INF8225 – Intelligence artificielle : technique probabilistes et d'apprentissage
Génération d'accords dans une partition de musique

Téo Marcin **Mathias Gonin** **Victor Petit** **Julien Hage**
Polytechnique Montréal
{teo.marcin, mathias.gonin, victor.petit, julien.hage}@polymtl.ca

Abstract

Notre travail s'est porté sur la génération d'accords à partir d'une mélodie en utilisant l'Apprentissage Machine. Nous utilisons comme données des fichiers MusicXML, format basé sur XML permettant de manipuler des partitions de musique. Ce problème est peu traité en Intelligence Artificielle et nous nous sommes donc inspirés des modèles de Natural Language Processing (NLP) afin de traduire les dépendances temporelles entre les notes, mais aussi entre les accords joués de chaque mesure. Nos résultats ont ensuite été mesurés à l'aide d'un ensemble de test, ainsi qu'en écoutant les audios générés grâce à nos scripts et au logiciel MuseScore.

1 Introduction

La génération de musique est un sujet de plus en plus couvert par la recherche en Apprentissage Machine. En effet des technologies récentes comme les GANs (Generative Adversarial Networks) ont permis des résultats très intéressants comme MuseGan [Dong et al.(2017)Dong, Hsiao, Yang, and Yang] ou SynthGAN de Google [Engel et al.(2019)Engel, Krishna Agrawal, Chen, Gulrajani, Donahue, and Roberts]. Mais si certains articles s'attachent à la génération de mélodies, c'est à la génération d'accords à partir de mélodies que nous nous intéressons. En effet, ils sont à la base des accompagnements d'une mélodie, mais peuvent cependant représenter des difficultés pour les musiciens novices, notamment pour ceux ayant des connaissances limitées en solfège. Ainsi certains articles se sont penchés sur le sujet et ont tenté une approche par mesures, c'est-à-dire qu'un accord est généré pour une mesure uniquement à l'aide des notes de cette mesure. L'article [Chen et al.(2015)Chen, Qi, and Zhou] propose d'appliquer plusieurs modèles basiques et plus avancés à cette tâche: la régression logistique, les réseaux bayésiens naïfs, les SVM, les Random Forests, le Boosting ainsi que les HMM. Les SVM Radial, Random Forest et Boosting donnent les meilleurs résultats pour la prédiction d'accord sur une seule mesure avec une précision dépassant 60% mais en restreignant à 7 classes d'accords. Cependant, la musique étant une information séquentielle, certains auteurs ont utilisé les dépendances temporelles entre les notes et réalisé un modèle séquentiel. Dans l'article [Lim et al.(2017)Lim, Rhyu, and Lee] par exemple, la génération des accords à partir d'une mélodie est réalisée par l'intermédiaire d'un réseau BiLSTM. Une couche TimeDistributed est appliquée à chaque mesure, les auteurs obtiennent aussi les meilleurs résultats (précision de 0.5) avec une séquence de 4 mesures. Nous avons ainsi voulu partir de cet article pour obtenir un premier modèle Seq2Vec et avons donc réalisé une architecture de type BiLSTM. Toutefois, dans la composition musicale, les musiciens choisissent généralement l'accord d'une mesure selon les notes présentes dans la mesure ainsi que les accords utilisés dans les mesures précédentes. En effet, deux mesures peuvent avoir exactement les mêmes notes dans le même ordre - ou même pas de note du tout (silences) - et correspondre à des accords différents. Nous avons alors décidé d'utiliser les dépendances entre les mesures, elles-mêmes séquences de notes, en s'inspirant du modèle Seq2Seq très utilisé en NLP [Sutskever et al.(2014)Sutskever, Vinyals, and Le]. Les résultats ont été mesurés sur un ensemble de test et notre deuxième approche Seq2Seq s'est révélée plus précise que l'approche BiLSTM. Mais la précision ne semble pas être une mesure suffisante de la cohérence d'une musique. En effet, cela dépend de la justesse - c'est-à-dire si l'accord est dans la gamme de la mélodie -, de la diversité des accords -car une musique avec toujours le même accord juste semblera vide d'émotions -, et de plusieurs autres mesures difficiles à décrire. Nous avons donc créer un script qui génère les accords en audio à la guitare en jouant la mélodie en parallèle afin de pouvoir juger à l'oreille de la progression générée. Ainsi nos tests ont aussi une partie auditive qui bien que subjective, nous parait toute aussi importante. Ce rapport a donc pour but d'expliquer nos démarches et nos résultats: nous verrons dans un premier temps l'extraction et le traitement des données, puis nous détaillerons l'architecture des modèles implémentés. Enfin nous analyserons les résultats et détaillerons notre génération de fichiers audios avant de conclure sur les potentielles améliorations de notre travail.

