République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université des Sciences et de la Technologie Houari Boumediene

Faculté d'Electronique et d'Informatique
Département Informatique

Mémoire de Licence

Filière : Informatique

Spécialité : Licence Informatique Académique.

Composition de Musique à l'Aide des Réseaux de Neurones

Sujet proposé par :

MOUDJARI Leila
AKLI Karima

Présenté Par :

FERGANI Kheireddine
MERZOUG Imad-eddine

Soutenu le : 17/10/2020

Devant le jury composé de :

Mr BESSAA Brahim **Président**
Mme BERKANI Lamia **Membre**

Binôme n^o : 055/2020

Remerciements

En préambule à ce mémoire de licence, nous tenons avant tout à remercier Dieu de nous avoir donné le courage et la volonté pour mener à bien ce modeste travail. Nous souhaiterions également adresser nos remerciements les plus sincères aux personnes qui ont su nous apporter leurs aides et contribuer à la réalisation de notre projet.

Aussi, nous aimerions exprimer nos plus sincères remerciements et notre profonde gratitude à nos encadrantes Melle MOUDJARI Leila et Mme AKLI Karima pour tous les efforts qu'elles ont fournis, pour les explications, remarques et suggestions précieuses, pour le temps qu'elles nous ont consacré. Sans elles, ce travail n'aurai pas été le même.

Enfin, nos sincère considérations et remerciements sont également exprimés aux membres du jury pour avoir accepté de nous consacrer du temps en examinant notre travail.

Résumé

Récemment, les réseaux de neurones générateurs ont ouvert la voie à des activités artistiques, telles que la génération d'images et les retouches de photos. Un autre domaine dans lequel ces réseaux d'apprentissage profond commencent à laisser des traces est la génération de musique. Dans ce projet de fin d'études de licence, notre objectif est d'explorer l'utilisation des réseaux de neurones de type GAN et LSTM pour générer une musique qui semble être d'origine humaine. Afin de mettre en évidence l'efficacité de nos modèles,nous établirons des critères d'évaluation et une base de comparaison.

Abstract

Recently, generating neural networks are used in artistic activities, such as image generation and photo editing. Another area in which these deep learning networks can be exploited is the music generation. In this graduate project, our goal is to explore the use of GAN and LSTM neural networks to generate music that appears to be human-made. In order to demonstrate the efficiency of our models, we will define some evaluation criteria and a comparison baseline.

Table des matières

Table des figures

Introduction Générale

La musique et l'informatique sont deux domaines extrêmement liés. Presque la totalité de la musique (passé et actuel) existe sous format numérique, ce qui est énorme, avec la quantité de musique disponible, on se retrouve avec du contenu divers et varié qui ne cesse de grandir, ce qui en fait une base de données intéressante à traiter sous divers contextes (suggestion de contenu, étude de sens ou encore génération musicale). La demande pour des outils qui génèrent, traitent et jouent de la musique ne cesse d'augmenter. Avec la disponibilité de technologies plus avancées notamment les GPU [1] qui sont devenus grâce à leurs calcul parallèle [2] une bénédiction pour les modèle d'apprentissage profond, a fait que de plus en plus de personnes/entreprises s'attaquent au problème, parmi lesquelles on peut citer Aiva [3], Amper [4]ou encore Magenta [5]. Dans cette étude, nous proposons des modèles basés sur l'apprentissage profond permettant de générer des pistes musicales originales du début à la fin ou compléter un bout de piste qui existe déjà.

1. Graphical Proccessing Unite : processeur Graphique/Carte Graphique.
2. Les GPU contrairement au CPU ont plusieurs milliers de coeurs ce qui leur permet de faire plusieurs milliers de calculs simples en même temps.
3. Aiva (Artificial Intelligence Virtual Artist) https://www.aiva.ai/.
4. Amper Music https://www.ampermusic.com/.
5. Google Magenta https://magenta.tensorflow.org/.

Chapitre 1

Contribution et Plan du Mémoire

La musique est un art qui peut de première abord sembler abstrait. Composer un morceau de musique semble être une manifestation obscure de l'esprit d'un artiste dont lui seul peut en comprendre les mécanismes, et encore peut-être n'en comprend-il lui-même pas les délicatesses. Mais en réalité la musique est l'une des expressions artistiques les plus structurées qui soient. Un morceau de musique qui sonne bien n'est pas dû au hasard ou juste l'instinct, il y a de la théorie et des règle qu'on doit respecter explicitement si on les connaît ou implicitement si on a une assez bonne oreille pour remarquer et éviter les dissonances. Cette suite de règles qui structure la musique est appelé théorie de la musique. Observé une machine apprendre les subtilités de ces règles ne peut-être que très fascinant.
Un des domaine de l'intelligence artificielle qui se rapproche le plus de la génération de musique est la génération de texte (on peut considérer les notes ou les accords comme des notes, les mesures comme des phrases, une partition comme un texte, etc.). Cela signifie que l'étude de l'un devient utile pour l'autre.

