Génération de musique à l'aide d'un modèle génératif type Transformer

PHÉLIZOT Yvan
Année 2021 / 2022 - Semestre 2
RCP217 - Intelligence artificielle
pour des données multimédia

le cnam

Introduction

Au cours des 10 dernières années, l'intelligence artificielle a fait des progrès incroyables. Son application à de nombreux domaines donne des résultats dépassant toutes les attentes, étant souvent meilleurs que des experts sur le sujet: identification d'objets, génération de "Deep Fakes" (Vidéos avec remplacement du visage d'un acteur), production de nouveaux médicaments mais aussi de nouvelles armes chimiques[1], joueur virtuel aux capacités surhumaines comme le Go ou StarCraft…
Parmi toutes ces applications, nous nous sommes intéressés dans ce projet à la génération de musiques à partir d'un jeu de musiques en entrée. On peut imaginer l'intérêt des producteurs de pouvoir produire une nouvelle chanson originale des Beatles qui raviront les fans.
L'objectif du projet est de partir d'un jeu de données en entrée et d'appliquer les connaissances issues du cours RCP217 à la génération de musique. Dans une première partie, nous introduisons la notion de musique et le problème que nous souhaitons résoudre. Ensuite, nous décrivons le modèle choisi, ses avantages et ses inconvénients. Enfin, les différents résultats obtenus sont présentés et analysés. Différentes pistes d'améliorations du modèle sont aussi discutées.

Le code du projet est disponible ici: https://github.com/cotonne/rcp217

[1] https://www.nature.com/articles/s42256-022-00465-9

Table des matières

Présentation du problème

La musique, le 4è art

Selon Wikipédia, "*La musique est un art et une activité culturelle consistant à combiner sons et silences au cours du temps*"[2].
Pour produire de la musique, nous utilisons différents instruments, par exemple un violon, une flûte ou encore la voix. Les différents instruments produisent des notes (Do, Ré, Mi, … dans la musique occidentale). Une note a une certaine durée et une hauteur. Les notes sont jouées à un certain rythme.
Les notes assemblées ensemble donnent des mesures. En les regroupant, on obtient des phrases, puis des sections. Ces sections peuvent être construites selon une certaine forme comme la forme ternaire A-B-A où la section A est jouée au début et à la fin pour créer une forme de répétition. On retrouve aussi des structures classiques de couplet-refrain.
Différents instruments peuvent aussi être combinés dans une musique. Chaque instrument peut suivre un style plutôt centré sur le rythme ou sur la mélodie. Des jeux entre instruments sont aussi possibles où les instruments se répondent les uns les autres.
A cela s'ajoute le style de la musique comme le jazz, le blues, le rock… qui imposent des règles de construction particulières à la musique.
Dans ces quelques paragraphes, nous avons évoqué quelques éléments constitutifs de la musique. Cependant le domaine est très riche. Si nous souhaitions créer un programme capable de produire de la musique à partir de règles métier, il pourrait être envisagé de coder les différentes règles. Nous aurions besoin d'un expert dans le domaine capable de nous expliquer comment les définir. Cependant, un expert seul ne pourrait avoir une connaissance exhaustive de ce domaine.
La musique présente des régularités dans sa construction. Elle respecte des règles de structure que l'on pourrait comparer à la grammaire d'une langue. On retrouve des structures répétitives, proches de la prose ou de la poésie (phrase, paragraphe, grammaire). La musique est donc un langage avec son vocabulaire et sa grammaire. De cette analogie avec le langage, nous allons utiliser les outils et les méthodes vues lors de RCP217 utilisés pour la génération de texte pour l'appliquer à la génération de musique.

Objectifs

L'objectif de ce projet est de construire une application de génération de musique. Plusieurs modèles génératifs existent et peuvent être sélectionnés. Il va donc s'agir de construire un modèle génératif qui va apprendre les régularités des différentes musiques et produire, à partir d'une entrée, une nouvelle musique.
En RCP211, nous avons vu différents modèles comme les "Generative Adversarial Networks" (GANs) ou les "Variational Autoencoders" (VAEs). Dans le cadre de RCP217 et du fait de la proximité avec le langage, nous allons nous appuyer sur les mêmes modèles utilisés que pour le Traitement du Langage Naturel.

[2] https://fr.wikipedia.org/wiki/Musique

Le modèle le plus efficace à ce jour est à base d'une architecture type Transformer.

