Algorithme génétique appliqué à la musique

Cyril Grelier – Cyril Lepinette

Encadrant :

Frederic Saubion

17 février 2020

Table des matières

1 Introduction

Les outils informatiques sont de plus en plus utilisés dans le domaine de la musique, tant pour la création que pour le traitement du signal. La communauté de l'intelligence artificielle s'est également intéressée à ce domaine en développant des outils. En effet, de nombreuses méthodes sont mises en place pour accompagner, arranger, ou encore composer dans un style.

Ce projet portera donc sur la création d'un système de composition automatique basé sur un algorithme génétique.

La génération de musique avec un algorithme génétique à été étudiée dans de nombreux articles dont le premier par A. Horner et D. Goldberg [4] en 1991. Dans l'article [6], Chien-Hung Liu et Chuan-Kang Ting présentent différentes possibilités pour générer de la musique avec des intelligences artificielles, dont une partie sur les algorithmes génétiques. Ils présentent alors trois manières d'évaluer les résultats. La première se base sur les réactions d'un public, la deuxième sur des règles d'écriture musicale et la dernière utilise des méthodes d'apprentissage. Ce rapport de projet présentera nos diverses recherches, allant de la représentation des données au choix d'une méthode d'évaluation, ceci en fonction des données disponibles et de leur utilisation possible.

2 Méthodes et algorithmes

2.1 Algorithme génétique

Les algorithmes génétiques sont aux centre de notre projet, étant la technologie imposée pour répondre au problème. C'est une branche des méthodes de résolution de problèmes d'optimisation par intelligence artificielle. Le concept découvert par John Holland et al. [3] s'inspire de l'évolution d'une espèce face à la sélection naturelle et aux changements génétiques de celle ci au cours du temps en se mêlant au reste de l'univers.

Pour résoudre un problème avec un algorithme génétique, il faut tout d'abord représenter les instances du problème en tant qu'individu. Chaque individu possède des caractéristiques permettant de résoudre le problème étudié, ces caractéristiques (parfois appelées chromosomes en analogie avec la biologie) sont représentés sous forme de tableau. À chaque individu sera attribué une note correspondant à sa capacité à résoudre le problème avec ses caractéristiques. Cette note sera appelée fitness (de l'anglais "aptitude"). Plusieurs individus seront réunis dans une population. En sélectionnant des individus dans la population en faisant évoluer leurs caractéristiques au moyen d'opérateurs de croisement et de mutation, l'ensemble des individus de la population aura tendance à obtenir une fitness qui répond de mieux en mieux au problème étudié.

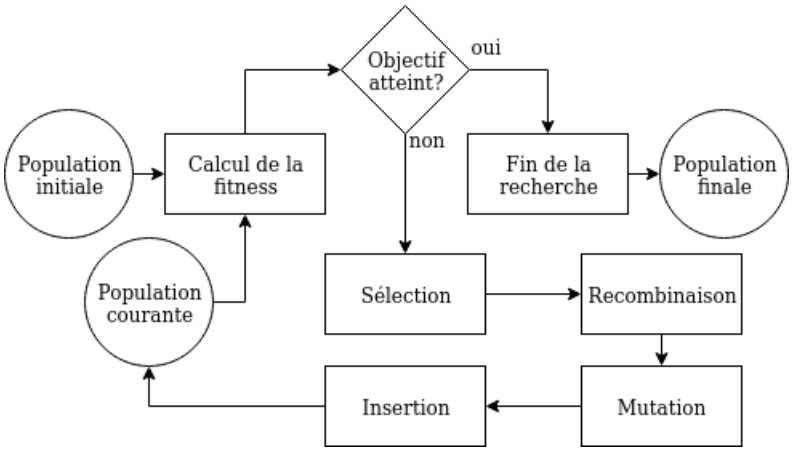


FIGURE 1 – Fonctionnement d'un algorithme génétique : L'algorithme commence avec une population initiale (avec des caractéristiques nulles, aléatoires ou prédéfinies). La fitness des individus est calculée une première fois. Tant que l'objectif n'est pas atteint, nous utilisons un opérateur de sélection d'individus sur la population, les individus choisis seront ensuite croisés (création d'individus "fils" mélange des caractéristiques des individus "parents") grâce à des opérateurs de croisement(recombinaison). Les individus créés lors du croisement seront ensuite mutés grâce à des opérateurs de mutation. Ils seront ensuite insérés dans la population en retirant les individus les plus vieux, les moins bons, les parents ou autres. Lorsque l'objectif est atteint (limite du nombre de générations ou fitness maximum atteinte) alors la recherche est terminée et nous pouvons analyser les individus obtenus.

2.2 Réseaux de neurones

Au cours de ce projet plusieurs outils ont du être mis en place afin de répondre à diverses problématiques. C'est le cas des réseaux de neurones que nous développerons ci après.

2.2.1 Réseaux de neurones classiques

Les réseaux de neurones (NN, Neural Network) sont très utilisés dans le domaine de l'informatique appliquée à la musique. La base d'un réseau de neurones est inspiré du fonctionnement de neurones dans le cerveau. Il existe des connexions entre différents neurones, lorsque l'un deux s'active, il envoie un message aux neurones connectés qui s'activeront à leur tour. Dans le cas des réseaux de neurones, le réseau est divisé en couches, chaque couche étant composée de neurones. Les entrées des neurones d'une couche sont les sorties des neurones de la couche précédente accompagné d'un poids pour chaque entrée. Chaque neurone est relié à tous les neurones de la couche précédente et à ceux de la couche suivante.

Lors de l'entraînement du réseaux, nous devons lui fournir des données. Prenons un exemple célèbre, celui de prédire les chances de survie des passagers du Titanic, les entrées sont l'âge, le sexe, la classe dans le bateau, etc...

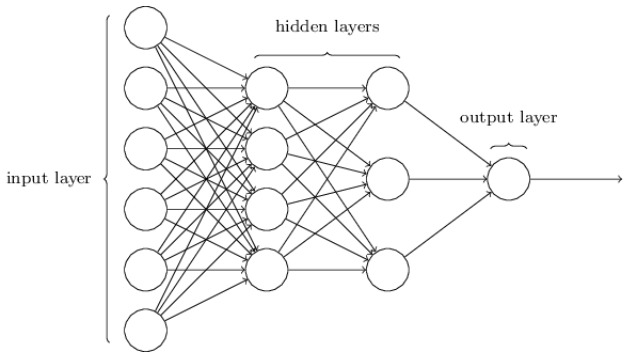


FIGURE 2 – Un réseau de neurone est composé d'une couche d'entrée et d'une couche de sortie au minimum. Des couches intermédiaires sont ajoutées en fonction de l'architecture souhaitée et du problème à résoudre. (Source : `https://meritis.fr/ia/deep-learning/`)

et la sortie du réseau est vrai ou faux si la personne en question à survécu ou non.

Nous donnons donc au réseau les entrées par lots (batch) sur lequels il fera des prédictions. Il calcule ensuite la différence entre les résultats obtenus et les résultats qu'il aurait dû obtenir et corrige les poids sur chaque neurones pour obtenir un résultat plus proche de la réalité (backpropagation).

2.2.2 Réseaux de neurones convolutifs

Les réseaux de neurones convolutifs ou CNN (Convolutional Neural networks) sont adaptés pour le traitement d'image. Ils ont la capacité de repérer des formes ou motifs dans les images en réalisant des convolutions et différentes méthodes de traitement de l'image, ainsi les premières couchent du réseau sont chargées de la reconnaissance des formes et les dernières d'établir un lien entre celles-ci. Pour la musique, les CNN peuvent être utilisés pour reconnaître des motifs récurrents dans des spectrogrammes entre autres.

