Composition automatique de musique actuelle

Maria Ashaeva, Pierrick Deshayes, Nicolas Dubrulle, *Université de Lille 1*

16 février 2017

Résumé

La génération automatique de musique est un problème très intéressant car il permet de tester la créativité d'un algorithme d'intelligence artificielle. Dans le passé, les différentes tentatives pour composer automatiquement de la musique avec les réseaux de neurones récurrents traditionnels (RNNs) se sont révélées infructueuses [3]. La raison est qu'ils possèdent une mémoire à court terme. Dans ce papier nous nous intéressons à différents types de réseaux de neurones récurrents : (a) les réseaux de neurones récurrents traditionnels (RNNs), (b) les réseaux Long Short Term Memory (LSTMs), ainsi qu'un réseau de neurones non récurrent : les Machines de Boltzmann restreintes (RBMs) . Dans un premier temps nous étudierons le fonctionnement de ces réseaux de neurones puis nous montrerons comment utiliser ces réseaux de neurones pour générer automatiquement de la musique. Enfin nous effectuerons des expériences dans lesquelles nous générons automatiquement des fichiers en format MIDI et MP3.

Sommaire

1 Introduction **3**

2 État de l'art **3**
2.1 Définition de la problématique . . . 3
2.2 Travaux existants . . . 4

3 Formats audio étudiés **6**
3.1 MIDI . . . 6
3.2 MP3 et WAV . . . 7

4 Architectures des réseaux de neurones étudiés **8**
4.1 Réseaux de neurones classiques - NN . . . 8
4.1.1 Généralités . . . 8
4.1.2 Qu'est-ce qu'un neurone ? . . . 8
4.1.3 Fonctionnement du réseau . . . 9
4.1.4 Apprentissage . . . 10
4.2 Réseaux de neurones récurrents - RNN . . . 10
4.3 Long term short memory - LSTM . . . 11
4.3.1 Architecture . . . 12
4.3.2 Apprentissage : Backpropagation Through Time . . . 13
4.4 Machine de Boltzmann restreinte . . . 14
4.4.1 Architecture . . . 14
4.4.2 Apprentissage . . . 16
4.4.3 Gibbs sampling dans les RBMs . . . 17
4.4.4 Divergence Contrastive - CD . . . 17

5 Google Magenta **18**

6 Expériences **19**
6.1 Reproduction d'expérience LSTM avec des fichiers MIDI . . . 19
6.1.1 Définition du problème . . . 19
6.1.2 Résultats . . . 19
6.2 Reproduction d'expérience LSTM avec des fichiers MP3 . . . 20
6.2.1 Définition du problème . . . 20
6.2.2 Représentation du modèle . . . 21
6.2.3 Génération . . . 21
6.2.4 Résultats . . . 22
6.3 Reproduction d'expérience RNN RBM avec des fichiers MIDI . . . 22
6.3.1 Représentation du modèle . . . 22
6.3.2 Génération . . . 23
6.3.3 Résultats . . . 23

7 Conclusion **24**

1 Introduction

La génération automatique de musique est un problème très intéressant car il permet de tester la créativité d'un algorithme d'intelligence artificielle. Ainsi, les modèles pour créer de nouvelles structures musicales font l'objet de nombreuses recherches. Bien que ces derniers aient une importance particulière du point de vue artistique, ils ont un potentiel commercial tout aussi important. Pour le moment, la finalité n'étant pas de créer un morceau qu'on retrouvera en tête des ventes mais plutôt d'aider des artistes en panne d'inspiration pour les aider à produire de la musique.

Parmi différentes approches, les réseaux récurrents de neurones (RNNs) sont apparus comme étant les plus prometteurs pour générer automatiquement de la musique. Après avoir été entraîné à prédire les enchaînements de notes les plus probables, le réseau peut alors créer et fournir ses propres compositions musicales. Cependant, ces réseaux possédant une mémoire à court terme, ils ne sont en pratique pas idéaux pour ce genre de tâche car, d'après Mozer (1994), « les pièces produites ne sont pas musicalement cohérentes et manquent de style et de rythme ». En effet, pour produire une note à un instant t, le réseau a besoin de connaître les notes jouées dans le passé (pas seulement les notes jouées récemment), c'est ce qu'on appelle les dépendances à long terme. Ainsi, bien que le réseau puisse apprendre les probabilités de transition d'une note à une note, les tentatives d'apprentissage sur un morceau complet sont restées vaines.

Pour pallier ce problème, on utilise des réseaux récurrents de neurones ayant une mémoire à long terme : les réseaux Long Short Term Memory (LSTMs). Ils ont été initialement proposés par Hochreiter et Schmidhuber en 1997 [8]. Eck & Schmidhuber (2002) ont été les premiers à les utiliser pour apprendre et composer du blues [3]. En 2006 Franklin les utilisera pour produire du jazz. Les LSTMs ont une mémoire persistante qui est transmise à chaque instant t. A chaque pas de temps on peut lui ajouter ou retirer des informations grâce à des structures appelées « gates ». Les LSTMs, qui par définition, sont capables de travailler sur des événements plus espacés les uns des autres, fournissent donc de meilleurs mécanismes pour la génération de musique. Ils paraissent donc avoir la structure la mieux adaptée pour produire de la musique, la qualité d'une pièce produite dépend alors de la phase d'apprentissage [9]. Nous expliquerons en détail le fonctionnement de ce type de réseau de neurones ainsi que la phase d'apprentissage.

En parallèle nous travaillerons avec un autre type de réseau de neurones : la machine restreinte de Boltzmann (RBM) initialement introduite par Paul Smolensky sous le nom de Harmonium. Cette dernière aura pour but de déterminer la probabilité pour chaque note d'être jouée à un instant t. La finalité étant de produire une pièce musicale qui soit la plus harmonieuse possible. Nous détaillerons ensuite l'algorithme d'apprentissage automatique qui permet de produire cette pièce musicale.

2 État de l'art

2.1 Définition de la problématique

Les premiers pas de la composition automatique de musique reposaient sur l'analyse statistique de musiques existantes. Cette démarche constitue la base la base de la plupart des expériences de composition automatisée depuis. Un outil répandu pour la composition de musique est l'emploi des chaînes de Markov, cette méthode nécessite d'abord d'analyser de la musique existante, c'est ce qu'on appelle l'apprentissage supervisé.

Si la musique automatique n'a pas encore touché le grand public, il y a un domaine dans lequel elle s'est depuis longtemps imposée comme une évidence : les jeux vidéos. Adapter la musique et l'ambiance à ce que fait le joueur est un défi que les compositeurs de musique de jeu vidéo ont toujours à l'esprit. Par exemple un programme réalisé par FX Dupas permet de générer de la musique d'ambiance variant selon une jauge d'intensité. La musique peut s'accélérer et devenir menaçante ou au contraire s'apaiser et se calmer sans que la transition ne soit brutale.

Finalement le but n'est pas d'apprendre à l'ordinateur d'écrire de la musique par lui-même mais de lui apprendre à écrire ce que nous voulons qu'il écrive.

2.2 Travaux existants

L'un des premiers modèles formels issus de l'informatique utilisés comme générateur musical, dès la fin des années cinquante, est celui des chaînes de Markov utilisé par Xenakis pour la composition d'*Analogiques A* pour 9 cordes (1958), de son complément électroacoustique *Analogiques B*, puis dans *Syrmos* pour 19 cordes (1959). Les chaînes de Markov sont des automates finis munis de probabilités de transitions entre états. Elles ont été introduites à l'origine pour décrire des statistiques d'enchaînement entre lettres ou entre mots dans un texte. La chaîne de Markov utilisée dans *Analogiques A* contrôle l'enchaînement de nuages de sons que Xenakis appelle des trames. Chaque trame remplit une demi-mesure de la partition (soient deux temps dans une mesure à quatre temps). Les trames sont définies par des densités moyennes de sons répartis dans différents registres et avec différentes intensités (Figure 1). Xenakis définit huit trames A, B, C, D, E, F, G, H. La trame A, par exemple, contient :

— trois sons forte entre mi_0 et mi_1,
— un son double-piano entre mi_1 et re_2,
— un son double-piano entre do_4 et si_4,
— neuf sons double-forte entre si_4 et la_5.

Les densités sont des valeurs moyennes qui sont réalisées par une loi de probabilité. Les notes obtenues sont distribuées entre les instruments (trois violons, trois violoncelles, trois contrebasses), chaque groupe d'instruments combinant trois subdivisions rythmiques superposées (croches, triolets de noires, quintolets de croches).

