Collège Calvin 2020-2021

L’algorithme du batteur:
création d’une boîte à rythmes améliorée, automatisée et basée sur les réseaux de neurones artificiels

GOELZER Roberto

Maître accompagnant: Bouget M.

1. Introduction

De nos jours, nous sommes de plus en plus entourés par la technologie et l'intelligence artificielle. Que ce soit à travers les algorithmes qui choisissent ce que l'on voit dans les réseaux sociaux ou bien les assistants de nos smartphones, l'intelligence artificielle est présente sous une multitude de formes pour une multitude de tâches. Et assez récemment, diverses compagnies ont commencé à utiliser l'intelligence artificielle à des fins créatives dans le domaine de la musique, comme le fait par exemple la nouvelle plateforme *open-source* Magenta[1], créée par Google.

Et c'est dans ce domaine que j'ai choisi de faire mon travail de maturité. Jouant d'un instrument (la batterie), je voulais appliquer mes connaissances du *machine learning* à la musique d'une manière qui me concerne directement. C'est pourquoi je me suis posé le défi de créer un algorithme de *deep learning* capable de générer un accompagnement musical de batterie à un extrait de musique qui n'en contient pas. Et le chemin pour y parvenir est tout aussi important, à mes yeux, que le résultat final; car le chemin n'a pas été pas facile.

En effet, le *machine learning* n'est pas le domaine de l'informatique le plus trivial de tous. Sa récence et les formules mathématiques associées n'en font pas, à première vue, un domaine simple à aborder. Cependant, grâce à des librairies très bien faites telles que Keras[2] ou Tensor Flow[3], et l'intuitivité du langage de programmation Python, le *machine learning* devient de plus en plus accessible et saisissable par tous ceux qui s'y intéressent. Ainsi, quiconque ayant des connaissances en informatique peut créer des algorithmes puissants et précis de *machine learning* en quelques lignes de code seulement. Et ces algorithmes ne doivent pas être parfaits dès le départ. Au contraire, ce sont des algorithmes qui apprennent tout seuls à s'améliorer. Le rôle du programmeur est donc de bien définir le cadre de l'algorithme dans un programme et de fournir les données nécessaires à son entraînement.

Pour présenter mon algorithme de manière compréhensible par tous, je vais commencer par faire une présentation un peu plus en détail du thème en question, le *deep learning*. J'évoquerai et expliquerai d'abord les différentes architectures et types de réseaux de neurones artificiels que je mentionnerai au cours de ce travail. Après cette explication plus globale du thème, je parlerai plus précisément du *deep learning* appliqué à la musique, en présentant les approches utilisées dans la littérature scientifique. Ensuite vient la présentation de mon programme, contextualisée par les deux premières parties. Je parlerai de l'architecture de mon modèle, des choix faits et des méthodes utilisées pour parvenir à mon but. Enfin viendra la présentation des résultats, accompagnée d'une analyse complète, suivie pour finir d'une conclusion synthétique.

---

[1]"Magenta Studio: Augmenting Creativity with Deep Learning in Ableton Live", ROBERTS A., ENGEL J., MANN Y., GILLICK J., KAYACIK C., NØRLY S., DINCULESCU M., RADEBAUGH C., HAWTHORNE C. et ECK D., *Google*, 2019, disponible sur https://magenta.tensorflow.org [consulté le 26/10/21]

[2]CHOLLET F. et al., "Keras: Deep Learning for Humans", *Github*, 2015, https://github.com/fchollet/keras [consulté le 26/10/21]

[3]"TensorFlow: Large-scale machine learning on heterogeneous systems", ABADI M., AGARWAL A., BARHAM P., BREVDO E., CHEN Z., CITRO C., CORRADO G. S., DAVIS A., DEAN J., DEVIN M., GHEMAWAT S., GOODFELLOW I., HARP A., IRVING G., ISARD M., JIA Y., JOZEFOWICZ R., KAISER L., KUDLUR M., LEVENBERG J., MANÉ D., MONGA R., MOORE S., MURRAY D., OLAH C., SCHUSTER M., SHLENS J., STEINER B., SUTSKEVER I., TALWAR K., TUCKER P., VANHOUCKE V., VASUDEVAN V., VIÉGAS F., VINYALS O., WARDEN P., WATTENBERG M., WICKE M., YU Y. et ZHENG X., *Google Research*, 2015,

2. Présentation du *deep learning*

2.1. *Machine learning*

Le *machine learning* – aussi appelé apprentissage automatique – est un champ de l'intelligence artificielle, dont l'idée centrale est la création d'algorithmes capables d'apprendre par eux-mêmes. En effet, ce n'est pas toujours facile de créer un algorithme pour accomplir une tâche analytique spécifique – à titre d'exemple, comment apprendre à un ordinateur à différencier les photos d'un chat de celles d'un chien? – car parfois la tâche est trop complexe pour être explicitée à l'algorithme par le programmeur.

Et c'est là que survient l'utilité du *machine learning*, car il permet la création de programmes qui à leur tour apprennent à l'algorithme comment s'améliorer lui-même à la tâche en question. Et la puissance du *machine learning* vient du fait qu'un algorithme peut être très performant dans une tâche sans avoir été explicitement programmé pour l'accomplir.

Ce domaine étant très vaste, on y trouve une multitude d'algorithmes différents, de tous types, et pour toutes tâches. Un grand nombre d'entre eux sont des algorithmes mathématiques relativement simples et élégamment puissants, tels que la régression linéaire ou logistique, le partitionnement de données, etc.; tandis que d'autres sont bien plus lourds mais plus polyvalents, notamment la majorité des réseaux de neurones artificiels, dont le bon fonctionnement repose sur des concepts plus complexes – mathématiquement parlant.

Malgré ces différences algorithmiques fondamentales, il existe un tronc commun du *machine learning* non-négligeable, l'importance des données.

Un ingrédient essentiel d'un bon algorithme est un bon jeu de données. Sans elles, l'algorithme ne peut être très performant[4]. Malgré cela, les données ne font pas tout le travail, un bon algorithme nécessite d'être entraîné avec les bonnes méthodes et de la bonne manière. Ceci nécessite de l'expérience dans le domaine, ou une phase de tests approfondie, afin d'évaluer la viabilité des différentes méthodes et architectures à disposition pour résoudre le problème. Ces tests sont donc nécessaires pour trouver une combinaison optimale d'hyperparamètres[5], afin de rendre non seulement l'apprentissage plus rapide mais surtout l'algorithme plus performant.

2.2. *Deep learning*

Le *deep learning*, aussi appelé apprentissage profond, est une sous-catégorie du *machine learning* et se focalise sur l'utilisation de réseaux de neurones artificiels comme structure pour ses algorithmes. Ceux-ci hiérarchisent les données en différents niveaux d'abstraction à travers leurs nombreuses couches de neurones empilées – d'où le "profond" dans le nom. Ils peuvent en extraire des caractéristiques de plus en plus complexes au fil des couches.

---

[4]Une expression commune de l'apprentissage automatique est «*garbage-in, garbage-out*» en anglais («poubelle en entrée, poubelle en sortie»). Cela illustre bien l'importance de la qualité des données sur la qualité des résultats obtenus.

[5]Les hyperparamètres sont des paramètres qui contrôlent le modèle dans son entièreté et le processus d'apprentissage. cf. BROWNLEE Jason, "What is the Difference Between a Parameter and a Hyperparameter", *Machine Learning Mastery*, 17 juin 2019, https://machinelearningmastery.com/difference-between-a-parameter-and-a-hyperparameter/ [consulté le 27/10/21]

De plus, la différence majeure entre les algorithmes de *machine learning* et de *deep learning* est que ces premiers algorithmes nécessitent en général un traitement des données plus approfondi afin d'en extraire "manuellement" les caractéristiques, tandis que ces derniers incluent dans leur structure-même des couches pour l'extraction des caractéristiques de plus basse complexité. Les algorithmes de *deep learning* sont donc d'autant plus automatisés dans leur capacité à traiter les données et les classifier.

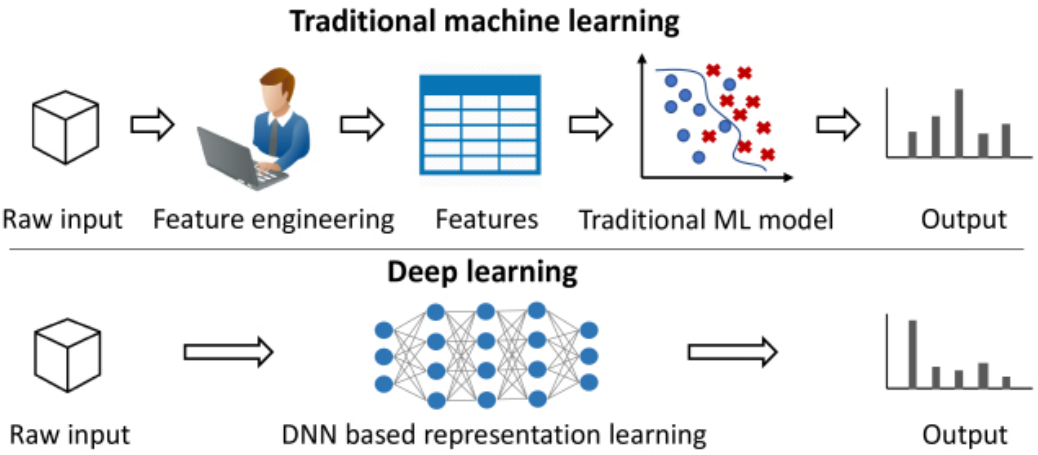

Image 1: différences entre le *machine learning* et le *deep learning*, vis à vis de l'extraction des caractéristiques des données et du modèle utilisé. Image tirée de l'article scientifique « Techniques for Interpretable Machine Learning » de Mengnan Du et al.[6]

En outre, ces algorithmes d'apprentissage profond utilisent un grand nombre de concepts de *machine learning* et de formules mathématiques sous-jacentes, se basant essentiellement sur l'algèbre linéaire. Or étant donné qu'une explication détaillée de ces formules dépasse l'envergure de ce travail de maturité, je ne vais pas m'y attarder. Les éléments importants à retenir sont les suivants:

- les réseaux de neurones artificiels sont l'équivalent d'une fonction régie par une multitude de paramètres, et séparée en couches de neurones successives;
- chaque neurone est aisément définissable – une simple opération matricielle linéaire –, pourtant l'ensemble des neurones est capable de modéliser n'importe quelle fonction mathématique[7];
- l'apprentissage d'un tel modèle se fait en comparant les résultats obtenus avec les résultats cibles, et en modifiant individuellement les valeurs des paramètres pour optimiser le résultat obtenu – une tâche triviale pour un ordinateur mais irréalisable pour un humain.[8]

Par ailleurs, la capacité d'un modèle de *deep learning* à effectuer une tâche correctement est une caractéristique émergente de son entraînement; c'est-à-dire qu'on ne

[6]cf. "Techniques for Interpretable Machine Learning", MENGNAN D., NINGHAO L. et XIA H., *arXiv Preprint arXiv:1808.00033v1*, 2018, disponible sur https://arxiv.org/pdf/1808.00033v1.pdf [consulté le 27/10/21]

[7]Je me réfère ici au théorème d'approximation universelle des réseaux de neurones. cf. "Multilayer feedforward networks are universal approximators", Hornik K., Stinchcombe M. et White H., *Neural Networks*, Volume 2, Issue 5, 1989, pp. 359-366, ISSN 0893-6080, disponible sur https://doi.org/10.1016/0893-6080(89)90020-8. [consulté le 27/10/21]

[8]Cette méthode s'appelle la rétropropagation du gradient, elle utilise les dérivées partielles des différents paramètres par rapport à l'erreur des neurones de la couche suivante pour modifier la valeur des paramètres. Et le modèle apprend en se rapprochant peu à peu de l'ensemble de paramètres optimal. cf. KOSTADINOV Simeon, "Understanding Backpropagation Algorithm", *Towards Data Science*, 8 août 2019, https://towardsdatascience.com/understanding-backpropagation-algorithm-7bb3aa2f95fd [consulté le 26/10/21]

peut pas véritablement savoir ni comment ni pourquoi un certain modèle est bon. En effet, l'apprentissage d'un modèle est un processus éminemment aléatoire. On ne peut prédire à l'avance l'efficacité d'un modèle; il faut constamment le tester, l'évaluer, le modifier et réessayer.