"Music Transformer"

L'article "**Music transformer: Generating music with long-term structure**" a été utilisé comme base pour ce projet. L'équipe de recherche a utilisé une architecture avec un Transformer pour pouvoir produire de la musique.
Il est possible de représenter la musique sous plusieurs formes. On peut voir la musique comme une onde sonore qui varie au cours du temps. On pourrait imaginer qu'un modèle génératif produise cette onde. Cependant, dans le cas d'un transformer, la complexité calculatoire est de $O(n^2 * d)$ avec n, la taille de l'entrée et d, la taille du vocabulaire. Or, pour l'équivalent d'une seconde, en respectant le théorème de Shanon - Nyquist et la capacité d'échantillonnage de l'oreille , il faudrait une entrée égale à n=44000 valeurs.
Une solution est d'utiliser une entrée différente. Pour ce faire, le format MIDI a été utilisé et transformé pour le compresser encore plus. Cela permet d'augmenter la quantité d'information en entrée du modèle sans perte.
Le jeu de d'entrée pour ce modèle a été le "Piano-e-Competition".

Dans le cadre de ce projet, nous avons choisi d'utiliser le format MIDI comme format d'entrée. Le jeu de données utilisé est un sous-ensemble du "Lakh MIDI Dataset[3]", constitué de 45,129 fichiers. Ces fichiers sont dépourvus d'erreurs et sont associés à un jeu de données plus gros, le "Million Song Dataset".

[3] https://colinraffel.com/projects/lmd/

Modèle

Dans ce chapitre, nous présenterons en détail le format utilisé en entrée, le format MIDI et l'architecture Transformer.

Représentation des données

Format MIDI

Le format de fichier MIDI est un format de fichier permettant de stocker les commandes MIDI. MIDI, pour Musical Instrument Digital Interface, est un protocole binaire normalisé dans les années 1980 et qui à pour but de standardiser la communication entre les différents éléments intervenant dans la production musicale (logiciels, instruments…). Le fichier MIDI stocke les commandes de ce protocole MIDI: note, durée, hauteur, instrument...

Le protocole définit des messages avec les informations suivantes :

- Si la note doit être jouée (ON) ou arrêtée (OFF)
- La note à jouer
- De l'intensité avec laquelle la note est jouée
- Le canal

Les messages sont au format binaire. Par exemple, le message suivant "**92 3C 64**" se traduit par:

- 92
    - 9: message de type "NOTE ON"
    - 2: canal 2
- 3C: note 60 (0x3C en hexadécimal) ce qui donne la note "middle C" ou "Do du milieu". Par exemple, pour la note "LA", les valeurs sont les suivantes

MIDI note number	Key number (61 keys)	Key number (88 keys)	Note names (English)	Frequency (Equal tuning at 440 Hz)
69	34	49	A4 concert pitch	440.00

(Réf: LA : https://www.inspiredacoustics.com/en/MIDI_note_numbers_and_center_frequencies)

- 64: vélocité, la force avec laquelle la note est frappée

D'autres messages sont présents comme des commandes de modulation ou de début/fin de musique.

Du fichier MIDI au batch d'apprentissage

La lecture du fichier MIDI se fait avec la librairie Python MIDO[4].

[4] https://github.com/mido/mido

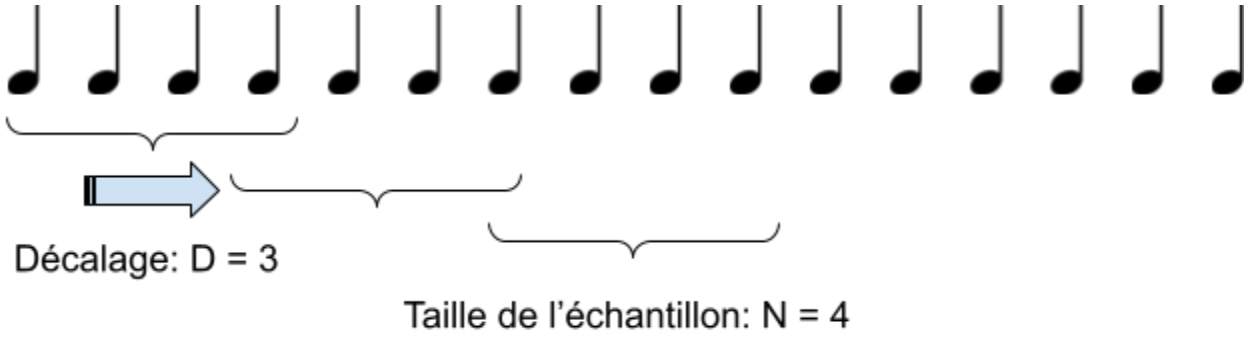
```
mid = MidiFile('new_song.mid')
for i, track in enumerate(mid.tracks):
   for msg in track:
       print(msg.dict())

# {'type': 'program_change', 'time': 0, 'program': 12, 'channel': 0}
# {'type': 'note_on', 'time': 12, 'note': 69, 'velocity': 27, 'channel':
0}
# {'type': 'note_off', 'time': 4, 'note': 69, 'velocity': 64, 'channel':
0}
# {'type': 'end_of_track', 'time': 0}
```

