}(t)\right) \tag{4.18}$$

Dans l'équation 4.18, la "squashing function" est optionnelle car la porte de sortie agit déjà comme atténuateur. La valeur de sortie du bloc ne pourra pas prendre des valeurs déraisonnables.

Dans les équations précédentes, nous pouvons prendre conscience de la particularité importante du modèle LSTM. La valeur d'une cellule n'est pas contrainte par une fonction d'activation. Ainsi, la cellule peut prendre n'importe quelle valeur et le gradient pourra facilement passer à travers plusieurs étapes dans le temps. Nous verrons dans le calcul du gradient que le gradient sera coupé proportionnellement à l'ouverture de la porte d'oubli (forget gate). Cette porte permettra le contrôle de la valeur de la cellule. LSTM devra apprendre à fermer et à ouvrir la porte pour s'assurer que la cellule ne prenne pas des valeurs déraisonnables.

Dans la dernière étape de la passe avant, il faut calculer la valeur du vecteur de la couche de sortie($O_n(t)$ pour $1 \leq n \leq m_t$) et le coût ($e_n^{ct}(t)$ pour $1 \leq n \leq m_t$) que l'on veut minimiser à chaque temps t. La couche de sortie a des connections directes avec la couche d'entrée alors il faut inclure une sommation sur les unités d'entrée dans le calcul.

$$\Omega_n(t) = \sum_{i=1}^{m_{\text{in}}} w_{in_i,out_n} I_i(t) + \sum_{i=1}^{N_b} \sum_{j=1}^{N_c^i} w_{B_j^i,out_n} B_{out_j}^i(t) \tag{4.19}$$

$$O_n(t) = f\left(\Omega_n(t)\right) \tag{4.20}$$

Nous voulons minimiser la valeur de E_{total} qui est défini par

$$E_{total} = \sum_{t=1}^{T} \sum_{n=1}^{m_t} e_n^{ct}(t) \text{où } ct \in \{mse, crent\} \tag{4.21}$$

Nous utilisons encore deux fonctions de coût soit la minimisation de l'entropie croisée ($e_n^{crent}(t)$) et la minimisation de la différence au carré ($e_n^{mse}(t)$).

$$e_n^{mse}(t) = \left(O_n(t) - T_n(t)\right)^2 \tag{4.22}$$

$$e_n^{crent}(t) = -T_n(t) \ln\left(\frac{a_n(t)}{T_n(t)}\right) \tag{4.23}$$

où

$$a_n(t) = \frac{\exp(O_n(t))}{\sum_{i=1}^{m_t} \exp(O_i(t))} \tag{4.24}$$

4.2.2. Calculs des dérivées partielles

Pour calculer le gradient sur les poids du réseau LSTM, nous devons calculer des dérivées partielles. Cette stratégie consiste à calculer la dérivée de la valeur des cellules $B_{cell}()$ par rapport aux poids du réseau. Nous coupons le gradient entre les blocs, ainsi nous pouvons calculer le gradient sans tout dérouler le réseau. Nous sauvons beaucoup de mémoire étant donné le grand nombre de poids dans un réseau LSTM. Il y a trois groupes de dérivées partielles qu'il faut calculer :

$$\frac{\partial B_{cell_c}^b(t)}{\partial w_{m,B_c^b}}, \frac{\partial B_{cell_c}^b(t)}{\partial w_{m,B_{g_{in}}^b}}, \frac{\partial B_{cell_c}^b(t)}{\partial w_{m,B_{g_{fg}}^b}}$$

Pour réduire le nombre d'équations, nous devons définir $v^m(t)$ qui indique la valeur du neurone source de l'arc w_{m,B_c^b} à l'instant t. $v^m(t)$ se définit mathématiquement comme

suit :

$$v^m(t) = \begin{cases} I_i(t) & \text{si } m = in_i \\ B^i_{out_j}(t-1) & \text{si } m = B^i_{out_j} \end{cases} \tag{4.25}$$

Nous obtenons les solutions suivantes à ces groupes de dérivées partielles à partir l'équation 4.16 : $\forall b \in \{1, .., N_b\}$, $\forall c \in \{1, ..., N^b_c\}$ et $\forall m \in \{in_1, ..., in_{m_{\text{in}}}, B^{i=1..N_b}_{out_{j=1..N^i_c}}\}$

$$\frac{\partial B^b_{cell_c}(t)}{\partial w_{m,B^b_c}} = \frac{\partial B^b_{cell_c}(t-1)}{\partial w_{m,B^b_c}} B^b_{g_{fg}}(t) + f'\left(\beta^b_{in_c}(t)\right) B^b_{g_{in}}(t) v^m(t) \tag{4.26}$$

$$\frac{\partial B^b_{cell_c}(t)}{\partial w_{m,B^b_{g_{in}}}} = \frac{\partial B^b_{cell_c}(t-1)}{\partial w_{m,B^b_{g_{in}}}} B^b_{g_{fg}}(t) + B^b_{in_c}(t) f'\left(\beta^b_{g_{in}}(t)\right) v^m(t) \tag{4.27}$$

$$\frac{\partial B^b_{cell_c}(t)}{\partial w_{m,B^b_{g_{fg}}}} = \frac{\partial B^b_{cell_c}(t-1)}{\partial w_{m,B^b_{g_{fg}}}} B^b_{g_{fg}}(t) + B^b_{cell_c}(t-1) f'\left(\beta^b_{g_{fg}}(t)\right) v^m(t) \tag{4.28}$$

Nous remarquons que l'on peut calculer ces dérivées partielles en même temps que les valeurs de la couche de sortie.

4.2.3. Passe arrière

Ceci est la dernière étape avant la mise à jour des poids. Une grande partie est déjà faite dans le calcul des dérivées partielles, mais il nous manque le gradient du coût par rapport à chacune des cellules. Ainsi, nous pourrons appliquer le principe de la dérivée en chaîne pour obtenir le gradient du coût par rapport aux poids. La passe arrière est seulement appliquée si des cibles sont disponibles dans l'ensemble d'entraînement.

En premier lieu calculons le gradient du coût total par rapport aux neurones de sortie $\Omega_i(t)$.

$$\frac{\partial E_{total}}{\partial \Omega_i(t)} = f'\left(\Omega_i(t)\right)\left(O_i(t) - T_i(t)\right) \tag{4.29}$$

Il est à noter que cette équation est valide pour la minimisation de l'entropie croisée et la minimisation de la différence au carré pour les même raisons que dans le cas du réseau de neurones récurrents standard (Voir la section 4.1.3) Pour calculer le gradient sur les portes de sortie et les cellules à l'intérieur des blocs, on propage le gradient à travers le réseau comme dans BPTT. En dérivant l'équation 4.17 par rapport à $\Omega_i(t)$, et

en coupant le gradient entre les blocs LSTM, nous obtenons :

$$\frac{\partial E_{total}}{\partial \beta^i_{g_{out}}(t)} = f'\left(\beta^i_{g_{out}}(t)\right)\left(\sum_{j=1}^{N^i_c} B^i_{cell_j}(t)\sum_{k=1}^{m_t} w_{B^i_j,out_k}\frac{\partial E_{total}}{\partial \Omega_k(t)}\right) \tag{4.30}$$

Nous pouvons calculer aussi le gradient du coût total par rapport aux valeurs des cellules des blocs. En dérivant l'équation 4.17 par rapport à $B^i_{cell_j}(t)$ et toujours en coupant le gradient entre les blocs LSTM, nous obtenons :

$$\frac{\partial E_{total}}{\partial B^i_{cell_j}(t)} = B^i_{g_{out}}(t)\left(\sum_{k=1}^{m_t} w_{B^i_j,out_k}\frac{\partial E_{total}}{\partial \Omega_k(t)}\right) \tag{4.31}$$

Le fait qu'on coupe le gradient entre les blocs LSTM, nous permet de calculer le gradient sans tout déplier le réseau dans le temps. Si on avait à le faire, nous aurions besoin d'une énorme quantité de mémoire et nous serions incapable de faire des expériences avec de gros réseaux et de très longues séquences. De cette façon, il nous suffit de garder l'information de l'étape courante et de l'étape précédente.

4.2.4. Gradients sur les poids du réseau LSTM

La dernière étape consiste à calculer la gradient de l'erreur totale sur chacun des poids du réseau($\frac{\partial E_{total}}{\partial w_{a,b}}$). Nous allons utiliser à nouveau la notation $v^m(t)$ définie dans l'équation 4.25

Pour les poids de la couche de sortie, le calcul est le même que pour un réseau de neurones standard :

$$\frac{\partial E_{total}}{\partial w_{m,out_i}} = \sum_t v^m(t)\frac{\partial E_{total}}{\partial \Omega_i(t)} \tag{4.32}$$

Nous procédons encore avec le même principe pour les poids qui entre dans les portes de sorties des blocs :

$$\frac{\partial E_{total}}{\partial w_{m,B^i_{g_{out}}}} = \sum_t v^m(t)\frac{\partial E_{total}}{\partial \beta^j_{g_{out}}(t)} \tag{4.33}$$

Pour calculer les trois autres groupes de poids, ceux qui entrent dans les portes d'oubli, portes d'entrée et dans l'entrée des blocs, il faut utiliser les dérivées partielles calculées auparavant. En utilisant, la règle de dérivée en chaîne avec l'équation 4.31 et les équations des dérivées partielles 4.26, 4.28 et 4.27, nous obtenons les gradients

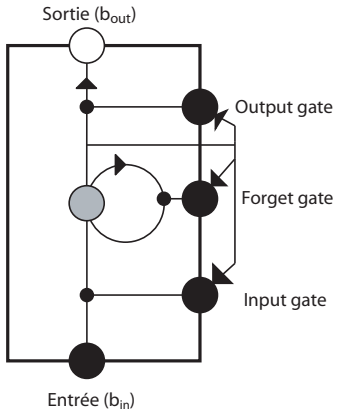

Fig. 4.4. Un bloc LSTM avec des peephole connections. La somme des valeurs des cellules pondérée par les poids des arcs est ajouté à la valeur des portes des blocs. Il n'y a pas de connexion entre les différents blocs.

suivants

$$\frac{\partial E_{total}}{\partial w_{m,B^i_{g_{fg}}}} = \sum_t \sum_{j=1}^{N_c^i} \frac{\partial E_{total}}{\partial B^i_{cell_j}(t)} \frac{\partial B^i_{cell_j}(t)}{\partial w_{m,B^i_{g_{fg}}}} \tag{4.34}$$

$$\frac{\partial E_{total}}{\partial w_{m,B^i_{g_{in}}}} = \sum_t \sum_{j=1}^{N_c^i} \frac{\partial E_{total}}{\partial B^i_{cell_j}(t)} \frac{\partial B^i_{cell_j}(t)}{\partial w_{m,B^i_{g_{in}}}} \tag{4.35}$$

$$\frac{\partial E_{total}}{\partial w_{m,B^i_j}} = \sum_t \frac{\partial E_{total}}{\partial B^i_{cell_j}(t)} \frac{\partial B^i_{cell_j}(t)}{\partial w_{m,B^i_j}} \tag{4.36}$$

4.2.5. Peephole connection

Dans le modèle LSTM, les portes ne sont influencées que par la sortie des blocs LSTM et la couche d'entrée. Si les portes de sortie sont fermées, il ne reste plus que la couche d'entrée pour influencer les portes. Les valeurs des cellules n'ont plus aucune influence. Pour remédier à cette situation, Gers [2001] utilise des peephole connections. Ces peephole connections (Figure 4.4) permettent une connexion directe entre les cellules d'un bloc et leurs portes. Ainsi, même les portes fermées, les cellules pourront agir et faire rouvrir les portes.

