LES TROPHÉES NSI

Édition 2023

DOSSIER DE CANDIDATURE

PRÉSENTATION DU PROJET

Logiciel de génération de musique

CRESCENDO

par David KARR--DUCCI
et Esteban VIGANO--FLUCKIGER

ANNEE SCOLAIRE 2022-2023

✓ Choisissez vos musiques préférées
✓ Donnez un début de morceau
✓ l'IA compose la suite !

NOM DU PROJET : Crescendo

> PRÉSENTATION GÉNÉRALE :

Crescendo est une application qui génère de la musique à partir d'un début de morceau choisi par l'utilisateur en utilisant un réseau de neurones.

Le « cœur » du programme est constitué de quatre réseaux de neurones *LSTM* (*Long Short Term Memory*). Ce type de réseau est bien adapté pour la génération. Les couches *LSTM* reçoivent en entrée, un par un, les éléments d'une séquence - dans notre cas, les notes d'un morceau. A chaque fois qu'un nouvel élément est reçu, la couche « décide » ce dont il faut se souvenir et ce qu'il faut oublier, ce qui lui permet d'avoir à la fois une mémoire à long terme et une mémoire à court terme. Ici, ces réseaux de neurones sont utilisés pour prédire la note qui suit, et en répétant cette opération nous obtenons un nouveau morceau.

Nous avons choisi d'utiliser le format MIDI pour les morceaux. Dans ce format, les sons ne sont pas représentés par des fréquences ni des durées en seconde, mais par des notes, ce qui est plus adapté pour utiliser les données.

Les réseaux de neurones peuvent être longs à entraîner. Les cartes graphiques (*GPU*) étant beaucoup plus efficaces que les processeurs (*CPU*) pour ce type de tâche il est préférable d'utiliser la carte graphique pour l'entraînement. Cela n'est pas possible pour tous les modèles de carte graphique ; si celle de l'utilisateur n'est pas compatible, le programme utilisera le processeur. David a fait tourer les réseaux de neurones sur sa carte graphique NVIDIA RTX 3070.

Pour l'interface graphique, à base de fenêtres, boutons et menus déroulants, nous avons utilisé le module *Tkinter*.

> ORGANISATION DU TRAVAIL :

Nous avons décidé qu'EstebanVIGANO--FLÜCKIGER serait responsable de l'interface et David KARR--DUCCI du réseau de neurones. La phase d'entraînement sur *GPU* utilisant des librairies spécifiques, il n'était pas possible de tester le code en ligne. C'est pourquoi nous n'avons pas créé de code partagé, mais nous nous sommes envoyé régulièrement nos nouvelles versions.

> LES ÉTAPES DU PROJET :

Les fichiers midi et la représentation des données :

Nous avons d'abord cherché comment manipuler les fichiers midi. Nous avons trouvé la librairie *music21*, qui permet de lire et enregistrer facilement de la musique. Nous avons essayé plusieurs façons de représenter les notes, soit avec leur nombre midi associé (*pitch.midi*), soit avec l'octave (*pitch.octave*) et le nom de la note (*pitch.pitchClass*). Nous représentons aussi le moment auxquelles elles sont jouées (*note.offset*) et leur durée (*note.duration.quarterLength*).

Finalement, ce qui nous a semblé plus efficace était de représenter les notes par ces quatre caractéristiques : leur octave, le nom de la note, combien de temps elles sont jouées après la note précédente, et leur durée.

Le traitement des données

En effectuant des recherches, nous avons vu qu'il existe deux options pour traiter les données : soit les normaliser, de manière à avoir un écart-type égal à 1 et une moyenne égale à 0 pour toutes les caractéristiques des notes, soit les représenter de façon binaire. Nous avons choisi cette deuxième solution. Chaque note est représentée par quatre caractéristiques, chacune étant

une liste où les valeurs possibles sont représentées avec un "1" dans l'élément correspondant et des "0" dans les autres. Par exemple, si l'octave d'une note est 3 et que toutes les octaves de 0 à 7 sont représentées, sa représentation peut être : [0,0,0,1,0,0,0,0].