La création d'un fichier MIDI se fait aussi de manière simple:

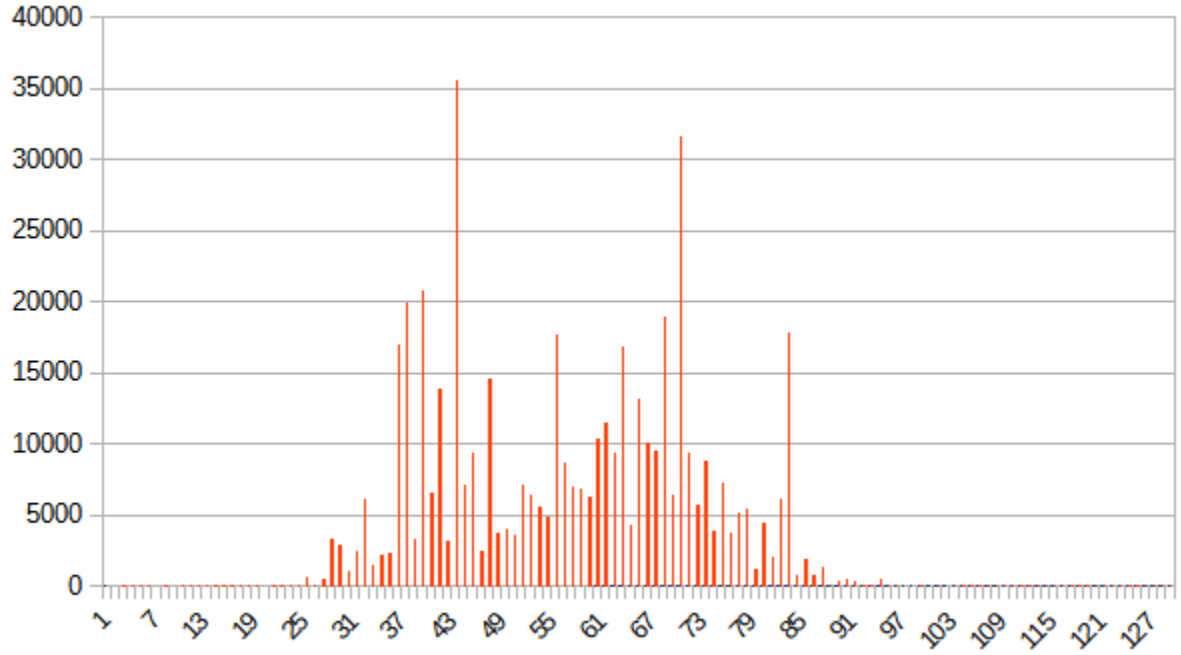
```
from mido import Message, MidiFile, MidiTrack

mid = MidiFile()
track = MidiTrack()
mid.tracks.append(track)
track.append(Message('program_change', program=12, time=0))
track.append(Message('note_on', note=69, velocity=27, time=12))
track.append(Message('note_off', note=69, velocity=64, time=4))
mid.save('new_song.mid')
```

Un fichier MIDI est constitué de plusieurs pistes (ou "tracks"). Chaque piste a plusieurs messages qui correspondent à des notes jouées à un certain rythme.