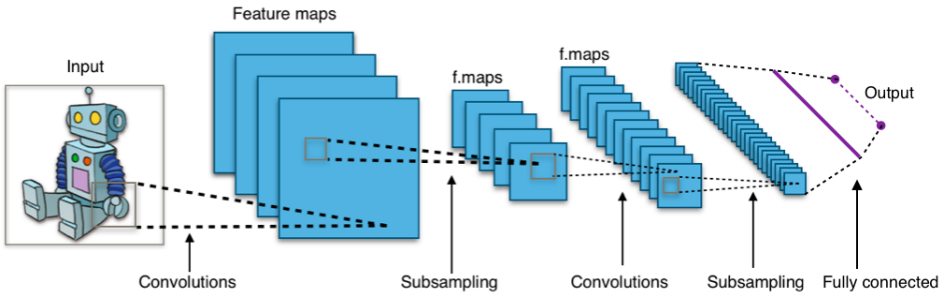


FIGURE 3 – Les premières couches du réseau extraient des informations comme des formes ou motifs et réunissent toutes ces informations avant de réaliser les prédiction avec les dernières couches du réseau correspondant à des couches complétement connectées comme dans un réseau de neurones classique. (Source : `www.wikipedia.org`)

2.2.3 Réseaux de neurones récurrents

Les RNN (Recurrent neural network) sont des réseaux de neurones récurrents. Comme pour un réseau de neurones classique chaque neurone de chaque couche est relié à la couche précédente et suivante. Il dispose cependant de connexions au sein d'une même couche permettant des cycles. Ces réseaux permettent de traiter des données dont l'information courante à un lien logique avec la précédente, comme des séries temporelles, des paroles ou du texte. Ce type de réseau est donc très bien adapté pour la musique dont les notes se suivent au cours du temps.

L'une des architectures de RNN les plus utilisées est le LSTM (Long Short-Term Memory), un réseau de neurones à mémoire à long terme et à court terme. Cette architecture dispose d'une mémoire permettant de garder les information sur une plus grande durée.

Vous pourrez trouver plusieurs projets implémentant des RNN tel que ce projet sur github : `https://github.com/Skuldur/Classical-Piano-Composer` qui génère de la musique en s'inspirant de plusieurs musiques de Final Fantasy.

2.3 Auto encodeurs variationnels

Les auto encodeurs appartiennent aux méthodes d'apprentissage non supervisées. Pour chaque exemple donné, le système va tenter de le compresser puis de le décompresser. Il comparera ensuite l'exemple original et sa reconstruction afin de s'améliorer et produire la meilleure reconstruction possible. De ce fait, les auto encodeurs apprennent comment encoder efficacement un élément à partir de son contenu. Ils sont composés de deux réseaux de neurones successifs, l'un encodant et l'autre décodant.

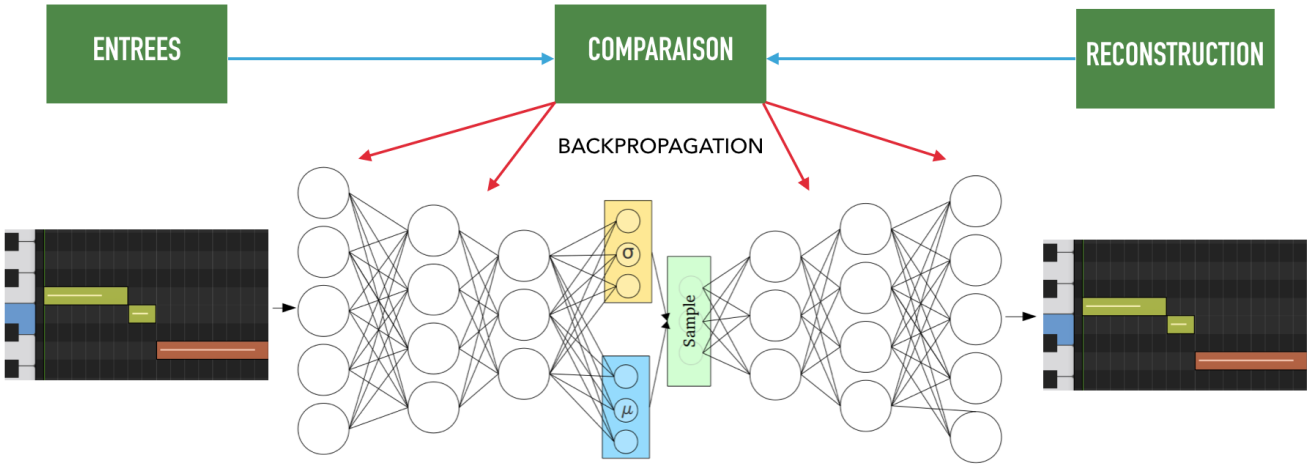


FIGURE 4 – Le premier réseau prend des données d'entrée et tente de les compresser dans un espace latent. Le second tente de reconstruire au plus juste les données à partir de la compression. Dans le cas de fichiers midi, l'auto encodeur va tenter de reconstruire au mieux la mélodie présentée en entrée.

Le premier réseau, qui encode va projeter les données compressées dans un espace latent. Si cet espace est en deux dimensions, alors nous pouvons le visualiser et obtenir une approche intuitive de son utilité.

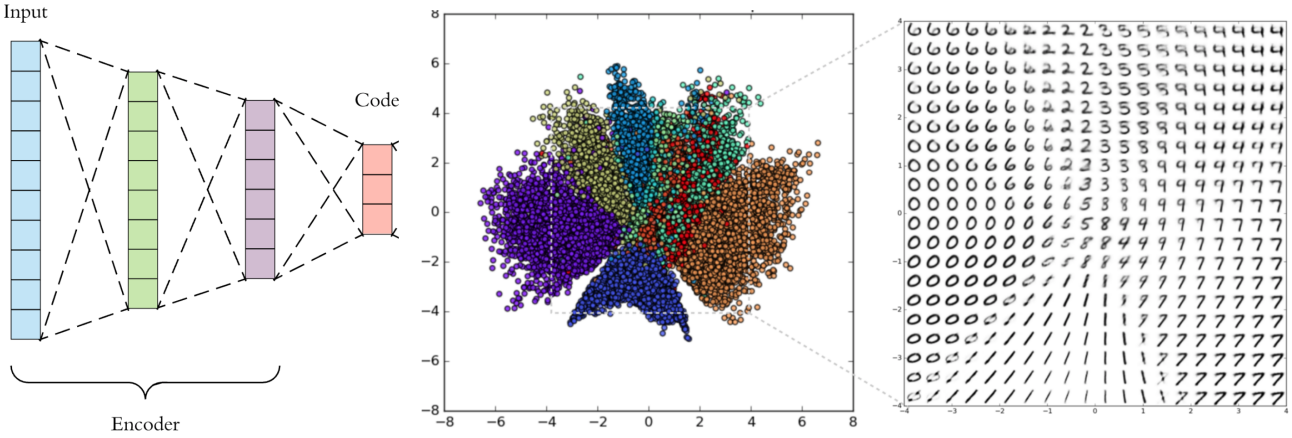


FIGURE 5 – Représentation de l'espace latent sur le jeu de données MNIST. Chaque point représente un exemple donné en entrée. Dans ce sens les auto encodeurs peuvent réaliser des clusters.

Les auto encodeurs variationnels ont la particularité de pouvoir générer une sortie à partir d'un point de l'espace latent. De ce fait, ils peuvent également réaliser des interpolations, en générant à partir d'une coordonnée située entre deux autres.