2.3. RNN

Une catégorie du *deep learning* très importante – autant pour ce travail de maturité qu'en informatique en général – est celle des réseaux neuronaux récurrents (RNNs). Ces derniers sont comme des réseaux neuronaux artificiels classiques, à la différence près qu'ils s'étendent en plus dans le temps et se spécialisent dans le traitement de séquences. Et ils ne traitent pas ces dernières d'un seul coup, mais ils la parcourent plutôt élément par élément. Ceux-ci développent donc une sorte de mémoire interne des éléments précédents de la séquence. Et là se trouve toute l'utilité de ce type de réseau.

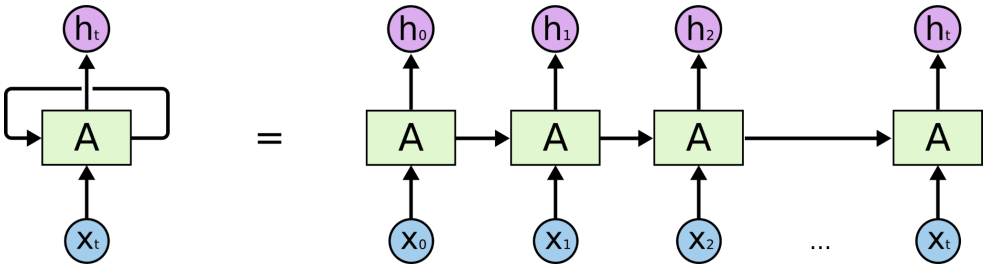

Image 2: un RNN sous ses deux formes les plus courantes: à gauche est représenté le RNN tel qu'il est défini (connecté à lui-même lors du prochain pas), et à droite sa version déroulée dans le temps. Le bloc A désigne la cellule récurrente, unité de base d'un RNN; les ronds bleus (x_t) et violets (h_t) représentent respectivement les entrées et les sorties de la cellule RNN; enfin les flèches représentent le cheminement des données, de l'input à l'output. Image tirée d'un article de Christopher Olah.[9]

Comme l'affirme très justement Jason Brownlee dans son livre à ce sujet, la temporalité des données donne une nouvelle dimension à la fonction à modéliser. À la place de modéliser l'output en fonction de l'input seul, un RNN est capable de modéliser les outputs en fonction de l'input au fil du temps. Le modèle a accès – grâce à sa mémoire interne – aux données précédentes en plus des données lui étant fournies. Cela donne la possibilité au modèle – en plus de posséder les avantages de réseaux neuronaux artificiels classiques – d'exploiter les motifs temporels des données.[10]

Je pense sincèrement que l'invention des RNNs – et de toutes les nouvelles architectures associées (LSTMs, GRUs, Attention RNNs, etc.) – a révolutionné le domaine du *machine learning* en ouvrant la porte pour le traitement de séquences. Et ce nouveau domaine est extrêmement vaste.

D'un côté, il permet de résoudre de nombreuses nouvelles classes de problèmes, comme notamment les tâches de prédiction, classification et génération de séquences[11]. Celles-ci incluent par exemple la prévision des actions en bourse, l'analyse sentimentale ou thématique d'un texte, la génération de musique selon le style d'un corpus donné, la génération automatique de légendes d'images, la génération d'un résumé d'un texte, etc.[12]

---

[9]OLAH Cristopher, "Understanding LSTM Networks", *Colah's Blog*, 27 août 2015, https://colah.github.io/posts/2015-08-Understanding-LSTMs/ [consulté le 26/10/21]
[10]BROWNLEE Jason, *Long Short-Term Memory Networks with Python*, Machine Learning Mastery, p. 9, disponible sur https://machinelearningmastery.com/lstms-with-python/ [consulté le 26/10/21]
[11]cf. *ibid.*, p. 3
[12]cf. *ibid.*, pp. 4-7

D'un autre côté, il s'applique à une assez grande variété de données: les séquences temporelles – comme les données météorologiques, les actions en bourse ou les données musicales –, le langage – que ce soit du texte, du code, ou de l'écriture manuscrite –, ou mêmes des images et des vidéos.

Ce qui va nous intéresser particulièrement dans ce domaine, c'est un type de réseaux neuronaux récurrents dits LSTMs (*Long Short-Term Memory networks*). Ils se différencient par le fonctionnement de leurs cellules particulières, qui leur confèrent divers atouts majeurs par rapport à un RNN classique – dont la cellule ne se différencie pas d'une couche d'un réseau de neurones artificiels classique. Plus particulièrement, les LSTMs ont une meilleure capacité de retenir les informations temporelles éloignées. Ils ont une meilleure mémoire interne et sont donc plus résistants aux problèmes courants de l'entraînement des RNNs classiques.

2.4. CNN-LSTM

Un autre type de réseaux neuronaux récurrents qui pourrait nous intéresser est le CNN-LSTM. C'est une combinaison d'un LSTM et d'un CNN (réseau neuronal convolutif). Ce CNN peut être apparenté à un ensemble de petites fenêtres glissantes qui parcourent les données et en extraient localement des caractéristiques visuelles simples – leur structure est inspirée du cortex visuel humain et des motifs de connectivité entre nos neurones[13].

Cette citation de Jason Brownlee peut nous aider à comprendre l'utilité de chacune des deux parties d'un CNN-LSTM:

> L'architecture CNN-LSTM consiste en l'utilisation des couches d'un réseau neuronal convolutif (CNN) pour l'extraction des caractéristiques des données input et en la combinaison de ces couches avec des LSTMs pour permettre [le traitement et] la prédiction de séquences.[14]

En effet, le CNN est très utile en première couche d'un modèle. Il permet un traitement rapide et économique d'images ou de vidéos. Mais ses avantages ne s'arrêtent pas là. Une variante particulière des CNNs va nous intéresser – les CNNs à une dimension – grâce à leur capacité de traiter aussi les données musicales. En effet, ceux-ci parcourent cette fois les données dans leur dimension temporelle uniquement et en extraient des caractéristiques locales à chaque pas – tout comme des LSTMs. Ainsi, ces CNNs peuvent être combinés à des LSTMs, et les deux forment ensuite une nouvelle cellule récurrente. Cette dernière peut être intégrée dans une couche RNN, conservant ainsi tous les avantages d'un LSTM classique, avec en plus une meilleure extraction des caractéristiques – pour autant que les données aient des caractéristiques ou des motifs visuels.

2.5. VIB/VAE

Pour comprendre ce que sont les architectures d'un VAE (auto-encodeur variationnel) ou d'un modèle avec VIB (*Variational Information Bottleneck*), il faut d'abord comprendre ce qu'est un auto-encodeur.

---

[13]cf. SAHA Sumit, "A Comprehensive Guide to Convolutional Neural Networks – the ELI5 way", *Towards Data Science*, 15 décembre 2018, https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53 [consulté le 27/10/21], §4

[14]*ibid.*, p.93, (trad. libre)

Un auto-encodeur est un modèle composé de deux parties: l'encodeur et le décodeur. Le premier s'occupe de réduire les dimensions des données en entrée, pour ainsi en extraire les caractéristiques discriminantes les plus importantes – *i.e.* il compresse les données originales. Le second, quant à lui, s'occupe de reproduire les données originales à partir des caractéristiques fournies par l'encodeur – *i.e.* il décompresse les caractéristiques pour retrouver les mêmes données. C'est donc un type d'apprentissage non supervisé – il n'y a pas de besoin de définir de données cibles, car les données input sont les données cibles.

En ce faisant, l'auto-encodeur crée un espace latent, qui est l'espace formé par toutes les caractéristiques trouvées par l'encodeur. Mais cet espace vectoriel n'est pas régularisé – certains couples de caractéristiques ne correspondent absolument à rien. Et c'est là qu'intervient l'auto-encodeur variationnel. Il modifie légèrement l'encodeur et ajoute un nouveau terme à l'erreur du modèle dans l'optique de régulariser l'espace latent.

Notamment, le VAE diffère d'un auto-encodeur normal car, à la place d'encoder directement les données dans l'espace latent en un vecteur z, il les encode sous la forme de distributions – plus particulièrement sous ses deux paramètres: la moyenne et la variance – qu'il va ensuite échantillonner pour générer z. Le fonctionnement du décodeur est identique.