Les fichiers MIDI sont lus et les notes sont extraites. Mais, il n'est pas possible d'utiliser directement une chanson entière comme entrée du modèle Transformer, le nombre de paramètres explosant et rendant impossible les phases d'apprentissage. Nous allons donc construire des entrées à partir d'extraits des chansons.
Une entrée de notre jeu d'apprentissage est un fenêtre glissante de N notes décalées de D notes à chaque fois. Le schéma suivant permet de représenter un exemple pour expliciter le principe de construction du jeu de données:



Afin d'avoir un modèle efficace tout en espérant capturer un maximum de règles, nous avons choisi une fenêtre de taille 50 décalée de 20 notes.
D'un point de vue ondulatoire, une note est représentée par une fréquence. Le format MIDI prévoit un total de 128 notes. Par exemple, la valeur 69 ou 0xA4 en hexadécimal correspond à la note LA utilisée comme référence et qui a une fréquence de 440Hz.
Nous avons ajouté deux notes spéciales représentant le début et la fin d'une chanson. Pour représenter les notes, nous allons donc coder les notes avec un codage disjonctif complet de 128 bits + 2 bits pour les deux notes spéciales .
L'histogramme suivant représente le nombre de notes dans l'ensemble du jeu d'apprentissage:

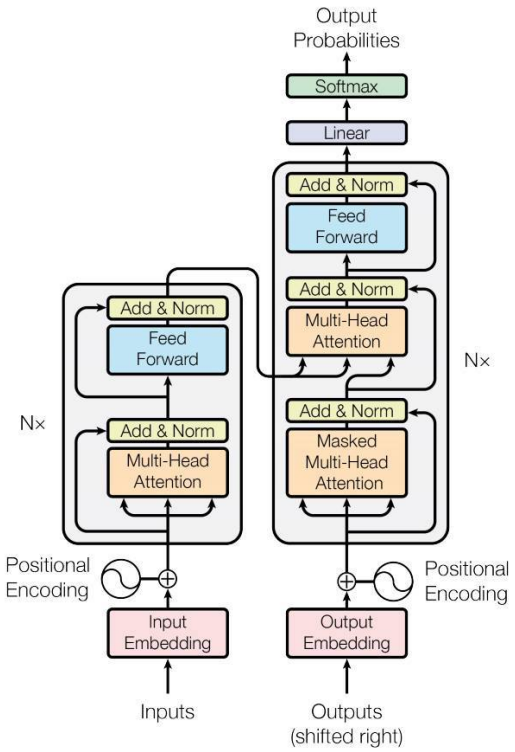


On peut remarquer que la répartition de notes est proche d'une gaussienne centrée autour des notes comme LA. Cela peut s'expliquer dans la mesure où les chansons utilisent rarement des notes dans les gammes les plus extrêmes. Cependant, cela entraîne un biais de représentativité dans la mesure où certaines notes sont très représentées et d'autres ne sont pas présentes.
En plus des notes, d'autres informations sont disponibles comme la vélocité qui représente l'intensité avec laquelle la note est jouée. Cette valeur est comprise entre 0 et 127. Pour éviter de saturer le réseau, ces paramètres sont normalisés pour être compris dans un intervalle entre 0 et 1.
L'erreur est donc égale à:

$$-y_{note} \times log(p_{note}) + (1 - y_{note}) \times log(1 - p_{note}) + (y_{velocity} - \widehat{y}_{velocity})^2$$

Architecture

L'architecture utilisée est un transformer. L'image suivante présente l'architecture interne d'un transformer:



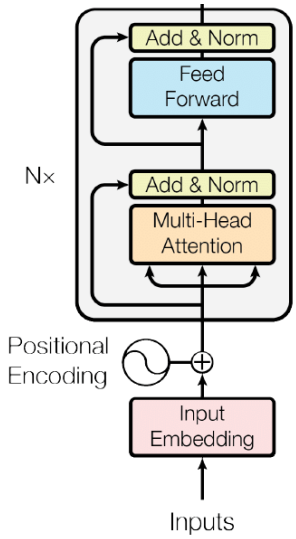
From https://arxiv.org/abs/1706.03762

Cette architecture est employée notamment pour traduire des phrases. Lors de la phase d'apprentissage, un exemple de d'entrées pourrait être:

- Input: "Le texte traduit"
- Output (shifted right): "<start> The translated"
- Ouput : "The translated **text**"

Le modèle doit donc prédire le mot "text" à partir de ces deux entrées. Dans notre cas, il ne s'agit pas d'un problème de traduction mais de génération de contenu appris à partir du jeu de données.