2 Données

Nous disposons de 2236 partitions de musique venant de la base de données de Wikifonia. Ces données sont sous la forme de fichiers MusicXML (.mxl). Ce sont des partitions classiques qui présentent en plus des accords accompagnant la mélodie. Elles peuvent être ouvertes et éditées

avec le logiciel MuseScore par exemple. La plupart de ces musiques sont des musiques Pop occidentales, mais il y aussi du jazz, du classique et quelques musiques du monde comme des musiques chinoises. Certains fichiers .mxl sont présents comme exemples dans le lien du code.

2.1 Extraction des données

Afin d'extraire les données, nous avons converti les fichiers MXL en XML, puis de XML en CSV. En effet, le format MXL est en fait une archive zip contenant un fichier META-INF (contenant un container.xml) et un fichier .xml contenant les informations voulues. Puis ces informations ont été extraites à l'aide de la libraire "xml" de Python et converties en fichier CSV en sélectionnant les informations utiles, à savoir les numéros de mesures, les hauteurs, octaves et durées des notes, les hauteurs et modes des accords, et les informations sur la clé jouée.

2.2 Pré-traitement des données

Afin de rendre ces informations les plus simples et utiles possibles. La key_fifths permet de connaître la gamme dans lequel le morceau est joué. Pour normaliser les données nous avons transposé toutes les chansons dans la clé de Do Majeur. Nous avons conservé le numéro de la mesure, les hauteurs et les durées des notes et les hauteurs des accords. Nous avons gardé un accord par mesure. Afin de simplifier les données, nous avons segmenté les modes des accords en deux modes distincts: majeur -associé à une sonorité joyeuse -, et mineur -associé à une sonorité plus sombre-. Ce sont les deux modes principaux en musique. Ainsi un G#m7dim par exemple sera transformé en G#:min (min pour mineur). Un silence correspond donc à une absence de note.

Ainsi, il y a 12 classes pour une note (12 hauteurs:A, A#,B,C ...). Nous transformons une note en vecteur One-Hot. Nous avons ensuite créé des tenseurs Numpy .npy où la première dimension correspond au nombre de groupes de n mesures (expliqué ci-après), la deuxième dimension à une mesure, et la troisième dimension à la note. Du padding a été effectué afin d'obtenir des séquences de notes (mesures) de même taille (taille 32). Ce sont donc des tenseurs de taille Nombre de chansons x Nombre de mesures x Taille du vecteur note. Ensuite, deux types de données sont utilisés pour entraîner deux types de modèles. Le BiLSTM prendra en entrée une séquence de notes appartenant à une unique mesure, le modèle Seq2Seq prendra lui une séquence de notes appartenant à plusieurs mesures. Nous utiliserons les séquences de mesures d'une même chanson se superposant: par exemple la séquence de mesure [1,2,3,4] puis la séquence [2,3,4,5], l'unique condition étant que les mesures doivent provenir de la même chanson. Toutefois, l'ensemble de test utilisera seulement des séquences de mesures ne se recoupant pas, c'est à dire [1,2,3,4] puis [5,6,7,8] pour obtenir des résultats comparables à ceux de l'article [Lim et al.(2017)Lim, Rhyu, and Lee].