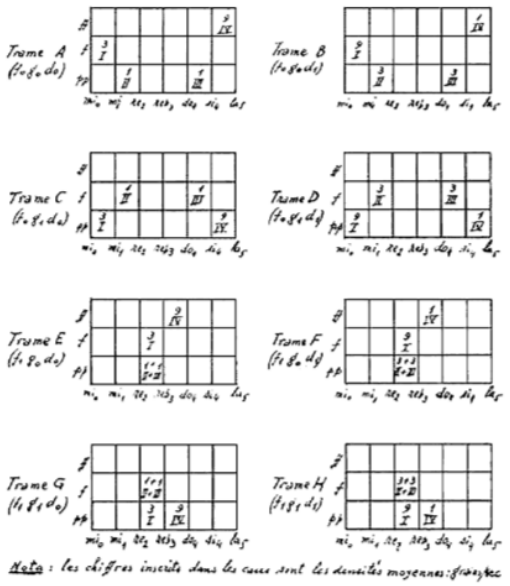


Figure 1 – *Les 8 trames d'Analogique A*

Pour définir la chaîne de Markov contrôlant la succession des trames, Xenakis introduit une matrice de probabilités de transitions [Xenakis 1963, p. 107] :

	A	*B*	*C*	*D*	*E*	*F*	*G*	*H*
A	0,021	0,357	0,084	0,189	0,165	0,204	0,408	0,096
B	0,084	0,089	0,076	0,126	0,150	0,136	0,072	0,144
C	0,084	0,323	0,021	0,126	0,150	0,036	0,272	0,144
D	0,336	0,081	0,019	0,084	0,135	0,024	0,048	0,216
E	0,019	0,063	0,336	0,171	0,110	0,306	0,102	0,064
F	0,076	0,016	0,304	0,114	0,100	0,204	0,018	0,096
G	0,076	0,057	0,084	0,114	0,100	0,054	0,068	0,096
H	0,304	0,014	0,076	0,076	0,090	0,036	0,012	0,144

Chaque colonne indique les probabilités d'enchaîner une trame avec les huit. Ainsi la trame A a une probabilité 0,021 d'être suivie de A, et une probabilité 0,336 d'être suivie de D.

En 1989 Todd a été l'un des premiers à générer automatiquement de la musique classique avec un réseau récurrent appelé le réseau de Jordan [9]. Ce réseau possède une connexion de la couche de sortie à la couche d'entrée comme le montre la Figure 2. Il est entraîné grâce à la méthode de la rétropropagation. Après cette phase d'apprentissage il peut générer de la musique en mélangeant et en variant les notes des données d'apprentissage composées de mélodies monophoniques.

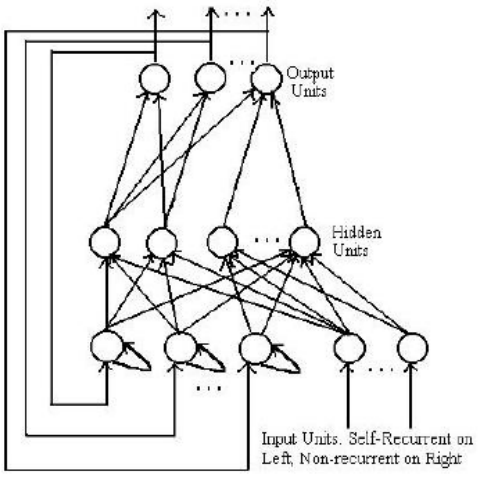


Figure 2 – *Le réseau de Jordan*

En 1994 Mozer a utilisé un réseau récurrent entièrement connecté qu'il a entraîné avec la même méthode de rétropropagation pour apprendre et reproduire de la musique [10]. Les sorties de la couche finale représentent la probabilité qu'une note soit on ou off, c'est à dire qu'une note soit jouée ou relâchée.

En 2001, Franklin a utilisé le même réseau que Todd et y a ajouté une seconde phase d'apprentissage par renforcement [9]. Durant cette phase de renforcement, une valeur est calculée parmi un ensemble de "règles musicales" pour déterminer la qualité de la sortie.

Pour régler le problème du *vanishing gradient* Eck & Schmidhuber (2002) ont utilisé 2 réseaux LSTM, un pour apprendre les mélodies et un pour apprendre les accords, pour composer du blues. La sortie du réseau des accords est connecté à l'entrée du réseau des mélodies. Le système est capable de reproduire des notes de musique qui suivent les accords.

Chung et al. ont utilisé des réseaux GRU (Gated Recurrent Unit) pour modéliser des séquences de musique polyphoniques à partir de fichiers MIDI. Leur résultats sont comparables aux réseaux LSTM.

3 Formats audio étudiés

3.1 MIDI

Le format MIDI (pour Musical Instrument Digital Interface) correspond à une représentation symbolique de l'information musicale. il ne contient pas de signal audio, mais uniquement des messages de commandes qui sont un ensemble d'informations sur les notes (volume, expression, panoramique, vélocité, hauteur, etc). Le protocole définit les messages note-on et note-off pour déclencher et arrêter chaque note. Une vélocité est associée à chaque note, permettant d'indiquer si la note est plus ou moins jouée fortement. Les notes de l'échelle chromatique sont représentées par un nombre entier codé sur 7 bits, permettant de coder plus de 10 octaves : du C-1 (note 0) au G9 (note 127) avec une résolution d'un demi-ton (Figure 4).

Par exemple la suite de notes $69, 69, 70, 70, 69, 69, 67, 67, 74, 74, 74, 72, 70, 70$ correspond à la partition suivante :
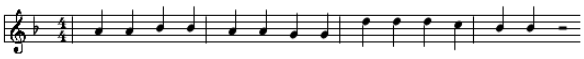


Figure 3 – *Représentation d'une partition*

Octave	Note Numbers											
	C	C#	D	D#	E	F	F#	G	G#	A	A#	B
-1	0	1	2	3	4	5	6	7	8	9	10	11
0	12	13	14	15	16	17	18	19	20	21	22	23
1	24	25	26	27	28	29	30	31	32	33	34	35
2	36	37	38	39	40	41	42	43	44	45	46	47
3	48	49	50	51	52	53	54	55	56	57	58	59
4	60	61	62	63	64	65	66	67	68	69	70	71
5	72	73	74	75	76	77	78	79	80	81	82	83
6	84	85	86	87	88	89	90	91	92	93	94	95
7	96	97	98	99	100	101	102	103	104	105	106	107
8	108	109	110	111	112	113	114	115	116	117	118	119
9	120	121	122	123	124	125	126	127				

Figure 4 – *Représentation des notes dans le format MIDI*

Plus précisément, un fichier MIDI est composé de une ou plusieurs pistes, chaque piste correspondant aux notes jouées par un instrument. Une piste est composée de plusieurs événements, un événement représente une certaine note (avec une valeur entre 0 et 127) jouée pendant une certaine durée et avec une certaine vélocité (entre 0 et 127). la vélocité sert à décrire le volume d'une note jouée, plus la valeur est grande et plus le volume est élevé. (Figure 5)

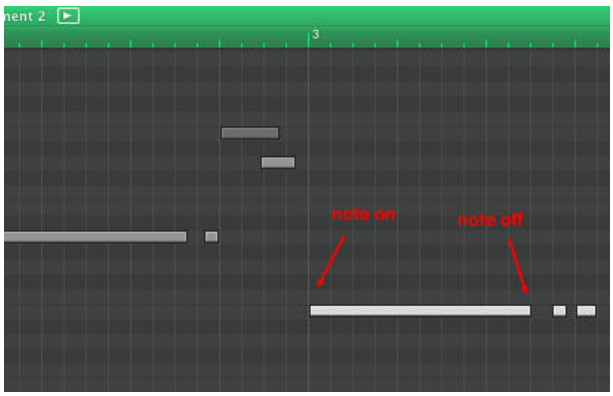


Figure 5 – *Structure d'un fichier MIDI*

Certains logiciels représentent la vélocité d'une note par la couleur. dans l'image ci-dessus plus une note est blanche, plus a vélocité est élevée, plus elle est grise et moins la vélocité est élevée.

Utiliser des fichiers MIDI comprend plusieurs avantages :

- le format MIDI est compact, une chanson ne contient que quelques centaines de messages MIDI,
- il est facile à modifier, on peut changer une note ou la durée d'une note sans avoir à ré-enregistrer la chanson,
- un fichier MIDI décrit seulement les notes à jouer, on peut changer l'instrument pour obtenir un son différent.

3.2 MP3 et WAV

Le format MP3 et WAV correspond à une représentation physique de l'information musicale. Les fichiers WAV contiennent une information sur le son en termes d'amplitude et de durée

retranscrits approximativement en numérique lors de la conversion analogique/numérique. En d'autre termes et étant donnée une source audio, un fichier WAV généré à partir de cette source contiendra une numérisation de cette source. La numérisation se charge d'échantillonner la valeur du signal puis de la quantifier.
- L'échantillonnage (ou *sampling*) prélève, le plus souvent à intervalles réguliers, la valeur du signal. Chacune de ces valeurs fournit un échantillon (aussi appelé *sample*). La fréquence d'échantillonnage, exprimée en Hz, est appelée *sample rate*.
- La quantification transforme ensuite chacun de ces samples en valeurs numériques discrètes. Le nombre de bits alloués pour la quantification est appelé *bit depth* et correspond à la "résolution" de chaque sample.