Nous gardons donc la partie "Encoder" du Transformer qui nous permet d'extraire les informations de la musique:



En sortie de l'encodeur, nous avons un vecteur représentant le vecteur en entrée. Nous ajoutons une couche entièrement connectée et un softmax pour avoir un vecteur de taille 128 représentant la probabilité de la note que nous devons utiliser.

Notre jeu d'apprentissage devient donc:

- Input: "Do Ré Mi"
- Ouput : "**Fa"**

La note à prédire est la note Fa.

Nous avons aussi conservé la partie "**Positional Encoding**", qui est un élément essentiel du transformer. Le modèle Transformer ne prend pas en compte par défaut la position des mots les uns par rapport aux autres. C'est pourquoi il est essentiel d'enrichir les données pour que l'ordre des notes ait une influence sur le résultat produit. C'est le but du "Positional Encoding"[5,6]. A chaque élément de l'entrée, on ajoute une valeur qui représente la distance entre deux notes. Le principe étant que si deux notes sont "proches", leurs valeurs soient plus importantes. Cela rend le modèle moins sensible à la présence de notes supplémentaires entre les notes qui ont un lien entre elles.

Pour la génération, nous pouvons donner en entrée une ou plusieurs notes et appeler de manière répétée notre modèle en décalant l'entrée avec la nouvelle note. Par exemple:

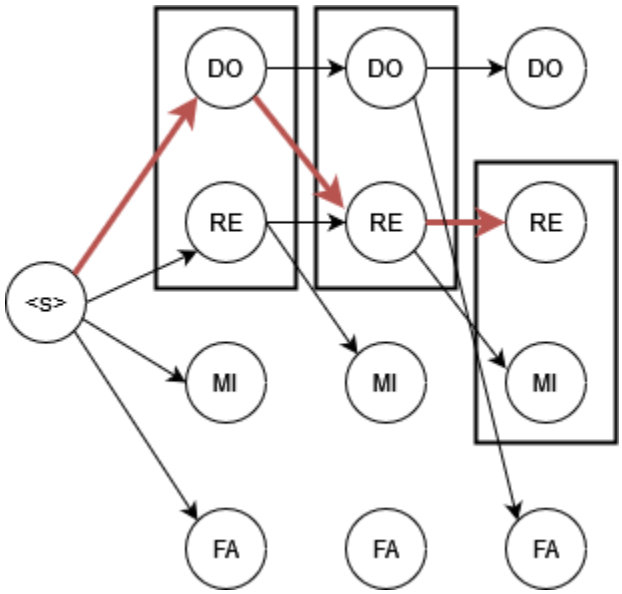
Appel	**1**	**2**	**3**	**4**
Input	Do Ré Mi	Ré Mi Fa	Mi Fa Sol	…
Output	Fa	Sol	La	

[5] https://kazemnejad.com/blog/transformer_architecture_positional_encoding/

[6] https://towardsdatascience.com/attention-is-all-you-need-e498378552f9

Résultat final	Do Ré Mi Fa	Do Ré Mi Fa Sol	Do Ré Mi Fa Sol La	

Enfin, pour essayer d'améliorer la musique produite, nous avons adapté l'algorithme de Viterbi (ou Beam Search) pour la recherche de la chaîne de notes la plus probable. On cherche donc à maximiser une suite de notes probables plus qu'une seule note au moment de la génération. A chaque étape, on produit plusieurs notes et on cherche à maximiser la probabilité de la chaîne. Le schéma suivant présente la séquence produite.



Soit la probabilité de la i-ème note produite:

$$P(Note_i = n_j / Note_{i-1} = n_{j-1}, ..., Note_1 = n_1, Note_0 =< s >)$$

On veut donc maximiser:

$$P(Note_{i+k} = n_{j+k}, ..., Note_i = n_j / Note_{i-1} = n_{j-1}, ..., Note_1 = n_1, Note_0 =< s >) = \prod_{i=m-k}^{m} P(Note_i = n_i / Note_{i-1} = n_{j-1}, ..., Note_{i-l} = n_{i-l})$$

Avec :

- k: la longueur de la chaîne que l'on veut maximiser
- l: la longueur de la séquence utilisée en entrée de notre modèle