Sur une note plus personnelle, en tant que guitaristes amateurs, nous avons toujours été fascinés par le processus de composition de musique. On peut imaginer plusieurs applications liées à la génération de musique, comme l'aide à la composition dans le cas d'un "Art Block" [1], notre IA peut intervenir en générant des morceaux musicaux. Cela permet d'un côté de compléter une composition ou d'un autre créer des morceaux originaux. Ce qui va permettre au musicien de s'inspirer voir collaborer avec la machine pour finir sa composition. Nous pouvons voir aussi notre IA combinée à la vision par ordinateur pour générer une musique de fond correspondant à l'ambiance d'un film. Ou encore génération ou suggestion de musique selon l'humeur et l'état émotionnel de la personne, car la musique dans d'autres cultures est considérée comme remède de l'âme humaine. Et tant d'autres, comme dit le proverbe anglais "sky's the limit", il n'y a pas de limites pour l'imagination. Il suffit juste d'être méthodologique avec un esprit ouvert pour pouvoir avancer et réaliser ses rêves.

1.1 Contribution

Afin de répondre à la première application i.e., "Art Block", nous proposons des modèles basées sur l'apprentissage profond, à savoir les LSTMs et les GANs. Effectivement, nous avons réussi à générer des mélodies en utilisant différents types de réseaux neuronaux. Les réseau neuronaux récurrents et les réseaux antagonistes génératif. Les résultats obtenus sont très encourageants et assez remarquables.

1. Art Block, une période durant laquelle l'artiste stagne et ne peut se résoudre à créer une nouvelle œuvre [3], c'est l'équivalent du syndrome de la feuille blanche des écrivains.

1.2 Plan du Mémoire

Afin de présenter notre travail, notre mémoire sera organisé en deux parties. La première soulignera les travaux de l'état de l'art, et la seconde notre contribution présentées en deux chapitres. Dans le premier chapitre, on abordera la phase de développement avec les architectures de nos modèles et les technologies utilisées, et dans le second chapitre, on présentera les résultats et les différentes évaluations entreprises. Pour atteindre les objectifs assignés, nous procédons comme suit. Dans une première étape, nous fournissons des notions de base de la théorie de musique et sur l'apprentissage automatique. Nous nous intéressons surtout à l'apprentissage profond où nous définissons deux types de modèles essentiels pour notre travail (RNN, GAN). Nous proposons ensuite une étude de l'état de l'art de diverses applications qui exploitent ces modèles dans le but de générer de la musique.
Dans une seconde étape, nous étudions les deux architectures des réseaux de neurones à savoir réseaux de neurones Récurrents et réseaux de neurones antagonistes génératifs dédiés à résoudre notre problématique. Pour chaque architecture, nous étudions les paramètres correspondants, en vue de les optimiser grâce à des expérimentation empiriques. Ceci nous mène vers la proposition de nos modèles.
La troisième étape, entame l'analyse et le pré-traitement des données. Cette étape est essentielle, et permet de transformer les données utilisées sous un format numérique traitable par la machine facile à utiliser par nos modèles. L'étape suivante de notre étude permet l'apprentissage des réseaux de neurones que nous avons proposé ainsi que leur optimisation grâce aux fonctions objectifs qui mesurent leurs performances en terme d'erreur (loss).
Dans une dernière étape, nous examinons les résultats obtenus par nos modèles et nous définissons nos critères d'évaluation.
Finalement, nous terminons ce travail par le chapitre final, qui présente les conclusions tirées de l'évaluation et esquisse les lignes de travail possibles pour l'avenir.

Chapitre 2

La Musique et l'Intelligence Artificielle

Ce chapitre a pour but de définir les notions importantes et nécessaires à la compréhension de notre travail. Tout d'abord, nous donnerons un aperçu sur la théorie de la musique et ensuite sur les notions de l'apprentissage automatique. Nous donnerons finalement une étude de l'état de l'art des diverses applications qui exploitent ces modèles d'apprentissage automatique pour la génération de la musique.

2.1 Quelques Concepts de la Théorie de Musique

La théorie de la musique est une discipline qui nous permet de comprendre le langage de la musique. C'est un ensemble de lignes directrices et de pratiques utilisées pour reconnaître les différentes façons d'exprimer des émotions à travers un son musical. Dans cette section, on abordera les notions de base du langage musical et quelques concepts de la théorie de la musique, nécessaire à la compréhension des notations utilisées dans notre travail.

Portée La portée est la base de l'écriture musicale, composée d'un ensemble de cinq lignes horizontales séparées (Figure 2.1). La portée peut être considérée comme un graphe musical sur lequel sont placées les notes de musique pour indiquer au lecteur la hauteur (pitch) spécifique d'une note [6].

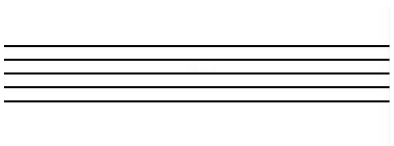


FIGURE 2.1 – Une Portée Musicale Vide.