4 Architectures des réseaux de neurones étudiés

4.1 Réseaux de neurones classiques - NN

4.1.1 Généralités

Un réseau de neurones est un modèle de calcul dont l'organisation et le fonctionnement sont inspirées du cerveau. A l'origine l'objectif des chercheurs était de construire une machine capable de reproduire certains aspects de l'intelligence humaine. Les réseaux de neurones sont donc conçus pour reproduire certaines caractéristiques du cerveau comme :

— la capacité d'apprentissage ;
— la capacité de mémoriser l'information ;
— la capacité de traiter des informations incomplètes.

Ils sont en quelque sorte une tentative de modélisation mathématique du cerveau humain.

Ils sont utilisés dans des domaines variés, par exemple :

— en biologie pour la classification d'espèces animales étant donnée une analyse ADN ;
— en imagerie pour la reconnaissance de motifs (notamment par les banques pour vérifier le montant des chèques, la Poste pour trier le courrier en fonction du code postal) ;
— en statistiques pour l'analyse de données, la prévision ou la classification ;
— dans la finance pour des estimations boursières ;
— en météorologie pour la prévision du temps ;
— en robotique pour le contrôle et le guidage de robots ou de véhicules autonomes.

4.1.2 Qu'est-ce qu'un neurone ?

Chaque neurone du réseau reçoit un certain nombre de variables d'entrées provenant de neurones en amont. A chacune de ces entrées est associé un poids qui représente l'importance de la connexion entre ces deux neurones. Chaque neurone possède une sortie, ces sorties représentent les entrées qui alimenteront d'autres neurones en aval.

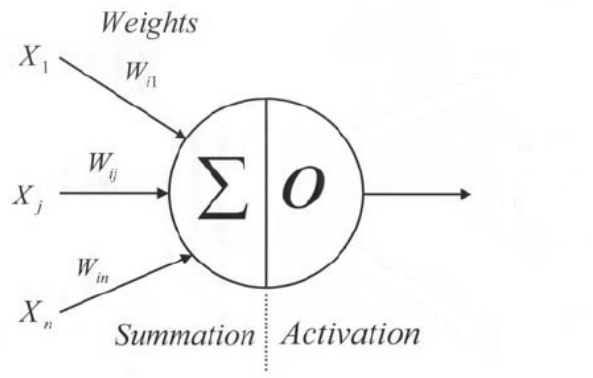


Figure 6 – *Schéma d'un neurone artificiel*

D'abord le neurone fait le calcul de la somme pondérée des entrées puis à partir de cette valeur, une fonction d'activation calcule la sortie du neurone. Les fonctions d'activation retournent une valeur dans $[0; 1]$ pour la fonction sigmoïde, dans $[-1; 1]$ pour la fonction tangente hyperbolique ou dans {-1,1} pour la fonction signe.
La sortie du neurone i est donnée par la formule suivante :

$$o_i = f(b + \sum_{j=1}^{n} (X_j.W_{ij}))$$

où f est la fonction d'activation, W_{ij} le vecteur des poids, X_j le vecteur des entrées et b le biais.

4.1.3 Fonctionnement du réseau

Un réseau est donc une association de plusieurs neurones, il existe principalement trois types de réseau. Ils diffèrent selon la manière dont les neurones sont connectés entre eux :

- les réseaux partiellement connectés : chaque neurone est lié à quelques neurones ;
- les réseaux entièrement connectés : chaque neurone est lié à tous les autres neurones du réseau ;
- les réseaux multicouches : les neurones d'une couche sont connectés uniquement aux neurones de la couche en aval.

La qualité de la sortie du réseau dépend en partie de la configuration et du nombre de neurones.

Actuellement les réseaux multicouches (ou perceptrons multicouches) sont les plus utilisés. Ils sont utilisés pour classifier des données qui ne sont pas linéairement séparables, ce que les réseaux monocouches (perceptrons) ne peuvent pas faire. Le réseau est organisé en plusieurs couches au sein desquelles une information circule de la couche d'entrée vers la couche de sortie uniquement, il s'agit donc d'un réseau de type feedforward. Chaque couche est constituée d'un nombre variable de neurones. Il n'y a pas de connexion entre les neurones d'une même couche. Chaque neurone d'une couche est connecté avec tous les neurones de la couche suivante. La couche d'entrée est l'ensemble des neurones en entrée, la couche de sortie est l'ensemble des neurones en sortie, les autres couches sont appelées couches cachées, elles n'ont aucun contact avec l'"extérieur". De plus chaque neurone est une unité de calcul indépendante des autres neurones.

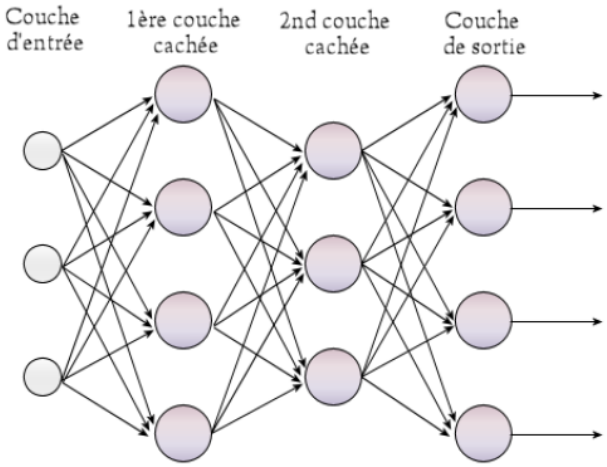


Figure 7 – *Schéma d'un perceptron multicouches*

4.1.4 Apprentissage

Les paramètres les plus importants dans un réseau de neurones sont les poids. Ce sont eux qui déterminent l'évolution du réseau en fonction de ses informations d'entrée. Dans le cas des réseaux à apprentissage supervisé, les données servant à l'apprentissage peuvent être représentées par le couple (entrée ; sortie désirée correspondante). Il faut choisir un mécanisme permettant de faire converger les poids de façon à ce que la sortie obtenue pour chaque entrée soit la plus proche de la sortie souhaitée. C'est la phase d'apprentissage.

Tout d'abord il faut définir une fonction de perte (par exemple l'erreur quadratique). Dans le cas d'une classification supervisée, si on possède N exemples d'apprentissage alors, pour $1 \leq k \leq N$ notons (x_k, y_k) où x_k est le vecteur contenant les n informations de l'exemple k et y_k la sortie attendue de l'exemple k (la vraie sortie). Enfin, on note s_k la sortie obtenue avec les poids actuels. L'erreur quadratique est définie comme suit :

$$E(W) = \frac{1}{N} \sum_{k=1}^{N} (y_k - s_k)^2$$

L'apprentissage consiste donc à trouver les poids qui minimisent cette fonction :

$$W^* = argmin E(W)$$

La minimisation de la fonction de perte s'effectue généralement grâce à la méthode de rétropropagation du gradient. C'est une méthode qui permet de calculer le gradient de l'erreur pour chaque neurone, de la dernière couche vers la première. Ainsi, Les poids qui contribuent à engendrer une erreur importante se verront modifiés de manière plus significative que les poids qui ont engendré une erreur marginale. On modifie alors les poids de telle sorte qu'à la prochaine itération, l'erreur commise entre la sortie calculée et connue soit minimisée. En appliquant cette étape plusieurs fois, l'erreur tend à diminuer et le réseau offre une meilleure prédiction.

4.2 Réseaux de neurones récurrents - RNN

Les réseaux récurrents ont été créés dans les années 1980. Ils sont particulièrement utiles dans le traitement de données séquentielles car, à la différence des réseaux traditionnels, chaque neurone possède une mémoire interne qui capture les informations qui ont été calculées avant.

Ce modèle est donc bien adapté pour, par exemple, prédire le prochain mot dans une phrase car dans ce cas il est indispensable de connaître quels mots sont apparus avant. Il est donc bien adapté pour générer de la musique car la musique n'est rien d'autre qu'une séquence de notes. Voici comment on peut représenter un réseau récurrent :

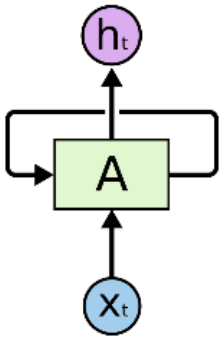


Figure 8 – *Schéma d'un réseau récurrent de neurones*

Dans le diagramme ci-dessus A représente une partie du réseau à l'instant t, x_t est l'entrée et h_t la sortie. La boucle permet à l'information d'être transmise à l'instant $t + 1$.