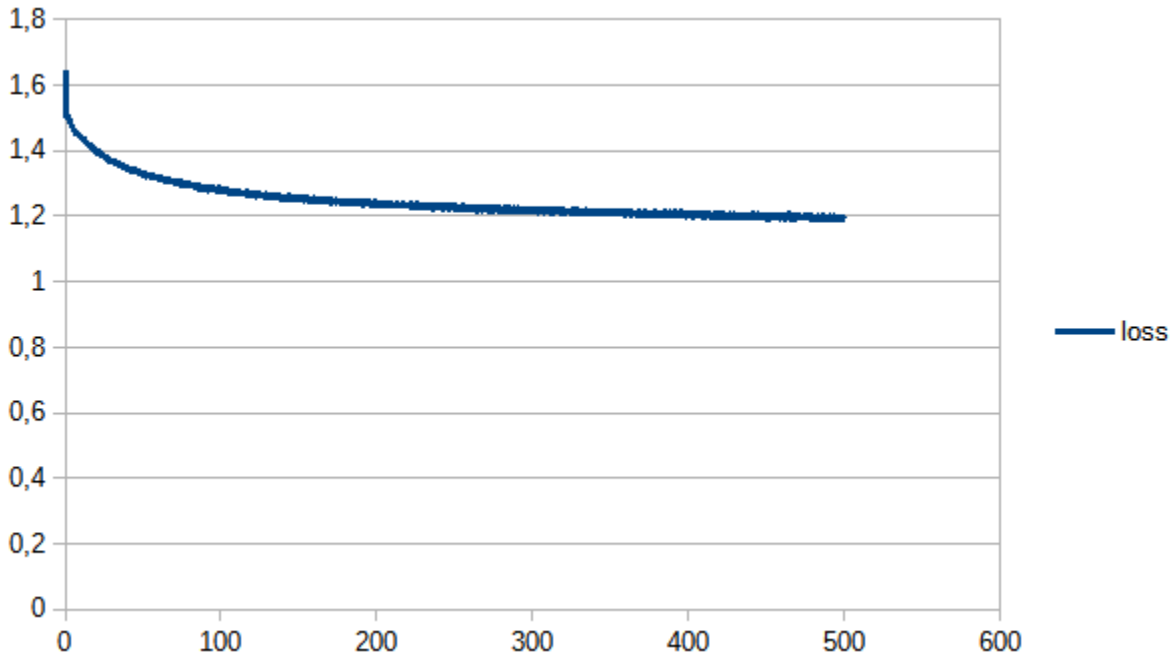
Tests et extensions

Un fois le modèle développé et le jeu de données construit, il a été nécessaire d'effectuer une phase d'apprentissage et de tests sur les données.

Tests, résultats et analyses

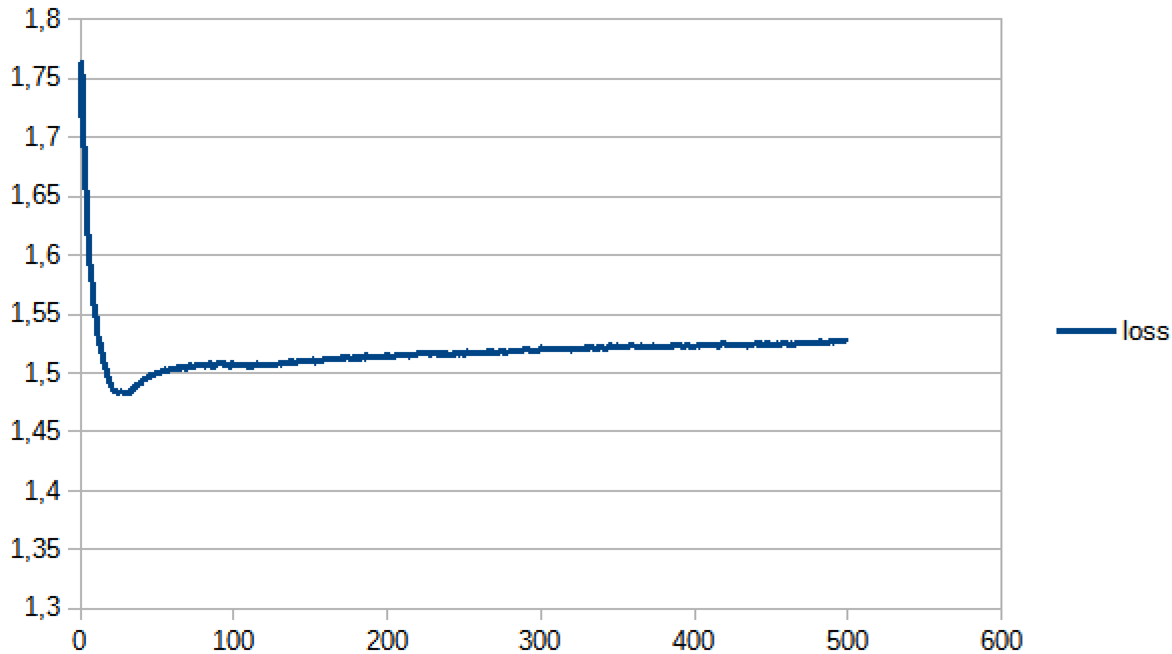
Le jeu de données est construit à partir de plusieurs chansons. Il est constitué de 10000 entrées d'une taille de 50 notes, décalé de 30 notes extraites des différentes chansons. Le nombre de 10000 entrées est choisi pour pouvoir être exécuté sur mon poste.
Le jeu de données a été divisé en deux ensembles : un jeu d'apprentissage et un jeu de test. L'exécution de 500 itérations prend environ 1h40.