Il faut changer les trois équations des portes (4.12,4.13,4.14) dans la passe avant pour ajouter ces connexions :

$$\begin{aligned}
\beta^b_{g_{in}}(t) &= \sum_{i=1}^{m_{\text{in}}} w_{in_i,B^b_{g_{in}}} I_i(t) + \sum_{i=1}^{N_b}\sum_{j=1}^{N^i_c} w_{B^i_{out_j},B^b_{g_{in}}} B^i_{out_j}(t-1) \\
&\quad + \sum_{j=1}^{N^i_c} w_{B^b_{cell_j},B^b_{g_{in}}} B^b_{cell_j}(t-1) \qquad (4.37)\\
\beta^b_{g_{fg}}(t) &= \sum_{i=1}^{m_{\text{in}}} w_{in_i,B^b_{g_{fg}}} I_i(t) + \sum_{i=1}^{N_b}\sum_{j=1}^{N^i_c} w_{B^i_{out_j},B^b_{g_{fg}}} B^i_{out_j}(t-1) \\
&\quad + \sum_{j=1}^{N^i_c} w_{B^b_{cell_j},B^b_{g_{fg}}} B^b_{cell_j}(t-1) \qquad (4.38)\\
\beta^b_{g_{out}}(t) &= \sum_{i=1}^{m_{\text{in}}} w_{in_i,B^b_{g_{out}}} I_i(t) + \sum_{i=1}^{N_b}\sum_{j=1}^{N^i_c} w_{B^i_{out_j},B^b_{g_{out}}} B^i_{out_j}(t-1) \\
&\quad + \sum_{j=1}^{N^i_c} w_{B^b_{cell_j},B^b_{g_{out}}} B^b_{cell_j}(t) \qquad (4.39)\\
&\quad B^b_g(t) = f\left(\beta^b_g(t)\right) \forall g \in \{g_{in}, g_{fg}, g_{out}\} \qquad (4.40)
\end{aligned}$$

Dans les équations 4.37, 4.38 et 4.39 le dernier terme correspond aux peephole connection.

Il faudra modifier notre algorithme pour la passe avant. Voici dans l'ordre les étapes à faire :

- Calculer les valeurs d'entrée, les portes d'entrée et les portes d'oubli de chacun des blocs.
- Calculer les valeurs de toutes les cellules des blocs.
- Calculer les portes de sortie.
- Calculer les valeurs de sorties de chacun des blocs.

Pour le gradient sur ces poids, il faudra calculer des dérivées partielles de plus pour les poids allant aux portes d'entrée (input) et d'oubli(forget). Les calculs sont les

mêmes, il faut changer la définition de $v^m(t)$:

$$v^m(t) = \begin{cases} I_i(t) & \text{si } m = in_i \\ B^i_{out_j}(t-1) & \text{si } m = B^i_{out_j} \\ B^i_{cell_j}(t-1) & \text{si } m = B^i_{cell_j} \end{cases} \tag{4.41}$$

Les équations 4.27 et 4.28 devront être calculées pour les $m \in \{B^b_{cell_{j=1..N^b_c}}\}$ en plus. Les équations des gradients sur les poids 4.34 et 4.35 pourront ainsi être calculé avec ces mêmes m.

Pour les poids des peephole connection qui se rendent à la porte de sortie. L'équation 4.33 doit être calculée avec $v^m(t)$ modifié pour que $v^m(t) = B^i_{cell_j}(t)$ lorsque $m = B^i_{cell_j}$. Ceci est dû au fait que la porte de sortie au temps t est influencée par la valeur des cellules du bloc au même temps t.

4.2.6. Couche cachée

Dans le modèle LSTM, il y a des connexions directes entre la couche d'entrée et la couche de sortie. Ainsi, l'information en entrée peut être directement utilisée pour calculer la sortie. Pour ajouter un peu de capacité lorsque les blocs LSTM ne trouvent pas de dépendance temporelle ou lorsque les portes sont fermées, nous avons ajouté une couche cachée parallèle aux blocs LSTM. Cette couche cachée n'a pas de connexions récurrentes. Elle a des effets positifs sur la composition automatique surtout lorsque la structure métrique est impliquée. Les améliorations de nos modèles avec cette couche cachée sont détaillés dans la section 7. Le modèle LSTM avec couche cachée est représenté à la figure 4.5

Les calculs pour la passe avant et le gradient de la couche cachée sont les mêmes que pour un réseau feed-forward. Supposons que nous avons N_{hid} neurones dans la couche cachée.

$$\begin{aligned} \eta_h(t) &= \sum_{i=1}^{m_{\text{in}}} w_{in_i,H_h} I_i(t) + \sum_{i=1}^{N_b} \sum_{j=1}^{N^i_c} w_{B^i_{out_j},H_h} B^i_{out_j}(t-1) \\ &\quad \forall h \in \{1, ..., N_{hid}\} \end{aligned} \tag{4.42}$$

$$H_h(t) = f\left(\eta_h(t)\right) \tag{4.43}$$

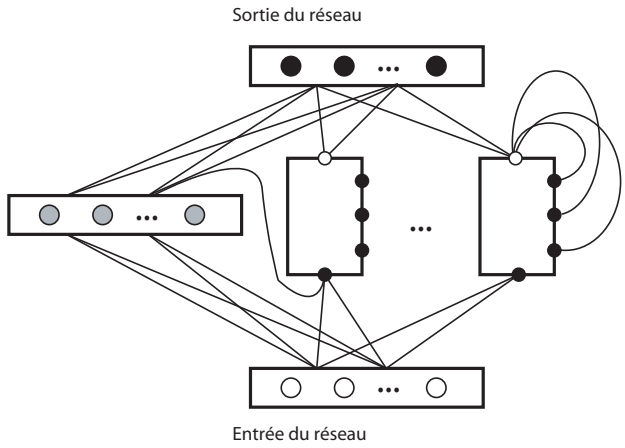

Fig. 4.5. Réseau LSTM avec couche cachée parallèle aux blocs LSTM. La couche cachée a aussi des connexions entre les sorties des blocs et l'entrée de la couche cachée. Elle permet d'apprendre des tâches un peu plus complexes directement entre la couche d'entrée et de sortie sans dépendance temporelle.

où $\eta_h(t)$ représente la valeur avant la fonction d'activation du neurone h de la couche cachée au temps t. $H_h(t)$ représente la valeur du neurone h au temps t après la fonction d'activation.

Il faut changer le calcul de la couche de sortie pour ajouter les valeurs de la couche cachée. L'équation 4.19 devient donc

$$\begin{aligned} \Omega_n(t) &= \sum_{i=1}^{m_{\text{in}}} w_{in_i,out_n} I_i(t) + \sum_{i=1}^{N_b} \sum_{j=1}^{N_c^i} w_{B_j^i,out_n} B_{out_j}^i(t) \\ &\quad + \sum_{i=1}^{N_{hid}} w_{H_i,out_n} H_i(t) \qquad (4.44) \\ O_n(t) &= f\left(\Omega_n(t)\right) \qquad (4.45) \end{aligned}$$

Pour calculer le gradient sur les poids de la couche cachée, on propage le gradient à travers les neurones cachées. Pour les poids provenant de la couche jusqu'à la couche cachée, on obtient directement

$$\frac{\partial E_{total}}{\partial w_{H_h,out_i}} = \sum_t H_h(t) \frac{\partial E_{total}}{\partial \Omega_i(t)} \qquad (4.46)$$

Pour calculer le gradient sur les poids de la couche d'entrée et la couche cachée, nous devons obtenir en premier lieu le gradient de l'erreur totale sur les neurones de la couche cachée.

$$\frac{\partial E_{total}}{\partial H_h(t)} = \sum_{i=1}^{m_t} w_{H_h,out_i} \frac{\partial E_{total}}{\partial \Omega_i(t)} \tag{4.47}$$

Ensuite, nous calculons le gradient sur le reste des poids de la couche cachée :

$$\frac{\partial E_{total}}{\partial w_{in_i,H_h}} = \sum_t I_i(t) \frac{\partial E_{total}}{\partial H_h(t)} \tag{4.48}$$

$$\frac{\partial E_{total}}{\partial w_{B^b_{out_c},H_h}} = \sum_t B^b_{out_c}(t) \frac{\partial E_{total}}{\partial H_h(t)} \tag{4.49}$$

4.2.7. Biais

Dans tous ces calculs, nous n'avons jamais tenu compte du biais sur les neurones ou les portes. Nous avons mis des biais dans nos modèles, mais les calculs sont les mêmes. Il faut ajouter un neurone fictif à l'entrée du réseau qui a toujours un valeur égale à 1. Le biais sera le poids de l'arc qui est attaché à ce neurone.

Les biais sur les portes joue un grand rôle car ils permettent de retarder leur action dans le processus d'entraînement. Ceci est expliqué dans la prochaine section.

4.2.8. Initialisation du réseau

Nous devons fixer des valeurs aux cellules des blocs LSTM et aux poids sur le réseau avant de commencer l'entraînement. Les cellules sont initialisées à zéro. Pour chacun des poids $w_{s,d}$ sauf ceux des biais des portes des blocs LSTM, on les fixe à une valeur aléatoire entre $-\sqrt{fanin_d}$ et $\sqrt{fanin_d}$ où $fanin_d$ est le nombre de poids qui entre dans le neurone d.