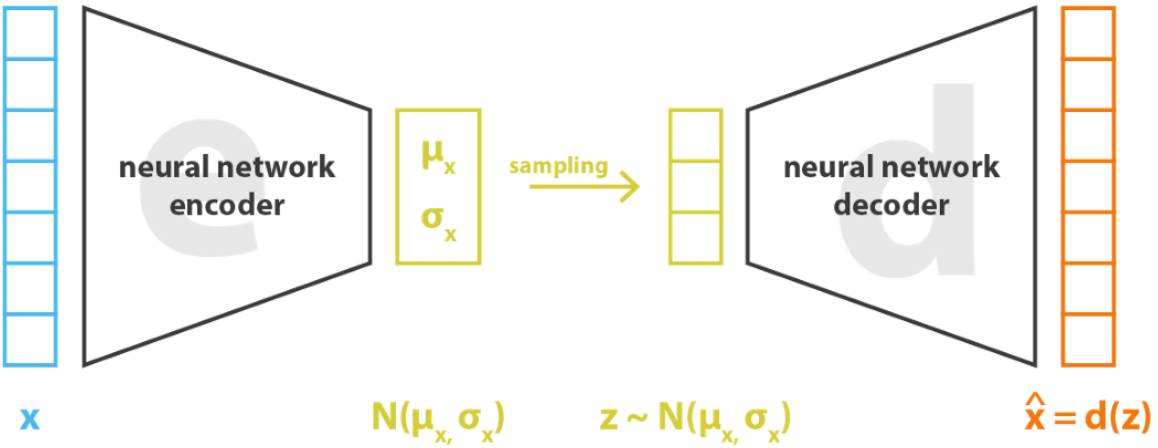

Image 3: structure d'un VAE. On y voit l'encodeur et le décodeur en noir, et la partie variationnelle en jaune. μ_x et σ_x représentent les matrices des paramètres des distributions – respectivement la matrice des moyennes et des variances; x et $\hat{x}$ désignent respectivement les données originales et reproduites; et $N(\mu_x, \sigma_x)$ représente les distributions gaussiennes, définies par les deux matrices de paramètres μ_x et σ_x. Image tirée d'un article de Joseph Rocca dans Towards Data Science.[15]

À la différence des VAEs, les modèles avec VIB ne sont pas des auto-encodeurs, mais sont plutôt basés sur des modèles nommés *encoder-decoder*. Ils comportent aussi un encodeur et un décodeur; et suivent le même format d'encoder les données en distributions, puis de les décoder. Mais ils se différencient par leur sortie, qui, elle, diffère de l'entrée. La partie variationnelle de l'encodeur – *i.e.* les distributions régularisées – est tout de même avantageuse. Elle permet au modèle d'avoir un espace latent continu et dense. Et cela confère au modèle plus de robustesse lorsque ce dernier est confronté à des données inconnues qui diffèrent largement des données d'entraînement. De plus, l'étape d'échantillonnage ajoute un peu de bruit dans l'entraînement et ainsi confère au modèle une meilleure capacité de généralisation.

[15]cf. "Understanding Variational Autoencoders", ROCCA Joseph et ROCCA Baptiste, *Towards Data Science*, 24 septembre 2019, https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73 [consulté le 25/10/21]

3. *Machine learning* appliqué à la musique

3.1. Représentation des données

La représentation des données est une partie très importante du *machine learning*, encore plus lorsque l'on se trouve dans le domaine musical. Les possibilités de représentation des données sont très vastes, et sont étroitement liées à l'objectif et l'architecture du modèle en question.

La plus grande distinction pour les données musicales se fait entre la représentation dite «symbolique» des données et la représentation d'un signal audio. Cette dernière se base majoritairement sur le traitement du signal, en traitant soit le signal audio pur – ce qui est informatiquement très lourd en termes de mémoire et traitement – ou bien en le transformant à pertes pour en faire ressortir des caractéristiques de plus haut niveau, au moyen de spectrogrammes, ou de "chromagrammes"[16].

La représentation symbolique, quant à elle, est liée à des représentations plus abstraites inspirées de la notation musicale. On y trouve notamment les notes, la durée, le rythme, les accords et de nombreux autres concepts musicaux.

Cette approche est bien plus utilisée dans la littérature scientifique pour diverses raisons. Une première raison, exposée par Briot et al. (2019), serait la suivante:

> L'essence de la musique (contrairement au son) se trouve dans le processus de composition musicale, qui se révèle au travers de représentations symboliques (comme des partitions musicales) et qui peut être le sujet d'analyses (telles que l'analyse harmonique).[17]

Cette citation illustre bien l'importance de la dimension symbolique dans les représentations de données musicales, notamment pour avoir accès à toute la dimension d'analyse harmonique et musicale – qui sera, à titre d'exemple, très utile lors de l'analyse des résultats de mon modèle (cf. § 5.2. Analyse).

Il y a cependant une deuxième raison pour laquelle la représentation symbolique est préférable. Et la raison se trouve dans la simplicité et l'efficacité liées à la représentation de caractéristiques de haut-niveau (telles que les notes) par rapport à des caractéristiques de plus bas niveau (telles qu'une onde sonore pure) – d'un point de vue algorithmique et du point de vue du volume des données à traiter.

Au sein de cette représentation symbolique, par ailleurs, on trouve encore de nombreuses possibilités pour l'encodage des données. J'ai personnellement opté pour un encodage se rapprochant plutôt de la notation de musique digitale MIDI, à quelques différences près. Notamment, je suis passé de la notation événementielle des notes dans le système MIDI à une notation temporelle absolue – une notation s'apparentant plus à un *piano-roll*[18]. Ensuite pour la représentation des accords et des notes, j'ai choisi d'utiliser un

---

[16] C'est un type de spectrogramme, où l'on discrétise les fréquences en les notes de la gamme chromatique. cf. "Deep Learning Techniques For Music Generation – *A Survey"*, BRIOT J., HADJERES G., PACHET F., *arXiv Preprint arXiv:1709.01620v4*, 2019, p. 20, disponible sur https://arxiv.org/pdf/1709.01620.pdf [consulté le 27/10/21]

[17] ibid, p.18 (trad. libre)

[18] C'est le terme utilisé dans la littérature scientifique pour indiquer ce type de partition. Cela représente un tableau défilant avec comme lignes les différentes notes jouables et comme colonnes les "pas temporels" (ou "*time steps*") – représentant les plus petits intervalles de temps choisis pour discrétiser la temporalité de la partition (cf. *ibid.*, p. 31). Le désavantage majeur est que l'on perd les données sur la fin de la note; ainsi, des

encodage *many-hot* – plusieurs notes peuvent être représentées simultanément dans une colonne du *piano roll*, contrairement à la notation *one-hot*[19] où chaque valeur du vecteur désigne un couple d'une ou plusieurs notes, et où le vecteur contient tous les couples possibles – car c'est une des manières les plus simples d'encoder une mélodie polyphonique (avec plusieurs notes jouées simultanément), ce qui est le cas de la batterie et, la plupart du temps, du piano.

Cet encodage fonctionne bien pour les notes, mais on a besoin de méthodes différentes pour les autres types de données, comme par exemple la vélocité des notes.

En effet, pour représenter un rythme de batterie assez fidèlement, une approche possible – exposée par Gillick et al. (2019) pour leur modèle GrooVAE[20] – est de considérer le rythme de batterie comme un assemblage de la partition ("*score*", composée des notes discrétisées) et du "*groove*", composé lui de la vélocité des notes et de leur léger décalage temporel par rapport aux notes discrétisées de la partition – nommé "*micro-timing*". Cela nous donne donc deux variables en plus des notes à encoder; et toutes deux sont continues et indépendantes entre elles. En temps normal, on aurait eu besoin de numériser les variables en plusieurs petits intervalles ou "boîtes" ("*bins*"); mais, comme l'ont habilement remarqué Gillick et al., la distribution de ces variables, notamment le *micro-timing*, se rapproche considérablement d'une distribution normale[21]. Ainsi l'on peut se permettre de modéliser ces variables de manière continue sur un intervalle donné: l'intervalle [-0.5,0.5) pour le *micro-timing* (où le 0 représente l'absence de décalage) et [0,1] pour la vélocité – représentant l'ensemble {0;..;127} des valeurs de vélocité possibles selon la convention MIDI.

Cela nous donne au final (pour la batterie uniquement) trois matrices de même taille – *H* caractérisant les notes, *V* caractérisant la vélocité des notes, et *O* caractérisant le *micro-timing* – qui, combinées, représentent le rythme de batterie à générer.

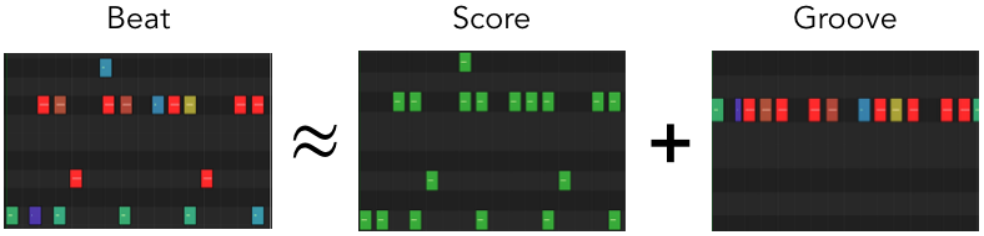

Image 4: illustration de la combinaison de la partition ("*score*") et du "*groove*" pour décrire le rythme ("*beat*"). Les tableaux représentent une mesure de rythme, dont les lignes correspondent aux différents éléments de la batterie (caisse claire, grosse caisse, toms, cymbales, etc.). Les couleurs représentent la vélocité des notes; et le décalage par rapport à la grille temporelle (lignes verticales) illustre le "*micro-timing*" des notes. Image tirée de la page internet de Magenta dédiée au modèle GrooVAE.[22]

3.2. Différentes approches dans la littérature scientifique

Il existe un grand nombre d'exemples de modèles de *machine learning* appliqué à la musique, que ce soit pour de la génération *ex nihilo* ou pour un accompagnement musical.

---

notes répétées au plus petit intervalle de temps et une longue note sont confondues dans le *piano roll*. cf. *ibid.*, p.26

[19] La notation *one-hot* a l'avantage massif de pouvoir représenter la colonne du piano roll comme un vecteur de probabilités, dont la somme des valeurs vaut donc 1. cf. *ibid.*, p. 36

[20]"Learning to Groove with Inverse Sequence Transformations", GILLICK J., ROBERTS A., ENGEL J., ECK D. et BAMMAN D., *arXiv Preprint* arXiv:1905.06118v2, 2019, p.3, §3.1

[21]cf. ibid., p. 4, §3.3

[22]"GrooVAE: Generating and Controlling Expressive Drum Performances", GILLICK J., ROBERTS A. et ENGEL J., *Magenta*, 2 mai 2019, https://magenta.tensorflow.org/groovae [consulté le 27/10/21]

L'article scientifique « Deep Learning Techniques for Music Generation – *A Survey* »[23] étudie en détail et catégorise les différentes approches utilisées jusque là. Malheureusement, on peut remarquer un très faible nombre de modèles se focalisant plus particulièrement dans la génération de batterie. Ce n'est pas la partie du domaine la plus étudiée, et les exemples sont peu nombreux.

Pour la création de mon programme, je me suis donc inspiré des deux articles scientifiques les plus pertinents à mes yeux: « Learning to Groove with Inverse Sequence Transformations »[24] et « A Hierarchical Latent Vector Model for Learning Long-Term Structure in Music »[25], tous deux issus de la plateforme Magenta, et correspondant respectivement aux modèles GrooVAE et MusicVAE.

Le premier est un modèle *seq2seq*[26], composé d'un encodeur – un LSTM bidirectionnel[27] suivi d'une couche VIB – et d'un décodeur – un LSTM générateur autorégressif. Ce modèle ne traite que des rythmes de batterie et sa tâche principale est celle dite de « *Humanization* »[28] – le but étant de générer une performance MIDI de batterie à partir d'une partition de batterie simple (pas de *micro-timing* ni d'informations sur la vélocité des notes). Le modèle ne prend donc en input que H et, après le goulot figuratif du VIB, tente de reproduire autorégressivement H, ainsi que de prédire V et O.