Le diagramme suivant présente l'évolution de l'erreur d'apprentissage au cours des différents itérations d'apprentissage:



Evolution de l'erreur d'apprentissage au cours du temps

Le diagramme suivant présente l'évolution de l'erreur de test au cours des différents itérations d'apprentissage:



Evolution de l'erreur de test au cours du temps

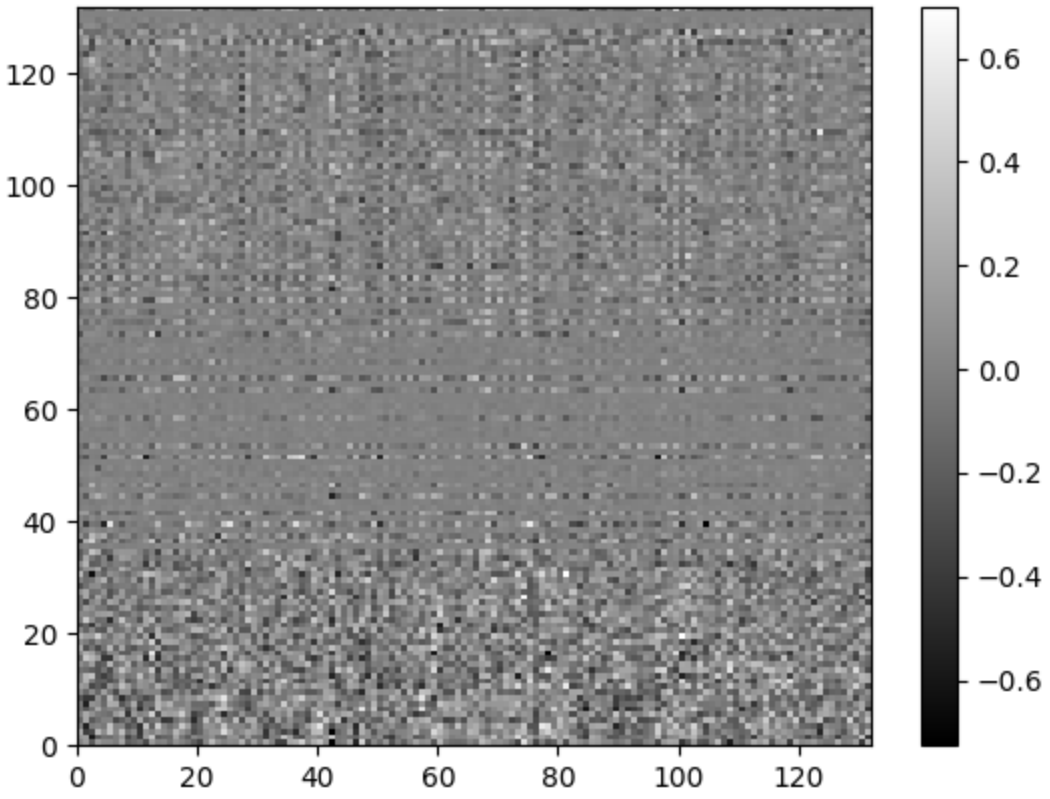
On peut remarquer que l'erreur d'apprentissage décroit de façon alors que l'erreur de test diminue puis augmente après un certain nombre d'itérations. Cela tendrait à indiquer que le modèle a sur-appris.

