**Génération de musique:**

Le but de ce projet était la création d’un réseau neuronal permettant la génération de musique.

Outils utilisés:

Google Colab:

Colab permet à quiconque d'écrire et d'exécuter un code python arbitraire via le navigateur. Il est particulièrement bien adapté à l'apprentissage automatique, à l'analyse de données et à l'enseignement.

Music21:

Le toolkit Python Music21 nous permet de travailler avec des fichiers MIDI ( Musical Instrument Digital Interface). Ces fichiers contiennent des sur le début , la durée et la hauteur d'une note, mais pas sur leur son.

Keras:

Keras est une librairie logicielle open-source qui fournit une interface Python pour les réseaux de neurones artificiels. Keras sert d'interface à la bibliothèque TensorFlow.

Dans ce projet j’ai utilisé la librairie Keras pour créer et entraîner un modèle LSTM puis pour générer de la musique grâce à ce modèle.

**Entrainement ( IA\_train.ipynb):**

Les données utilisées pour l'entraînement du modèle sont des fichier MIDI disponible gratuitement sur le site : <http://www.partnersinrhyme.com>

Entraînement du modèle se divise en 4 étapes qui se font grâce à 4 fonctions.

La première étape, réalisée par la fonction get\_notes() consiste à préparer les données.

Pour cela j’ai utilisé la librairie Music21 qui grâce à sa fonction .partitionByInstrument() permet de séparer les fichiers MIDI en instruments individuels. Chaque instrument contient une liste de notes qui est extraite et ajoutée à une liste pour être ensuite sauvegardée dans un fichier en utilisant pickle.

Les fichiers MIDI lu par music21 se présent sous cette forme:

...

<music21.note.Note F>

<music21.chord.Chord A2 E3>

<music21.chord.Chord A2 E3>

<music21.note.Note F>

<music21.note.Note D>

<music21.chord.Chord B-2 F3>

<music21.note.Note E>

<music21.chord.Chord A2 E3>

...

La donnée est séparée en 2 types d'objets : les notes et les chords. Les notes contiennent des informations sur la hauteur, l'octave et l'offset de celle-ci. La hauteur fait référence à la fréquence du son, c'est-à-dire à son intensité, et est représentée par les lettres [A, B, C, D, E, F, G]. L'octave désigne l'ensemble des hauteurs de son que l'on utilise sur un piano. L'offset fait référence à l'endroit où la note est située dans le morceau.

La deuxième étape est faite par la fonction prepare\_sequences() qui consiste en une fonction de mapping pour passer des données en string aux données numériques basées sur des nombres entiers. Je fais cela car les réseaux neuronaux fonctionnent mieux avec des données numériques basées sur des nombres entiers qu'avec des string.

Ensuite, nous devons créer des séquences d'entrée pour le réseau et leurs sorties respectives. La sortie de chaque séquence d'entrée sera la première note ou le premier accord qui suit la séquence de notes de la séquence d'entrée dans notre liste de notes.

Finalement nous créons l'architecture du modèle grâce à la fonction create\_model(). Celui ci comporte 4 couches:

Les couches LSTM sont des couches de réseau neuronal récurrent qui prennent une séquence en entrée et peuvent renvoyer des séquences (return\_sequences=True) ou une matrice.

Les couches de Dropout sont une technique de régularisation qui consiste à fixer une fraction d'unités d'entrée à 0 à chaque mise à jour au cours de la formation afin d'éviter un surajustement. La fraction est déterminée par le paramètre utilisé avec la couche.

Les couches denses ou couches entièrement connectées sont des couches de réseau neuronal entièrement connectées où chaque nœud d'entrée est connecté à chaque nœud de sortie.

La couche d'activation détermine la fonction d'activation que notre réseau neuronal utilisera pour calculer la sortie d'un nœud.

Pour chaque couche LSTM, Dense et Activation, le premier paramètre est le nombre de nœuds que la couche doit avoir. Pour la couche Dropout, le premier paramètre est la fraction des unités d'entrée qui doivent être abandonnées pendant la formation.

Pour la première couche, nous devons fournir un paramètre unique appelé input\_shape. Le but de ce paramètre est d'informer le réseau de la forme des données qu'il va entraîner. La dernière couche doit toujours contenir le même nombre de nœuds que le nombre de sorties différentes de notre système. Cela garantit que la sortie du réseau correspondra directement à nos classes.

Enfin pour calculer la perte pour chaque itération de la formation, nous utiliserons l'entropie croisée catégorielle, car chacune de nos sorties n'appartient qu'à une seule classe et nous avons plus de deux classes à utiliser. Et pour optimiser notre réseau, nous utiliserons un optimiseur RMSprop.

Enfin la dernière étape est l'entraînement du modèle. Pour cela nous utilisons la fonction model.fit() de Keras pour entraîner le réseau. Le premier paramètre est la liste des séquences d'entrée que nous avons préparées précédemment et le second est une liste de leurs sorties respectives, 'epchos' correspond aux nombre d'itérations et 'batch\_size' le nombre d'échantillons. Ces deux derniers éléments sont modifiables et auront un impact sur la qualité de ce que l'on pourra générer mais aussi sur le temps d'entraînement.

Nous utilisons aussi des ModelCheckPoint. Ces points de contrôle du modèle permettent d'enregistrer les poids des nœuds du réseau dans un fichier après chaque époque ce qui nous permet de ne pas perdre l'entraînement du modèle en cas d'arrêt volontaire ou involontaire de son entraînement.

**Génération de musique (IA\_predict.ipynb):**

Maintenant que le modèle est entraîné nous pouvons l'utiliser pour générer de la musique.

Pour pouvoir utiliser le réseau neuronal afin de générer de la musique, nous devons le mettre dans le même état que précédemment. Pour cela nous réutilisons le code de la section d’entrainement pour préparer les données et configurer le modèle de réseau de la même manière que précédemment. Sauf qu'au lieu de former le réseau, nous chargeons dans le modèle les poids que nous avons sauvegardés pendant l’entrainement.

Nous devons également créer une fonction de mapping pour décoder la sortie du réseau. Cette fonction permettra de transformer les données sous formes d’entiers en notes.

Nous pouvons maintenant générer de nouvelles notes. Le point de départ est un indice aléatoire de la séquence d'entrée, le modèle va ensuite prédire les notes suivantes pour générer la musique.

Nous créons un tableau numpy de toutes les prédictions, puis nous indexons la note avec la probabilité la plus élevée. Cette note est alors la sortie de la prédiction, qui est ajoutée à la séquence d'entrée dans l'itération suivante.

Maintenant que nous avons une liste de notes et d'accords générés par le réseau, nous pouvons créer un objet Music21 Stream en utilisant la liste comme paramètre. Enfin, pour créer le fichier MIDI contenant la musique générée par le réseau, nous utilisons la fonction write de Music21 pour écrire le flux dans un fichier.

**Résultat et conclusion:**

Les résultats que j’ai obtenus sont assez peu satisfaisants. Cela est, je pense, principalement dû au manque d'entraînement du modèle. Beaucoup de paramètres sont modifiables pour l'entraînement mais j’ai étais limité par les ressources de Google Colab et l’impossibilité de laisser mon modèle s'entraîner pendant plus de 12h.

Cependant cela montre bien le fonctionnement de la génération de musique grâce à un réseau neuronal qui peut avoir du potentiel si l’on dispose de plus de ressources et que l’on utilise plus de données pour l'entraînement du modèle.