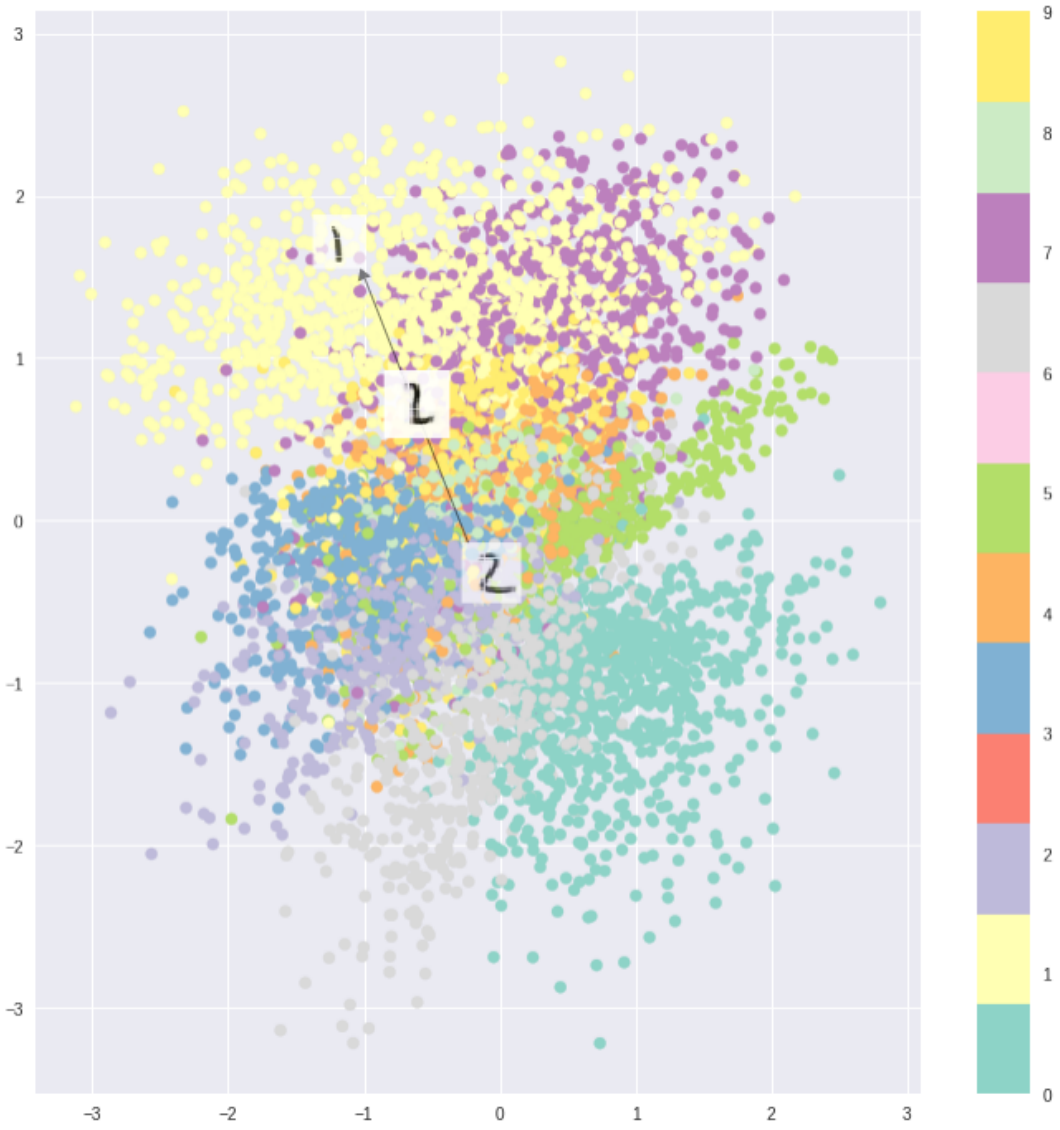


FIGURE 6 – Interpolation entre un "1" et un "2".

Ainsi les auto encodeurs, et plus particulièrement variationnels peuvent avoir plusieurs fonctions :

1. Représenter les données dans un espace latent (= réduction de dimensionnalité), clustering
2. Interpoler entre différentes sources
3. Générer à partir de l'espace latent

Tout comme pour les réseaux de neurones, il est possible d'ajouter différentes couches. Ainsi nous pouvons développer un VAE Convolutif ou bien un LSTM-VAE.

3 Modélisation

3.1 Choix du type de données

3.1.1 Audio

D'un point de vue informatique, la musique est généralement représenté par des fichiers audio : mp3, wav, flac etc. Bien qu'ils soient très répandus, ils présentent un désavantage majeur. En effet, en pratique il est impossible de les modifier. Si un fichier audio contient plusieurs instruments (ce qui est très souvent le cas), alors nous ne pourrons pas modifier les notes d'un instrument en particulier. Notre seule option possible est d'ajouter un traitement à ce fichier audio (exemple : filtrer des fréquences, le ralentir, ajouter un effet de réverbération ou d'écho, etc.). Il faut voir ces fichiers audio comme une compilation : plusieurs sources audio sont encodées dans un fichier non éditable par la suite. Ceci est peu intéressant dans notre cas car ils ne sont pas manipulables et donc inappropriés à notre problème.

3.1.2 Midi

Nous avons donc choisi de travailler avec le format des musiciens, à savoir le midi. Protocole initialement conçu pour communiquer entre les instruments de musique électroniques (ex : envoyer un do). Il est désormais devenu une norme dans le domaine de l'informatique musicale. Un fichier midi est en quelque sorte un conteneur (ou une liste) de notes. Elle pourront ensuite être jouées par tout tout type d'instruments ou être converties en partition pour le musicien. En soit, il ne produit aucun sons, ce sont ses données, envoyées à un instrument qui le produira. Il peut être virtuel (piano logiciel, batterie logicielle etc.) ou bien physique : synthétiseur, clavier arrangeur et numérique, batterie numérique etc.

L'avantage indéniable de ce format et qu'il puisse être modifié : nous pouvons prendre un message, par exemple une note et changer sa hauteur (do->ré), sa durée (croche->noire), la supprimer ou bien en ajouter une nouvelle. Nous pouvons ensuite écouter le résultat en le faisant jouer par un instrument.

Ce format est donc très adaptée pour notre problème, et c'est celui que nous utiliserons. Nous pourrons facilement réaliser les diverses mutations et croisement avec ces fichiers.

De plus, il existe de nombreux jeux de données regroupant ces fichiers. C'est le cas par exemple de la *Lakh Midi Dataset* : `https://colinraffel.com/projects/lmd/`

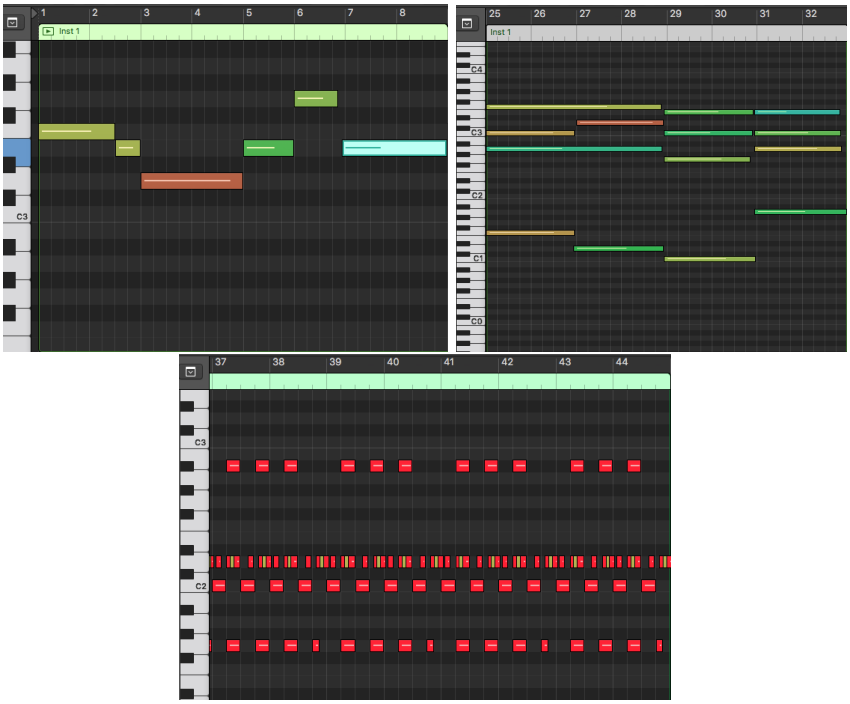


FIGURE 7 – À gauche, un exemple de représentation d'un fichier midi contenant une mélodie, à droite, des accords, en bas, une batterie. Sur l'axe des X le temps, sur l'axe des Y la hauteur de la note.