Pour la phase d'entraînement on utilise une technique appelée *BackPropagation Through Time* (BPTT) qui permet d'ajuster les poids du réseau et ainsi lui permettre de mieux apprendre. La phase d'apprentissage est détaillée dans la partie suivante.

4.3 Long term short memory - LSTM

Le problème avec les réseaux récurrents traditionnels est que la mémoire est à court terme. Une valeur en sortie à un instant t devient une valeur d'entrée à l'instant t+1 mais cette valeur sera ensuite perdue à moins que la valeur de sortie à l'instant t+1 soit la même que la valeur en entrée. Ce qui rend le réseau incapable de traiter les dépendances à long terme (les notes qui ont été jouées avant), or en musique les dépendances à long terme sont primordiales, en effet elles permettent de définir la structure globale d'un morceau et donc définir un style particulier [3].

Pour pallier à ce problème, on utilise des réseaux de neurones ayant une mémoire à long terme : les LSTMs. Ils ont été initialement proposés par Hochreiter et Schmidhuber en 1997 [8]. Les LSTMs ont une mémoire persistante, elle est transmise à chaque instant t. A chaque pas de temps on peut lui ajouter ou retirer des informations grâce à des structures appelées "gates".

Les LSTMs ont été spécialement conçus dans le but de palier aux problèmes des dépendances sur le long terme. La sauvegarde d'informations au cours du temps étant leur particularité, ils permettent d'exceller dans les domaines où les différentes valeurs choisies dans le passé sont nécessaires pour créer et influencer celles du futur.

Une variante des LSTMs "classiques" est celle introduite par Gers et Schmidhuber en 2000 [5], qui ajoute des "peephole connections". Ces connexions permettent à certaines gates d'accéder à la valeur de la mémoire de la cellule courante.

4.3.1 Architecture

On prendra ici les notations :
— x_t le vecteur d'entrée au temps t
— c_{t-1} la mémoire de la cellule au temps t-1
— h_{t-1} le vecteur de sortie au temps t-1
— W_* les matrices des poids d'entrée
— R_* les matrices des poids pour les valeurs récurrentes
— p_* les vecteurs de poids associés aux peepholes
— b_* les vecteurs de biais

Les fonctions sigmoïdes sont utilisées comme fonctions d'activation des "*gates*", tandis que les tangentes hyperboliques sont utilisées pour l'activation des *input* et *output block*.

Le produit de Hadamard (ou pointwise product) est désigné par $\odot$.

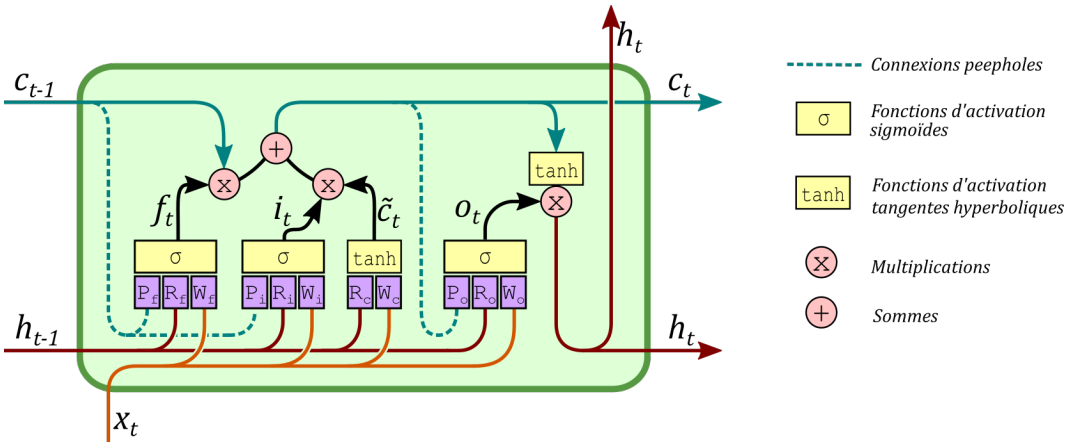


Figure 9 – *Schéma d'un LSTM avec peepholes*

La première étape dans le fonctionnement d'un LSTM est de décider quelles informations seront supprimées de la cellule. Cette décision est prise par une fonction sigmoïde appelée *forget gate*. Elle prend en entrée les vecteurs h_{t-1} et x_t, et retourne un vecteur dont les composantes sont comprises entre 0 et 1, où 0 indique la suppression totale de l'information et 1 indique que l'information sera complètement gardée :

$$\bar{f}_t = W_f x_t + R_f h_{t-1} + p_f \odot c_{t-1} + b_f \quad (1)$$
$$f_t = \sigma(\bar{f}_t) \quad (2)$$

forget gate

De la même façon, l'input gate décide quelles valeurs vont être mises à jour :

$$\bar{i}_t = W_i x_t + R_i h_{t-1} + p_i \odot c_{t-1} + b_i \quad (3)$$
$$i_t = \sigma(\bar{i}_t) \quad (4)$$

input gate

Une fonction tanh s'occupe de créer de nouvelles valeurs candidates :

$$\bar{\tilde{c}}_t = W_c x_t + R_c h_{t-1} \quad (5)$$
$$\tilde{c}_t = tanh(\bar{\tilde{c}}_t) \quad (6)$$

block input

L'ancienne valeur de la mémoire c_{t-1} est ensuite mise à jour. Pour cela, on utilise les valeurs calculées précédemment (pour oublier quelques informations tout en en ajoutant de nouvelles). On multiplie l'état précédent c_{t-1} par f_t (pour oublier les informations) puis on ajoute $i_t \odot \tilde{c}_t$ (les valeurs candidates "scalées" par i_t) :

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \tag{7}$$

cell state

Enfin, la décision de quelles valeurs seront en sortie s'effectue en deux étapes. On calcule d'abord la valeur de l'output gate o_t via une fonction sigmoïd qui décide quelle partie de la mémoire sera exposée en sortie. Sa valeur est définit par :

$$\bar{o}_t = W_o x_t + R_o h_{t-1} + p_o \odot c_t + b_o \tag{8}$$
$$o_t = \sigma(\bar{o}_t) \tag{9}$$

output gate

Pour finir, la valeur de la cellule c_t est passée à une fonction tanh puis multipliée par la sortie de l'output gate o_t. On obtient ainsi en sortie le vecteur h_t qui contient les valeurs désirées :

$$h_t = o_t * tanh(c_t) \tag{10}$$

block output

4.3.2 Apprentissage : Backpropagation Through Time

Ici Δ_t est le vecteur des deltas transmis par la couche suivante.

Si E est la fonction de perte, elle correspond normalement à $\frac{\partial E}{\partial \mathbf{h}_t}$, mais n'inclut pas les dépendances récurrentes [6]. Les deltas dans un bloc de LSTM sont ensuite calculés comme suit :

$$\begin{aligned}
\delta\mathbf{h}_t &= \Delta_t + \mathbf{R}_{\tilde{c}}^T \delta\tilde{\mathbf{c}}_{t+1} + \mathbf{R}_i^T \delta\mathbf{i}_{t+1} + \mathbf{R}_f^T \delta\mathbf{f}_{t+1} + \mathbf{R}_o^T \delta\mathbf{o}_{t+1} \\
\delta\mathbf{o}_t &= \delta\mathbf{h}_t \odot h(\mathbf{c}_t) \odot \sigma'(\bar{\mathbf{o}}_t) \\
\delta\mathbf{c}_t &= \delta\mathbf{h}_t \odot \mathbf{o}_t \odot h'(\mathbf{c}_t) + \mathbf{p}_o \odot \delta\mathbf{o}_t + \mathbf{p}_i \odot \delta\mathbf{i}_{t+1} \\
&\quad + \mathbf{p}_f \odot \delta\mathbf{f}_{t+1} + \delta\mathbf{c}_{t+1} \odot \mathbf{f}_{t+1} \\
\delta\mathbf{f}_t &= \delta\mathbf{c}_t \odot \mathbf{c}_{t-1} \odot \sigma'(\bar{\mathbf{f}}_t) \\
\delta\mathbf{i}_t &= \delta\mathbf{c}_t \odot \tilde{\mathbf{c}}_t \odot \sigma'(\bar{\mathbf{i}}_t) \\
\delta\tilde{\mathbf{c}}_t &= \delta\mathbf{c}_t \odot \mathbf{i}_t \odot g'(\bar{\tilde{c}}_t)
\end{aligned}$$

