L'IA musicienne : Étude des réseaux récurrents pour la génération de musique

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

6 janvier 2017

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 [1]. La raison est qu'ils possèdent une mémoire à court terme. Dans ce papier nous nous intéressons à un type de réseau de neurones récurrent : le réseau Long Short Term Memory (LSTM). Dans un premier temps nous étudierons le fonctionnement de ce réseau de neurones puis nous montrerons comment utiliser ce réseau de neurones pour générer automatiquement de la musique. Enfin nous effectuerons une expérience dans laquelle nous générons automatiquement des fichiers en format MIDI.

Automated music generation turned out to be a fascinating problem than challenge minds of many scientists around the globe since it could prove the creative capacity of artificial intelligence. In the past numerous approaches were proposed to address the problem, but the ones using recurrent neural networks proved to be the most successful among them. Ability to use information from previous iterations enabled to improve significantly the quality of these algorithms. In this paper we study different types of recurrent neural networks commonly used for the automated music generation, namely : (i) Long Short Term Memory (LSTM) and Restricted Boltzmann Machine(ii). First of all, we will discover the structure and functionality of discussed algorithms and then we will show how to apply them to music composition. Finally, we will perform a computational experiment described in the paper of D. Johnson [2] in order to generate the music in the format of MIDI files.

1 Introduction

Dans ce papier nous nous intéresserons uniquement à la composition automatique de musique qui est probablement la forme de média la plus largement consommée. 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.

Nénamoins on peut citer Will.i.am, leader du groupe Black Eyed Peas (au magazine Gala, en juillet 2015) : « Bientôt on ne fera plus de musique. Ou du moins, la musique ne sera plus faite par des musiciens [...]. Il y aura bien une intelligence artificielle qui sera meilleure pianiste que Mozart, ou meilleur rappeur que Kanye West ou Jay Z. Je ne dis pas que c'est ce que je souhaite voir arriver, mais malheureusement, c'est ce qui est en train de se passer. C'est la terrible réalité. La révolution technologique est en

marche. ≫

Une façon très simple pour générer de la musique est d'utiliser les RNNs. Après avoir été entraîné à prédire les enchainements de notes les plus probables (de t à t+1), le réseau peut alors créer et fournir ses propres compositions musicales [3] (en utilisant ses sorties comme entrée par exemple). Cependant, et bien qu'ils puissent posséder une mémoire à court terme, les RNNs ne sont en pratique pas idéaux pour ce genre de tâche [1] car, d'après Mozer (1994), "les pièces produites ne sont pas musicalement cohérentes et manquent de style et de rythme". 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.

Les LSTMs, qui par défnition sont capables de travailler sur des évènements plus espacés les uns des autres, fournissent donc de meilleurs méchanismes pour la génération de musique.

De la même façon, les performances des GRUs (proposés en 2014) sur la génération de musique ont été prouvés similaires aux LSTMs [4].

Nous utiliserons donc des réseaux de neurones récurrents, et en particulier ceux possédant une mémoire à long terme : les LSTMs. Nous expliquerons en détail le fonctionnement de ce type de réseau de neurones.

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 [5]. Cette dernière aura pour but de déterminer la probabilité que chaque note soit 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.

L'expérience réalisée consiste à produire automatiquement une série de notes grâce à un algorithme qui utilise des LSTMs.

2 Etats de l'art

2.1 Les LSTMs pour la composition automatique de musique

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 [1].

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 [6]. Les LSTMs ont une mémoire persistente, 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 ≫.

2.1.1 Architecture

La Figure 1 représente l'architecture globale d'un réseau LSTM traditionnel [7] :

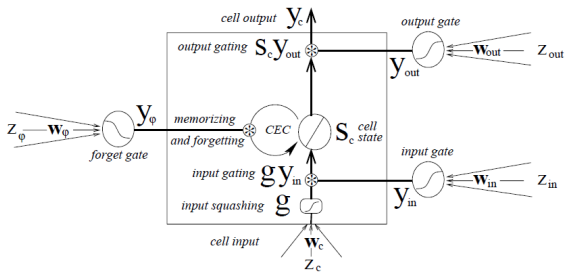


Figure 1 – *Architecture d'un réseau LSTM*

L'élément basique d'un réseau LSTM est le bloc mémoire qui contient une ou plusieurs cellules mémoire (S dans le schéma) et 3 portes [7] :

- l'≪ input gate ≫ qui détermine quelles informations il faut stocker en mémoire,
- la ≪ forget gate ≫ qui détermine combien de temps il faut stocker ces informations,
- l'≪ output gate ≫ qui détermine quand ces informations seront lues.