3.2 Représentation

3.2.1 Algorithme génétique

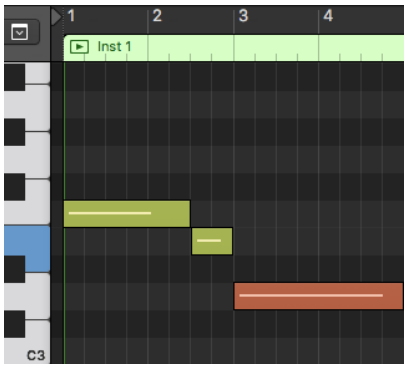
Grâce aux fichiers midis, la représentation des données est simplifiée. Afin de pouvoir communiquer entre nos algorithmes et le standard midi, nous avons développé un parser relativement simple qui lit un fichier midi pour ajouter ses données dans une classe en Python. Celle ci est très simple, c'est une liste contenant des objets *Note*. Un individu de l'algorithme génétique est donc tout simplement en ensemble de *Note*. Chaque objet *Note* est une classe contenant 3 informations :

1. Une hauteur : entier compris entre 36 et 72 représentant la position de la note sur le clavier, 48 étant un do3.
2. Une durée : flottant, représentant une croche, une noire etc.
3. Une position : endroit à laquelle la note est jouée dans la partition.

Grâce à cette structure, nous avons pu facile et intuitivement extraire les données d'un fichier midi, les modifier puis recréer un fichier lisible par les logiciels de musique.

3.2.2 Réseaux de neurones et auto encodeurs

Pour l'entraînement des réseaux de neurones et des auto encodeurs variationnels, la représentation diffère quelque peu. En effet, les données d'entrées doivent être les plus simples possibles. Dans ce sens nous avons transformé notre structure en une matrice de booléens à deux dimensions.



1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

FIGURE 8 – À gauche une mélodie simple, à droite, la matrice la représentant.

3.3 Langage et bibliothèques

Pour ce projet, nous avons choisi le Python, d'une part car beaucoup de bibliothèques pour la musique sont implémentés en Python mais aussi parce qu'il est très utilisé dans le domaine du machine learning. Pour les réseaux de neurones, après avoir commencé à utiliser le framework Pytorch, nous sommes finalement passé à Tensorflow, plus facile à prendre en main. Pour gérer les fichiers midi nous avons utilisé les bibliothèques pretty_midi, music21, magenta et MIDIUtils en coopération avec numpy pour le lien avec les réseaux de neurones. Pour l'algorithme génétique, nous avons utilisé le code développé pour le cours d'algorithmes intelligents pour l'aide à la décision, développé en Python. Le code est disponible sur github à l'adresse `https://github.com/Cyril-Grl/AlgoGen` et le package est disponible sur `pypi.org` avec la commande `pip install algo-gen`. Toutes les bibliothèques utilisées étant indiquées dans le fichier *requirements.txt*, nous vous recommandons d'utiliser un environnement virtuel tel que virtualenv pour les installer et lancer le programme.

4 Problématique de l'évaluation

Comme nous l'avons vu précédemment, un élément essentiel dans un algorithme génétique est sa fonction de "fitness" ou d'évaluation. Et voici notre premier problème rencontré :

Comment noter une musique ? Quels sont les critères, -reconnaissables par un algorithmes -qui définissent si une mélodie est belle ? A cette question philosophique "Qu'est ce que le beau" qui pourrait rappeler de longs débats dans un amphithéâtre d'histoire de l'art soulève un réel problème inhérent aux ordinateurs :

Un humain peut avec une certaine facilité, de par des conventions, différencier le beau du non, ce que l'ordinateur est incapable.

Et bien que la musique et sa beauté soit subjective, accordons nous ici à dire qu'une musique est belle si elle respecte deux critères :

1. Tonale : exit le dodécaphonisme, exit la seconde école de Vienne, exit Schönberg, exit certaines musiques du monde, exit le free jazz (assez de fausses notes dans le jazz). En somme, des harmonies occidentales.
2. Si des éléments rythmiques sont présents, alors ils doivent être en place.

Ces deux critères ouvrent sur les moyens déployés pour les satisfaire. Doit on implicitement donner des règles de musiques, d'harmonies et d'écriture occidentale, ou doit on laisser l'algorithme les déduire à partir d'exemples ?

4.1 Évaluation interactive

Le problème étant très difficile pour les ordinateurs, bon nombre d'articles tels que [1] utilisent une audience pour pallier à ce problème. Ce système interactif est simple : un public est réuni et vote à main levée à chaque nouvelle itération de l'algorithme pour dire quel individu (= musique) est le meilleur dans la population pour le garder. Ce système est irréalisable dans notre cas pour plusieurs raisons :

1. Nous n'avons pas d'audience à disposition.
2. Ce serait très long.
3. Impossible d'apporter des modifications à l'algorithme une fois lancé.
4. Il serait dommage d'avoir énormément de puissante de calcul (des milliers d'itérations par secondes) et de les limiter à 1 par minute (dans le meilleur des cas).
5. L'audience aurait du mal à voter après une centaine d'écoutes.

D'autres articles vont au delà du vote à main levée en analysant les signaux physiologiques comme le rythme cardiaque, pouls ou autre lors de la composition du morceau.

4.2 Évaluation à partir de règles

Une autre méthode possible est de définir un ensemble de règles qui pénaliseront ou récompenseront la musique. En voici quelques exemples :

1. Ne pas jouer trop de fois à la suite la même note
2. La dernière note doit être un demi-ton en dessous de la première. Lors d'un cycle, la septième sera donc résolue sur la fondamentale.
3. Les notes qui se succèdent doivent être proches en terme de hauteur. De ce fait, une mélodie continue est formée.

Le problème essentiel avec ce système est que nous pouvons dans une certaine mesure réaliser de la musique à partir de ces règles sans algorithme génétique. Il suffirait de tirer au hasard une tonalité, puis des notes dans cette tonalité et de les placer suivant un schéma rythmique. De plus, ces règles sont très restrictives à un type de musique, se rapprochant de la musique baroque, très "mathématique". Ensuite, elles réduisent quelque peu la diversité et l'exploration. Les grands génies de la musiques ont tous apportés quelque chose de nouveaux de part leur exploration. Avec un tel système nous tirons donc un trait sur une potentielle révolution musicale -saint graal des musiciens. Elles sont tout de même très avantageuses comparées à l'évaluation interactive, car elles sont calculables par un ordinateur.

4.3 Évaluation par l'apprentissage

Dans le meilleur des mondes, nous aurions donnée toute la musique existante à un algorithme qui aurait lui même déterminés des éléments récurrents à la musique. C'est une approche permise par l'apprentissage artificiel, et notamment les réseaux de neurones. Suivant l'architecture, il est possible de classifier, analyser ou regrouper les morceaux créés. Les avantages de cette méthode sont la précision des réseaux de neurones ainsi que leur capacité à générer de la musique. Le problème inhérent à cette méthode est les données d'entrées. En effet, il n'existe pas de *dataset* contenant des fichiers midis notés. Nous disposons de fichiers midis en grande quantité, mais rien ne nous indique leur qualité. Ce sous problème sera d'ailleurs étudié dans ce projet : comment à partir de fichiers existants (représentant des morceaux) peut on générer une *dataset* de notation ?

5 Approches étudiées

Afin de répondre à la problématique de l'évaluation, nous pouvons diviser le problème en suivant les deux critères précédemment évoqués, à savoir *tonal* et *rythmique*. Est il plus simple d'essayer d'apprendre un algorithme à jouer en rythme, ou lui donner une notion mélodique ?

Dans ces deux cas, nos approches tentent de répondre à ces problématiques avant tout grâce aux données que nous disposions et leur champs d'exploitation. Pour chaque approche, une *dataset* a été générée et une méthode d'évaluation choisie afin de répondre à un problème précis.