Nous initialisons les biais des portes des blocs de façon différente car nous voulons que chacun des blocs apprennent des dépendances de durée différente. Pour forcer les blocs à travailler différemment, nous leur affectons des biais différents. Pour les portes d'entrée, le premier bloc a un biais de 0.5, le deuxième 1.0, le troisième 1.5 et ainsi de suite. Pour les portes d'oubli, le premier bloc a un biais de -0.5, le deuxième -1.0, le troisième -1.5 et ainsi de suite. Les portes de sortie sont initialisées exactement comme ceux des portes d'entrée.

4.2.9. Mise à jour des poids

Dans les deux modèles, nous utilisons la même technique de mise à jour des poids. Nous utilisons la technique de la descente du gradient stochastique. Nous calculons le gradient des poids sur n exemples (appelé batch size en anglais). Avant de continuer dans les prochaines étapes dans les séquences, nous mettons les valeurs des poids à jour avec ce calcul.

$$w_{s,d} = w_{s,d} - \alpha \frac{\partial E_{total}}{\partial w_{s,d}} \qquad \text{où } \alpha \in]0, 1] \tag{4.50}$$

Nous répétons ceci plusieurs fois et nous choisissons l'ensemble des poids qui donne une erreur minimale sur un ensemble de validation, un ensemble de séquences qui n'ont pas été vues en entraînement. Ceci évite le surentraînement sur les exemples d'entraînement (problème fréquent des réseaux de neurones).

4.3. Conclusion

Dans cette section, nous avons vu deux sortes de réseaux de neurones récurrents : le standard où l'on calcule le gradient à l'aide BPTT, et LSTM où l'on calcule le gradient à l'aide de RTRL. Dans le premier cas, on peut calculer le gradient exact mais le modèle souffre du problème du gradient dissipé (vanishing gradient). Ceci nous empêche de trouver des corrélations entre des événements éloignés. C'est une des principales caractéristiques de la musique. Nous utilisons donc le modèle LSTM pour nos expériences car il souffre moins de ce problème.

Nous avons implanté les deux réseaux récurrents en environ 10000 lignes de code. Le temps d'apprentissage des modèles dépend de surtout de la dimension des vecteurs d'entrée et de sortie. Dans nos expériences avec les reels, le temps d'apprentissage est d'une heure pour les représentations sans structure métrique et de 4 heures pour les représentations avec structure métrique sur un PowerMac G5 2.5 Ghz. Par contre, la génération de musique prend moins de 10 secondes.

Chapitre 5

REPRÉSENTATION

Les réseaux de neurones ne peuvent comprendre une note directement. Nous devons fixer une représentation des notes à présenter au réseau de neurones.

Dans chaque représentation, nous utilisons le même principe pour pouvoir représenter n'importe quelle note. En premier lieu, nous choisissons un intervalle de note (n_{min} jusqu'à n_{max}). Nous définissons plus une formule permettant de prendre n'importe quel note et la modifier pour qu'elle soit dans l'intervalle et de sorte qu'elle conserve le même nom. Un Do dans le sixième octave d'un piano deviendra le Do le plus près qui est contenu dans l'intervalle n_{min} et n_{max}.

Si la note est comprise dans l'intervalle on n'y touche pas, sinon on la ramène à l'intérieur de l'intervalle en la modifiant par octave poru qu'elle conserve le même nom. Si nous avons une note n qui est l'entier équivalent à la notation MIDI et l'intervalle choisi est entre n_{min} et n_{max}, alors la nouvelle note sera n_{mod}.

$$n_{mod} = \begin{cases} n_{min} + (n - n_{min}) \bmod 12 & \text{si } n < n_{min} \\ n_{max} - (n_{max} - n) \bmod 12 & \text{si } n > n_{max} \\ n & \text{sinon} \end{cases} \tag{5.1}$$

Une fois la note dans l'intervalle, nous pourrons alors la transformer dans la représentation choisie.

Notre réseau de neurones va prédire des notes dans la représentation choisie, mais jamais il va reproduire une sortie équivalente à notre ensemble de notes possibles. Nous construisons alors une distribution de probabilité des notes possibles à l'aide de la méthode du softmax (Éq. 5.2) : Soit un ensemble de notes possibles $\{x_1, ..., x_n\}$, la probabilité $P(x_i)$ de la note x_i étant donnée la sortie du réseau de neurones est x_o est :

$$P(x_i) = \frac{e^{-\lambda||x_i - x_o||}}{\sum_{j=1}^{n} e^{-\lambda||x_j - x_o||}} \tag{5.2}$$

Note	Représentation
Do3	0 1 0 0 0 0 0 0 0 0 0 0 0 0
Mi3	0 0 0 1 0 0 0 0 0 0 0 0 0 0
Do4	0 0 0 0 0 0 0 0 0 0 0 0 0 1
Silence	1 0 0 0 0 0 0 0 0 0 0 0 0 0

Tab. 5.1. Représentation locale de notes de musique. Nous traitons le silence et toutes les notes comprises entre le Do3 et le Do4 inclusivement.

où nous avons choisit un $\lambda = 1$ et $||x_i - x_j||$ correspond à la distance entre x_i et x_j.

5.1. Représentation locale (Représentation one-hot)

Une représentation locale. consiste à avoir un vecteur aussi grand que le nombre de notes différentes à générer ou à traiter. Le vecteur doit avoir des zéros partout sauf pour un 1 correspondant à une seule note.

Étant donnée que nous voulons traiter le silence, notre vecteur sera de dimension $n+1$, où est n est le nombre de notes dans l'intervalle. Des exemples de cette représentation sont montrés dans la table 5.1

Nous remarquons que la distance euclidienne est la même entre toutes les notes. Ceci peut occasionner des problèmes car deux notes peuvent "sonner" plus proches que deux autres notes. La représentation de Shepard, expliquée dans la prochaine section, traite ce problème.

5.2. Shepard

Cette représentation (Shepard [1982]) est basée sur la distance psychologique des notes. Elle a la particularité que la distance euclidienne entre deux notes est relative à la distance que l'on perçoit psychologiquement entre deux notes. Il a utilisé les cercles chromatiques (Figure 2.2) et des quintes (Figure 2.3). Avec ces deux cercles (Figure 5.1), il obtient deux paires de coordonnées (x, y) qui correspondent à l'emplacement où se trouve une note sur chacun des cercles. Pour l'octave, il a une échelle linéaire en la note la plus haute et la plus basse traitées.

Mozer [1994] a utilisé cette représentation pour représenter n'importe quelle note dans un vecteur de dimension 13 séparé en 3 parties. Il n'utilise pas les coordonnées

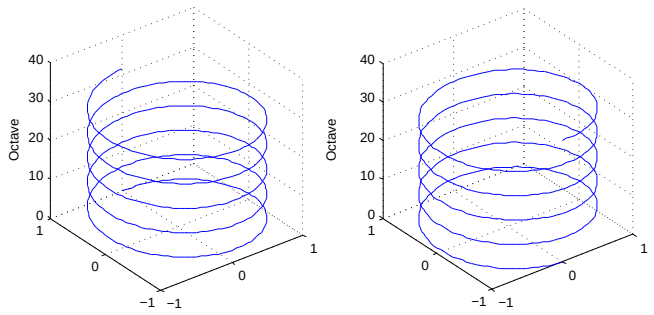

Fig. 5.1. Les spirales représentant la proximité des notes psychoacoustiquement selon Shepard. Une rotation de cercle représente un octave. Une spirale représente le cercle chromatique et l'autre représente le cercle des quintes. Deux notes séparées d'un octave sont près l'une de l'autre car elles sont une au-dessus de l'autre. Deux notes séparées d'une quinte seront près dans une spirale et loin dans l'autre, mais pas aussi loin que deux séparés d'un triton.

(x, y) car les réseaux de neurones n'ont pas de bons résultats avec ces paires de coordonnées. Il les encode en vecteur de taille 6. Les trois parties sont expliquées dans les trois prochains paragraphes.

La première partie, de dimension 1, représente l'octave de la note. C'est un nombre compris entre -9.798 pour le do le plus grave (C1) et +9.798 pour le do plus aigu (C5).

La deuxième partie, de dimension 6, représente le cercle chromatique des notes. Ce sont les 12 notes dans un octave dans l'ordre à partir du do. Dans la table 5.2, nous représentons les notes selon le cercle chromatique.

La dernière partie, aussi de dimension 6, représente le cercle des quintes. Ce sont les 12 notes dans un octave mais dans l'ordre des quintes. Deux notes qui ont une quinte de différence (mélodieux à entendre en même temps) auront une distance euclédienne petite. Dans la table 5.3, nous représentons les notes selon la partie du cercle des quintes :

La table 5.4 donne des exemples de représentation de notes avec la notation complète.

Dans nos expérimentations, nous avons fixé la hauteur des notes (le premier nombre du vecteur) entre 0 et 1 et fait débuter les cercles au La au lieu du Do. Ceci n'a aucune incidence sur les propriétés de la représentation. Cette représentation est utilisée en entrée dans les réseaux de neurones.

Note	Représentation
Do	-1 -1 -1 -1 -1 -1
Do♯	-1 -1 -1 -1 -1 +1
Ré	-1 -1 -1 -1 +1 +1
Ré♯	-1 -1 -1 +1 +1 +1
Mi	-1 -1 +1 +1 +1 +1
Fa	-1 +1 +1 +1 +1 +1
Fa♯	+1 +1 +1 +1 +1 +1
Sol	+1 +1 +1 +1 +1 -1
Sol♯	+1 +1 +1 +1 -1 -1
La	+1 +1 +1 -1 -1 -1
La♯	+1 +1 -1 -1 -1 -1
Si	+1 -1 -1 -1 -1 -1

Tab. 5.2. Représentation de Shepard. Ceci est seulement la partie qui correspond au cercle chromatique, ce qui implique que deux notes près dans le cercle chromatique on une distance euclidienne petite.