L'output y_c de la cellule mémoire S_c est calculé à partir de l'état courant de cette cellule et 4 entrées : z_c, l'input de la cellule elle-même, z_{in}, z_ϕ et z_{out} qui sont les entrées des 3 portes vues précédemment. [7]

2.1.2 Apprentissage

L'apprentissage se déroule en 2 étapes :

— le forward pass où l'on calcule l'activation des portes,
— le backward pass (ou backpropagation) où l'on met à jour les poids de façon à minimiser l'erreur en sortie pour chaque neurone.

On commence par calculer l'input de la cellule grâce à la formule suivante [7] :

$$z_c(t) = \Sigma w_c.y(t-1) \quad (1)$$

où w représente le vecteur des poids.

On applique ensuite à (1) une fonction g sigmoïde. Le résultat est multiplié par l'activation de l'≪ input gate ≫ obtenue en appliquant une fonction sigmoïde f_{in} (à valeurs dans [0, 1]) à l'entrée z_{in} [7] :

$$y_{in}(t) = f_{in}(z_{in}(t), z_{in}(t) = \Sigma w_{in}.y(t-1) \quad (2)$$

L'activation y_{in} détermine combien d'informations on laisse entrer. Plus y_{in} tend vers 0 (la porte est ≪ fermée ≫) et plus les inputs sont inappropriés donc on ne les laisse pas passer d'informations, plus y_{in} tend vers 1 (la porte est ouverte) et plus on laisse entrer les informations dans la mémoire.

L'activation de la ≪ forget gate ≫ est obtenue grâce à la formule suivante [7] :

$$y_\phi(t) = f_\phi(z_\phi(t), z_\phi(t) = \Sigma w_\phi.y(t-1) \quad (3)$$

où f_ϕ est une fonction sigmoïde à valeurs dans [0, 1].

Le nouvel état de la cellule est obtenu en ajoutant les input y_{in} et z_c (appliqué à une fonction sigmoïde) à l'état de la cellule à l'instant t-1 multiplié par l'activation de la ≪ forget gate ≫ [7] :

$$S_c(t) = y_\phi(t).S_c(t-1) + y_{in}(t).g(z_c(t) \quad (4)$$

Comme l'≪ input gate ≫ détermine quelles informations stocker en mémoire, la ≪ forget gate ≫ détermine combien de temps les informations restent en mémoire. Les informations obsolètes sont donc effacées, ce qui empêche la mémoire d'exploser et rend donc possible le stockage de données récentes. [7]

L'output y_c de la cellule est obtenu en multipliant l'état de la cellule S_c par l'activation y_{out} de l'≪ output gate ≫ [7] :

$$y_c(t) = y_{out}(t).S_c(t) \quad (5)$$

L'≪ output gate ≫ permet de contrôler la lecture de la mémoire. Son activation est obtenue en appliquant une fonction sigmoïde (à valeurs dans [0, 1] à l'entrée z_{out} [7] :

$$y_{out}(t) = f_{out}(z_{out}(t), z_{out}(t) = \Sigma w_{out} y(t-1) \quad (6)$$

Ce qui clôt le forward pass.

Après le forward pass, l'output du réseau est comparé à l'output désiré et on calcule l'erreur pour chaque neurone de l'output layer. Les erreurs sont ensuite rétropropagées dans tous les neurones du réseau de façon à ce que chaque neurone ait une erreur qui lui soit propre.

La backpropagation utilise ces erreurs pour calculer le gradient d'une fonction de coût par rapport aux poids du réseau. Ce gradient est ensuite optimisé puis est utilisé pour mettre à jour les poids du réseau afin de minimiser la fonction de coût.

La fonction de coût est une fonction qui en entrée reçoit l'erreur de chaque neurone et retourne un nombre réel.

2.2 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 [5].

2.2.1 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).

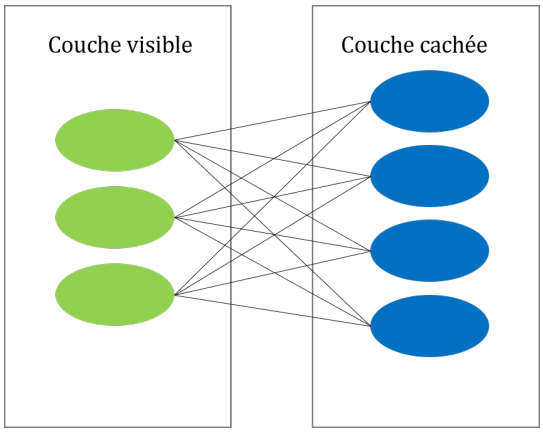


Figure 2 – *Les connections entre les couches de neurones*

Une énergie d'activation pour RBM est donnée par :

$$E = -\sum_{i,j} w_{ij} x_i h_j - \sum_i b_i x_i - \sum_j c_j h_j \quad (7)$$

Avec :
- $w_{i,j}$ la matrice de poids entre le neurone i et le neurone j
- x_i est l'état, $x_i \in 0, 1$, du neurone i.
- b_i et c_j sont respectivement les bias des neurones x_i et h_j.