5.1 Évaluation d'éléments rythmiques par classification

5.1.1 Présentation et création de la dataset

Notre première approche pour répondre à la question de l'évaluation a été de donner un score à des fichiers midi existant puis de faire apprendre à un réseau de neurones ce qu'était un bon fichier midi ou non.

Pour ce, nous n'avions à notre disposition qu'un seul ensemble de données : des fichiers midi *bon*, provenant de musiques existantes. Notre idée a ensuite été de générer de *mauvais* fichiers, c'est à dire aléatoire.

Ayant un ensemble de *mauvais* et *bons* fichiers, nous avons par la suite procédé à une interpolation grâce à un VAE entre chacun de ces fichiers. Cependant, l'interpolation est coûteuse en temps, et doit être réalisée entre des fichiers de même taille. C'est la raison pour laquelle nous avons choisis de ne prendre que des fichiers de batteries, sur deux mesures.

En effet, découper un fichier contenant une mélodie peut poser problème : à quel moment doit on commencer à découper, allons nous couper avant la fin de la mélodie ? Le choix de batteries était donc plus judicieux, il nous a suffit de découper des parties de deux mesures aléatoirement dans les fichiers. A noter que dans ce cas, toutes les batteries jouent dans le rythme. Par contre, les fichiers aléatoires n'ont ni de cohérence entre les éléments (grosse caisse - caisse claire) ni de pattern récurrents (jouer la grosse caisse sur tous les temps).

En résumé, voici le procédé :

1. Génération de fichiers aléatoires de deux mesures
2. Découpage de batteries dans des fichiers existants
3. 20 interpolations entre un fichier aléatoire et un existant

Le fichier aléatoire a donc un score de 0, la première interpolation de 5, l'avant dernière de 95 et le fichier existant de 100. Par ce procédé, nous avons généré un jeu de données contenant 10000 fichiers. Elle est disponible dans *datasets/interpolate_ dataset*

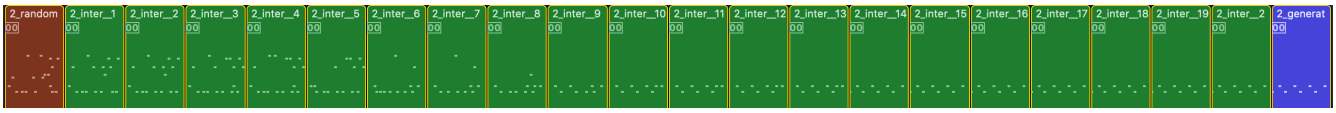


FIGURE 9 – Un exemple d'interpolation sur 20 fichiers, à gauche, des notes aléatoires, à droite le *bon* fichier, entre les deux, les interpolations.

5.1.2 Réflexions et améliorations

Les sorties de nos réseaux correspondent aux interpolations recherchées. Nous avons commencé par traiter le problème comme un problème de régression étant donné que les valeurs de sortie correspondaient à un taux d'interpolation compris entre 0 et 1 avec un lien logique entre chaque interpolation. Cependant nos modèles ne déduisaient rien et ne dépassaient pas les 15% de précision lors de l'entraînement. Puis nous avons remarqué que le réseaux fonctionnait mieux si nous lui donnions moins d'interpolations à deviner. Nous sommes alors passé à des valeurs tous les 10% d'interpolation (0, 0.1, 0.2, 0.3, 0.4, 0.5 , 0.6, 0.7, 0.8, 0.9, 1) soit 11 valeurs. Avec cette configuration, nos modèles arrivaient à 50% de réussite. Nous avons ensuite transformé le problème de régression en problème de classification en convertissant chaque sortie en tableau de taille 11. Chaque tableau correspondant à un encodage one hot de la sortie (pour 0, un 1 suivi de dix 0, pour 0.1, un 1 sur la deuxième case du tableau et des 0 sinon, pour 0.2, le 1 sur la troisième case, ... etc). La précision est alors monté jusqu'à 0.9 sur les données d'entraînement (voir plus de détail dans la partie suivante).

5.1.3 Architectures implémentées

NN

Pour les réseaux de neurones classiques, nous pourrions aplatir la matrice représentant les notes jouées au fil du temps. Cependant, traiter un tableau de 4800 booléens pour classer 11 classes ne semble pas adapté. Nous avons donc utilisé la bibliothèque *music*21 qui propose différents calculs de $features$ (caractéristiques) telles que $RepeatedNotesFeature$, $NoteDensityFeature$, $MostCommonPitchFeature$ et bien d'autres, nous avons donc utilisé ces caractéristiques pour entraîner le réseau à reconnaître les interpolations.

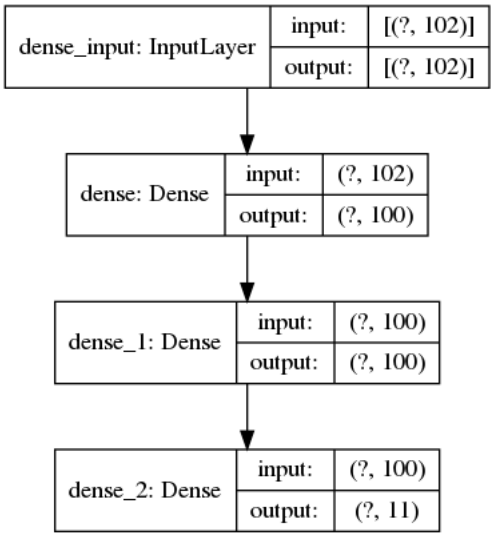


FIGURE 10 – À gauche, l'architecture du réseau de neurone mise en place. 102 éléments extraits des caractéristiques données par *music*21 en entrée puis 2 couches Dense (couches linéaires de Tensorflow) et pour la sortie 11 classes à trouver.

CNN

Les réseaux de neurones convolutifs prennent en général une image en entrée, notre matrice des notes jouées donnée par le fichier MIDI transformé en matrice booléenne jouera ce rôle, les 1 étant des pixels blanc et les 0 des noirs.

FIGURE 11 – Exemple "d'image" de mélodie. 12 notes et 400 temps durant lesquels les notes sont jouées ou non.

L'architecture du réseau de neurones convolutif étant plus imposante, nous ne l'affichons pas dans le rapport mais vous pourrez la trouver dans le fichier *src/modeles.py* dans la fonction *cnn*1. L'architecture ce présentant comme ceci, deux couches de convolution suivis d'une couche pour aplatir leurs sorties. Puis trois couches Dense accompagnées de couches de dropout avec la sortie sur 11 classes. Les couches de dropout servent à désactiver des neurones lors de l'entraînement pour éviter le sur-apprentissage.

RNN

Le RNN prend en entrée la matrice de 0 et 1 correspondant aux notes jouées et analyse la suite des notes pour trouver un lien entre l'entrée et le niveau d'interpolation.

L'architecture du réseau de neurone récursif le plus performant que nous ayont produit ce compose de deux couches de LSTM (Long Short-Term Memory) servant à gérer la récursion puis d'une couche Dense suivie de la couche de sortie avec les 11 classes. Les différentes versions de RNN sont présentes dans le fichier *src/modeles.py*.

5.1.4 Résultats et observations

NN

Les réseaux de neurones classiques entraînés sur les caractéristiques données par *music*21 donnaient de bons résultats sur les données d'entraînement(environ 0.92 de précision au maximum). Sur les données de test, nous obtenons une précision de 0.39.