Après avoir testé nos modèles, nous avons décidé d'ajouter la durée de la note dans notre modèle. En effet cette caractéristique semble très importante car une ronde (qui dure 4 temps) ne devrait pas avoir le même poids qu'une croche (qui dure un demi-temps) sur l'harmonie. Nous avons choisi de le faire de deux façons différentes. Premièrement en ajoutant une 13e colonne au vecteur note correspondant à la durée normalisée de la note. Deuxièmement, en répétant la note dans la mesure (qui est une séquence de notes) un nombre de fois correspondant à sa durée. Par exemple une blanche (2 temps) sera une même note (un même vecteur) répétée 16 fois car une mesure est une séquence de 32 notes, et la blanche représente une demi-mesure. Les résultats obtenus sont meilleurs lorsqu'on prend la durée en compte et similaires pour les deux méthodes choisies, bien que légèrement meilleurs pour la séquence de notes répétées. Nous choisirons donc dans la suite cette deuxième méthode.

2.3 Division en ensemble d'entraînement, de validation et de test

Les données ont été divisées en ensemble d'entraînement, de test et de validation. L'ensemble d'entraînement comprend 1788 chansons, celui de test 458. On utilise 10% de l'ensemble d'entraînement (179 chansons) pour créer un ensemble de validation nécessaire pour affiner l'apprentissage en évitant le sur-apprentissage, et en choisissant les bons hyper-paramètres.

3 Architectures et méthodes utilisées

Les architectures retenues ont été implémentées en Python avec la librairie PyTorch, permettant l'intégration du GPU pour accélérer l'apprentissage. Nous avons aussi eu recourt à Google Colab pour tirer profit des processeurs graphiques plus puissants.

3.1 Réseau de neurones récurrent type BiLSTM

La première méthode envisagée fut une architecture de type BiLSTM.

Ce modèle permet de prendre en compte les dépendances des notes au sein d'une unique mesure. En effet, un accord est généré par mesure, il est principalement relié aux notes qui composent la mesure. Dans cette approche, les entrées du modèle sont constituées de la séquence de notes composant chaque mesure. Les mesures de l'ensemble d'entraînement et de l'ensemble de validation sont séparées les unes des autres, on ne regarde plus leurs relations au sein d'une même chanson.

L'architecture retenue, donnant les meilleurs résultats, est un modèle comportant deux couches LSTM bidirectionnelles, de dimensions cachées 256, avec un dropout de 0.3 entre chaque couche, une couche linéaire réduisant la dimension des sorties à 24, et une couche log_softmax. La fonction de perte est la négative log-likelihood, qui, couplée à la couche de log_softmax nous permet d'utiliser la cross-entropy.

Les notes sont passés séquentiellement en entrée du réseau. Ainsi, on passe les 32 notes d'une mesure en entrée du réseau, chaque note étant un vecteur onehot correspondant à la note jouée, de dimension 12, auquel on rajoute parfois des features (comme expliqué en partie 3). La sortie est un accord correspondant à la mesure, onehot de dimension 24.

Pour l'apprentissage, nous avons utilisé ADAM, ainsi qu'un learning rate de 0.001 couplé à un scheduler pour le diminuer lorsque la loss sur l'ensemble d'entraînement

stagne. Une quarantaine d'epochs suffit à l'apprentissage du réseau. L'apprentissage est réalisé en suivant la valeur de la perte sur l'ensemble de validation, le modèle retenu est le modèle qui correspond à la plus faible valeur de perte sur cet ensemble.

3.2 Modèle Seq2Seq type encodeur-décodeur

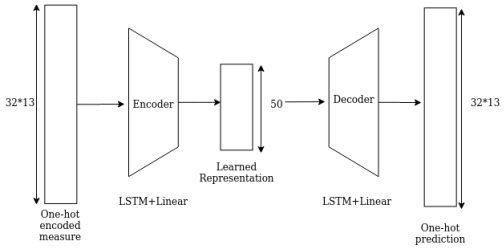
Un modèle de type Seq2Seq permet de prendre en compte non seulement les dépendances de chaque accord par rapport aux notes qui composent leur mesure, mais aussi la dépendance de chaque accord par rapport aux accords des mesures voisines. Un accord est en effet le résultat d'une harmonie par rapport aux accords le précédant et aux accords le suivant, en plus des dépendances évidentes avec les notes de la mesure à laquelle il est associé.