Note	Représentation
Do	-1 -1 -1 -1 -1 -1
Sol	-1 -1 -1 -1 -1 +1
Ré	-1 -1 -1 -1 +1 +1
La	-1 -1 -1 +1 +1 +1
Mi	-1 -1 +1 +1 +1 +1
Si	-1 +1 +1 +1 +1 +1
Fa♯	+1 +1 +1 +1 +1 +1
Do♯	+1 +1 +1 +1 +1 -1
Sol♯	+1 +1 +1 +1 -1 -1
Ré♯	+1 +1 +1 -1 -1 -1
La♯	+1 +1 -1 -1 -1 -1
Fa	+1 -1 -1 -1 -1 -1

Tab. 5.3. Partie de la représentation de Shepard qui prend compte du cercle des quintes. La distance euclidienne de deux notes séparées d'une quinte est petite sur cette partie.

5.3. Fréquentielle

Nous utilisons une autre représentation qui tient compte des harmoniques des notes.

Note	Représentation
Do1	-9,798 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
La4	9,573 +1 +1 +1 -1 -1 -1 +1 +1 +1 -1 -1 -1
Mi3	1,633 -1 -1 +1 +1 +1 +1 +1 +1 +1 +1 -1 -1

Tab. 5.4. Représentation de Shepard complète. Elle contient 13 dimensions et la distance euclidienne est relative à la distance psychoacoustique entre deux notes. La distance est dominée par la première dimension lorsque deux notes sont séparées par plusieurs octaves. Lorsqu'elle sont plus proches de façon chromatique, la distance sera dominée par les autres dimensions ce qui fait que les intervalles de quintes et chromatique prendront plus d'importance dans la distance. On peut le remarquer dans la figure 5.1 avec les deux spirales

Cette représentation a été utilisée par Paiement et al. [2005] pour faire de la génération de progression d'accords jazz. On veut une petite norme de la différence entre deux notes près l'une de l'autre psycho-acoustiquement.

C'est une représentation à 12 dimensions. Pour représenter une note n, nous calculons la série des notes $(h_1, h_2, ..., h_k)$ qui correspond aux harmoniques de n. Pour chaque h_i, nous donnons une valeur de ρ^i où $0 < \rho \leq 1$ et nous ramenons cette série de notes sur un octave en additionnant les valeurs. Une façon de calculer cette représentation est d'utiliser deux fonctions :

- $f(m)$ qui calcule la fréquence fondamentale d'une note étant donné le nombre équivalent MIDI m
- $m(f)$ qui calcule la nombre équivalent MIDI qui a comme fréquence fondamentale f

On définit les deux fonctions comme suit

$$f(m) = (8.1758)2^{\frac{m}{12}} \tag{5.3}$$

$$m(f) = 12(\log_2(f) - \log_2(8.1758)) \tag{5.4}$$

Par exemple pour le La qui a une fréquence fondamentale de 220Hz, ces harmoniques sont $F = (220, 440, 660, 880, 1100, 1320, ...)$. Nous transformons cette suite par la suite de notes qui ont cette fréquence comme fréquence fondamentale avec la fonction $m(f)$. Nous obtenons donc la suite $F_m = (57, 69, 76, 81, 85, 88, ...)$ et nous appliquons un modulo 12 pour ne pas tenir compte de l'octave pour obtenir les noms de notes $F_n = (\text{La}, \text{La}, \text{Mi}, \text{La}, \text{Do}\sharp, \text{Mi}, ...)$. Dans le vecteur final, de dimension 12,

on associe une dimension à une note n qui aura la somme des ρ^i où $F_m(i) = n$. Il faut se référer à Paiement et al. [2005] pour la suite des détails mathématiques de cette représentation.

5.4. Accords

Pour chaque représentation locale, fréquentielle et Shepard, on peut aussi représenter les accords.

Dans le cas de la représentation locale, on utilise un vecteur one-hot où le 1 représente un accord. On doit donc énumérer tous les accords dans la base de données et construire un vecteur aussi grand que le nombre d'accords différents.

Pour la représentation de Shepard, on construit un vecteur d'un accord en faisant la moyenne des vecteurs des notes qui constituent l'accord. La représentation d'un accord en Shepard est de la même grandeur qu'une note en Shepard.

Pour la représentation fréquentielle, nous ne prenons pas la moyenne comme dans Shepard, mais plutôt le maximum des dimensions des vecteurs qui constituent l'accord afin de tenir compte de l'effet de masque sur les fréquences (Moore [1982]).

5.5. Conclusion

Nous avons présenté trois représentations de notes différentes : locale, Shepard et fréquentielle. Les représentations de Shepard et fréquentielle essaient de modéliser la différence psycho-acoustique entre les notes. Ainsi, les modèles qui se trompent dans un choix de notes devrait avoir plus de chance de choisir une note plus près psycho-acoustiquement.

Dans la section 7.5, nous avons effectué des expériences avec chacune de ces représentations. Nous avons eu de bons résultats avec chacune des représentations. Avec la génération de mélodie, nous n'avons pas pu départager l'une de l'autre. La représentation locale a l'avantage que le modèle apprend un peu plus rapidement lorsque le nombre de notes différentes est petit. Si le nombre de notes devient grand, la dimension de la représentation locale devient plus grande ce qui n'est pas le cas avec les deux autres représentations.

Chapitre 6

COMPOSITION DE MÉLODIES

Pour faire la composition de mélodies à l'aide de notre modèle, nous suivrons les étapes suivantes :

- Pré-traitement de la base de données ;
- Apprentissage du modèle ;
- Génération de nouvelles chansons à l'aide du modèle entraîné ;
- Post-traitement des nouvelles chansons.

Dans cette section, nous discutons des bases de données utilisées et des étapes nécessaires à la composition de nos chansons.

6.1. Bases de données

Nous avons utilisé deux bases de données de musique en format MIDI : une ne contenant que des mélodies et l'autre avec des mélodies et accords. Nous ne voulions pas nous concentrer à apprendre le rythme, il nous fallait choisir un style où le rythme est quasi constant et où les mélodies sont définies surtout par les hauteurs des notes. Les reels irlandais ont un style qui s'approche de ces propriétés.

6.1.1. The session

La première base de données s'appelle "The session". Elle se trouve a www.thesession.org et contient environ 1700 reels irlandais en format MIDI. La plupart des chansons ne contiennent que des mélodies dans cette base de données.

Étant donné le grand nombre de chansons dans un même style, nous avons pu choisir seulement les reels en Do majeur(Cmaj) ce qui évite de transposer les chansons automatiquement. Aussi, nous permet d'économiser de la mémoire et du temps d'entraînement étant donnée que nos modèle charge en mémoire centrale toutes les données

d'entraînement. Ceci donne un ensemble de 46 chansons. Nous entraînons un modèle qu'avec des chansons dans la même tonalité.

6.1.2. Nottingham

La seconde base de données est "Nottingham". On peut la consulter à http://www.cs.nott.ac.uk/^ef/music/database.htm. Cette base de données contient 435 reels. Par contre, elles ne sont pas toutes dans la même tonalité alors nous prenons toutes les chansons en majeur et nous les transformons en Do majeur (Cmaj). Nous obtenons un ensemble de 92 chansons.

Cette base de données est intéressante car elle offre les mélodies ainsi que les accords d'accompagnement. Nos modèles pourront donc prédire la mélodie ou les accords en ayant les accords et la mélodie comme entrée. Par contre, nous tentons de générer des chansons avec des mélodies ainsi que des accords.

6.2. Pré-traitement

On doit faire un pré-traitement à nos chansons avant de pouvoir entraîner les modèles avec les chansons dans les bases de données. En premier lieu, on transpose toutes les chansons dans la même tonalité.

6.2.1. Sous-échantillonage

On doit sous-échantilloner les chansons pour obtenir une chanson où toutes les notes ont la même longueur car notre modèle ne peut gérer le rythme. Les étapes représentent une quantité de temps constante. Nous pourrons facilement encoder la structure métrique (Section 6.2.2). On choisit une fréquence d'échantillonnage et la note qui est jouée est affectée à chaque échantillon. S'il n'y pas de note de jouée, nous mettons un silence. La figure 6.1 montre un exemple d'une partition originale et de la partition sous-échantillonée.

Il est maintenant possible de construire des séquences de notes avec les représentations décrites dans le chapitre 5.

6.2.2. Encodage de la structure métrique

Pour encoder l'entrée du réseau de neurones afin de tenir compte de la structure métrique des chansons, nous allons ajouter des notes en entrée. Au lieu de présenter

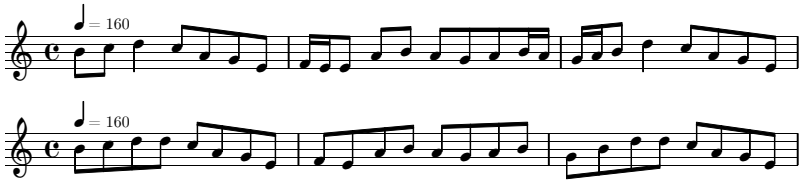

Fig. 6.1. La partition du haut est une section d'une chanson originale dans notre ensemble d'entraînement. La partition du bas est la nouvelle partition après notre sous-échantillonnage pour n'avoir que des notes de même durée.

la note au temps t au réseau pour prédire la note au temps $t + 1$, nous présentons non seulement la note au temps t, mais les autres notes aux temps $t - k_1$, $t - k_2$, $t - k_3$, etc... Les k_i doivent représenter des intervalles correspondant à la structure métrique.

Dans des chansons en 4/4, comme tous nos reels, les intervalles de notes seront 4, 8, 16, ainsi de suite. On peut remarquer ceci dans la figure 2.9. On obtient un k_i pour chacun des niveaux en comptant le nombre de croche qu'il y a entre chaque point sur un même niveau. Ce sera toujours le même nombre pour chacun des niveaux. La série des k_i est spécifique aux chiffres indicateurs (time signature) des mesures.

Nous avons justifié dans la section 3.2.2.1 que LSTM peut apprendre des événements avec des corrélations à long terme. Mais, en pratique, il peut être très long avant que LSTM trouve les intervalles de notes qui ont des corrélations fortes entre elles. Or, en musique, pour un style, tous ces intervalles sont fixes dès le départ de la chanson. LSTM est capable d'apprendre des grammaires du style $a^n b^n$, comme l'on montré Schmidhuber et al. [2002].