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

$$P(x_i, h_j) = \frac{1}{Z} e^{-E(x_i, h_j)} \quad (8)$$

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

2.2.2 Apprentissage

Dans la phase d'entraînement on cherche à minimiser le negative log-likelihood, donné par la formule suivante :

$$\mathcal{L}_{gen}(\mathcal{D}_{train}) = \frac{1}{n} \sum_{x_t \in \mathcal{D}_{train}} -log(p(x_t)) \quad (9)$$

La dérivée du log-vraisemblance alors donnée par :

$$\frac{\delta(-log(p(x_t)))}{\delta\theta} = \mathop{\mathbb{E}}_{h}[\frac{\delta\, \mathbb{E}(x_t, h)}{\delta\theta}|x_t] - \mathop{\mathbb{E}}_{x,h}[\frac{\delta\, \mathbb{E}(x_t, h)}{\delta\theta}] \quad (10)$$

Avec :
- θ les variables du système (les poids ou le biais)
- $E_{x,y}$ l'espérance mathématique pour les variables aléatoires x et y
- $x^{(t)}$ une valeur du jeu de données
- $E(x, h)$ l'énergie du système

Cette expresion contient deux termes. Le premier est appelé la phase positive et le deuxième est appelé la phase négative. La phase positive se calcule pour les données originales et la phase négative avec la distribution qu'on a obtenu (reconstruction de données).

On peut donc obtenir :

$$\mathop{\mathbb{E}}_{h}[\frac{\delta\, \mathbb{E}(x_t, h)}{\delta W_{ij}}|x_t] = -h(x^{(t)}) * (x^{(t)})^T \quad (11)$$

L'état de la couche cachée est défini par :

$$h(x) = sigm(W * x + b) \quad (12)$$

La reconstruction de l'entrée $x^{(t)}$ est effectuée comme suit :

$$x_{rec} = W^T * h(x) + c \quad (13)$$

avec c le bias de la couche cachée. 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.

3 Expérience

3.1 Setting

Pour l'expérience, nous avons utilisé music-rnn [1] (créé par Jamonglab) qui est un outil développé en Python permettant la composition automatique de musique à partir d'un jeu de données. Cette expérience permet de produire de la musique sous la forme de fichiers MIDI grâce aux réseaux LSTMs. L'apprentissage est effectué sur des fichiers MIDI situés dans le dossier data-for-train/waltzes/.

Nous avons effectué deux expériences :
- la première avec des compositions de piano,
- la deuxième avec des compositions de Mozart (comportant différents instruments de musique).

3.2 Résultats

Les fichiers MIDI produits sont plutôt harmonieux et ressemblent aux morceaux utilisés pour l'apprentissage

4 Conclusion

Nous avons présenté et mis en place un modèle de composition de musique MIDI basé sur sur les réseaux de neurones de type LSTM. Suite à leur phase d'apprentissage à partir de différents morceaux, ces derniers ont été capable d'en générer de nouveaux.

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 génères 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.

1. Code source : github.com/jamong/music-rnn

Références

[1] D. Eck and J. Schmidhuber, *A First Look at Music Composition using LSTM Recurrent Neural Networks*. 2002.

[2] D. Johnson, *Composing Music With Recurrent Neural Networks*. 2015.

[3] M. C. Mozer, *Neural network music composition by prediction : Exploring the benefits of psychophysical constraints and multiscale processing*. 1994.

[4] J. Chung, C. Gulcehre, K. Cho and Y. Bengio, *Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling*. 2014.

[5] P. Smolensky, *"Chapter 6 : Information Processing in Dynamical Systems : Foundations of Harmony Theory"* in *Parallel Distributed Processing : Explorations in the Microstructure of Cognition, Volume 1 : Foundations*. 1986.

[6] S. Hochreiter and J. Schmidhuber, *Long short-term memory*. 1997.

[7] Felix A. Gers, Nicol N. Schraudolph and Jürgen Schmidhuber, *Learning Precise Timing with LSTM Recurrent Networks*. 2002.

[8] M. C. Mozer, *Induction of multiscale temporal structure*. 1992.

[9] Z. Wu and S. King, *Investigating gated recurrent neural networks for speech synthesis*. 2016.

[10] A. Graves, *Generating Sequences With Recurrent Neural Networks*. 2013.

[11] Q. Lyu, Z. Wu and J. Zhu, *Polyphonic Music Modelling with LSTM-RTRBM*. 2015.

[12] N. Boulanger-Lewandowski, Y. Bengio, and P. Vincent, *Modeling temporal dependencies in high-dimensional sequences : Application to polyphonic music generation and transcription*. 2012.