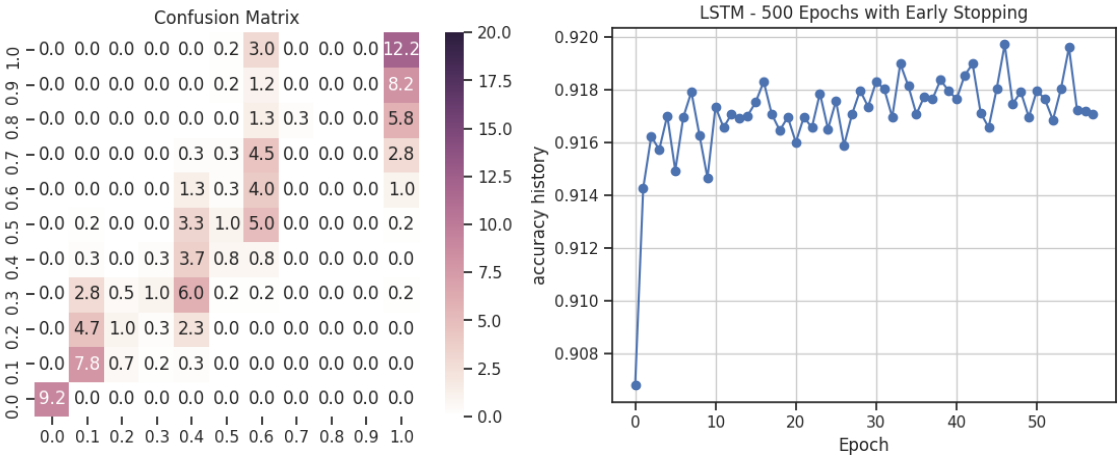


FIGURE 12 – Voici la matrice de confusion (vraie interpolation en fonction de l'interpolation prédite) et la précision lors de l'entraînement du NN. La matrice de confusion est relativement inégale, il prédit beaucoup d'interpolation au dessus de 0.7 comme étant de bons morceaux.

CNN

Les réseaux de neurones convolutifs entraînés avec les images des fichiers midi atteignent 0.97 de précision sur les données d'entraînement et 0.39 sur les données de test.

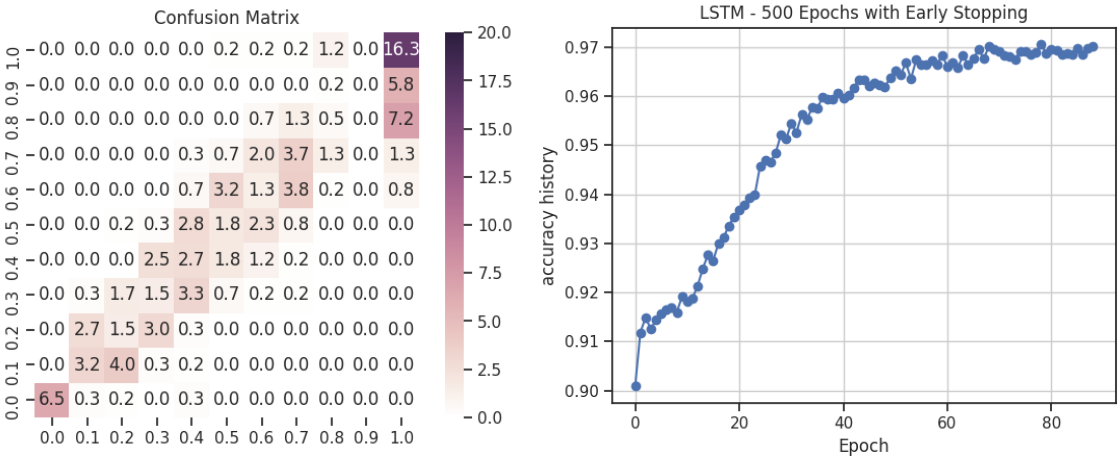


FIGURE 13 – Voici la matrice de confusion (vraie interpolation en fonction de l'interpolation prédite) et la précision lors de l'entraînement du CNN. La matrice de confusion est mieux répartie malgré une difficulté pour les interpolations supérieures à 0.8.

RNN

Les réseaux de neurones récursifs présentent les meilleurs résultats sur les 3 méthodes testés. Ils dépassent les 0.97 de précision en moins d'époques d'entraînement que les CNN et ont une meilleure précision sur les jeux de tests, 0.47.

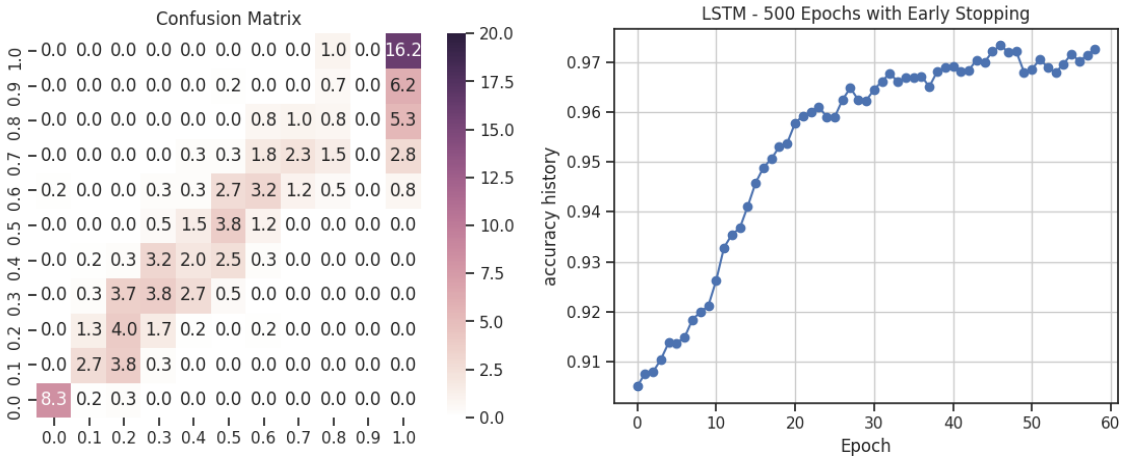


FIGURE 14 – Voici la matrice de confusion (vraie interpolation en fonction de l'interpolation prédite) et la précision lors de l'entraînement du RNN. La matrice de confusion est plus précise, malgré encore une fois, la difficulté pour les interpolations supérieures à 0.8.

Nous avons donc décidé d'utiliser la notation avec Le RNN pour l'utilisation avec l'algorithme génétique.

5.1.5 Mutations de l'algorithme génétique

Les mutations de l'algorithme génétique dans ce cas sont :

1. Ajouter/Retirer une note
2. Déplacer une note vers le haut ou vers le bas (changer l'élément rythmique)

Observations sur la qualité des fichiers générés

Du fait que la qualité des fichiers générés ne soient pas au rendez-vous, nous supposons que les données extraite par music21, ne soit pas pertinentes pour des pattern de batteries. En effet, beaucoup trop se basent sur des mélodies et hauteurs de notes. Pourtant le réseau arrive tout de même à en déduire quelque chose. Il est aussi possible que le réseau de neurones apprenne d'autres liens entre les données que nous ne sommes pas capables de voir. Il faudrait faire une étude sur l'explicabilité de notre réseau pour mettre en avant le processus qui lui font prendre ces décisions.

Afin d'améliorer cette approche il faudrait travailler plus sur le jeu d'entrées, qui présente des défauts (parfois deux fichiers interpolés sont identiques et faussent donc le résultat).

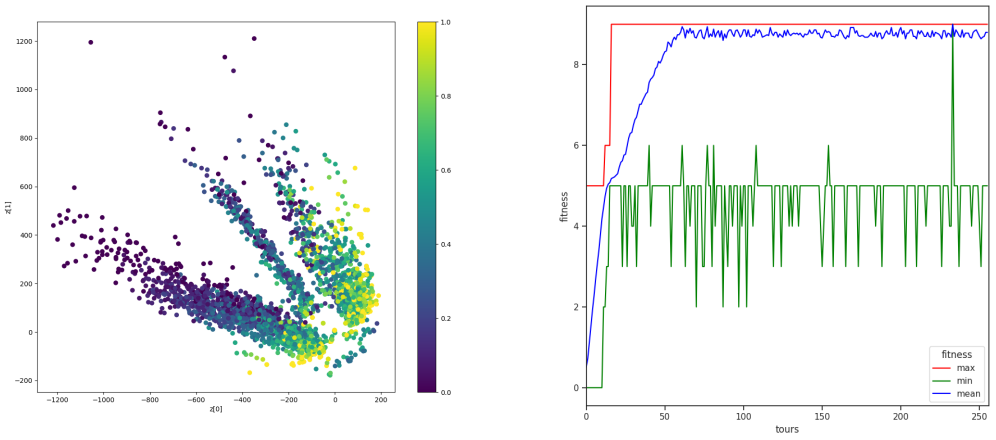


FIGURE 15 – À gauche, la projection de notre dataset dans l'espace latent de notre VAE. La couleur des points représente leur score. Nous observons que les différents points convergent vers l'origine. L'interpolation produite est donc en corrélation son espace latent. À droite, le résultat de l'exécution avec l'algorithme génétique. 10 étant le score maximal atteindre 9 peut sembler impressionnant mais le résultat à l'écoute l'est beaucoup moins.