Notre objectif est de prédire, pour une séquence de notes de longueur donnée, la note qui suit. Nous avons donc besoin, comme données d'entraînement (appelées *samples*), de toutes les séquences d'une longueur donnée présentes dans le morceau. Ainsi, les quantités prédites par les réseaux de neurones (appelées *labels*) sont les caractéristiques de la note suivante de chaque séquence des *samples*.

Il y a quatre caractéristiques (octave, note, moment, durée) donc quatre réseaux en tout. A noter que nous devons transformer les labels et les *samples* en listes *numpy* pour qu'ils puissent être utilisés par un réseau de neurones.

Type de couches des réseaux de neurones

On a d'abord utilisé des *layers Dense*, les plus simples, mais on a ensuite découvert que les *layers LSTM* étaient plus adaptés pour les séquences et la génération, et que le layer Dropout permettait d'accélérer l'entraînement : c'est une couche qui désactive une certaine proportion de neurones, ce qui permet de corriger plus rapidement les neurones actifs.

Fonctionnement des réseaux de neurones :

Un réseau de neurones est composé de couches, elles-mêmes composées de neurones. Un neurone prend en entrée les sorties de tous les neurones de la couche précédente, et envoie un nombre en sortie. La sortie est calculée ainsi : on additionne les sorties de tous les neurones, multipliées par un poids, et on ajoute à cette somme le biais du neurone. Enfin, la fonction d'activation de la couche à laquelle appartient le neurone est appliquée au résultat. Durant la phase d'apprentissage, les réseaux de neurones ajustent leurs poids et leurs biais afin de gagner en précision dans leurs prédictions. Nous avons utilisé des fonctions d'activations non linéaires : la fonction *relu* et la fonction *softmax*.

L'interface et les chemins d'accès

Pour l'interface, nous avons une fenêtre principale et deux fenêtres secondaires. La première permet d'entraîner un modèle, et la deuxième permet de composer une musique. Puisque nous voulions une interface simple, nous avons décidé d'utiliser *Tkinter*. Pour demander différents paramètres à l'utilisateur, nous devons parcourir les dossiers, et nous utilisons pour cela le module *os*.

Nous avons remarqué que pendant l'entraînement et la composition, la fenêtre se gèle et ne répond pas. Pour éviter cela, nous avons utilisé le *threading*, qui permet de réaliser plusieurs tâches en parallèle.

Enfin, pour éviter des erreurs ou des événements indésirables, nous désactivons les boutons qui ne peuvent pas être utilisés.

> FONCTIONNEMENT ET OPÉRATIONNALITÉ :

Le programme fonctionne comme prévu : nous avons testé la lecture et la sauvegarde de musiques, l'entraînement des réseaux de neurones et leur utilisation sur d'autres ordinateurs avec succès.

La musique générée n'est pas exceptionnelle, mais venant d'une IA, le résultat est assez satisfaisant. Cependant, on remarque une tendance à répéter en boucle une même séquence de notes. Pour y remédier, nous utilisons des probabilités pour générer les notes, au lieu de choisir toujours celle qui a le résultat le plus élevé. Malgré cela, le problème ne disparait pas totalement.

> OUVERTURE :

Pour améliorer le réseau de neurones, nous pourrions modéliser les notes par plus d'informations, par exemple le moment absolu auxquelles elles sont jouées, le moment dans la mesure auxquelles elles sont jouées…

Nous pourrions aussi imaginer un modèle qui génère des musiques pour plusieurs instruments.

Au niveau de l'interface, une amélioration possible serait que l'utilisateur puisse écrire son début de morceau dans une partition plutôt que sous la forme d'un fichier midi.

DOCUMENTATION

L'utilisation du *GPU* nécessite l'installation de certains composants qui sont *CUDA toolkit* et *CUDnn* qui est expliquée sur ce site : https://www.tensorflow.org/install/pip?hl=fr.

Les modules à avoir sont : *tensorflow, keras, Tkinter, pillow, os, threading, music21, numpy, pickle, queue.*

Les musiques doivent être au format MIDI.

L'exécution peut prendre plusieurs minutes.

L'explication complète de l'utilisation du programme est dans la vidéo de présentation.

Aperçu du logiciel :