Nous avons utilisé comme entrée du réseau des séquences de notes provenant de plusieurs mesures successives, avec la condition triviale que les mesures de la séquence doivent toutes appartenir à la même chanson. Les meilleurs résultats sont obtenus pour une séquence de 4 mesures, soit 32*4 notes. En effet, le modèle retient mal les dépendances temporelles pour une plus grande séquence de mesure. De plus, un accord est principalement lié aux mesures qui lui sont les plus proches.

Le modèle Seq2Seq est de type encodeur-décodeur. L'encodeur est unidirectionnel. Les tests que nous avons effectués montrent qu'en utilisant un encodeur bidirectionnel, et en initialisant chaque couche du décodeur avec la concaténation du dernier état caché / de cellule pour les deux directions, les performances sont très similaires à celles obtenues avec l'encodeur unidirectionnel. L'encodeur se compose de deux couches de LSTM de dimensions cachées 256, un dropout de 0.3 entre ces couches. Le décodeur récupère le dernier état caché/de cellule pour chaque couche pour s'initialiser. Similairement, il est construit à partir de deux couches LSTM de dimensions cachées 256, dropout de 0.3, et en sortie d'une couche linéaire réduisant la dimension des accords générés à 24, avant une couche log_softmax.

Pour l'entraînement de ce modèle, on utilise comme pour le BiLSTM la négative log-likelihood qui combinée à la couche de log_softmax du modèle, est équivalent à la fonction de perte crossentropy. On utilise le gradient clipping, qui définit la borne maximum du gradient à 1.0 (problème du exploding gradient), et l'optimiseur ADAM. L'entraînement a été réalisé avec l'aide du teacher-forcing. Avec une probabilité fixée à 0.5, l'entrée suivante du décodeur est l'accord vrai qui devait être prédit en sortie précédente. Sinon, l'entrée du décodeur est la prédiction précédente. Remarquons aussi que pour être en capacité de générer une séquence d'accords, le décodeur est initialisé par un token $< start >$ (vecteur onehot de dimension 25, 1 en dernière position). Cela nous permet d'éviter un apprentissage de l'identité en décalant d'une position les entrées du décodeur et les targets. De plus, cela nous permet aussi de ne pas utiliser de targets (c'est à dire le première accord de la séquence à générer) lors de la phase de test du modèle. Ainsi, les sorties du modèles sont des vecteurs de dimension 25, il y a 24 classes d'accord et 1 classe correspondant au token start. L'entraînement du modèle est plus long et demande une centaine d'epochs.

3.3 Modèle Seq2Seq type encodeur-décodeur couplé à un autoencodeur



Architecture de l'auto-encoder

Afin d'améliorer notre solution, nous avons pensé qu'il pourrait être intéressant de procéder à une compression de nos données avant de les passer dans notre modèle Seq2Seq. En effet, chaque groupe de 4 mesures était représenté précedemment par une matrice de taille (32*4,12) composée de vecteurs one-hot, et on pourrait imaginer qu'en apprenant une représentation dense de nos données, l'apprentissage serait plus efficace. Un autoencoder apprends une représentation dense des données en ayant en entrée et en sorties les mêmes données, et au milieu un bottleneck, à partir duquel il essaye de retrouver les données initiales. Cette srucuture pourrait donc compresser chaque mesure, l'idée étant ensuite de donner en entrée de notre architecture Seq2Seq, un groupe de 4 mesures qui ne serait plus représenté par une matrice de taille (32*4,12) mais pas une matrice de taille (4, *dim_compressed*), où *dim_compressed* correspond à la taille du vecteur représentant une mesure après encodage. Puisque l'on veut compresser les données pour chaque mesure, le dataset en entrée ici ne correspond pas aux notes groupées par 4 mesures comme pour l'architecture Seq2Seq, mais aux notes groupées par mesures. Les mesures sont ainsi représentées par des matrices (32,12), que nous aplatissons en un grand vecteur avant de le faire entrer dans notre autoencoder, qui aura pour but de réduire la dimension de ce vecteur à une taille *dim_compressed*.