Les deltas pour les inputs ne sont nécessaires que si les couches précédentes ont besoin d'être entraînées :

$$\delta\mathbf{x}_t = \mathbf{W}_{\tilde{c}}^T \delta\tilde{\mathbf{c}}_t + \mathbf{W}_i^T \delta\mathbf{i}_t + \mathbf{W}_f^T \delta\mathbf{f}_t + \mathbf{W}_o^T \delta\mathbf{o}_t$$

Enfin, les valeurs des poids sont mises à jour comme ceci (où $\star$ désigne n'importe quel élément de $\{\tilde{\mathbf{c}}, \mathbf{i}, \mathbf{f}, \mathbf{o}\}$) :

$$\delta \mathbf{W}_\star = \sum_{t=0}^{T} \delta \star_t \otimes \mathbf{x}_t \qquad \delta \mathbf{p}_i = \sum_{t=0}^{T-1} \mathbf{c}_t \odot \delta \mathbf{i}_{t+1}$$

$$\delta \mathbf{R}_\star = \sum_{t=0}^{T-1} \delta \star_{t+1} \otimes \mathbf{h}_t \qquad \delta \mathbf{p}_f = \sum_{t=0}^{T-1} \mathbf{c}^t \odot \delta \mathbf{f}_{t+1}$$

$$\delta \mathbf{b}_\star = \sum_{t=0}^{T} \delta \star_t \qquad \delta \mathbf{p}_o = \sum_{t=0}^{T} \mathbf{c}_t \odot \delta \mathbf{o}_t$$

où $\otimes$ désigne le produit dyadique de deux vecteurs.

4.4 Machine de Boltzmann restreinte

Les machines de Boltzmann restreintes (RBM) sont des modèles particuliers de réseaux de neurones qui permettent de modéliser des distributions de probabilité de jeux de données. Elles peuvent être également vues comme des modèles génératifs, puisque elles sont associées à un processus de génération de données suivant la distribution qu'elles encodent. RBM a initialement été inventée sous le nom de "*Harmonium*" en 1986 par Paul Smolenski [12].

4.4.1 Architecture

Une machine de Boltzmann est composée d'une couche de neurones, qui reçoit l'entrée, ainsi que d'une couche de neurones cachée. De plus on suppose que les neurones d'une même couche sont indépendants entre eux, donc on appelle cette configuration une machine de Boltzmann restreinte (RBM).

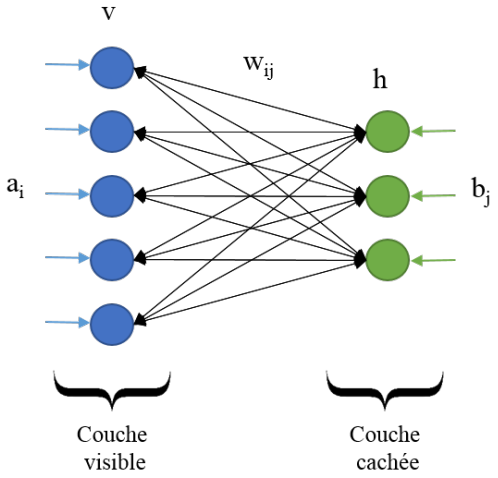


Figure 10 – *Architecture de RBM*

Une énergie d'activation pour RBM est donnée par :

$$E = -\sum_{i}^{n} a_i v_i - \sum_{j}^{m} b_j h_j - \sum_{i,j}^{n,m} w_{ij} x_i h_j \tag{11}$$

Avec :

— $w_{i,j}$ la matrice de poids entre le neurone i et le neurone j ;
— v_i est l'état, $v_i \in 0, 1$, du neurone i de la couche visible ;
— h_j est l'état, $h_j \in 0, 1$, du neurone j de la couche cachée ;
— a_i et b_j sont respectivement les biais des neurones v_i et h_j.

La probabilité conjointe d'avoir une configuration (v_i, h_j) est définie comme :

$$P(v, h) = \frac{1}{Z} e^{-E(v,h)} \tag{12}$$

$$Z = \sum_{r}^{N} \sum_{t}^{M} e^{-E(v^{(r)}, h^{(t)})} \tag{13}$$

Avec :
— E la fonction d'énergie ;
— Z la fonction de normalisation (pour que la somme de toutes les probabilités soit égale à 1).

La probabilité totale est donc définie par :

$$P(v) = \sum_{t}^{M} P(v, h^{(t)}) = \frac{1}{Z} \sum_{t}^{M} e^{-E(v,h^{(t)})} \tag{14}$$

Considérons maintenant la probabilité de $h_k = 1$ étant donné l'état v. Supposons donc que l'énergie du système soit égale à E_0 si $h_k = 0$ et à E_1 si $h_k = 1$. Alors :

$$P(h_k = 1|v) = \frac{e^{-E_1}}{e^{-E_1} + e^{-E_0}} = \frac{1}{1 + e^{E_1 - E_0}} = \frac{1}{1 + e^{-b - \sum_i^n v_i w_{ik}}} = \sigma(-b - \sum_{i}^{n} v_i w_{ik}) \tag{15}$$

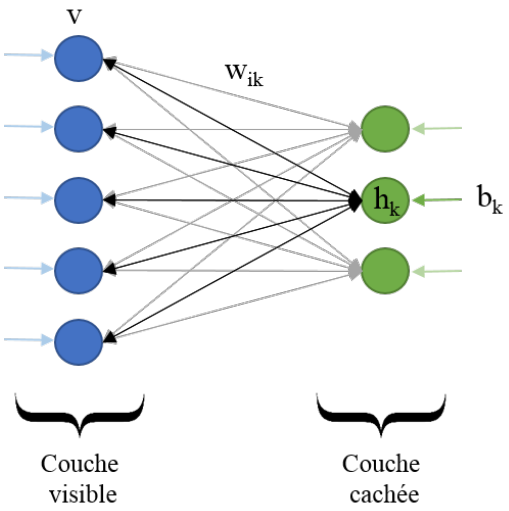


Figure 11 – *les éléments influençant l'état du neurone h_k*

Sachant que les neurones dans un groupe sont indépendants entre eux, alors :

$$P(h|v) = \prod_{j}^{m} P(h_j, v) \tag{16}$$

De la même façon :

$$P(v|h) = \prod_{i}^{n} P(v_i, h) \tag{17}$$

4.4.2 Apprentissage

La but de l'apprentissage est de faire en sorte que le vecteur reconstitué soit très proche du vecteur original, autrement dit, maximiser la probabilité $P(v)$. Pour faire cela il faut trouver les dérivées partielles de la probabilité selon les paramètres du modèle. Commençons par dériver la fonction d'énergie :

$$\frac{\delta E(v,h)}{\delta a_i} = -v_i$$
$$\frac{\delta E(v,h)}{\delta b_j} = -h_j$$
$$\frac{\delta E(v,h)}{\delta w_{ij}} = -v_i h_j$$

Les dérivées de l'exposant de l'énergie négative sont définies comme suit :

$$\frac{\delta e^{-E(v,h)}}{\delta a_i} = v_i e^{-E(v,h)}$$
$$\frac{\delta e^{-E(v,h)}}{\delta b_j} = h_j e^{-E(v,h)}$$
$$\frac{\delta e^{-E(v,h)}}{\delta w_{ij}} = v_i h_j e^{-E(v,h)}$$

La dérivée de la fonction de normalisation Z est donnée par :

$$\frac{\delta Z}{\delta a_i} = \sum_r^N \sum_t^M v_i^{(r)} e^{-E(v^{(r)},h^{(t)})}$$
$$\frac{\delta Z}{\delta b_j} = \sum_r^N \sum_t^M h_j^{(t)} e^{-E(v^{(r)},h^{(t)})}$$
$$\frac{\delta Z}{\delta w_{ij}} = \sum_r^N \sum_t^M v_i^{(r)} h_j^{(t)} e^{-E(v^{(r)},h^{(t)})}$$

Prenons la dérivée de $P(v^{(k)})$:

$$\frac{\delta P(v^{(k)})}{\delta w_{ij}} = \frac{1}{Z^2}(Z(\sum_t^M v_i^{(r)} h_j^{(t)} e^{-E(v^{(r)},h^{(t)})}) - (\sum_t^M e^{-E(v^{(r)},h^{(t)})})(\sum_r^N \sum_t^M v_i^{(r)} h_j^{(t)} e^{-E(v^{(r)},h^{(t)})}))$$