En fournissant au réseau déjà les intervalles correspondant à la structure métrique, LSTM n'aura pas besoin de chercher ces intervalles. Il pourra alors chercher d'autres corrélations entre les notes représentant le style de musique qui n'est pas spécifique à la structure métrique. Ces intervalles sont calculés auparavant à l'aide du modèle proposé par Eck and Casagrande [2005]. Ce modèle nous donne la hiérarchie de la structure métrique, nous permettant de retrouver ces intervalles. LSTM prédit mieux les prochaines notes à l'aide de la structure métrique avec de véritables séquences musicales. En plus, cette structure permet à LSTM de générer des mélodies ayant une structure globale.

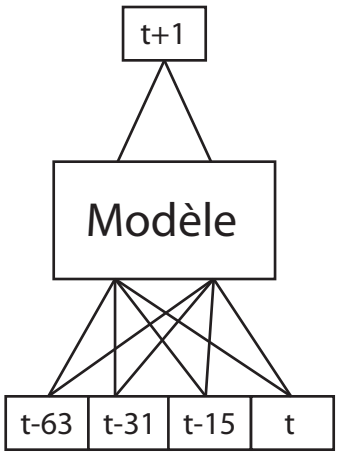

Fig. 6.2. Représentation de l'architecture pour représenter l'encodage de la structure métrique à notre réseau. Nous prédisons la note au temps $t+1$ avec les notes au temps t, $(t+1)-16$, $(t+1)-32$ et $(t+1)-64$.

6.3. Apprentissage du modèle

Nous entraînons tous nos modèles avec la technique du next-step prediction. Il s'agit de prédire la prochaine note dans la séquence. Nous prenons donc un sous-ensemble d'une base de données de musique et nous entraînons un modèle avec la technique de la descente de gradient. Dans la figure 6.2, nous montrons comment nous entraînons le réseau au temps t.

6.4. Génération de chansons

Une fois le modèle entraîné, nous avons une série de poids pour notre réseau de neurones. Nous pouvons générer des chansons à l'aide du réseau et de ces poids. Nous choisissons une chanson qui n'a pas été présentée au réseau mais dans le même style et la même tonalité que le réseau a été entraîné.

Nous commençons par présenter un début de chanson choisie au réseau pour le mettre dans un état où les cellules ne sont pas à zéro. Par la suite, nous prenons la sortie au temps t et nous choisissons au hasard une note n_t dans la distribution générée par le réseau (Voir équation 5.2). L'entrée du réseau au temps $t+1$ ne sera pas celle de la

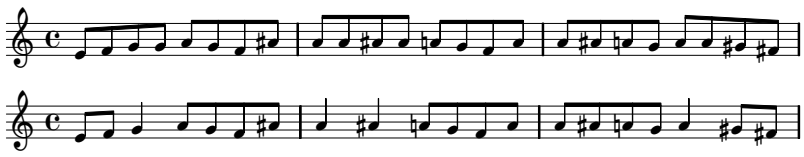

Fig. 6.3. La partition du haut est une chanson générée directement par le modèle. Il n'y a qu'une seule longueur de note. La partition du bas représente la partition du haut lorsque le post-traitement est fait. On fusionne les notes consécutives de même hauteur sans faire de notes liées entre les mesures.

chanson choisie mais plutôt la note n_t. Ainsi, nous pouvons générer des chansons aussi longues que nous voulons.

Après plusieurs expériences, nous avons remarqué que très rarement le modèle prédit une note avec une probabilité 1, en fait jamais. Ainsi, si le modèle génératif choisit au hasard une note de très faible probabilité, le modèle peut tomber dans un état très instable. Le modèle ne sera plus capable de revenir dans un état où les mélodies et/ou accords sont beaux. Pour éviter ce genre de problème, nous avons ajouté une petite heuristique. Nous calculons la distribution de probabilité et nous rejetons toutes les notes dont la probabilité est en-dessous de $\frac{1}{N}$ où N est le nombre de notes possibles. Ensuite, nous recalculons une distribution, avec softmax toujours (Voir équation 5.2), juste avec ces notes plus probables.

6.5. Post-traitement

Une fois notre chanson générée, nous effectuons un post-traitement. Étant donné que la chanson ne contient que des notes de même durée, nous fusionnons les notes consécutives ayant la même hauteur. Par contre, nous ne faisons pas de notes liées entre les mesures car, pour faire percevoir à l'auditeur que la chanson suit bien la structure métrique, un début de note sera toujours jouée aux début des mesures. La figure 6.3 illustre le post-traitement.

6.6. Conclusion

Nous avons décrit notre méthode de composition automatique de chansons. Dans la figure 6.4, on représente le processus automatique de composition. Nous avons choisi de composer à l'aide de la technique de prédire pas à pas (next-step prediction). Dans le

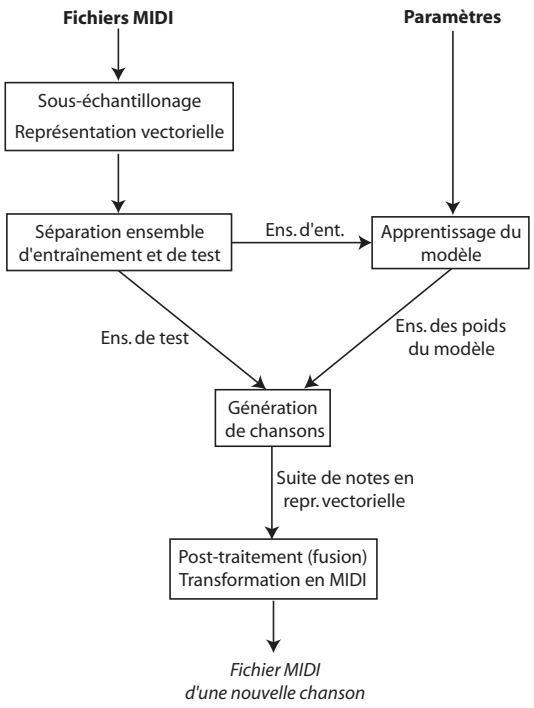

Fig. 6.4. Schéma de notre méthode pour générer des chansons d'un style donné à l'aide d'un ensemble de fichiers MIDI. Nous avons en entrée un ensemble de fichiers MIDI et nous finissons avec un nouveau fichier MIDI qui est dans le même style que ceux en entrée. L'utilisateur doit fournir ce qui est gras et la sortie du processus est en italique. Les paramètres sont, par exemple, le nombre de blocs LSTM.

prochain chapitre, nous utilisons cette méthode pour effectuer différentes expériences et valider l'utilité de la structure métrique.

Chapitre 7

RÉSULTATS

Nous avons effectué plusieurs séries d'expériences pour vérifier la capacité de LSTM à effectuer des tâches musicales. Nous avons aussi comparé avec un réseau de neurones feed-forward ou un réseau de neurones récurrents standard. Dans tous les cas, nous avons utilisé la technique du "next-step prediction" (Voir section 6.3).

Nous présentons quatre séries d'expérimentation, de la tâche la plus simple à la plus compliquée.

- Apprendre à jouer une note à un intervalle régulier parmi du bruit ;
- Apprendre des "Walking bass" de blues ;
- Apprendre des mélodies avec de vrais reels irlandais ;
- Apprendre des mélodies et accords avec de vrais reels irlandais .

7.1. Note dans du bruit

Cette expérience consistait à faire jouer à notre modèle une note à un intervalle régulier et, entre ces notes, à jouer toute autre note. Nous avons construit un ensemble de données de fichiers MIDI avec un Do à toutes les n croches. Toutes les autres notes sont des notes choisies dans le même octave sauf le Do. Nous avons fait apprendre cet ensemble à LSTM avec différents n.

Nous effectuons cette expérience pour trouver comment LSTM réagit avec des événements de plus en plus distants entourés de bruit. Avec ce résultat, nous avons pu construire une représentation avec la structure métrique plus juste.

La table 7.1 montre des expériences faites avec différentes représentations. Nous avons utilisé un réseau LSTM avec un bloc et une cellule à l'intérieur. Notre coût est la différence au carré entre les sorties et cibles. Nous utilisons une sigmoïde comme fonction d'activation. Elle varie entre 0 et 1 pour la couche de sortie et les portes des blocs LSTM. Elle varie entre -2.0 et 2.0 pour l'entrée des blocs LSTM. Nous entraînons

	Représentation		
Dist(n)	Locale	Shepard	Fréquentielle
8	0,036	0,106	0,038
10	0,034	0,109	0,040
12	0,036	0,111	0,039
16	*0,038	0,117	*0,044

Tab. 7.1. Expérience de génération d'une note à intervalle régulier. Nous utilisons le modèle LSTM avec un bloc et une cellule en tout temps. L'astérisque (*) indique que le modèle n'a pas réussi à générer un Do toutes les n croches. Dans les deux cas où LSTM n'a pas réussi, il faut noter qu'il générait des Do à la mauvaise fréquence mais quand même très proche. Avec un peu plus d'entraînement, LSTM aurait sans doute réussi.

pendant 600000 étapes où nous mettons à jour les poids toutes les 60 étapes avec un taux d'apprentissage de 0.05. Nous avons choisi de mettre à jour les poids à un nombre d'étapes plus grand que 1 pour s'assurer de faire des changements de poids plus grand et éviter de tomber dans les limites de précisions ou presque de l'ordinateur. Ainsi, nous ferions de trop petit saut et nous resterions toujours au même endroit.

Nous n'avons indiqué que des expériences avec LSTM car avec une réseau récurrent standard le temps d'apprentissage est excessivement élevé pour de mauvais résultats. Le temps d'apprentissage peut prendre jusqu'à 10 fois plus de temps pour obtenir le même résultat, si on l'obtient. Dans les cas où le n est plus grand, par exemple 12 ou 16, un réseau récurrent n'a pas réussi à l'apprendre.

Nous remarquons que LSTM est capable d'apprendre ce type d'expérience car il peut compter. Dans la figure 7.1, on peut voir le comportement de la cellule et des portes du bloc LSTM. Même si la valeur de la cellule dans un bloc LSTM n'est pas limité par une fonction d'activation, elle ne prend pas des valeurs démesurées pour saturer le réseau. Ce sont les portes qui limitent la valeur en les ouvrant et fermant. Dans ce genre de tâche, l'évolution de la valeur de la cellule prend une période aussi grande que la distance entre les notes fixes (soit le Do dans notre cas).