Le deuxième modèle, quant à lui, est un VAE "*multi-track*", qui encode donc un trio d'instruments simultanément: une mélodie monophonique, une ligne de basse, et un rythme de batterie. C'est d'ailleurs aussi un *seq2seq*, car il fut l'inspiration de GrooVAE et son prédécesseur. Il est donc aussi formé d'un BLSTM[29] en encodeur et d'un LSTM autorégressif en décodeur. Mais sa différence majeure – outre sa taille imposante, par rapport à GrooVAE et mon modèle – est son décodeur hiérarchique inédit à travers la proposition originale du RNN conducteur ("*Conductor RNN*"). Cette variante du décodeur classique segmente d'abord la séquence output en U sous-séquences consécutives distinctes. Le RNN conducteur génère ensuite un vecteur conducteur c_u pour chaque sous-séquence y_u. Ce dernier vecteur sert à initialiser le LSTM autorégressif qui va décoder la séquence finale par morceaux, tout en restant borné à sa sous-séquence y_u. Cela incite le modèle à utiliser le vecteur conducteur c_u et par extension z, représentant les séquences input. Sans ces bornes, le LSTM autorégressif aurait plutôt tendance à ignorer z, pour tenter de générer seul la séquence en sortie; et le résultat ne serait pas optimal dans ce cas de figure, car il ne tiendrait pas compte des séquences input.

---

[23]BRIOT J. et al., "Deep Learning Techniques For Music Generation – *A Survey"*, *op. cit.*

[24]GILLICK J. et al., "Learning to Groove with Inverse Sequence Transformations", *op. cit.*

[25] "A Hierarchical Latent Vector Model for Learning Long-Term Structure in Music", ROBERTS A., ENGEL J., RAFFEL C., HAWTHORNE C. et ECK D., *arXiv Preprint arXiv:1803.05428v5*, 2019, disponible sur https://arxiv.org/pdf/1803.05428.pdf [consulté le 27/10/21]

[26]"*sequence to sequence model*" en anglais; autrement dit un modèle qui manipule des séquences, en entrée comme en sortie (à l'inverse d'un modèle de classification par exemple), et qui le fait au moyen de RNNs.

[27]"*Bidirectional LSTM*" en anglais. Ce sont des LSTMs qui traitent la séquence input dans les deux sens pour ainsi générer une paire de vecteurs cachés ("*hidden states*" en anglais), le but étant d'extraire un maximum d'informations de la séquence.

[28]cf. GILLICK J. et al., "Learning to Groove with Inverse Sequence Transformations", *op. cit.*, p. 1

[29]LSTM bidirectionnel

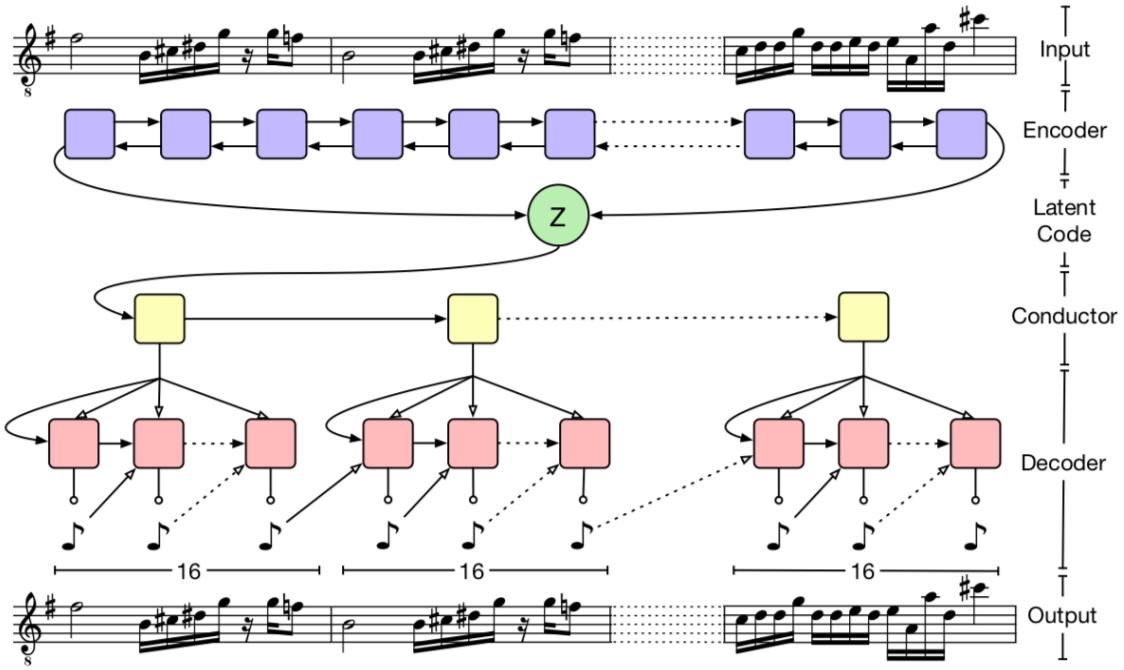

Image 5: Schéma du modèle MusicVAE, illustrant bien l'encodeur bidirectionnel et le décodeur hiérarchique inédit. Image tirée de « A Hierarchical Latent Vector Model for Learning Long-Term Structure in Music ».[30]

3.3. Création des données

Tout bon algorithme de *machine learning* se doit d'avoir un bon jeu de données. Mais dans mon cas, un tel jeu de données déjà préparé (au moins en partie) n'existait pas. J'ai donc décidé d'écrire mon propre programme pour créer mon jeu de données.

J'ai commencé par chercher un bon ensemble de fichiers MIDI comme point de départ – le Lakh MIDI Dataset (LMD)[31] –, ainsi qu'une librairie intuitive en Python pour les traiter – Mido[32]. La création du programme fut ensuite un peu fastidieuse, car la plupart des articles scientifiques que j'ai lus n'incluent pas le code utilisé pour la création des données, ou alors le format n'est pas toujours compatible (comme fut le cas du format propriétaire de MusicVAE et GrooVAE, et très peu intuitif).

D'un côté, cela ne m'a laissé de choix que d'écrire de zéro l'algorithme; d'un autre, cela m'a permis d'avoir ma propre approche et d'ainsi faire mes propres choix pour les instruments choisis, le format des données, leur degré d'abstraction et leur taux de compression (par rapport aux données musicales originales). N'oublions pas qu'une grande partie de ces choix restent arbitraires et sont plutôt dûs à la simplicité et/ou complexité de les incorporer dans l'algorithme. Notamment, je désirais bien créer un modèle capable d'accepter un nombre variable d'instruments (entre les différents types et pour chaque type). Mais la

---

[30]ROBERTS et al., "A Hierarchical Latent Vector Model for Learning Long-Term Structure in Music", *op. cit.*

[31]C'est un ensemble de chansons en format MIDI très populaire et assez précis. Pour mon modèle j'ai utilisé un sous-ensemble de celui-ci ("*lmd-matched*"), qui a été aligné et vérifié avec les chansons du *Million Song Dataset*.
Pour plus d'informations sur le *Lakh MIDI Dataset*, se référer à: RAFFEL Colin, "The Lakh MIDI Dataset v0.1", *colinraffel.com*, https://colinraffel.com/projects/lmd/ [consulté le 27/10/21]
Pour plus d'informations sur le *Million Song Dataset*, se référer à: "The Million Song Dataset", BERTIN-MAHIEUX T., ELLIS D.P.W., WHITMAN B. et LAMERE P., Dans *Proceedings of the 12th International Society for Music Information Retrieval Conference* (ISMIR 2011), 2011

[32]BJØRNDALEN Ole Martin, "Mido Documentation", *readthedocs.org*, 10 mai 2021, https://readthedocs.org/projects/mido/downloads/pdf/latest/ [consulté le 27/10/21]

complexité de l'algorithme de création de données et du modèle pour un tel choix me semblait trop grande à ce moment-là – en rétrospective, je pense qu'un tel algorithme complexe n'est pas nécessaire, les modifications pourraient se faire en grande partie directement sur le modèle; et c'est une des modifications que je vais sûrement apporter au modèle plus tard, en tant que projet personnel cette fois.

En fin de compte, j'ai donc opté pour un modèle n'acceptant qu'un nombre fixe d'instruments, un seul par catégorie, pour les catégories suivantes: guitares, basses et piano – définies par les familles d'instruments de la conventions *General MIDI*[33].

Dans mon algorithme de création des données – qui parcourt chaque chanson dans LMD – j'effectue d'abord rapidement un triage, en vérifiant si la chanson contient tous les instruments désirés. L'algorithme commence ensuite par transformer les partitions MIDI des différents instruments en matrices de type *piano-roll*; à cette étape se situe la compression, car je ne garde pas les données sur la fin de la note – et donc sur sa durée – mais seulement sur sa hauteur (sauf pour la batterie).

Cette partie de l'algorithme peut être apparentée à une sorte de fenêtre glissante de 8 mesures de longueur et traitant simultanément les matrices de tous les instruments. Celle-ci génère les échantillons recherchés à chaque pas et les combine pour créer le jeu de données, qui lui sera composé d'échantillons de 3 séquences (piano, guitare et basse) en input, et une séquence de batterie comme cible.

[33]cf. "GM 1 Sound Set", MIDI™ Association, *midi.org*, https://www.midi.org/specifications-old/item/gm-level-1-sound-set [consulté le 27/10/21]

4. Algorithme

4.1. Présentation du modèle et des choix

L'objectif de ce TM est de réussir à créer un programme qui puisse générer une séquence de batterie en accompagnement à une séquence d'autres instruments.

Pour résoudre ce problème de nombreuses approches sont possibles. Le programme doit d'abord être un modèle *seq2seq*, étant donné que l'on veut générer des séquences. Pour cette tâche, l'utilisation de réseaux neuronaux récurrents contre des réseaux neuronaux classiques est particulièrement avantageuse – en effet, un modèle n'utilisant pas de réseaux de neurones récurrents serait difficilement envisageable, à cause de sa taille et de la difficulté à l'entraîner. Ensuite, parmi les RNNs, plusieurs architectures se proposent: d'une part les modèles dits "*encoder-decoder*" (cf. §2.5.) et d'autre part les modèles dits "*Attention RNNs*"[34]. La première option est dans notre cas plus simple à implémenter et c'est la méthode qui s'applique mieux à la musique – c'est l'approche utilisée par les deux autres modèles similaires (cf. §3.2.). De plus, l'encodeur du modèle pourra être amélioré à l'aide d'une couche CNN initiale (cf. §2.4.) ou utilisant des LSTMs bidirectionnels.