Dans la phase d'entraînement on cherche à minimiser le log-vraisemblance, donnée par la formule suivante :

$$\mathcal{L}_{gen} = \frac{1}{N} \sum_r^N -log(P(v^{(r)})) \tag{18}$$

Les dérivées du log-vraisemblance par rapport à w_{ij} données par :

$$\frac{\delta logP(v^k)}{\delta w_{ij}} = \frac{\sum_t^M v_i^{(k)} h_j^{(t)} e^{-E(v^{(k)},h^{(t)})}}{\sum_t^M e^{-E(v^{(k)},h^{(t)})}} - \frac{\sum_r^N \sum_t^M v_i^{(r)} h_j^{(t)} e^{-E(v^{(r)},h^{(t)})}}{\sum_r^N \sum_t^M e^{-E(v^{(r)},h^{(t)})}}$$

$$\frac{\delta logP(v^k)}{\delta w_{ij}} = \sum_t^M v_i^{(k)} h_j^{(t)} P(h^{(t)}|v^{(k)}) - \sum_r^N \sum_t^M v_i^{(r)} h_j^{(t)} P(h^{(t)}, v^{(k)})$$

De la même façcon on peut obtenir les dérivées pour les paramètres a_i et b_j. Finalement, on obtient les dérivées suivantes :

$$\frac{\delta logP(v^k)}{\delta w_{ij}} = \mathbb{E}[v_i^{(k)} h_j]_{data} - \mathbb{E}[v_i h_j]_{model}$$

$$\frac{\delta logP(v^k)}{\delta a_i} = v_i^{(k)} - \mathbb{E}[v_i^{(k)}]_{model}$$

$$\frac{\delta logP(v^k)}{\delta b_j} = \mathbb{E}[h_j]_{data} - \mathbb{E}[h_j]_{model}$$

Nous pouvons après mettre à jour les paramètres en utilisant les règles suivantes :

$$\Delta w_{ij} = \eta(\mathbb{E}[v_i^{(k)} h_j]_{data} - \mathbb{E}[v_i h_j]_{model})$$

$$\Delta a_i = \eta(v_i^{(k)} - \mathbb{E}[v_i^{(k)}]_{model})$$

$$\Delta b_j = \eta(\mathbb{E}[h_j]_{data} - \mathbb{E}[h_j]_{model})$$

En réitérant la procédure on effectue une descente de gradient. Dans la génération de musique cette approche est utilisée pour harmoniser les notes jouées ensemble.

4.4.3 Gibbs sampling dans les RBMs

Bien qu'il soit assez simple de déterminer $P(h|v)$ dans une RBM, calculer P(v) ou P(v,h) ne peut pas se faire de manière exacte et se fait donc par Gibbs sampling (forme de Markov Chain Monte Carlo), où l'on prend avantage du fait que les tirages $P(h|v)$ et $P(v|h)$ sont facile. On obtient donc :

$$v^{(1)} \sim \text{ exemple d'apprentissage}$$

$$h^{(1)} \sim P(h|v^{(1)})$$

$$v^{(2)} \sim P(v|h^{(1)})$$

$$h^{(2)} \sim P(h|v^{(2)})$$

$$\ldots$$

Pour la visualisation de données générées à l'étape k, il est plus raisonnable d'utiliser les espérances (i.e. $\mathbb{E}[v_i^{(k)}|h^{(k-1)}] = P(v_i^{(k)} = 1|h^{(k-1)})$) qui sont moins bruitées que les échantillons $v^{(k)}$ eux-mêmes.

4.4.4 Divergence Contrastive - CD

Pour calculer les approximation de $\mathbb{E}[v, h]$ (i.e. pour obtenir des "exemples négatifs" pour la "partie négative" du gradient), il suffit de faire une courte chaîne de Gibbs (de k étapes) où l'on

commence avec un exemple d'apprentissage. Cet algorithme a été proposé par J.Hinton dans [7].

$v \Leftarrow v(t)$
$h \Leftarrow W * v + b$
$phasePositive \Leftarrow -h * v^T$
for *i allant de 1 à k* **do**
  $v \Leftarrow W^T * h(v) + a$
  $h \Leftarrow W * v + b$
end
$phaseNegative \Leftarrow -h * v^T$
$gradient \Leftarrow phasePositive - phaseNegative$
$W \Leftarrow W + \eta * gradient$
$a \Leftarrow a + \eta * (v(t) - v)$
$b \Leftarrow b + \eta * (h(v(t)) - h)$

Algorithm 1: CD-k alrorithm

Ce qui est étonnant avec cet algorithme qui est assez simple c'est que même avec k=1, on obtient des résultats qui fonctionnent bien dans le sens qu'elles extraient des bonnes caractéristiques des entrées.
Il est possible de montrer que CD-1 est très proche de l'entraînement d'un auto-encodeur par minimisation de l'erreur de reconstruction, et on voit que l'erreur de reconstruction diminue de manière monotone pendant l'entraînement par CD-1.
De plus, CD-k tends vers le vrai gradient (en espérance) quand k devient grand, mais à ce moment on multiplie le temps de calcul par k. En pratique le petit nombre de k est suffisant ou on peut augmenter k au fur et à mesure de l'exécution d'un algorithme.

5 Google Magenta

Magenta est un projet open source dédié à la création musicale conçu par des ingénieurs de la Google Brain Team qui a été présenté au début du mois de Juin 2016. Ce projet tente de répondre à la question suivante : peut-on utiliser une machine pour créer de l'art de manière aussi convaincante que l'Homme ? Si oui, comment ? Sinon, pourquoi ?

Le machine learning a déjà été largement utilisé pour interpréter du contenu, que ce soit la reconnaissance vocale, la traduction, la reconnaissance d'images ou encore la compréhension d'écriture manuscrite. Le but de Magenta est d'explorer l'autre face : apprendre et créer de l'art notamment de la musique.

La difficulté réside dans le fait que l'art est dynamique, il ne suffit pas d'apprendre des chansons et de les reproduire car dans ce cas les différentes reproductions seront plus ou moins similaires. Or, pour attirer l'attention du public, les artistes essaient de jouer sur l'effet de surprise. Ainsi, l'album d'un artiste 10 ans auparavant peut être très différent d'un album actuel. Alors comment représenter l'effet de surprise ou l'évolution d'un style musical dans un algorithme de machine learning ? C'est le principal challenge de Magenta. Un modèle prometteur, le *Show, Attend and tell Model* a été développé par Xu et al. du laboratoire MILA de Montréal. Ce modèle est utilisé pour générer des descriptifs d'images.

Bien que magenta soit actuellement en mesure de collecter un nombre incommensurable de données, il manque encore de talent puisque sa première composition [1] est loin de faire l'unani-

1. https ://cdn2.vox-cdn.com/uploads/chorus_asset/file/6577761/Google_-_Magenta_music_sample.0.mp3

mité mais elle semble encourageante pour les ingénieurs de Google.

Au-delà de la création, les chercheurs de Google Brain espèrent prochainement constituer une communauté qui transmettra ses connaissances à la machine pour l'aider à évoluer.

6 Expériences

6.1 Reproduction d'expérience LSTM avec des fichiers MIDI

6.1.1 Définition du problème

L'expérience consiste à générer une pièce musicale à partir d'un code open source [2]. Le projet est implémenté en Python et utilise les librairies *theano*, *unroll*, *numpy*, *music21* pour le traitement des fichiers MIDI, *keras* pour l'implémentation du modèle. Les réseaux LSTM sont utilisés pour apprendre et reproduire une série de notes à partir d'un ensemble d'apprentissage constitué de 229 chorales de Jean-Sébastien Bach. Pour comparer l'efficacité de l'apprentissage, nous avons généré plusieurs fichiers à différentes périodes de l'apprentissage.

Durant la phase d'apprentissage, le réseau apprend plusieurs pièces de musique avec des réseaux LSTM. Le but est de prédire la probabilité pour chaque note d'être jouée à un instant t. Cette tâche est donc similaire à un problème de classification, une note est soit *on* ou soit *off*. Les sorties aux temps $t' \geq t$ sont ensuite prédites par rapport aux précédentes prédictions. Tous les neurones dont la valeur de la fonction d'activation est supérieure à un certain seuil sont considérés comme *on*, les notes associées à ces neurones sont donc jouées. La fonction d'activation utilisée est la fonction tangente hyperbolique.