Puisque l'autoencoder est un cas particulier d'encoder/decoder, notre implémentation Seq2Seq a été adaptée pour créer l'architecture autoencoder. Elle est composée d'un encoder -un LSTM suivi d'une couche linéaire, dont la dimension de sortie correspond à *dim_compressed*-, puis d'un decoder - un LSTM et une couche linéaire -, qui prédit chaque note de la mesure encodée.

Comme pour le modèle Seq2Seq, il est nécessaire d'initialiser le decoder avec un token start pour éviter le surapprentissage et également éviter d'utiliser la première note de chaque mesure, qui est donc ici la target du modèle, lors de la phase de test. Nos notes, qui composent ici les targets du modèle, sont en fait représentées par des vecteurs one-hot de longueur 13, où les 12 premiers éléments correspondent bien aux 12 classes de notes et le 13ème élément correspond au token start.

Pour l'évaluation, afin de ne pas accorder une trop grande importance aux vecteurs nuls ajoutés aux matrices des mesures par le padding (effectué pour que toutes les mesures

soient de taille (32,12)), on calcule une accuracy où on ne prend pas en compte les prédictions de l'autoencoder sur ces vecteurs nuls.

Nous obtenons de bons résultats pour la compression des données en encodant chaque mesure par un vecteur de dimension 50, avec un teacher_forcing_ratio de 0.3.

Malheureusement, en essayant de passer en entrée de notre modèle Seq2Seq ces mesures encodées, l'apprentissage du réseau ne fonctionne pas du tout. Nous n'avons pas eu le temps de comprendre ce qui pose problème, nous n'avons donc pas pu intégrer l'autoencoder à notre pipeline.

4 Résultats

On note D1 l'ensemble des données avec la feature temporelle répétée le nombre de fois correspondant à sa durée.

Di avec i différent de 1: ensemble de i mesures obtenues par le même traitement que D1, se superposant (mais pas de superposition pour l'ensemble de test comme expliqué en section 2.2).

4.1 Précision des modèles

Les résultats obtenus sont présentés dans le tableau 1 ci dessous. Ils ont été obtenus avec une taille de batch de 2048.

Modèle	Données	Accuracy validation	Accuracy test
BiLSTM	D1	0.523	0.452
Seq2Seq	D2	0.548	0.451
Seq2Seq	D4	0.550	0.471
Seq2Seq	D6	0.516	0.460
Seq2Seq	D8	0.604	0.443

Table 1: Résultats

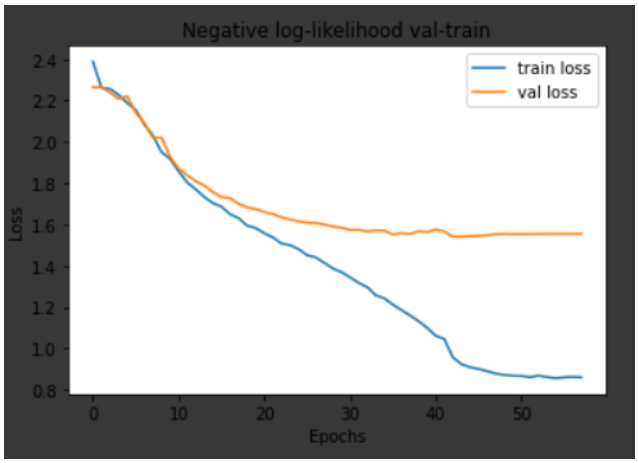


Figure 1: Évolution de la valeur de la loss pour le modèle Seq2Seq, D4, au cours de l'entraînement

L'autoencoder donne de bons résultats en étant entraîné sur 50 époques, avec une accuracy (pour laquelle les vecteurs nuls du padding sont ignorés) sur les inputs de test de **0.948** et les courbes d'apprentissage suivantes présentées dans les figures 3 et 4. L'apprentissage d'une forme plus dense des données est efficace.