Ensuite pour le décodeur, les LSTMs sont en général la meilleure solution, mais la manière dont on les implémente peut varier drastiquement[35]. J'ai testé divers types de modèles vis à vis du nombre de prédictions à chaque pas du décodeur – p.ex. une variante qui prédit une mesure entière à la fois (*multi-step model*), ou une variante qui ne prédit que la prochaine note à chaque pas (*autoregressive model*). Pour finir j'ai opté pour un modèle autorégressif dont la génération est bornée en sous-séquences – suivant l'exemple du décodeur hiérarchique de MusicVAE – et dont l'output suit la structure proposée par GrooVAE (3 types de sorties pour la batterie). Les spécificités du modèle seront expliquées dans la partie suivante.

4.2. Architecture du modèle

Pour la première partie du modèle – l'encodeur –, j'ai choisi un CNN-LSTM en tant que première couche. Celui-ci est formé de 2 couches de réseaux neuronaux convolutionnels à 1 dimension puis d'un LSTM bidirectionnel. Cette couche-là traite donc la séquence dans un sens puis dans l'autre, produisant ainsi une paire de vecteurs "cachés" (nommés *hidden states*) qui sont concaténés ensemble pour la couche suivante. J'ai répété ce groupe de couches pour les trois catégories d'instruments de mes données en entrée.

Une couche de neurones classique combine les outputs des trois groupes pour en faire ressortir un seul vecteur. Ce dernier est ensuite décliné par la partie VIB du modèle en les

---

[34]Ce type de modèle utilise une méthode particulière (démontrée dans "Attention is all you need") qui permet à l'encodeur de retenir les valeurs importantes dans la séquence input. Ensuite à chaque pas du décodeur, l'encodeur lui fournit un vecteur latent qui n'a été calculé qu'à partir des valeurs jugées importantes – pour ce pas spécifique du décodeur – dans la séquence input. Cette méthode s'applique très bien au traitement de texte ou NLP (*Natural Language Processing*), un peu moins bien à la génération de musique.
cf. "Attention is all you need", VASWANI A., SHAZEER N., PARMAR N., USZKOREIT J., JONES L., GOMEZ A., KAISER Ł. et POLOSUKHIN I., *Advances in Neural Information Processing Systems*, pp. 5998-6008, 2017

[35]Il existe un tutoriel de Tensorflow qui explique bien les différentes approches pour la prédiction de *Time series*. Bien que ce type de prédiction ne s'applique pas directement à notre cas de figure, l'analyse est utile et les types de modèles utilisés sont similaires. cf. ABADI et al., "Time series forecasting", *Tensorflow*, 13 août 2021, https://www.tensorflow.org/tutorials/structured_data/time_series#multi-step_models [consulté le 28/07/21]

paramètres des distributions – les moyennes et les variances. Et la dernière partie de l'encodeur est la couche qui échantillonne les distributions pour générer le vecteur latent z.

Ensuite vient la deuxième partie du modèle, le décodeur. La première couche de cette partie est un RNN conducteur personnalisé qui traite de manière autorégressive le vecteur latent z, afin de générer les vecteurs conducteurs correspondant à chaque sous-séquence de la séquence finale. Une couche de neurones commune traite chacun de ces vecteurs conducteurs pour en générer les états initiaux de la prochaine couche – le décodeur inférieur. Ce dernier est formé de 2 LSTMs empilés, qui décodent autorégressivement chaque sous-séquence. À chaque pas, cette double couche prend en entrée l'output du pas précédent concaténé au vecteur conducteur de la sous-séquence concernée.

Finalement la sortie des LSTMs à chaque pas est traitée par trois réseaux neuronaux classiques distincts pour générer les trois types de sorties: le vecteur *many-hot* des notes prédites, leur vélocité et leur décalage temporel par rapport à la structure discrétisée de la mesure – leur *micro-timing*.

En résumé, le modèle commence avec 3 séquences d'instruments. Puis, à l'aide de l'encodeur, il les compresse en un seul vecteur latent (z) – contenant les informations extraites des séquences input; puis le décodeur décompresse celui-ci en un rythme de batterie, décliné en ses trois composantes principales.

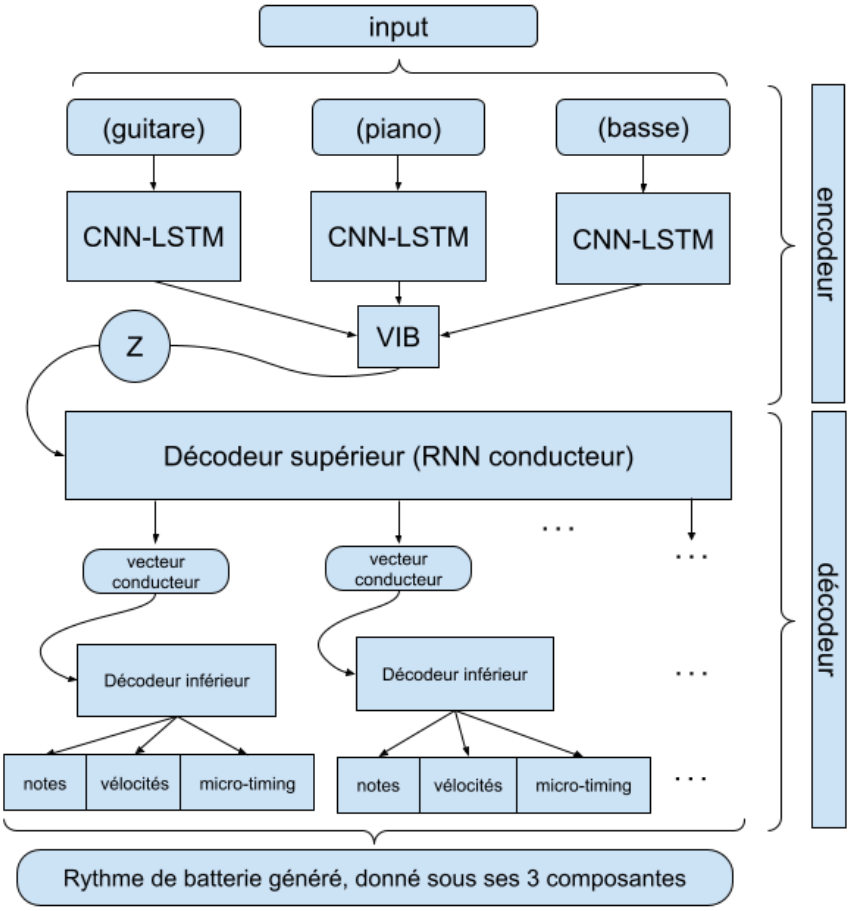

Image 6: schéma simplifié du modèle. On y voit les différentes composantes de l'encodeur (les CNN-LSTMs et le VIB), ainsi que la structure hiérarchique du décodeur, et son fonctionnement en sous-séquences. Les flèches sur la gauche d'un RNN illustrent l'initialisation de ses états avec le vecteur concerné. L'image 5 illustre aussi très bien le fonctionnement du décodeur: on y voit particulièrement le caractère autorégressif de ce dernier.

4.3. Entraînement du modèle et méthodes utilisées

Le but de l'entraînement de mon modèle est de le rendre capable de générer un bon rythme de batterie en accompagnement à n'importe quelle séquence des trois instruments choisis (cf. §3.3.). Pour accomplir ce but, on réduit le problème à des échantillons standardisés de 8 mesures de longueur. Le programme fournit donc au modèle 8 mesures d'instruments, et le pousse à tenter d'imiter la vraie batterie sur ces mêmes 8 mesures. Pour ce faire, le programme compare la batterie générée à la batterie cible (*i.e.* la vraie), afin de calculer l'erreur du modèle et de modifier ses paramètres en conséquence[36]. Ainsi le modèle, en devenant de plus en plus performant à la tâche demandée, génère un rythme de batterie de plus en plus réel.

Par ailleurs, il existe plusieurs méthodes disponibles pour l'entraînement d'un modèle, car souvent entraîner le modèle de la manière classique[37] ne suffit pas. Parmi mes nombreux tests, j'ai examiné deux méthodes particulièrement intéressantes: le *layer-wise pretraining* et le *transfer learning*.

La première méthode est une méthode itérative qui repose sur l'idée de n'entraîner qu'un nombre très réduit de couches à la fois. Traditionnellement cette méthode fonctionne de la sorte: on commence par mettre une première couche dans notre modèle que l'on entraîne jusqu'à stagnation des résultats (peu de temps pour une seule couche). Puis on "congèle" les paramètres de la couche précédente et l'on ajoute une autre couche par dessus que l'on entraîne seule à son tour. Ces étapes se répètent jusqu'à l'obtention d'un modèle profond et dont chaque couche a été suffisamment entraînée.

J'ai du légèrement adapter cette méthode à mon cas cependant. Toutes les couches étaient en place dès le début de l'entraînement, mais une seule couche – ou parfois un groupe de couches – était active à chaque instant, traversant le modèle incrémentalement du haut (les couches proches de la sortie) vers le bas (les couches proches de l'entrée). Ensuite a lieu la période d'entraînement, nommée *fine-tuning,* où toutes les couches sont décongelées, mais le taux d'apprentissage (*learning rate* ou *lr* en anglais) est réduit considérablement pour ne pas trop écraser les paramètres obtenus auparavant.

La deuxième méthode – plus utilisée dans la littérature scientifique en vue de son utilité dans sa niche de compétence – consiste à importer des paramètres d'un autre modèle pour initialiser une partie des couches initiales du modèle. Cette approche est très utilisée dans le domaine de *computer vision*, car souvent tous ces modèles possèdent des premières couches similaires et se différencient plutôt par leur dernières couches de classification. Cette étape économise beaucoup de ressources lors de l'entraînement: seules les quelques dernières couches doivent être rigoureusement entraînées, les autres ne nécessitent qu'une courte phase de *fine-tuning*.

Cette méthode, cependant, nécessite l'existence d'un modèle de structure similaire pour une tâche similaire – ce qui n'est pas exactement le cas pour mon modèle. Nonobstant cela, j'ai effectué un essai en utilisant une partie des paramètres du modèle GrooVAE pour initialiser les couches les plus importantes du décodeur, notamment les deux LSTMs principaux, le LSTM conducteur et les couches VIB.

---

[36] C'est la rétropropagation du gradient (cf. note 8)

[37] Par méthode "classique", je parle ici de la méthode plus traditionnelle du *deep learning*: on fournit simplement au modèle une grande quantité d'exemples sur une machine virtuelle puissante, et on laisse la rétropropagation "tirer" le modèle vers la perfection. Malheureusement cette méthode nécessite maintes ressources et ne permet pas toujours d'arriver en un temps raisonnable à un degré satisfaisant de précision.

Malheureusement l'essai n'a pas été suffisamment concluant sur l'efficacité du *transfer learning* (dans mon cas de figure précis). J'ai donc opté pour un entraînement se basant sur ma variante du *layer-wise pretraining*.

4.4. Modèle d'inférence et modifications

Afin de pouvoir déployer le modèle et générer des résultats utilisables – c'est-à-dire non plus sous la forme de données d'entraînement, mais sous la forme de données réelles. Il a fallu modifier et réécrire une partie du modèle.