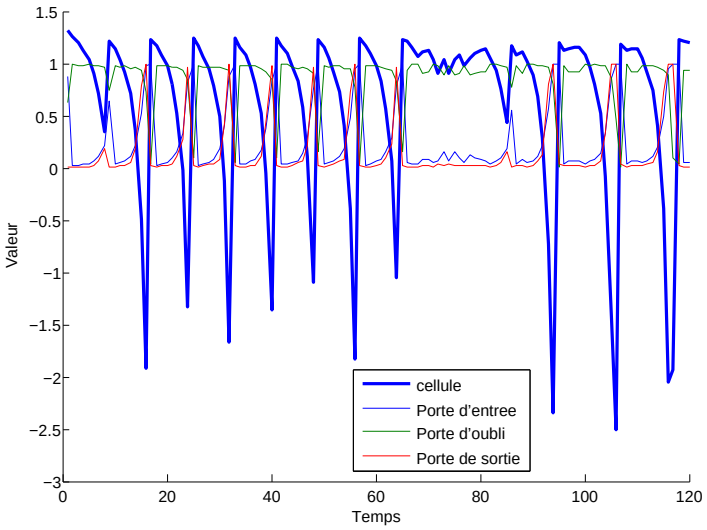

Fig. 7.1. Valeurs de la cellule (ligne bleue et épaisse) et des portes (lignes fines) d'un bloc LSTM lors de la génération avec un modèle entraîné à générer des do à toutes les 8 notes. On remarque une période de 8 temps entre les changements d'ouverture et fermeture de porte. On remarque une période de 8 notes entre un grand changement de valeur de la cellule. Ce graphe ne montre pas une génération parfaite d'un Do à toutes les 8 notes. On peut voir que LSTM peut ne pas fermer les portes pendant quelques périodes mais réussir plus tard.

7.2. Walking bass

Cette tâche consiste à faire apprendre par LSTM des walking bass de blues. On construit notre ensemble d'entraînement avec 6 suites de 8 notes, représentées à la figure 7.2, placées aléatoirement les unes après les autres.

Avec cette expérience, nous voulons savoir si LSTM peut apprendre de petites formes de notes qui surviennent aléatoirement. La première étape est qu'il trouve les limites entre chaque forme et ensuite les généralise pour en composer de nouvelles.

Nous avons utilisé les mêmes fonctions d'activation, coût à minimiser et étapes d'entraînement que dans les expériences de la section 7.1. Cette fois-ci, nous avons utilisé la représentation avec la structure métrique (Section 6.2.2). Pour générer les chansons, nous prenons une chanson qui n'a pas été vue ni en entraînement, ni en validation et nous présentons les 40 premières notes aux modèles. Par la suite, la n-ième note générée est la $(n+1)$-ième en entrée.

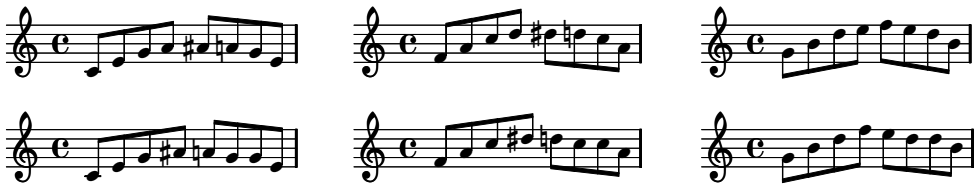

Fig. 7.2. Les 6 suites de notes utilisées pour générer l'ensemble d'entraînement pour la série d'expérience des walking bass.

On remarque que l'erreur de la validation est plus grande avec l'encodage de la structure métrique. Ceci est dû à la façon dont l'ensemble de données a été construit. Étant donné que les suites de 8 notes sont choisies au hasard les unes après les autres, il n'y a aucune corrélation entre les notes à des délais des 16, 32 et 64 croches. Le modèle a ainsi une entrée de 4 fois plus de dimensions à traiter et rend donc le modèle beaucoup plus difficile à entraîner.

Dans tous les cas, LSTM a trouvé que des formes à 8 croches se répétaient car les chansons générées ont des formes avec 8 croches. Cependant, les représentations fréquentielles et de Shepard n'ont pas réussi à bien mélanger les formes les unes après les autres, ils tombent souvent, ou toujours, dans une boucle répétant les même notes. Ceci est dû au fait que ces représentations sont plus complexes car les distances entre les notes psycho-acoustiquement proches sont plus petites. Ce n'est pas le cas dans la représentation locale. Dans la base de données des walking bass, les notes sont toutes proches, alors il est plus difficile pour un réseau de neurones de prédire exactement la bonne note. Il y a donc une plus grande probabilité de prédire une autre note psycho-acoustiquement proche et ainsi de débalancer le réseau LSTM. Un exemple de trois mesures générées par LSTM est montré à la figure 7.3.

LSTM a bien appris les passages entre les différentes formes. Dans les chansons générées, on identifie souvent des formes pareilles à celles trouvées dans l'ensemble de données. Il est aussi capable de générer de nouvelles formes, comme dans la figure 7.3, en changeant quelques notes. Ceci arrive fréquemment avec les représentation de Shepard et fréquentielle, mais LSTM tombe rapidement dans une boucle.

Représentation	Nb. de bloc 1 cell./bloc	Erreur de validation sans SM	avec SM
Locale	1	0,013	0,017
Locale	2	0,006	0,010
Shepard	1	0,047	0,058
Shepard	2	0,027	0,037
Fréquentielle	1	0,015	0,014
Fréquentielle	2	0,010	0,012

Tab. 7.2. Expérience de génération de walking bass de blues par LSTM. La colonne Err. correspond à l'erreur de validation. La colonne MIDI contient le nom du fichier MIDI que le modèle a généré. Ce fichier peut être obtenu sur le web à l'adresse suivante : www.iro.umontreal.ca/^lapalmej/memoire/. La figure 7.3 présente un exemple de trois mesures générées par la représentation Locale sans structure métrique.

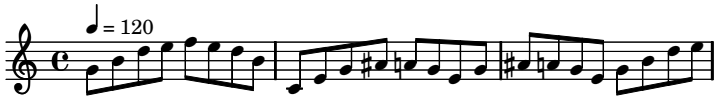

Fig. 7.3. Trois mesures de walking bass générées par un modèle LSTM. Les formes sont très semblables à celles de la figure 7.2.

7.3. Mélodies (reels)

Cette série d'expérimentation consiste à utiliser des fichiers MIDI d'une base de données de véritables reels irlandais. On utilise la base de données de the session (Section 6.1.1). À chaque entraînement de modèle, nous enlevons deux chansons au hasard : une pour calculer notre erreur de validation, l'autre pour faire la génération d'une nouvelle chanson.

Cette expérience permet de vérifier que LSTM peut gérer un vrai fichier MIDI sans modification manuelle. Toutes les étapes sont faites automatiquement et permettent de générer de nouvelles mélodies.

Nous avons essayé un réseau de neurones feed-forward standard avec la représentation de la structure métrique. Ceci nous donnera une référence. Ensuite, nous avons essayé plusieurs topologies de réseau LSTM.

Pour les réseaux de neurones feed-forward, nous avons entraîné pendant 50000 étapes et mis à jour les poids à chaque 50 étapes avec un taux d'apprentissage de $0,05$.

Nous minimisons la différence au carré entre les sorties et les cibles et utilisons la sigmoïde comme fonction d'activation.

Pour les réseaux LSTM, nous utilisons les même fonctions d'activation que dans les expériences précédentes. Nous entraînons pendant 1200000 étapes où nous mettons à jour les poids à chaque 60 étapes avec un taux d'apprentissage de $0,005$. Nous minimisons toujours la différence au carré entre les sorties et les cibles. Dans la table 7.3, nous avons mis les erreurs de validation du modèle LSTM avec 2 blocs et 2 cellules à l'intérieur de chaque bloc. Nous n'en avons pas mis d'autre car les autres topologies avec plus de capacité avaient pratiquement la même erreur en validation.

Nous avons fait des expériences avec la représentation locale et Shepard. On ne peut pas comparer les deux représentations avec l'erreur de validation. Ceci est dû au fait qu'avec la représentation local, le réseau peut facilement mettre à zéro une sortie lorsqu'une note n'est jamais vu ou presque jamais et ainsi faire baisser la différence au carré. Avec la représentation de Shepard, le réseau ne pourra se comporter ainsi pour faire baisser la différence au carré. On peut cependant les comparer avec la génération des chansons. Les mélodies générées avec la représentation locale sont plus souvent variées que la représentation de Shepard. La représentation Shepard tend à générer soit une boucle de notes ou une seule note.

La couche cachée en parallèle ne reflète pas une amélioration en erreur de validation dans tous les cas, mais les mélodies tendent à avoir un contour mélodique plus intéressant. Il y a un lissage qui se fait entre la note courante et la note prédite.

On remarque que LSTM est plus efficace à prédire la prochaine note en moyenne. Dans tous les cas, avec une même représentation, LSTM a une plus petite erreur en validation qu'un réseau feed-forward avec une structure métrique. Si on met la structure métrique avec LSTM, on obtient encore une erreur de validation plus petite.

Nous avons généré des mélodies avec ces modèles. Les meilleures mélodies sont celles avec la représentation locale avec structure métrique. On remarque que les mélodies générées avec le réseau feed-forward ont très peu de contour mélodique mais la structure globale est toujours retenue. Ceci est dû à la représentation de la structure métrique. Les mélodies générées avec LSTM peuvent être très bonnes, mais il y en avait de très mauvaises. LSTM peut tomber dans une boucle de quelques notes et les répéter indéfiniment ou, encore pire, toujours générer la même note. On explique ce ceci par le fait que, lorsque LSTM commence à générer ces entrées par lui-même, s'il génère une note à laquelle il n'est pas "habitué" de voir en entrée, il aura tendance à fermer

Modèle	Représentation	Err. de validation sans SM	avec SM
FF-16	Locale	—	0,075
FF-16	Shep.	—	0,259
LSTM-0	Locale	0,016	*0,013
LSTM-16	Locale	0,015	0,013
LSTM-0	Shep.	0,080	0,071
LSTM-16	Shep.	0,080	*0,055

Tab. 7.3. Expérience de génération de chansons à partir de reels de la base de données de the session. SM est indiqué dans la représentation lorsque nous utilisons la structure métrique en entrée comme représentation. Les modèles FF-n indiquent que ce sont des réseaux de neurones feed-forward à n unités cachées. Les modèles LSTM-n sont des réseaux LSTM avec 2 blocs qui contiennent 2 cellules chacunes et avec n unités cachées parallèles aux blocs. L'erreur de validation est la moyenne de 3 essais pour le modèle et la représentation. Les chansons générées avec un astérisque (*) sont disponibles à www.iro.umontreal.ca/^lapalmej/memoire/.

les portes et à tomber dans un état où il génère toujours la même note. En utilisant la technique de teacher forcing, on continurait à donner en entrée les notes de la chanson prise pour initialiser le réseau mais on jouerait les notes données en sortie. Cette technique permettrait de garder le réseau dans un état stable. Par contre, la chanson générée pourrait être la même que la chanson choisie pour initialiser le réseau. Le contour locale entre les notes pourrait ne pas être bon, car le réseau voudrait faire une série de note dans une direction, mais la chanson fait une autre série de note. La longueur de la chanson limiterait la longueur de la chanson générée.