Malheureusement, nous n'avons pas pu observer les performances de cet encodage des données insérées en entrée du modèle Seq2Seq. Nos tentatives n'ont pas permis au modèle d'apprendre à partir de cet données encodées, nous n'avons donc pas pu intégrer cet autoencoder au modèle. On peut par conséquent se demander si la représentation apprise pour chaque mesure est une représentation pertinente et utilisable des données.

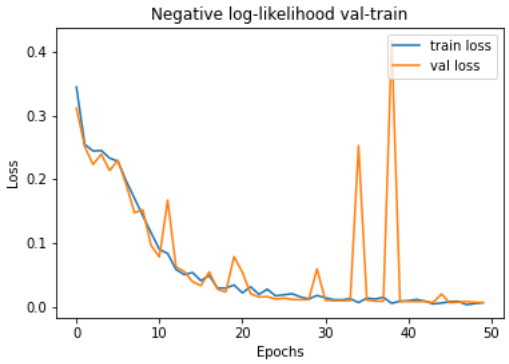


Figure 2: Évolution de la valeur de la loss lors de l'entraînement de l'autoencoder

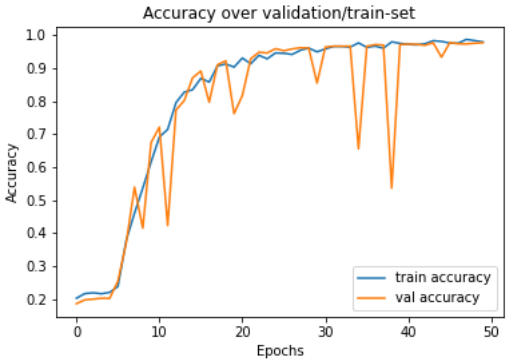


Figure 3: Évolution de la valeur de l'accuracy lors de l'entraînement de l'autoencoder

4.2 Écoute des résultats

Comme expliqué en introduction, la mesure de précision n'est pas suffisante pour déterminer si les résultats sont corrects ou non. Afin de mesurer la cohérence des résultats obtenus, nous avons généré des fichiers audio jouant en parallèle la mélodie et les accords à la guitare à chaque mesure. Pour ce faire, nous avons créé un script Python ajoutant une deuxième voix (jouée à la guitare) au fichier mxl original. Cette deuxième voix joue les accords générés à chaque mesure à l'aide d'un dictionnaire de la musique et de théorie de base en musique (pour les accords mineurs/majeurs). Voir Figure 4

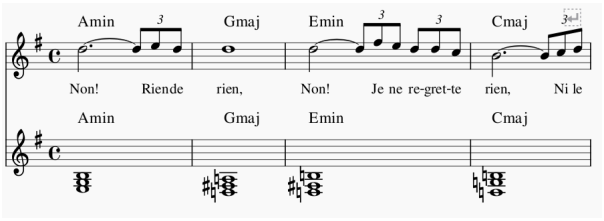


Figure 4: Partition générée. La ligne du dessus correspond à la mélodie, la ligne du dessous à l'harmonie à la guitare. Les accords sont marqués en noir.

Une fois les fichiers .mxl générés, le logiciel MuseScore permet d'obtenir un fichier audio .wav. Des fichiers audio résultats et groundtruth sont donnés dans le lien du code à titre d'exemple.