6.1.2 Résultats

Nous avons généré deux partitions, la première étant générée au début de la phase d'apprentissage, la deuxième est créée après 5 heures d'apprentissage :
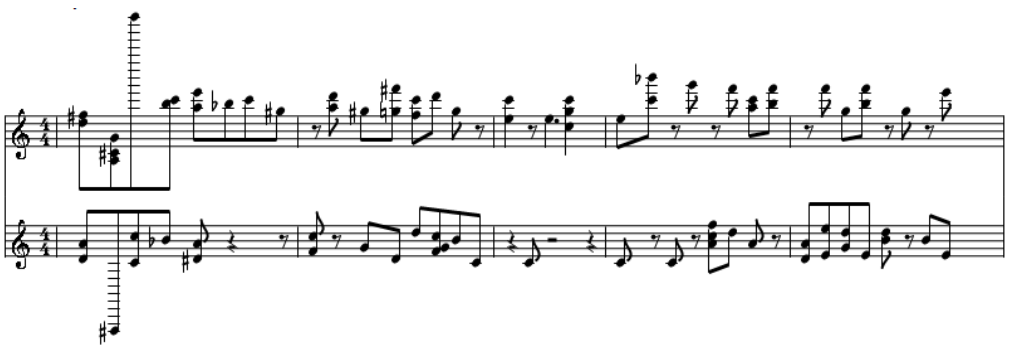


Figure 12 – *Début de la partition créée en début d'apprentissage*

2. https ://github.com/jamong/music-rnn
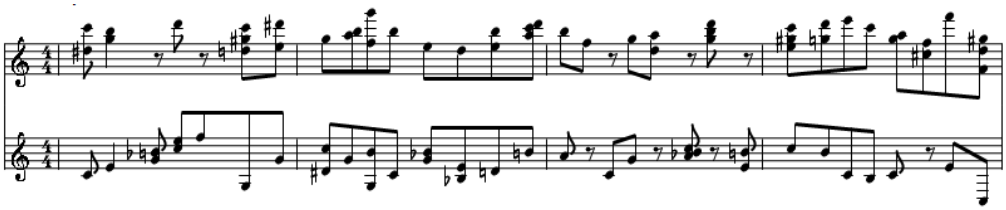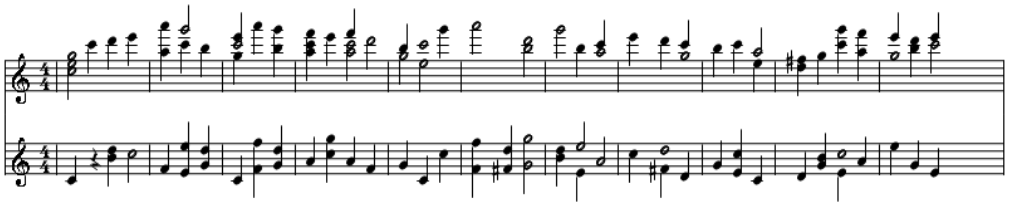


Figure 13 – *Début de la partition créée après 5 heures d'apprentissage*

En comparaison voici le début de la partition d'un morceau de l'ensemble d'apprentissage :



6.2 Reproduction d'expérience LSTM avec des fichiers MP3

Eck and Schmidhuber [4] ont utilisé les LSTMs pour analyser la structure de musiques Blues à l'aide de vecteurs pour modéliser chaque note d'une séquence. Chung et al. [2] ont eux utilisé les GRUs pour modéliser des musiques polyphoniques à base de datasets MIDI et ont trouvé leurs performances comparables à celles des LSTMs.

Bien que les travaux cités ci-dessus ont été appliqués à la génération de musique , aucun n'a essayé la génération récurrente de musique directement à partir des signaux audio bruts d'une musique. En ce sens, le projet de A. Nayebi et M. Vitelli [11] est entièrement nouveau puisque le réseau qu'ils ont créé essaye de produire de la musique en se basant directement sur les signaux audio bruts [3].

6.2.1 Définition du problème

Le problème de génération de musique peut être défini comme suit : soit un ensemble de vecteurs $X_t, X_{t-1}, ..., X_0$ représentant les ondes audio aux temps $t, t-1, ..., 0$, le but est d'essayer de générer le vecteur X_{t+1} représentant l'onde audio au temps $t+1$.

Il s'agit donc d'une tâche de modélisation de séquence dans laquelle le but est d'essayer d'estimer la séquence suivante, cette dernière étant conditionnée par les précédentes. Dans le cas de ce problème, les entrées sont naturellement continues et on peut donc formuler le problème d'estimer X_{t+1} comme une tâche de régression.

Cette régression est effectuée en utilisant un réseau de neurones récurrent de type LSTM dans lequel chaque calcul de feed-forward donne une nouvelle estimation $\hat{X}_{t+1}$ représentant la meilleure estimation pour X_{t+1}. En entrainant le réseau à partir d'un ensemble d'apprentissage, la fonction de perte L2 est donnée par :

$$l(\theta) \equiv \frac{1}{T}\sum_{t}(X_t - \hat{X}_t)^2$$

où T est le temps t actuel.

3. https ://github.com/MattVitelli/GRUV

6.2.2 Représentation du modèle

L'un des avantages de formuler le problème comme une tâche de régression à partir de signaux audio est que les entrées du réseau de neurones peuvent être n'importe quelle sorte de données audio. Ainsi, toutes les données du dataset d'apprentissage sont représentées sous forme de samples audio extraits depuis des musiques au format WAV.

Afin de simplifier l'apprentissage, toutes les musiques sont converties en mono et échantillonnées à 44,1 Khz. La complexité des données audio s'en trouve alors réduite. Le taux d'échantillonnage fixé à 44,1 Khz permet d'utiliser des tailles d'entrée en de sortie constantes pour l'apprentissage et la génération.

Bien que les données audio soient typiquement représentées sous la forme d'un signal dans le domaine temporel, il est généralement plus significatif d'analyser ces données audio dans le domaine fréquentiel. Après avoir lu les samples audio, ces derniers sont divisés en blocs de taille N et chaque bloc est converti en sa représentation dans le domaine fréquentiel en utilisant l'algorithme de Transformation de Fourier discrète (DFT).

Cette transformation fournit un vecteur de N nombres réels et un vecteur de N nombres imaginaires (qui constituent à eux deux les phases et les magnitudes du signal audio dans le domaine temporel). Ces deux vecteurs sont concaténés ensemble pour créer un vecteur de taille $2N$ qui est utilisé comme représentation du modèle du réseau.

6.2.3 Génération

Après la phase d'entraînement du réseau, de nouvelles compositions musicales peuvent être générées. On initialise d'abord le réseau en fournissant une séquence de "seeds" $\mathcal{S}_{seed} = S_t, S_{t-1}, ..., S_0$ représentant les signaux audio aux temps $t, t-1, ..., 0$. La première étape consiste à effectuer un calcul de feed-forward à partir ce cette séquence. Une fois que le calcul du feed-forward est terminé, on obtient alors un nouvel ensemble de vecteurs $R_{t+1}, R_t, ..., R_1$, représentant les signaux audio prédits aux intervalles de temps $t+1, t, ..., 1$. Cet ensemble de vecteurs est appelé séquence générée (notée $\mathcal{S}_{gen}$).

Dans la deuxième étape, un autre calcul de feed-forward est effectué avec comme entrée $\mathcal{S}_{gen}$. La dernière prédiction du réseau est alors ajoutée à $\mathcal{S}_{gen}$. Par exemple, en supposant que $\mathcal{S}_{gen}$ était $R_{t+k}, R_{t+k-1}, ..., R_1$, on ajouterait R_{t+k+1} à $\mathcal{S}_{gen}$ à la fin de la deuxième étape.

La deuxième étape est ensuite effectuée en boucle jusqu'à ce que $\mathcal{S}_{gen}$ soit suffisamment longue (paramètre spécifié par l'utilisateur). Il est important de noter qu'il existe trois paramètres principaux pour l'algorithme de génération : la taille de $\mathcal{S}_{seed}$, le contenu de $\mathcal{S}_{seed}$ et le nombre d'itérations exécutés pour la prédiction de séquence.

6.2.4 Résultats

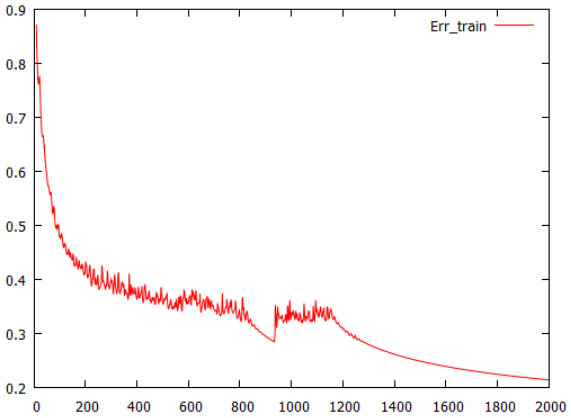


Figure 14 – *Erreur d'apprentissage du LSTM pour 2,000 epochs sur la musique "William Tell Overture" du film* Orange mécanique *(arrangée par Wendy Carlos) avec 3072 dimensions cachées.*

Les LSTMs instanciés pour le problème de génération de musique WAV étant particulièrement grands (au moins 22050 entrées/sorties, au moins 2048 dimensions cachées ainsi que l'ensemble des séquences générées doivent tenir en mémoire), et du fait de la puissance de calcul trop peu élevée, mais surtout d'une quantité de mémoire bien trop faible sur les ordinateurs disponibles, les reproductions d'expériences sur les réseaux LSTMs - s'ils n'ont pas échoué - n'ont pas pu être réalisés dans de bonnes conditions (l'expérience originale, bien qu'effectuée à Stanford sur les clusters Rye, a également manqué de mémoire).