5.2 Correction d'éléments rythmiques par similarité

5.2.1 Présentation

La dataset précédente présentant des défauts liés à l'interpolation, nous avons essayer d'ajuster notre approche. Cette deuxième a donc été d'apprendre à une batterie de jouer en rythme. Tout comme dans la première approche, nous disposons de fichiers rythmiques *bons*. Cette fois-ci, au lieu de générer des fichiers aléatoires qui jouent dans les temps, nous en avons générés qui ne jouent pas dans les temps.

Ainsi, le nouveau problème est de corriger un fichier qui ne joue pas en rythme vers un qui joue en rythme.

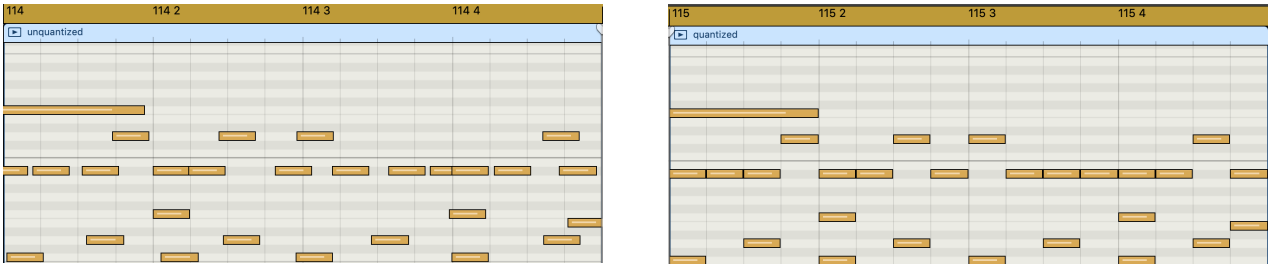


FIGURE 16 – A gauche un mauvais pattern de batterie qui n'est pas dans les temps, à droite la version corrigée.

Pour ce, nous avons également généré une dataset contenant 5000 fichiers qui jouent en rythme et 5000 aléatoires qui ne jouent pas en rythme. La dataset est disponible dans */datasets/quantized_rythm_dataset_v2_temperature*

Afin d'exploiter des données, nous avons développé 3 types d'auto encodeurs variationnels : classique, convolutionnel et récurrent (lstm). Ils sont disponibles à l'adresse : `https://github.com/KyrillosL/SimpleVAEMidi`

Dans un premier temps, nous avons encodé toutes nos données dans l'espace latent de l'auto encodeur. Voici ce que nous avons obtenu :

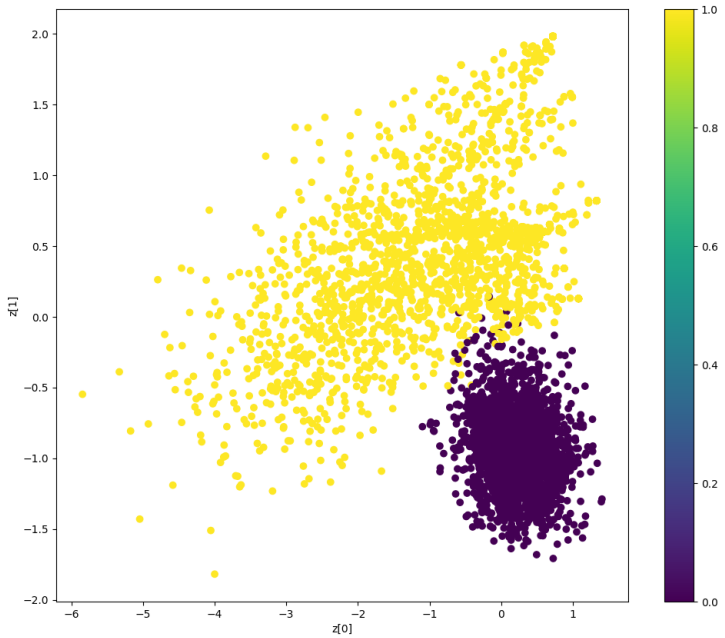


FIGURE 17 – Espace latent de notre VAE. En bleu les fichiers aléatoires, en jaune les fichiers *bons*

Afin de noter un fichier midi produit par l'algorithme génétique nous avons donc utilisé cet espace latent. A chaque nouveau fichier, nous le donnons au vae et le projetons dans l'espace latent. Le score de ce fichier est sa distance avec le centroide des *bons* fichiers, ici (0,0).

Intuitivement, ce que nous essayons de faire ici est de prendre un point dans l'ensemble des aléatoires et le faire converger vers un point dans l'ensemble des *bons*.

5.2.2 Architecture du VAE

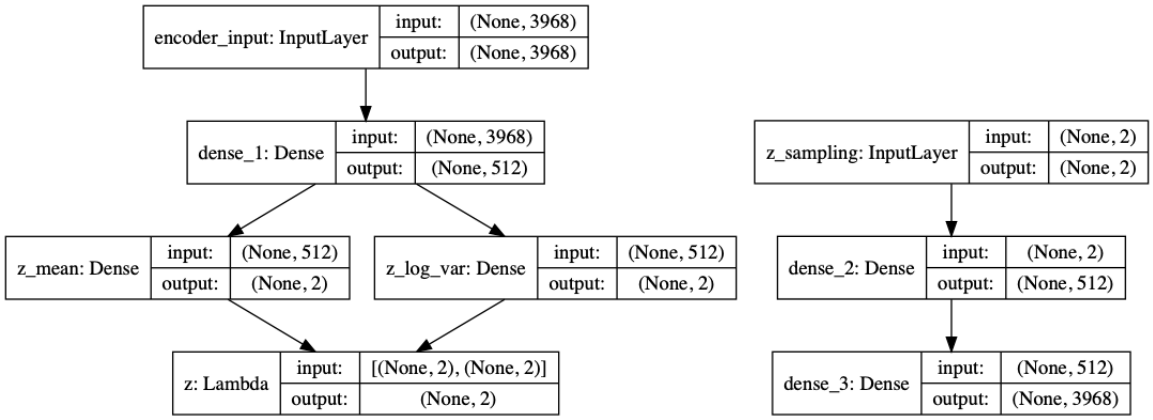


FIGURE 18 – L'encodeur à gauche, le décodeur à droite.

Nous remarquons bien que l'encodeur prends un fichier midi aplati dans une matrice (3968 = 16 notes sur une résolution de 248) et le compresse dans un espace à deux dimensions. Le décodeur prend bien ces deux dimensions pour reconstruire une matrice aplatie de 3968.

5.2.3 Mutations de l'algorithme génétique

Les mutations de l'algorithme génétique dans ce cas sont :

1. Ajouter/Retirer une note
2. Déplacer une note vers la gauche ou vers la droite (=tenter de la remettre dans les temps).

5.2.4 Résultats et observations

Les résultats n'ont pas étés très convaincants et nous nous sommes aperçu d'un biais qui faussait les résultats : les fichiers aléatoires contenaient plus de notes et le réseau a appris qu'un mauvais pattern de batterie était un pattern contenant beaucoup de notes. De ce fait, plus l'algorithme fait d'itérations, plus il enlève de notes. Afin d'améliorer cette approche nous aurions pu essayer de générer une dataset contenant exactement les mêmes notes pour les fichiers *bons* et *mauvais*, mais les mauvais n'étant pas dans les temps.