En effet, le modèle utilisé pour l'entraînement utilisait diverses méthodes qui ne peuvent physiquement pas être utilisées lors de la phase d'inférence; notamment la méthode de *Teacher-forcing* et la couche *Dropout*. Cette première méthode – utilisée sur des RNNs autorégressifs – nécessite la connaissance au préalable des valeurs à prédire et les utilise pour forcer le modèle à ne devoir uniquement prédire la valeur au prochain pas temporel, en lui fournissant la valeur correcte du pas précédent. Ainsi les erreurs ne s'accumulent pas vers la fin de la séquence. La couche Dropout, quant à elle, est une couche censée réguler l'erreur de généralisation en ajoutant du bruit aléatoire aux sorties d'autres couches[38]. Elle rend ainsi le modèle plus robuste aux variations dans le jeu de données, lui conférant donc une meilleure capacité de généralisation. Son utilité dans le modèle en dehors de la phase d'entraînement est cependant nulle.

En outre, une grande différence du modèle d'inférence réside dans sa capacité d'accepter un input de taille variable – cela est permis grâce à la nature récurrente de la plupart de ses couches. C'est la modification la plus importante et la plus dure. Il a fallu se détourner des couches fournies par la librairie et réécrire les couches récurrentes en grande partie à la main, pour les transformer en boucles de taille variable. Fort heureusement, grâce au principe de "divulgation progressive de complexité"[39] de la librairie Keras, ces modifications ne furent pas aussi laborieuses que je le pensais.

Une fois le modèle d'inférence adapté, on peut l'intégrer au sein d'un programme plus complet, pour pouvoir pleinement utiliser la puissance de prédiction du modèle. Mais ce programme lui-même comporte certains hyperparamètres importants, qui influent beaucoup sur le rythme généré en accompagnement. Les plus pertinents sont d'abord la "température" – c'est le seuil utilisé pour déterminer quelles notes jouer, parmi la matrice de probabilités des notes générées – et la taille des sous-séquences – c'est l'espace attribué au LSTM autorégressif pour générer seul la séquence output. Pour résoudre le problème du choix de ces paramètres (et par extension celui de leur optimisation), j'ai décidé de générer plusieurs rythmes avec différentes valeurs de ces paramètres et de les classer, dans le but d'en extraire le meilleur. Pour ce faire, je définis, arbitrairement mais judicieusement, une fonction discriminante permettant de classer les rythmes générés en évaluant leur qualité. On aurait pu faire appel à un autre réseau de neurones – plus petit cette fois. Mais, pour des raisons de simplicité, j'ai opté pour une fonction évaluant la différence entre le rythme généré et un

[38]Le bruit peut se rajouter soit en désactivant la sortie de certains neurones aléatoirement, soit en ajoutant une certaine quantité de bruit à chaque sortie, tout en gardant la norme totale constante.

[39]La "*progressive disclosure of complexity*" est un principe fondamental de Keras, permettant aux utilisateurs de personnaliser leurs modèles progressivement – vis à vis de la complexité. Ainsi on n'a pas soudainement affaire aux bases mathématiques sous-jacentes pour une petite déviation des fonctions proposées par Keras, parce que celles-ci reposent chacune sur des fonctions de moins en moins abstraites et de plus en plus complexes.
cf. CHOLLET François, "Customizing what happens in *fit( )*", *Keras*, 15 avril 2020, https://keras.io/guides/customizing_what_happens_in_fit/#introduction [consulté le 27/10/21]

“rythme type” judicieusement choisi. Et cela ajoute un biais non négligeable dans le rythme final. Mais je pense que mes connaissances approfondies de batteur et les tests audio effectués pour déterminer ce “rythme type” – et par extension la fonction discriminante – suffisent pour justifier ce biais.

Enfin, je peux aussi intégrer dans le programme encadrant le modèle d’inférence une partie de l’algorithme de création des données. Ainsi, ce nouveau programme est capable d'accepter directement des fichiers MIDI en input; puis, en ajoutant à la fin de ce dernier une sorte de réciproque de l'algorithme de création des données de batterie, le programme peut générer directement un fichier MIDI en output, écoutable avec n'importe quel ordinateur ayant un logiciel adapté.

4.5. Code

L’algorithme a été entièrement programmé en Python et j’ai utilisé la plateforme Google Colab pour faire tous mes essais. Mon projet ne se décline pas qu’en un seul fichier d’ailleurs, mais en différents programmes, chacun ayant une fonction précise au sein du projet entier – que ce soit le programme de création des données, le modèle destiné à l’entraînement, les fonctions de visualisation des résultats, le modèle d’inférence final, ou tous les essais infructueux qui m’ont progressivement orienté vers le modèle final.

Afin de permettre une meilleure compréhension de mon programme, je mets à disposition de tous sur Github[40] une version commentée des fichiers les plus importants: le programme de création des données, le programme du modèle d’entraînement (qui contient un schéma bien plus détaillé que l’image 6), et le programme du modèle d’inférence. J’y mets aussi des extraits audio du modèle d’inférence, pour mieux contextualiser l’analyse des résultats qui suit.

[40]Code disponible sur https://github.com/Keboby/TM

5. Présentation des résultats et analyse

5.1. Résultats

Le but de ce travail de maturité étant après tout une réalisation, nous allons désormais évaluer la performance du modèle et la qualité de ses résultats.

Une première approche – intuitive mais, comme nous allons le voir, pas assez représentative – serait de se fier aux valeurs d'évaluation de l'entraînement du modèle. En effet, lors de l'entraînement de tout modèle, il nous faut définir une certaine mesure pour évaluer son efficacité et ainsi modifier ses paramètres en conséquence. Pour l'entraînement de mon modèle, j'ai utilisé plusieurs fonctions pour calculer l'erreur du modèle, sous différents aspects: une fonction personnalisée (une modification avec coefficients de l'entropie croisée[41]) pour les notes, l'erreur quadratique moyenne[42] pour les vélocités et le *micro-timing*, et la divergence Kullback-Leibler[43] pour évaluer la qualité de l'espace latent (*i.e.* les deux paramètres définissant le vecteur z). Et j'ai utilisé encore d'autres fonctions pour évaluer l'entraînement (aussi nommées indicateurs de performance ou "*metrics"* (en anglais)) telles que l'exactitude, la précision (valeur prédictive positive), et le rappel – tous les trois extraits de la matrice de confusion[44] entre les notes prédites et leur vraie valeur.

Nous pouvons résumer les résultats, dans le tableau ci-présent. D'un côté, l'exactitude est prometteuse. Ce résultat est cependant quelque peu illusoire, à cause du nombre bien plus élevé des notes non jouées (négatives) par rapport aux notes jouées (positives) – parmi toutes les notes possibles. Ce qui nous intéresse c'est plutôt les notes jouées (et moins les autres); une fausse note ajoutée a plus d'impact qu'une vraie note en moins – donc les faux positifs sont plus importants que les faux négatifs. D'un autre côté, la précision et le rappel laissent un peu à désirer.

Les valeurs d'erreur sont plus difficiles à interpréter. Mais on peut y voir entre autres que le modèle est particulièrement doué pour prédire correctement le *micro-timing* des notes.

---

[41]Je ne fais pas référence à l'entropie croisée telle qu'elle est définie dans les probabilités, mais plutôt à son acceptation dans le domaine du *machine learning* en tant que fonction d'erreur (ou fonction économique).
cf. KOECH Kiprono Elijah, "Cross-Entropy Loss Function", *Towards Data Science*, 2 octobre 2020, https://towardsdatascience.com/cross-entropy-loss-function-f38c4ec8643e [consulté le 23/10/21]

[42]cf. SEIF George, "Understanding the 3 most common loss functions for Machine Learning Regression", *Towards Data Science*, 21 mai 2019, https://towardsdatascience.com/understanding-the-3-most-common-loss-functions-for-machine-learning-regression-23e0ef3e14d3 [consulté le 23/10/21]

[43]cf. BROWNLEE Jason, "How to calculate the KL Divergence for Machine Learning", *Machine Learning Mastery*, 19 octobre 2019, https://machinelearningmastery.com/divergence-between-probability-distributions/ [consulté le 23/10/21]

[44]La matrice de confusion, en *machine learning*, est une manière de visualiser les prédictions dans un problème de classification souvent binaire (vrai/faux; positif/négatif; etc.). Elle comprend dans ce cas 4 catégories, les vrais positifs, les faux-positifs, les vrais négatifs et les faux-négatifs. À partir de ces catégories, on peut définir divers indicateurs de performance comme la précision, l'exactitude, le rappel, la sensibilité, la spécificité, etc. (cf. BROWNLEE Jason, "What is a Confusion Matrix in Machine Learning", *Machine Learning Mastery*, 18 novembre 2016, https://machinelearningmastery.com/confusion-matrix-machine-learning/ [consulté le 26/10/21])

Indicateurs de performance	Valeur numérique	Erreur de chaque sortie et fonction utilisée	Valeur numérique
Exactitude	91.78%	Notes [entropie croisée modifiée]	0.3437
Précision	63.82%	Vélocités [Erreur quadratique moyenne]	0.0210
Rappel	77.64%	*Micro-timing* [Erreur quadratique moyenne]	0.0054

Image 7: Tableau résumant les valeurs d'entraînement du modèle. Toutes les valeurs sont mesurées sur un jeu de données différent des données d'entraînement après 76 époques (N.B.: chaque époque correspond au temps que le modèle prend à parcourir une fois chaque lot du jeu de données). Les indicateurs de performance ci-présents ne concernent d'ailleurs que les notes prédites (donc ni les vélocités, ni le *micro-timing*) et les comparent aux vraies notes du rythme cible.

5.2. Analyse

Cependant, comme l'affirment Roberts et al. en faisant référence à leur modèle MusicVAE:

> Il est difficile de déterminer si des échantillons de notre modèle semblent réels, uniquement à l'aide d'indicateurs de performance quantitatifs. Pour comparer la qualité perçue des échantillons, nous avons donc réalisé une étude basée sur des tests d'écoute.[45]

En effet, se fier uniquement à ces indicateurs de performance de l'entraînement est trompeur pour faire un jugement sur la qualité du modèle. Car cela présuppose non seulement qu'une exactitude de 100% est atteignable, mais aussi que l'atteindre est le but. Or la tâche à résoudre n'est pas définie par une fonction mathématique idéale, mais c'est plutôt une tâche artistique et créative. Il n'existe donc pas de solution unique au problème, mais une multitude de rythmes qui satisfont les critères d'un bon accompagnement musical.

Dans cette optique, des indicateurs de performance basés sur une matrice de confusion entre les notes ne sont pas suffisants pour illustrer la qualité du modèle. Il faut considérer les résultats dans leur contexte particulier – *i.e.* au sein de la musique qu'ils sont censés accompagner – et en comparaison aux autres méthodes de génération de rythmes de batterie en accompagnement – les boîtes à rythmes ("*drum machines"* en anglais). Et c'est là que la qualité et l'utilité de mon modèle devient d'autant plus évidente.