Néanmoins, certaines expériences ont pu fonctionner. Par exemple, en utilisant des datasets plus petits (plusieurs musiques d'une dizaine de secondes). Cependant, peu d'entre-elles ont donné des résultats concluants. La plus concluante étant celle où le dataset ne contient qu'une seule musique. Dans ce cas particulier, le réseau doit pouvoir, au moins, régénérer la musique de départ. On peut observer sur la figure 14 l'évolution de l'erreur d'apprentissage sur ce dataset.

6.3 Reproduction d'expérience RNN RBM avec des fichiers MIDI

6.3.1 Représentation du modèle

RNN RBM est un modèle basé sur le concept de l'énergie pour l'estimation de densité des séquences temporelles où le vecteur $v^{(t)}$ à chaque instant t peut être à plusieurs dimensions. Cela permet de décrire des distributions conditionnelles multimodales de $P(v^{(t)}|\mathcal{A}^{(t)})$, où $\mathcal{A}^{(t)} \equiv \{v_\tau|\tau < t\}$ qui définit l'histoire de séquence à l'instant t, via une série de RBMs conditionnelles (une pour chaque instant) dont les paramètres $b_v^{(t)}, b_h^{(t)}$ dépendent de sorties de RNN déterministe avec des unités cachées $u^{(t)}$:

$$b_v^{(t)} = b_v + W_{uv} * u^{(t-1)} \tag{19}$$

$$b_h^{(t)} = b_h + W_{uh} * u^{(t-1)} \tag{20}$$

Et la relation de récurrence RNN d'une seule couche est définie par :

$$u^{(t)} = tanh(b_u + W_{uu} * u^{(t-1)} + W_{vu} * u^{(t)}) \tag{21}$$

On peut donc visualiser le modèle avec l'image suivante :

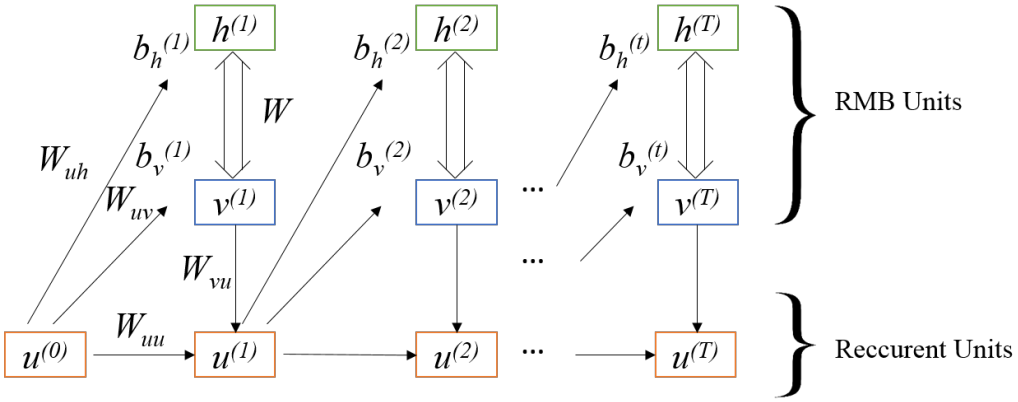


Figure 15 – *Architecture de RNN RBM*

La distribution de probabilités marginales est donnée par la somme sur tous les instants t dans une séquence donnée :

$$P(v^{(t)}) = \sum_{t=1}^{T} P(v^{(t)}|\mathcal{A}^{(t)}) \tag{22}$$

Où la partie droite est la probabilité marginalisée du t^{th} RBM.

6.3.2 Génération

Dans ce projet nous avons reproduit une partie de l'expérience citée dans l'article de N.Boulanger-Lewandowski [1]. [4].

Le projet est implémenté en Python et utilise les bibliothèques :

— Theano
— numpy
— Python MIDI package

Le modèle contient deux parties :

— Classe pour générer RBM avec Gubbs sampling
— Classe pour générer RNN en deux modes : apprentissage et génération

Dans la phase d'apprentissage l'algorithme apprend les paramètres $W_{uu}, W_{uv}, W_{vu}, W_{uh}, b_v, b_h$ pour les utiliser après dans la phase de génération.

L'apprentissage est effectué avec le descente du gradient stochastique mini batch. Comme "training set" nous avons utilisé "Nottingham Database of folk tunes music" qui contient environ 1000 chansons de longueur variable dans le format .midi.

6.3.3 Résultats

Après la phase d'apprentissage le programme peut générer des fichiers .midi de la taille donnée.

Lorsque la version implémentée est très simple, les résultats produits ne sont pas pertinents, cependant on peut remarquer que l'algorithme a réussit à répartir les partitions basse et mélodie et il sait produire les accords harmoniques. Les meilleurs résultats peuvent être obtenus par la considération des aspects suivants :

— Prétraitement de données : transposer les séquences dans une tonalité commune
— Prétraitement de paramètres : initialiser les paramètres $W, b_v, b_h, W_{uv}, W_{uu}, W_{vu}, b_u$ avec les techniques intelligentes.
— Techniques d'optimisation pour une descente du gradient.

4. http ://deeplearning.net/tutorial/rnnrbm.html

— Recherche d'hyperparamètres : le taux d'apprentissage, la taille du mini batch, le nombre d'unités cachées (récurrentes et RBM), la longueur de la chaîne de Gibbs k.
— Moyen d'intégrer l'information de plusieurs itérations précédentes pour mieux connaître le contexte.

Certaines de ces améliorations ont été proposées et implémentées par N.Boulanger-Lewandowski [1]. Cependant il reste beaucoup de choses à faire et à développer.

7 Conclusion

Nous avons présenté et mis en place deux modèles de composition de musique MIDI basés sur sur les réseaux de neurones de type LSTM et RBM ainsi qu'un modèle de composition de musique WAV basé sur les réseaux de neurones LSTM. Suite à leur phase d'apprentissage à partir de différents morceaux, ces derniers ont été capables d'en générer de nouveaux. De plus, le projet a démontré que la génération de musique par des réseaux de neurones récurrents à partir d'un signal audio est possible.

Bien que les musiques générées soient plutôt bonnes, il reste cependant difficile, à l'oreille, de bien cerner le caractère "Long Term", caractéristique des LSTMs. De plus les musiques générées semblent parfois se dégrader vers la fin (leur structure devient plus simple, moins recherchée). Le travail à effectuer dans le futur sera (1) d'exécuter les algorithmes sur les mêmes jeux de données mais en jouant sur les différents paramètres du réseau (la prédiction toutes les n itérations par exemple), (2) d'exécuter ces mêmes algorithmes sur des jeux de données plus importants ou de styles différents.

Références

[1] Nicolas Boulanger-Lewandowski, Yoshua Bengio, and Pascal Vincent. Modeling temporal dependencies in high-dimensional sequences : Application to polyphonic music generation and transcription. *arXiv preprint arXiv :1206.6392*, 2012.

[2] Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555, 2014.

[3] Douglas Eck and Juergen Schmidhuber. A first look at music composition using lstm recurrent neural networks. Technical report, 2002.

[4] Douglas Eck and Jürgen Schmidhuber. Learning the long-term structure of the blues. In *Proceedings of the International Conference on Artificial Neural Networks*, ICANN '02, pages 284–289, London, UK, UK, 2002. Springer-Verlag.

[5] Felix A. Gers and Jürgen Schmidhuber. Recurrent nets that time and count. In *IJCNN (3)*, pages 189–194, 2000.

[6] Klaus Greff, Rupesh Kumar Srivastava, Jan Koutník, Bas R. Steunebrink, and Jürgen Schmidhuber. LSTM : A search space odyssey. *CoRR*, abs/1503.04069, 2015.

[7] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7) :1527–1554, 2006.

[8] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8) :1735–1780, November 1997.

[9] I-Ting Liu and Bhiksha Ramakrishnan. Bach in 2014 : Music composition with recurrent neural network. *CoRR*, abs/1412.3191, 2014.

[10] Michael C. Mozer. Musical networks. chapter Neural Network Music Composition by Prediction : Exploring the Benefits of Psychoacoustic Constraints and Multi-scale Processing, pages 227–260. MIT Press, Cambridge, MA, USA, 1999.

[11] Aran Nayebi and Matt Vitelli. GRUV : Algorithmic music generation using recurrent neural networks. 2015.

[12] P. Smolensky. Information processing in dynamical systems : Foundations of harmony theory. In *Parallel distributed processing : Explorations in the microstructure of cognition*, pages 194–281–. MIT Press, Cambridge, MA, 1986.