Afin de mesurer la cohérence de la génération d'accords, nous avons fait écouter les audio avec les accords réels et les accords générés à 4 personnes différentes. Leur tâche était de déterminer si les accords entendus correspondaient aux accords originaux ou aux accords générés. Malheureusement, à cause d'un petut nombre d'accords qui sonnaient mal, les 4 sujets ont identifié correctement les accords générés. Cependant, ils ont tous les 4 souligné la cohérence de la progression générée. Si ce test n'a pas été concluant, il reste néanmoins important afin de mesurer la pertinence de nos résultats. Un test à plus grand échelle aurait été un bon indicateur de cohérence afin d'améliorer notre modèle.

5 Conclusion

Pour conclure, les meilleurs résultats ont été obtenus avec un modèle Seq2Seq de type encodeur-décodeur. La bidirectionnalité de l'encodeur dans ce modèle, ou de l'autre modèle BILSTM, n'apporte pas d'amélioration nette, et nécessite des temps de calcul beaucoup plus longs.Un accord semble ainsi être lié aux accords qui le suivent mais moins à ceux qui le précède.

Le palier de précision atteints par nos différents modèles peut être lié à la structure intrinsèque des données. En effet, les musiques utilisées proviennent de genres très variés. Les accords du jazz ne sont pas régis par les mêmes règles que les accords de la musique chinoise, ou de la musique pop. Le modèle peut alors avoir des difficultés à généraliser son apprentissage. Nous avons ainsi observé une diminution de la valeur de la loss sur l'ensemble d'entraînement jusqu'à des valeurs très faibles (overfitting jusqu'à des valeurs de l'ordre de 0.1). Néanmoins la loss sur l'ensemble de validation baisse beaucoup moins bas et beaucoup moins vite. On pourrait alors améliorer nos résultats en utilisant plus de données pour l'entraînement, par exemple avec la cross-validation, et surtout en ciblant ces données, en se restreignant par exemple à la musique pop.

De plus, il serait peut être utile d'ajouter une couche de batch normalisation pour accélérer l'entraînement, qui reste très lent dans nos différents essais. Il faudrait aussi explorer de nouvelles features pertinentes, comme les octaves. Nous pourrions explorer de nouvelles techniques de réduction de dimensions, en utilisant notre autoencoder, ou encore une couche d'embeddings dans le modèle Seq2Seq.

Enfin, ous avons segmenté les accords en uniquement deux modes différents. Nous pourrions ajouter des modes comme les accords septièmes qui ajoute de la tension au sein de la progression d'accords afin d'obtenir des progressions plus complexes, plus diverses et musicalement cohérentes.

References

[Chen et al.(2015)Chen, Qi, and Zhou] Ziheng Chen, Jie Qi, and Yifei Zhou. 2015. Machine Learning in Automatic Music Chords Generation.

[Dong et al.(2017)Dong, Hsiao, Yang, and Yang] Hao-Wen Dong, Wen-Yi Hsiao, Li-Chia Yang, and Yi-Hsuan Yang. 2017. http://arxiv.org/abs/1709.06298 MuseGAN: Multi-track Sequential Generative Adversarial Networks for Symbolic Music Generation and Accompaniment. *arXiv e-prints*, page arXiv:1709.06298.

[Engel et al.(2019)Engel, Krishna Agrawal, Chen, Gulrajani, Donahue Jesse Engel, Kumar Krishna Agrawal, Shuo Chen, Ishaan Gulrajani, Chris Donahue, and Adam Roberts. 2019. http://arxiv.org/abs/1902.08710 GANSynth: Adversarial Neural Audio Synthesis. *arXiv e-prints*, page arXiv:1902.08710.

[Lim et al.(2017)Lim, Rhyu, and Lee] Hyungui Lim, Seungyeon Rhyu, and Kyogu Lee. 2017. http://arxiv.org/abs/1712.01011 Chord Generation from Symbolic Melody Using BLSTM Networks. *arXiv e-prints*, page arXiv:1712.01011.

[Sutskever et al.(2014)Sutskever, Vinyals, and Le] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. http://arxiv.org/abs/1409.3215 Sequence to Sequence Learning with Neural Networks. *arXiv e-prints*, page arXiv:1409.3215.