Une fois le modèle appris, nous l'utilisons pour générer une nouvelle musique. Nous donnons en entrée un exemple de notre jeu de données et nous produisons une nouvelle note selon l'algorithme de Viterbi décrit précédemment pour obtenir 100 notes.

Nous obtenons la suite de notes suivantes au format MIDI:
65, 81, 81, 81, 81, 81, 81, 81, 81, 81, 81, 81, 81, 81, 81, 81, 81, 81, 81, 81, 81, 65, 81, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 65, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40

La suite devient constante après un certain nombre de notes. Une raison peut être que certains poids ont pris plus d'importance que d'autres. Ce phénomène s'est reproduit à chaque tentative d'apprentissage.

Le graphique suivant présente les valeurs d'attention par note.



Au vu des résultats obtenus, le modèle ne semble pas avoir été capable de généraliser et d'apprendre la structure musicale sous-jacente. On peut envisager différentes causes:

- Le jeu d'entrées n'est pas suffisamment bien construit. On peut essayer avec plusieurs tailles de sous-ensembles de musiques (10, 20, 100, … notes) avec différents décalages.
- Le jeu d'entrées est trop faible. La machine utilisée pour apprendre ne permettait pas d'avoir des instances de données importantes.
- La couche linéaire en sortie n'est pas assez "riche" en termes de paramètres, ce qui fait que le modèle n'aura pas assez de puissance pour apprendre.
- Le nombre d'itérations n'est pas correct. On voit apparaître du sur-apprentissage. Des mécanismes pour limiter cette situation ont été implémentés (clipping des gradients, dropout…) mais n'ont pas permis d'empêcher cela.

Extensions possibles du modèle

Plusieurs extensions du modèle sont possibles. En s'inspirant des architectures présentées dans le module RCP217, on peut envisager des axes d'évolutions du modèle.

Tout d'abord, une architecture type transformer a été utilisée. Une architecture pour les données récurrentes sont les LSTM et GRU. Bien que ces modèles aient été surpassés au cours des

dernières années par l'architecture Transformer, ils pourraient être adaptés dans notre cas au vu de la taille d'une instance de notre jeu de données (50 notes).

Il pourrait être intéressant de réutiliser des architectures existantes et essayer d'appliquer des approches de type "transfert learning". En effet, la musique présentant une structure pouvant rappeler la prose, on peut imaginer ré-utiliser des modèles pré-entraînés comme BERT auquel nous appliquerons une phase d'apprentissage mais plus courte.

Conclusion

Dans ce projet, j’ai étudié le domaine de la musique et j’ai cherché à générer de nouveaux morceaux sans connaissance particulière sur les règles musicales. j’ai produit un modèle s’appuyant sur l’architecture Transformer dans le but de générer de nouvelles partitions.

Bien que le modèle n'ait pas été capable de produire une musique acceptable, il donne une base pour étendre et poursuivre cette idée.

Dans le cadre de l’UE RCP217 “Intelligence artificielle pour des données multimédia”, ce projet m’a permis de développer un modèle complet sur un sujet complexe, la génération de données musicales.

Ce projet m’a aidé à mieux comprendre l’architecture type “Transformer” en me donnant un axe d’étude sur un sujet qui présente une complexité technique (compréhension du protocole MIDI) et fonctionnelle (musique).

Bibliographie

Huang, Cheng-Zhi Anna, and Ashish Vaswani. “An Improved Relative Self-Attention Mechanism for Transformer with Application to Music Generation.” *CoRR*, vol. abs/1809.04281, 2018.

Huang, Cheng-Zhi Anna, and Ashish Vaswani. “Music Transformer: Generating Music with Long-Term Structure.” *ICLR*, 2019.

“Language Modeling with nn.Transformer and TorchText — PyTorch Tutorials 1.12.0+cu102 documentation.” *PyTorch*, https://pytorch.org/tutorials/beginner/transformer_tutorial.html. Accessed 15 July 2022.

Shaw, Peter, et al. “Self-Attention with Relative Position Representations.” *CoRR*, vol. abs/1803.02155, 2018.

Lakh MIDI Dataset : Colin Raffel. **"Learning-Based Methods for Comparing Sequences, with Applications to Audio-to-MIDI Alignment and Matching"**. *PhD Thesis*, 2016.