7.4. Mélodies et accords (reels)

En plus de la mélodie, nous avons également fait apprendre les accords, à partir d'une autre base de données. Nous avons les même motivations à effectuer cette expérience que la précédente. En plus, nous voulons savoir comment réagit LSTM avec plus d'information, tel que les accords.

La représentation est semblable à celle que pour une note seule à laquelle un accord est associé. (Section 5.4). Pour la représentation avec la structure métrique, on aura

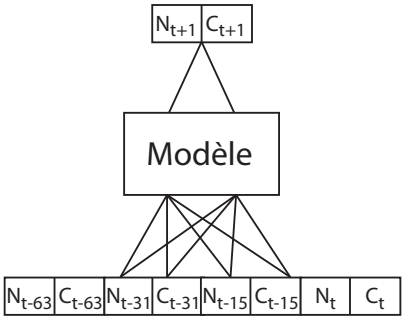

Fig. 7.4. Organisation des entrées et sorties de notre modèle pour gérer les accords (Voir la figure 6.2pour l'organisation sans les accords). N_t est la représentation de la note au temps t et C_t est la représentation de l'accord au temps t.

les notes à intervalles fixes ainsi que les accords aux mêmes intervalles. La figure 7.4 représente la nouvelle structure d'entrées et sorties de notre modèle.

Nous avons utilisé la base de données Nottingham (Section 6.1.2). Nous avons pris un ensemble de 22 chansons au hasard parmi les 92 chansons transposées. À chaque entraînement, nous prenons 20 chansons au hasard de ces 22 pour faire l'apprentissage de notre modèle. Les deux autres servent à calculer l'erreur de validation et à générer une nouvelle chanson en prenant les 64 premières notes d'une de ces chansons.

Nous avons effectué des expériences avec un réseau feed-forward pour avoir une référence d'erreur de validation. Nous avons utilisé les mêmes paramètres que dans les expériences avec les mélodies (Section 7.3).

Nous avons ensuite fait des simulations avec notre réseau LSTM, avec et sans structure métrique. C'est une tâche très compliquée pour LSTM car la mélodie varie à chaque étape ou presque et les accords peuvent rester les mêmes pendant une mesure complète (soit 8 étapes). Nous avons donc séparé l'erreur de validation en deux, une pour la mélodie et une pour les accords. Nous prenons encore l'ensemble des poids qui minimise l'erreur totale sur un ensemble de validation.

Le tableau 7.4 montre les résultats d'entraînement de différents modèles. On remarque que l'erreur de validation des accords est très petite par rapport aux mélodies. Étant donné que les accords sont les mêmes pendant plusieurs étapes, le réseau peut facilement apprendre que l'accord au temps t est le même qu'au temps $t + 1$. Si LSTM

Modèle	Représentation	Err. de val mélodies	
		sans SM	avec SM
FF-16	Locale	—	0,0888
FF-16	Fréq	—	0,1419
LSTM-0	Locale	0,0189	0,0147
LSTM-16	Locale	0,0180	*0,0134
LSTM-0	Fréq	0,0315	*0,0162
LSTM-16	Fréq	0,0242	0,0153
		accords	
FF-16	Locale	—	0,0949
FF-16	Fréq	—	0,2940
LSTM-0	Locale	0,0076	0,0050
LSTM-16	Locale	0,0068	*0,0063
LSTM-0	Fréq	0,0054	*0,0052
LSTM-16	Fréq	0,0049	0,0027
		totale	
FF-16	Locale	—	0,0912
FF-16	Fréq	—	0,2179
LSTM-0	Locale	0,0144	0,0108
LSTM-16	Locale	0,0135	*0,0106
LSTM-0	Fréq	0,0184	*0,0107
LSTM-16	Fréq	0,0146	0,0090

Tab. 7.4. Expérience de génération de mélodies et d'accords avec une base de données de reels provenant de la base de données de Nottingham. On utilise la même notation pour les modèles et représentations que dans la table 7.3. L'erreur de validation des mélodies est l'erreur sur les neurones qui prédisent les mélodies. L'erreur de validation des accords est l'erreur sur les neurones qui prédisent les accords.

et la représentation de la structure métrique font bien leur travail, LSTM pourra prédire les changements d'accords.

L'erreur de validation sur les accords est très petite à comparer avec la mélodie mais pour la génération, on remarque que LSTM a de la difficulté à effectuer les changements d'accord. Il va souvent générer un seul accord et la mélodie va suivre cet accord.

Nous avons présenté, cette fois-ci, les résultats avec la représentation fréquentielle. Nous n'avons pas noté de grande différence entre les deux représentations de Shepard et fréquentielle. Nous voulons montrer que la représentation n'a pas tellement d'effet sur capacité à LSTM d'apprendre avec l'une ou l'autre des représentations.

La génération de chansons avec les accords ont donné de belles mélodies. Les changements d'accords se font plutôt rares mais les mélodies qui sont induites sont plus variées. Certaines personnes ayant écouté quelques chansons générées ne croyaient pas que la mélodie avait été générée par un ordinateur.

7.5. Discussion

Les résultats sont dans l'ensemble concluants. LSTM peut faire beaucoup mieux qu'un réseau de neurones feed-forward pour prédire les séquences de notes et d'accords. Nous avons omis tous résultats avec un réseau récurrent standard car le temps d'apprentissage est d'environ 5 fois plus longs pour les première et deuxième expériences (Section 7.1 et 7.2). LSTM a pu atteindre un minimum en 10 minutes tandis qu'un réseau récurrent peut prendre jusqu'à 50 minutes pour des résultats à peu près équivalent dans ces expériences. Pour les deux dernières expériences, le temps d'apprentissage de LSTM est d'environ 1 heure pour les représentations sans structure métrique et d'environ 4 heures pour les représentations avec structure métrique. Dans tous les cas, la génération de musique prend moins de 5 secondes. Avec ces temps d'apprentissage, les réseaux récurrents standard auraient pris jusqu'à 20 heures d'apprentissage.

Les représentations fréquentielles et Shepard n'ont pas tellement d'avantages l'un par rapport à l'autre. Il était impossible de les partager lors de la génération de chansons que chacun faisait. La représentation locale a plus de facilité à générer des formes mélodiques différentes. LSTM a tendance à être déstabilisé quand il emprunte des suites de notes différentes et finalement saturer complètement les portes. Dans les représentations fréquentielle et de Shepard, les notes qui "sonnent pareils" sont près l'une de l'autre, il est alors plus facile pour le réseau de générer une autre note que celle qu'il aurait fallu prédire. C'est ainsi que LSTM peut facilement voir de nouvelles formes et être déstabilisé.

Un grand avantage des représentations fréquentielles et de Shepard est que la dimension des vecteurs n'est pas affectée par le nombre de notes différentes à générer. À dimension égale, la représentation locale obtient rapidement un minimum dans l'ensemble de validation, environ 2 fois plus rapidement. Dans les expériences avec les reels, nous avons choisi un intervalle de 24 notes possibles et dans ce cas LSTM avec les représentations fréquentielles et de Shepard, le calcul est plus rapide durant l'apprentissage.

La structure métrique diminue en tout temps l'erreur de validation. La génération de chansons avec la structure métrique permet de garder la structure globale de la chanson ce qui est compliqué avec seulement un modèle next-step prediction. Avec LSTM, on a pu avoir un contour mélodique plus intéressant qu'un réseau feed-forward car il est capable de capter les dépendances locales.

7.6. Conclusion

La structure métrique est une composante importante de la musique. En étant capable de l'encoder pour qu'un réseau de neurones LSTM en tienne compte, nous avons pu mieux prédire et générer des séquences musicales. Nous avons effectué quatre séries d'expériences pour comprendre les limites de LSTM.

La première série (note dans du bruit, section 7.1) a montré que LSTM est capable de trouver des événements constants à intervalle régulier.

La deuxième série (walking bass, section 7.2) a montré que LSTM peut trouver des formes répétitives et les répéter. Par contre, il n'a pas été capable d'en générer des nouvelles de façon efficace.

La troisième série (mélodies, 7.3) a conclu que LSTM peut prédire des séquences musicales réelles mieux qu'un réseau feed-forward. En ajoutant l'encodage de la structure métrique, les prédictions étaient encore meilleures. Au niveau, de la génération, les mélodies sont intéressantes mais LSTM peut facilement tomber dans un état où les neurones sont saturés.

La quatrième série (mélodies et accords, 7.4) a conclu que LSTM est encore meilleur qu'un réseau feed-forward. En ajoutant les accords, on obtient une meilleure prédiction de la mélodie mais l'erreur de validation n'a pas montré ceci. En génération, les mélodies sont plus intéressantes car elles sont plus variées mais les changements d'accords sont très rares.

Chapitre 8

CONCLUSION

8.1. Travaux futurs

Notre recherche et nos expériences sucitent de nouvelles questions. On montre que LSTM peut bien prédire le temps où arrivent les événements, mais a-t-il a de bonnes propriétés pour prédire le rythme de séquences musicales en ignorant la hauteur des notes ? Avec l'encodage de la structure métrique, le rythme sera très bien aligné sur cette structure.

Nous nous sommes concentré sur un seul style de musique : les reels irlandais. En entraînant plusieurs réseaux LSTM, chacun avec son style musical, on peut réussir à faire de la classification en choisissant le style qui obtient la plus basse erreur en validation. Si nous voulons classifier les fichiers audio, on peut utiliser un onset-detector pour trouver où surviennent les notes dans le fichier. Avec ces séquences de onset, on peut entraîner un modèle LSTM comme avec les fichiers MIDI. Notre modèle peut donc être appliqué aux fichiers audio pour faire de la classification en utilisant le rythme. Les modèles de classification de genre qui utilise le rythme sont rares.