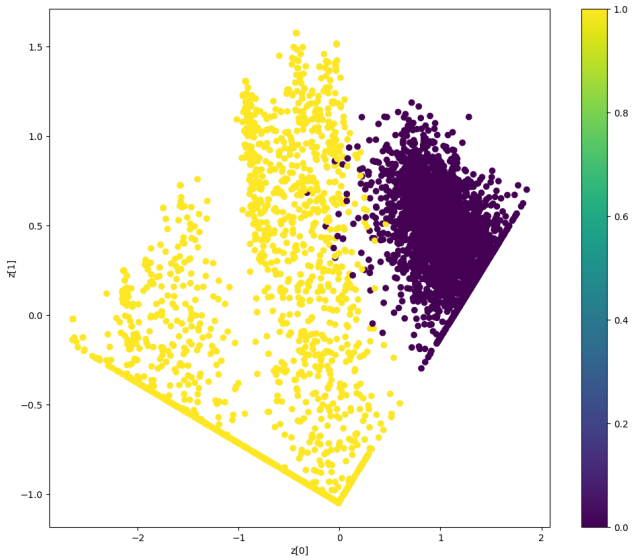


FIGURE 19 – Nous pouvons également noter que les hyper paramètres sont très important dans ces systèmes d'apprentissage. Voici un exemple ou une couche cachée était trop petite et où le vae avait du mal à apprendre :
Un problème de dimension sur une couche cachée provoque une altération des résultats.

5.3 Génération de mélodies par Reinforcement Learning

5.3.1 Présentation

La dernière approche est celle qui a produite les meilleurs résultats. Elle combine à la fois une évaluation par apprentissage (réseau de neurones récurrents) et par règles. Ces deux évaluations sont tirées de l'article [5] et le code est disponible à l'adresse :
`https://github.com/tensorflow/magenta/tree/master/magenta/models/rl_tuner`.

Afin de pouvoir donner un score à une mélodie, les auteurs procèdent à une notation progressive : ils prennent successivement les notes de la mélodie et les évaluent relativement à la précédente. L'intuition derrière ce fonctionnement est de comparer la note dans la mélodie et celle prédite par le réseau de neurone récurrent.
Par exemple, soit une mélodie *do mi fa ré#*, nous avons déjà analysé *do mi fa*. Le rnn prédit la note suivante *la*. Le système compare la prédiction avec la note contenue *ré#* : elles sont éloignées, donc le score global de la mélodie est diminuée.

L'itération suivante, grâce à une mutation réalisée par l'algorithme génétique, nous obtenons la mélodie *do mi fa la*. Cette fois le rnn va donner un meilleur score à cette mélodie car en adéquation avec sa prédiction.

En plus du rnn, un ensemble de règles vient compléter l'évaluation. Ces règles sont basées sur l'article [2]. En voici quelques unes :

— Évaluer la tonalité
— Évaluer la tonique
— Pénaliser les répétitions
— Évaluer le motif
— Évaluer l'intervalle

Voici un récapitulatif du fonctionnement :

1. L'automate génère une mélodie comme une liste de notes
2. Tant que toutes les notes de la liste n'ont pas étés visitées :
   (a) Évaluer la note par le rnn
   (b) Évaluer la note par les règles
   (c) Faire la somme et ajouter ce score pour la note dans la liste
   (d) Passer à la note suivante
3. Faire la moyenne des notes

La moyenne n'étant pas forcément représentative de la qualité globale de la mélodie nous avons mis des poids sur les notes. En effet, nous préférons avoir une mélodie globalement moyenne mais qui se tient plutôt qu'une mélodie ayant de très bonnes parties et de très mauvaises. Pour ce, nous avons pénalisé la pire note en lui associant un gros poids (0.7 contre 0.3 pour la moyenne) dans l'évaluation. Ceci permet à l'algorithme génétique d'augmenter l'évaluation générale de la mélodie et de le forcer à améliorer la plus mauvaise note.

A noter que cette architecture ne marche que pour les mélodies monophoniques, et c'est la raison pour laquelle nous avons abandonnés les batteries dans cette partie. En effet, le système prenant les notes successivement, il ne pourrait pas différencier un deux notes d'un accord de deux notes successives.

L'aspect rythmique des mélodies est générée aléatoirement par un automate que nous avons développé. En choisissant un signature rythmique de 4/4 nous obtenons cet automate.

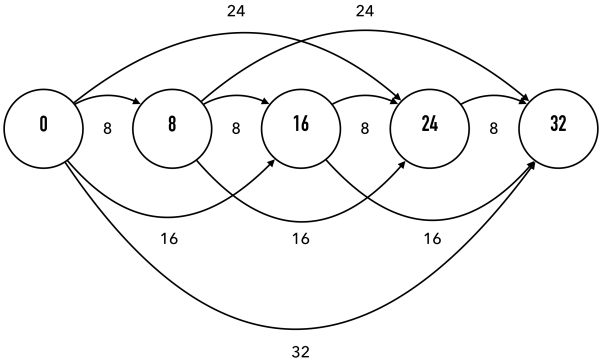


FIGURE 20 – Automate générant l'aspect rythmique

Nous n'avons pas eu de dataset à générer car le rnn était déjà entraîné et les règles n'en nécessitent pas.

5.3.2 Mutations de l'algorithme génétique

Les mutations de l'algorithme génétique dans ce cas sont :

1. Ajouter/Retirer une note
2. Déplacer une note vers le haut ou vers le bas (do->do#)

5.3.3 Résultats et observations

Le cumul des deux fonctions (et notamment l'évaluation des règles) est très coûteux et de ce fait le système est très lent. Nous ne faisons qu'une itération toutes les deux secondes, contre des milliers précédemment. Cependant c'est l'approche qui nous a donné les meilleurs résultats. On a également remarqué que les résultats étaient moindre avec une seule évaluation (règle seulement ou rnn seulement) qu'avec le cumul des deux. L'ajustement de la mélodie par une base de connaissance humaine est donc nécessaire. Pour l'améliorer nous pourrions ajouter des règles, et notamment sur le rythme afin qu'il soit pris en compte. Cela permettrait de répéter à la fois des motifs mélodiques et rythmiques, renforçant l'homogénéité des mélodies.

6 Conclusion

Ce projet a été très enrichissant car il nous a permis de découvrir et de mettre en application de nouvelles technologies. En effet, nous avons pu aborder diverses méthodes d'apprentissages via les bibliothèques Tensorflow et Keras, très utilisées à la fois dans la recherche et dans l'industrie. La plupart des architectures de réseaux de neurones les plus répandus à l'heure actuelle ont étés implémentés et nous avons pu les comparer. Nous avons également pu souligner que les résultats dépendent essentiellement des données en entrée. Dans ce sens, il faudrait investir du temps dans le domaine de l'explicabilité des méthodes par apprentissage. L'utilisation d'un outil tel que LIME, qui met en avant les éléments déterminant à l'apprentissage pourrait nous éclaircir, et notamment pour la partie NN et CNN.

La problématique de ce projet n'a donc pas été de générer de la musique mais bel et bien de l'évaluer, fonctionnalité indispensable dans un algorithme génétique.

Nos résultats des diverses approches étudiées nous ont permis de mettre en évidence la nécessité de faire intervenir la connaissance humaine dans ce type de problématique, afin d'aider la partie apprentissage.

Références

[1] Özcan Ender. A genetic algorithm for generating improvised music. 2008.

[2] Robert Gauldin. A practical approach to 18th century counterpointy. *Intégral*, 3 :227–234, 1989.

[3] John H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI, 1975. second edition, 1992.

[4] Andrew Horner and David Goldberg. Genetic algorithms and computer-assisted music composition. *Urbana*, 51 :437–441, 01 1991.

[5] Natasha Jaques, Shixiang Gu, Richard E. Turner, and Douglas Eck. Tuning recurrent neural networks with reinforcement learning. 2016.

[6] Chien-Hung Liu and Chuan-Kang Ting. Computational intelligence in music composition : A survey. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 1 :2–15, 2017.