La différence clé entre les rythmes générés par les boîtes à rythmes et par mon modèle est que ces premiers ne comportent pas de variations supplémentaires sur la vélocité et le *micro-timing* des notes. Or ces légères variations sont l'essence même d'un rythme de batterie.

En effet, comme le pointe justement Tom Barnes dans son article « Science shows why drum machines will never replace live drummers », les auditeurs ont envie d'imperfections. Les meilleurs batteurs ne jouent jamais exactement sur le tempo du

[45] *A Hierarchical Latent Vector Model for Learning Long-Term Structure in Music*, ROBERTS et al., op. cit., p.8, (trad. libre)

métronome. Leur rythme fluctue, et ils jouent parfois les notes légèrement en avance ou en retard, dépendant du style de musique. Les variations temporelles (*micro-timing*) sont de l'ordre de 10 à 20 millisecondes, mais restent pourtant perceptibles. Et ces infimes variations sont agréables à écouter. C'est pour cela que la plupart des programmes de musique électronique (cf. boîtes à rythmes) incluent des fonctions "humanisantes", qui ajoutent un peu de bruit aléatoire dans les notes, pour rendre le rythme plus humain.[46]

Cela illustre bien l'importance du *groove* (vélocité et *micro-timing*) dans un rythme de batterie. Et en analysant selon cette perspective les rythmes générés par mon modèle, on peut véritablement voir ces variations. Les caisses claires principales sont imperceptiblement en retard sur le métronome; les vélocités des charlestons ne sont pas exactement constantes, des légers accents sont visibles sur les temps; de plus, lors des doubles coups de caisse claires et grosse caisse, un accent distinct est reconnaissable sur le coup joué sur le temps, que ce soit le premier ou le deuxième coup des deux. On peut aussi remarquer une différence de volume perceptible entre les coups principaux du rythme (caisse claire et grosse caisse sur les temps) et les coups des autres éléments de la batterie, tels que les toms, le charleston ouvert, et les cymbales.

De surcroît, on peut aussi remarquer que le rythme n'est pas constant au cours du morceau – comme le serait celui d'une boîte à rythmes. D'une part, les légères variations (temporelles et de vélocité) changent un peu au fil des mesures, de sorte que le rythme ne sonne pas trop robotique et répétitif. D'autre part, le rythme en lui-même (les notes) varient aussi: certains double-coups apparaissent ou disparaissent à des moments clefs du morceaux; des petits *fills*[47] apparaissent parfois en fin de mesure; les coups du charleston varient un peu. Toute cette légère inconstance donne un *groove* plus humain au rythme généré – même si le rythme en soi (*i.e.* les notes) paraît simple.

Et il est important de noter que cette inconstance n'est pas le fruit du hasard – comme le sont certaines fonctions "humanisantes" des boîtes à rythmes. Elle est en grande partie dûe à la structure-même du décodeur hiérarchique: la couche inférieure s'occupe du rythme à petite échelle, et la couche supérieure s'occupe des variations au fil des mesures (cf. Images 5 et 6).

Donc dans la mesure où mon modèle est censé générer un rythme réaliste et relativement bon, je pense honnêtement que mon modèle est un succès.

Analysons désormais le rythme dans la perspective d'un accompagnement musical, c'est-à-dire tel qu'il se déploie dans l'entièreté du morceau qu'il accompagne.

Tout d'abord on peut remarquer l'absence de *fills* de transition entre les différents mouvements du morceau. La cause se trouve très probablement dans le RNN conducteur (couche supérieure du décodeur), de par sa petite taille – comparée aux modèles équivalents – et son ambition moindre. En effet, n'étant entraîné au départ qu'à fournir des vecteurs conducteurs pour des séquences de 8 mesures, ce dernier semble désormais avoir de la peine à orienter la couche suivante du décodeur pour des mouvements entiers (refrain, couplet,

---

[46]BARNES Tom, "Science shows why drum machines will never replace live drummers", *mic.com*, 23 mars 2015, https://mic.com/articles/113504/science-shows-why-drum-machines-will-never-replace-live-drummers [consulté le 24/10/21]

[47]Un "*fill*", dans le domaine des rythmes de batterie, est une rupture momentanée du rythme principal, incluant souvent divers toms et/ou cymbales, et dont la durée varie – allant d'un seul temps, à une mesure entière. Le *fill* sert souvent de transition entre les grands mouvements du morceau. Il faut bien le différencier du *break*, qui lui s'apparente plutôt à un court intermède instrumental, pendant lequel les autres instruments s'arrêtent. cf. CANET Frédéric, "Lexique du batteur", *Rim Shot & Ghost Note*, 26/02/2020, https://rimshotetghostnote.fr/lexique/ [consulté le 24/10/21]

etc.), qui s'étendent sur des dizaines de mesures. Aussi, une autre cause plausible pourrait se trouver dans le RNN de l'encodeur qui ne serait pas assez large – donc n'ayant pas assez de neurones – pour pouvoir "mémoriser" (et donc représenter) tout un morceau au moyen de ses 200 neurones. Même si mon modèle était performant sur 8 mesures, le saut vers un morceau entier semble avoir été trop grand. Donc à la plus grande échelle, le rythme ne semble pas trop changer: on n'y voit notamment pas de changement de cymbales pour les refrains. Et dans cette mesure, le rythme généré ne serait pas un bon accompagnement musical.

Tout de même, le modèle génère parfois des petits *fills* à la fin de groupes de 2 mesures, et précède certains mouvements (comme l'arrivée d'un nouvel instrument) par un léger changement de rythme, comprenant des double-coups de caisse claire. À mon avis, cela est dû en plus grande partie à la couche RNN inférieure, qui elle-même génère ces *fills* – illustrant la puissance d'un RNN autorégressif seul. En effet, l'output de cette dernière était borné dans des sous-séquences de 2 mesures de long. Et cela se reflète dans le rythme à travers des petits *fills* et autres motifs récurrents à cette même échelle.

5.3. Bilan de l'analyse

Ainsi on peut remarquer une différence de qualité du rythme généré à différentes échelles du morceau.

En regardant d'abord les notes individuelles ou en petits groupes, on peut remarquer une grande complexité, qui s'illustre par les variations légères du *micro-timing* et audibles de vélocité. En observant ensuite le rythme en tant que tel, on peut se rendre compte de sa qualité supérieure à ceux que l'on pourrait produire avec une simple boîte à rythmes. Enfin ce n'est qu'en agrandissant notre perspective et en étudiant le rythme tel qu'il se déploie tout au long du morceau qu'on peut constater ses quelques points faibles. Ceux-ci sont notamment le manque de *fills* de transition et de variations de rythme pour suivre les plus grands mouvements du morceau.

Néanmoins, le rythme est tout de même un accompagnement musical satisfaisant. Et, en vue de la difficulté majeure du projet, je considère cela comme un succès.

5.4. Pistes possibles d'amélioration

Après cette analyse bien complète, je pense que l'on peut en tirer diverses leçons, qu'elles soient pour une amélioration future du programme ou à titre instructif.

Tout d'abord, en vue de l'amélioration future continue de mon programme, je peux relever les causes possibles des points faibles du modèle – notamment l'envergure insuffisante de certaines couches.

D'un côté, le RNN conducteur ne semble pas avoir été suffisamment puissant pour bien diriger les couches d'en dessous. Une amélioration possible serait de réduire la taille des sous-séquences à une seule mesure, tout en ajoutant un autre RNN conducteur au-dessus du premier. Ainsi le décodeur aurait 3 niveaux de hiérarchie, correspondant aux différentes échelles de complexité d'un rythme de batterie: le rythme en soi (notes, vélocités, *micro-timing*), les motifs à moyenne échelle (*fills*, etc.) et les grands mouvements du morceau (différents rythmes pour le refrain, le couplet, etc.).

D'un autre côté, l'encodeur aussi pourrait être amélioré. Le hiérarchiser comme le décodeur pourrait résoudre le problème de son envergure temporelle limitée. Mais je pense qu'il serait plus judicieux de suivre l'exemple de MusicVAE, en utilisant plutôt des LSTMs empilés dans le RNN bidirectionnel.

Ensuite, j'ai pu constater l'importance de comparer et d'étudier les résultats dans leur contexte propre. Se fier à des simples indicateurs de performance n'est pas représentatif pour les tâches de génération créative, il faut trouver une autre méthode pour les évaluer. On peut donc se demander ce qu'il se passerait si l'on utilisait un autre modèle pour évaluer la génération créative. En suivant ce raisonnement, on arrive à la structure des GANs (*Generative Adversarial Networks*), très utilisés lors de la génération d'images – une tâche éminemment difficile à évaluer à l'aide d'une formule mathématique. Ce type de réseaux neuronaux est composé d'un générateur et d'un discriminateur. Ce dernier évalue la réalité de la génération de ce premier en la comparant à l'aveugle avec une image réelle du jeu de données, le but étant de deviner laquelle est la vraie.

C'est ce discriminateur qui nous intéresse, car il permettrait d'évaluer la dimension artistique et créative du rythme généré au sein de l'entraînement-même. Ainsi le modèle pourrait continuer à s'améliorer plus longtemps lors de l'entraînement, n'étant pas limité par des indicateurs de performance insuffisamment représentatifs. Lors de mes améliorations futures du modèle, j'essaierai d'intégrer un discriminateur, une fois que l'entraînement du modèle commencera à stagner.

Le modèle pourra ainsi être évalué non plus sur sa capacité à imiter un rythme vrai, mais plutôt sur sa capacité à générer un accompagnement vraisemblable – car, *in fine*, c'est ce que l'on cherche à atteindre.

6. Conclusion

En résumé, dans ce travail nous avons en premier lieu parcouru le thème dans sa globalité, en analysant et expliquant les diverses architectures et types de réseaux de neurones artificiels, qui nous intéressent particulièrement. En deuxième lieu, nous avons parcouru le thème du *deep learning* appliqué cette fois à la musique. Nous avons analysé les différentes méthodes de représentation des données, ainsi que les approches présentes dans la littérature scientifique pour ce domaine précis, qui furent l'inspiration de mon modèle. En troisième lieu, nous avons pu illustrer mon modèle, au moyen d'une explication détaillée des choix de son architecture, des méthodes utilisées pour son entraînement, et d'une présentation de sa variante d'inférence. En dernier lieu, nous avons évalué la performance de ce dernier à travers différents axes d'analyse, pour ensuite essayer d'en faire sortir des pistes d'amélioration possibles.

En conclusion, j'ai pu présenter, à travers ce travail écrit, l'algorithme que j'avais désiré créer, ainsi que les résultats obtenus – fruits de ce long chemin que j'ai décidé de suivre. Et ce chemin n'était pas vraiment pavé à l'avance: bien qu'il existe déjà divers algorithmes – comme les boîtes à rythmes – pour la même tâche, mon modèle est le premier algorithme de génération d'un accompagnement musical de batterie "humain" basé entièrement sur l'apprentissage profond. Mon algorithme est d'ailleurs loin d'être parfait (cf. §5.3.); il y a diverses pistes d'amélioration qui s'ouvrent devant moi (cf. §5.4). Malgré cela, l'algorithme est largement capable de générer un accompagnement de batterie, réaliste et bon. Dans cette optique, le fait que j'aie pu créer un modèle automatique qui puisse être à la hauteur des approches équivalentes peut être considéré comme un vrai succès.