La base de données avec mélodies et accords n'était pas la meilleure car les changements d'accord surviennent de façon piquée : un accord est soutenu longtemps, ensuite surviennent quelques changement rapides et finalement on revient sur le premier accord. C'est normal, c'est le style irlandais. Il serait maintenant intéressant d'utiliser la base de données de Paiement et al. [2005] où des accords et des mélodies ont été ajoutées.

La structure métrique est une composante qui peut être utilisée dans d'autres domaines que la génération de séquences musicales. Dans la transformation de fichiers audio à fichiers MIDI, la structure métrique est importante car les répétitions de séquences y sont très fortement alignées. En déterminant ces répétitions on obtient plusieurs exemples de séquences audio qui correspondent à une même séquence MIDI.

Des méthodes d'évaluation des chansons générées seraient intéressantes pour s'assurer que la chanson n'est pas copiée de l'ensemble d'entraînement et que la chanson soit "écoutable". Ces deux volets doivent être considérés en même temps car le fait de copier l'ensemble d'entraînement rend la chanson générée "écoutable". Elle a été fait par un humain finalement. Si la chanson n'a pas été copier de l'ensemble d'entraînement, il faut que la chanson ne soit pas de simples notes aléatoires, mais une chanson avec une structure.

Pour le cas de la copie de l'ensemble d'entraînement, on pourrait utiliser une méthode automatique tel que la distance d'édition, mais cette méthode est plus complexe lorsque le rythme varie. Pour le cas de la beauté de la composition, il faudrait faire écouter les chansons à des gens et que ces gens notent les compositions.

Comme notre modèle peut apprendre en temps réel, nous pourrions construire un accompagnateur qui joue en même temps qu'un musicien. Dans les système d'exploitation tel que Mac OS X et Windows, les mécanismes pour traiter les événements MIDI en temps réel sont très efficaces. En obtenant les données d'entraînement d'un instrument MIDI tel qu'un piano au lieu d'un fichier, nous pouvons entraîner ou générer des mélodies avec un modèle entraîné.

Nous avons aussi pu construire des piano roll probabilistes (Figure 8.1). En encodant ces informations dans un nouveau type de fichier MIDI, nous pourrions produire des pièces qui jouent toujours la même chanson mais de façon différente à chaque interprétation.

Le type d'entraînement next-step prediction est casse-coup. Il peut facilement être déstabilisé du fait qu'il tente toujours de minimiser l'erreur au prochain temps. En tentant ceci, lors de la génération il essaie de ne pas se tromper au prochain et s'il se trompe le système se déstabilise et n'a pas été entraîné à revenir dans un bon état. Il serait intéressant de prendre une approche où on minimise l'erreur de ne pas être déçu à long terme. Nous pourrions prendre plutôt une approche de reinforcement learning.

8.2. Coda

La génération automatique de musique est une tâche difficile. Plusieurs avenues sont explorées pour générer de la musique, entre autre les méthodes à base de règles et par apprentissage. Nous nous sommes concentré sur un modèle par apprentissage avec les réseaux de neurones récurrents. Ils sont reconnus pour avoir de la difficulté

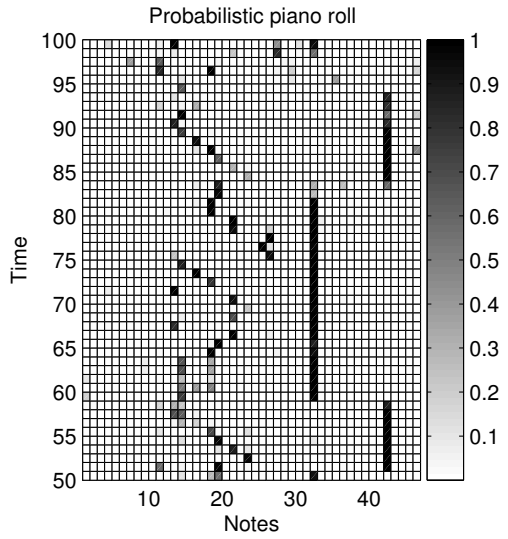

Fig. 8.1. Piano-roll probabiliste générée à partir d'un modèle LSTM. Une avancée possible dans un nouveau type de fichier MIDI. Les probabilité des notes (colonnes 1 à 26) et des accords (27 à 48) sont représenté par les teintes de gris. Les points foncées sont plus probables que les points plus pâles. Le temps suit de bas en haut.

à apprendre des séquences avec des événements qui ont des corrélations à de longs intervalles. Comme les réseaux de neurones LSTM ont plus de facilité à apprendre ce genre de séquences, nous les avons utilisés comme modèle principal. LSTM a un temps d'apprentissage beaucoup plus court qu'un réseau de neurones récurrents standard. Il utilise un peu plus de mémoire car on doit mémoriser les valeurs des dérivées partielles. Nous sommes par contre loin d'atteindre la limite d'un ordinateur de bureau.

Un aspect important de la musique est la structure métrique, car elle permet une structure globale dans une chanson. Nous avons réussi à encoder cette structure pour qu'elle soit prise en compte dans notre modèle LSTM. Nous avons montré qu'on pouvait ainsi prédire plus efficacement les séquences.

Un inconvénient de la structure métrique est que la dimension du vecteur d'entrée du réseau de neurones quadruple. En utilisant des représentations qui permettent de représenter plusieurs notes sans affecter la dimension de la représentation, le vecteur d'entrée permet d'avoir une taille raisonable. Le temps d'apprentissage augmente considérablement (environ 4 fois), mais une fois le modèle entraîné la génération de chanson est très rapide.

La génération de chansons avec LSTM avec la structure métrique a permis d'obtenir une meilleure structure globale dans les mélodies. Pour la génération des accords, LSTM n'a pas pu générer de belles progressions d'accord. Les chansons sont intéressantes surtout quand elles sont interprétées par un musicien qui ajoute de l'expression ce qui n'est pas fait par le générateur.

Les modèles de génération automatique de musique sont encore très loin de ce que peut faire l'humain. Les humains ont des très fortes connaissances à priori sur le style et la structure globale de chansons. C'est ce que nous avons voulu ajouter à notre modèle et mesurer sa capacité à apprendre et générer des séquences en ajoutant ces connaissances. Nous avons eu de meilleurs résultats qu'un modèle de prédiction pas-à-pas sans structure métrique.

Bibliographie

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

Christopher M. Bishop. Neural networks for pattern recognition. Oxford University Press, Oxford, UK, UK, 1996. ISBN 0-19-853849-9.

A. T. Cemgil, H. J. Kappen, and D. Barber. Generative model based polyphonic music transcription. In Proc. of IEEE WASPAA, New Paltz, NY, October 2003. IEEE Workshop on Applications of Signal Processing to Audio and Acoustics.

Grosvenor Cooper and Leonard B. Meyer. The Rhythmic Structure of Music. The Univ. of Chicago Press, 1960.

David Cope. Computers and Musical Style. A-R Editions, Inc., 1991.

Douglas Eck and Norman Casagrande. Finding meter in music using an autocorrelation phase matrix and Shannon entropy. In Proc. 6th International Conference on Music Information Retrieval (ISMIR 2005), 2005. Accepted.

Douglas Eck and Juergen Schmidhuber. Finding temporal structure in music : Blues improvisation with LSTM recurrent networks. In H. Bourlard, editor, Neural Networks for Signal Processing XII, Proc. 2002 IEEE Workshop, pages 747–756, New York, 2002. IEEE.

Judy A. Franklin. Recurrent neural networks and pitch representations for music tasks. In FLAIRS Conference, 2004.

F. A. Gers. Long short-term memory in recurrent neural networks, 2001.

Stephen Handel. Listening : An introduction to the perception of auditory events. MIT Press, Cambridge, Mass., 1993.

Martin Henz, Stefan Lauer, and Detlev Zimmermann. COMPOzE — intention-based music composition through constraint programming. In Proceedings of the 8th IEEE International Conference on Tools with Artificial Intelligence, pages 118–121, Toulouse, France, November16–19 1996. IEEE Computer Society Press.

Sepp Hochreiter and Juergen Schmidhuber. Long Short-Term Memory. Neural Computation, 9(8) :1735–1780, 1997.

Paul Kantor. Foundations of statistical natural language processing. Inf. Retr., 4(1) : 80–81, 2001. ISSN 1386-4564. doi : http://dx.doi.org/10.1023/A:1011424425034.

Fred Lerdahl and Ray Jackendoff. An overview of hierarchical structure in music. Music Perception, 1(2) :229–252, 1983–1984.

Brian C. Moore. An Introduction to the Psychology of Hearing. Academic Press, April 1982. ISBN 0125056281.

Michael C. Mozer. Neural network composition by prediction : Exploring the benefits of psychophysical constraints and multiscale processing. Cognitive Science, 6 :247–280, 1994.

Jean-François Paiement, Douglas Eck, and Samy Bengio. A probabilistic model for chord progressions. In Proc. 6th International Conference on Music Information Retrieval (ISMIR 2005), 2005. Accepted.

Juergen Schmidhuber, Felix Gers, and Douglas Eck. Learning nonregular languages : A comparison of simple recurrent networks and LSTM. Neural Computation, 14(9) : 2039–2041, 2002.

R. N. Shepard. Geometrical approximations to the structure of pitch. Psychological Review, 89 :305–333, 1982.

Peter M. Todd and Gregory M. Werner. Frankensteinian approaches to evolutionary music composition. In Niall Griffith and Peter M. Todd, editors, Musical Networks : Parallel Distributed Perception and Performance, pages 313–340. MIT Press, 1999. ISBN 0-262-07181-9. URL http://www-abc.mpib-berlin.mpg.de/users/ptodd/publications/99evmus/99evmus.pdf.

R. J. Williams and D. Zipser. Gradient-based learning algorithms for recurrent networks and their computational complexity. In Y. Chauvin and D. E. Rumelhart, editors, Back-propagation : Theory, Architectures and Applications, chapter 13, pages 433–486. Hillsdale, NJ : Erlbaum, 1995.

R. J.v Williams and D. Zipser. A learning algorithm for continually running fully recurrent neural networks. Neural Computation, In this volume, 1989. URL citeseer.ist.psu.edu/williams89learning.html.

Allen Winold and John Rehm. Introduction to music theory. Prentice-Hall, Englewood Cliffs, N.J., 2nd edition, 1979. ISBN 0134896661.