De plus, la création d'un modèle de *deep learning* de telle envergure m'a permis d'apprendre une multitude de leçons, d'accumuler un savoir-faire considérable et de nettement améliorer mes compétences dans le domaine à la fois de l'informatique en général et du *machine learning* – ce qui était un de mes objectifs principaux au début de mon entreprise.

7. Remerciements

Je tiens à remercier tout particulièrement mon père Luis Goelzer – batteur et informaticien – pour ses conseils et pistes d'analyses intéressantes, particulièrement pour la partie 5.2. Je tiens ensuite à remercier mon professeur de batterie Eric Fournier, pour m'avoir donné l'idée et la motivation d'entreprendre un projet aussi difficile. Évidemment je tiens aussi à remercier mon maître accompagnant Matthieu Bouget pour ses conseils très utiles. Enfin je tiens à remercier les équipes derrière Keras et TensorFlow, pour avoir rendu le *deep learning* à portée de tous avec leur documentation à la fois simple et intuitive, mais toujours puissante.

8. Bibliographie

8.1. Livres

BROWNLEE Jason, *Long Short-Term Memory Networks with Python*, Machine Learning Mastery, disponible sur https://machinelearningmastery.com/lstms-with-python/ [consulté le 26/10/21]

BROWNLEE Jason, *Better Deep Learning*, Machine Learning Mastery, disponible sur https://machinelearningmastery.com/better-deep-learning/ [consulté le 26/10/21]

8.2. Articles scientifiques

"Magenta Studio: Augmenting Creativity with Deep Learning in Ableton Live", ROBERTS A., ENGEL J., MANN Y., GILLICK J., KAYACIK C., NØRLY S., DINCULESCU M., RADEBAUGH C., HAWTHORNE C. et ECK D., *Google*, 2019

"TensorFlow: Large-scale machine learning on heterogeneous systems", ABADI M., AGARWAL A., BARHAM P., BREVDO E., CHEN Z., CITRO C., CORRADO G. S., DAVIS A., DEAN J., DEVIN M., GHEMAWAT S., GOODFELLOW I., HARP A., IRVING G., ISARD M., JIA Y., JOZEFOWICZ R., KAISER L., KUDLUR M., LEVENBERG J., MANÉ D., MONGA R., MOORE S., MURRAY D., OLAH C., SCHUSTER M., SHLENS J., STEINER B., SUTSKEVER I., TALWAR K., TUCKER P., VANHOUCKE V., VASUDEVAN V., VIÉGAS F., VINYALS O., WARDEN P., WATTENBERG M., WICKE M., YU Y. et ZHENG X., *Google Research*, 2015, disponible sur https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/45166.pdf [consulté le 27/10/21]

COLIN Raffel, "Learning-Based Methods for Comparing Sequences, with Applications to Audio-to-MIDI Alignment and Matching",*Columbia University*, Thèse PhD, 2016, disponible sur https://colinraffel.com/publications/thesis.pdf [consulté le 27/10/21]

"Techniques for Interpretable Machine Learning", MENGNAN D., NINGHAO L. et XIA H., *arXiv Preprint arXiv:1808.00033v1*, 2018, disponible sur https://arxiv.org/pdf/1808.00033v1.pdf [consulté le 27/10/21])

"Multilayer feedforward networks are universal approximators", HORNIK K., STINCHCOMBE M. et WHITE H., *Neural Networks*, Volume 2, Issue 5, 1989, pp. 359-366, ISSN 0893-6080, disponible sur https://doi.org/10.1016/0893-6080(89)90020-8. [consulté le 27/10/21]

"Deep Learning Techniques For Music Generation – *A Survey*", BRIOT J., HADJERES G., PACHET F., *arXiv Preprint arXiv:1709.01620v4*, 2019, disponible sur https://arxiv.org/pdf/1709.01620.pdf [consulté le 27/10/21]

"Learning to Groove with Inverse Sequence Transformations", GILLICK J., ROBERTS A., ENGEL J., ECK D. et BAMMAN D., arXiv Preprint arXiv:1905.06118v2, 2019, disponible sur https://arxiv.org/pdf/1905.06118.pdf [consulté le 27/10/21]

"A Hierarchical Latent Vector Model for Learning Long-Term Structure in Music", ROBERTS A., ENGEL J., RAFFEL C., HAWTHORNE C. et ECK D., *arXiv Preprint*

arXiv:1803.05428v5, 2019, disponible sur https://arxiv.org/pdf/1803.05428.pdf [consulté le 27/10/21]

"The Million Song Dataset", BERTIN-MAHIEUX T., ELLIS D.P.W., WHITMAN B. et LAMERE P., *Proceedings of the 12th International Society for Music Information Retrieval Conference* (ISMIR 2011), 2011

"Attention is all you need", VASWANI A., SHAZEER N., PARMAR N., USZKOREIT J., JONES L., GOMEZ A., KAISER Ł. et POLOSUKHIN I., *Advances in Neural Information Processing Systems*, pp. 5998-6008, 2017

8.3. Sites

BROWNLEE Jason, "What is the Difference Between a Parameter and a Hyperparameter", *Machine Learning Mastery*, 17 juin 2019, https://machinelearningmastery.com/difference-between-a-parameter-and-a-hyperparameter/ [consulté le 27/10/21]

KOSTADINOV Simeon, "Understanding Backpropagation Algorithm", *Towards Data Science*, 8 août 2019, https://towardsdatascience.com/understanding-backpropagation-algorithm-7bb3aa2f95fd [consulté le 26/10/21]

OLAH Cristopher, "Understanding LSTM Networks", *Colah's Blog*, 27 août 2015, https://colah.github.io/posts/2015-08-Understanding-LSTMs/ [consulté le 26/10/21]

SAHA Sumit, "A Comprehensive Guide to Convolutional Neural Networks – the ELI5 way", *Towards Data Science*, 15 décembre 2018, https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53 [consulté le 27/10/21]

"Understanding Variational Autoencoders", ROCCA Joseph et ROCCA Baptiste, *Towards Data Science*, 24 septembre 2019, https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73 [consulté le 25/10/21]

"GrooVAE: Generating and Controlling Expressive Drum Performances", GILLICK J., ROBERTS A. et ENGEL J., *Magenta*, 2 mai 2019, https://magenta.tensorflow.org/groovae [consulté le 27/10/21]

RAFFEL Colin, "The Lakh MIDI Dataset v0.1", *colinraffel.com*, https://colinraffel.com/projects/lmd/ [consulté le 27/10/21]

BJØRNDALEN Ole Martin, "Mido Documentation", *readthedocs.org*, 10 mai 2021, https://readthedocs.org/projects/mido/downloads/pdf/latest/ [consulté le 27/10/21]

"GM 1 Sound Set", MIDI™ Association, *midi.org*, https://www.midi.org/specifications-old/item/gm-level-1-sound-set [consulté le 27/10/21]

ABADI et al., "Time series forecasting", *Tensorflow*, 13 août 2021, https://www.tensorflow.org/tutorials/structured_data/time_series#multi-step_models [consulté le 28/07/21]

CHOLLET François, “Customizing what happens in *fit( )*”, *Keras*, 15 avril 2020, https://keras.io/guides/customizing_what_happens_in_fit/#introduction [consulté le 27/10/21]

KOECH Kiprono Elijah, “Cross-Entropy Loss Function”, *Towards Data Science*, 2 octobre 2020, https://towardsdatascience.com/cross-entropy-loss-function-f38c4ec8643e [consulté le 23/10/21]

SEIF George, “Understanding the 3 most common loss functions for Machine Learning Regression”, *Towards Data Science*, 21 mai 2019, https://towardsdatascience.com/understanding-the-3-most-common-loss-functions-for-machine-learning-regression-23e0ef3e14d3 [consulté le 23/10/21]

BROWNLEE Jason, “How to calculate the KL Divergence for Machine Learning”, *Machine Learning Mastery*, 19 octobre 2019, https://machinelearningmastery.com/divergence-between-probability-distributions/ [consulté le 23/10/21]

BROWNLEE Jason, “What is a Confusion Matrix in Machine Learning”, *Machine Learning Mastery*, 18 novembre 2016, https://machinelearningmastery.com/confusion-matrix-machine-learning/ [consulté le 26/10/21]

BARNES Tom, “Science shows why drum machines will never replace live drummers”, *mic.com*, 23 mars 2015, https://mic.com/articles/113504/science-shows-why-drum-machines-will-never-replace-live-drummers [consulté le 24/10/21]

CANET Frédéric, “Lexique du batteur”, *Rim Shot & Ghost Note*, 26/02/2020, https://rimshotetghostnote.fr/lexique/ [consulté le 24/10/21]

9. Références algorithmiques

9.1. Documentation TensorFlow utilisée:

“Tutorials | TensorFlow Core”, ABADI M., AGARWAL A., BARHAM P., BREVDO E., CHEN Z., CITRO C., CORRADO G. S., DAVIS A., DEAN J., DEVIN M., GHEMAWAT S., GOODFELLOW I., HARP A., IRVING G., ISARD M., JIA Y., JOZEFOWICZ R., KAISER L., KUDLUR M., LEVENBERG J., MANÉ D., MONGA R., MOORE S., MURRAY D., OLAH C., SCHUSTER M., SHLENS J., STEINER B., SUTSKEVER I., TALWAR K., TUCKER P., VANHOUCKE V., VASUDEVAN V., VIÉGAS F., VINYALS O., WARDEN P., WATTENBERG M., WICKE M., YU Y. et ZHENG X., *TensorFlow*, 2015, https://www.tensorflow.org/tutorials [consulté le 26/10/21]

ABADI et al., “Time series forecasting”, *Tensorflow*, 13 août 2021, https://www.tensorflow.org/tutorials/structured_data/time_series#multi-step_models [consulté le 28/07/21]

9.2. Documentation Keras utilisée:

CHOLLET François et al., “Keras API Reference”, *Keras*, 2015, https://keras.io/api/ [consulté le 26/10/21]

CHOLLET et al., “Developer guides”, *Keras*, 2015, https://keras.io/guides/ [consulté le 26/10/21]

CHOLLET et al., “Code examples”, *Keras*, 2015, https://keras.io/examples/ [consulté le 26/10/21]

9.3. Code *open-source* des modèles MusicVAE et GrooVAE:

ROBERTS et al., “MusicVAE: A hierarchical recurrent variational autoencoder for music”, *Github*, mis à jour le 30 juin 2021, https://github.com/magenta/magenta/tree/main/magenta/models/music_vae [consulté le 28/10/21]