Notes Lorsqu'elles sont écrites sur une portée, les notes indiquent une hauteur (pitch) et une valeur rythmique. Une note consiste en une tête (format rond, vide ou rempli), et peut éventuellement inclure une tige, une poutre, un point ou des drapeaux [6] (voir Figure 2.2). Les notes ne peuvent pas transmettre leurs informations de hauteur sans être placées sur une portée [6].

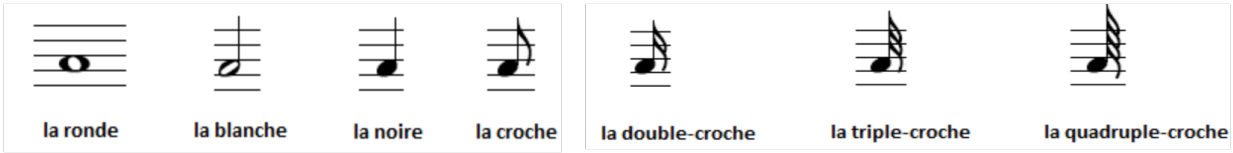


FIGURE 2.2 – Différentes Valeurs de Note.

Le Clavier du Piano Le clavier est idéal pour aider à développer une compréhension visuelle, orale et tactile de la théorie musicale.

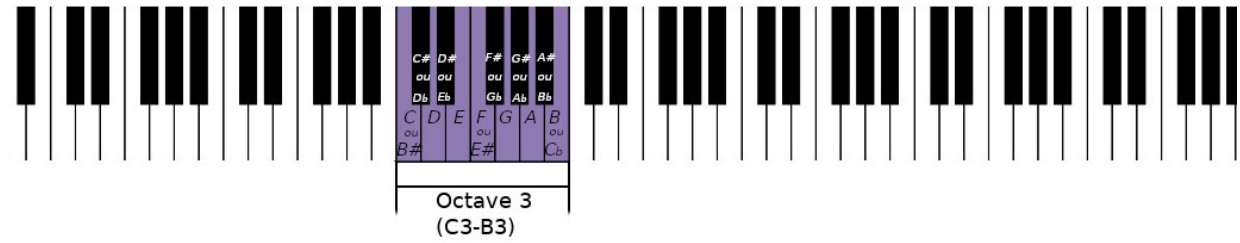


FIGURE 2.3 – Clavier de Piano avec les Noms des Lettres de la Classe de Hauteur.

Les notes séparées par une octave (ou plusieurs octaves) ont le même nom de lettre et sont de la même classe de hauteur (un piano contient en général 7 octaves). Il existe 12 notes dans une octave, 7 notes dite naturelles $C\ D\ E\ F\ G\ A\ B$ et 5 altérations. L'altération d'une note est soit un "demi-ton" au dessus et on mettra $\sharp$ à coté : $C\sharp\ D\sharp\ F\sharp\ G\sharp\ A\sharp$ ou un demi-ton en dessous et on mettra le signe $\flat$ à coté : $D\flat\ E\flat\ G\flat\ A\flat\ B\flat$ [6].

Il est à noter qu'une désignation complète contient à la fois le nom de la classe de hauteur (un nom de lettre plus un dièse ou un bémol facultatif) et le registre (le numéro indiquant l'octave dans laquelle se trouve la hauteur). La fréquence double plus ou moins à chaque octave, la note $C1$ (la note de C octave une) a une fréquence de 110, $C2$ et $C3$ ont une fréquence de 220 et 440 respectivement [1].

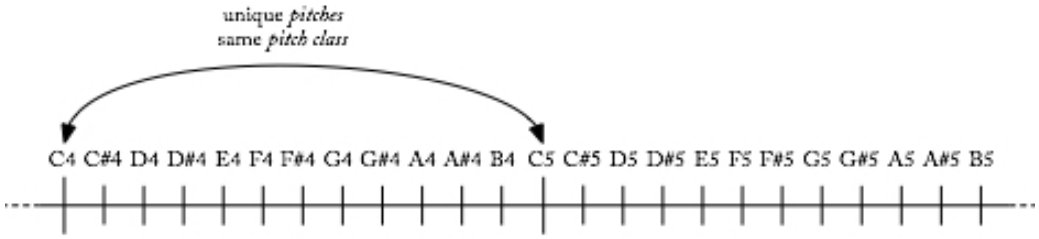


FIGURE 2.4 – Deux Hauteurs Séparées par une Octave.

Accords Un accord est un ensemble de hauteurs jouées simultanément qui peut créer une harmonie (deux ou plusieurs notes se complètent). Ils ajoutent de la texture à une mélodie, et peuvent même donner du rythme à une chanson.

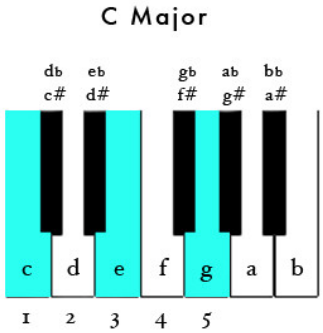


FIGURE 2.5 – Clés Primaires pour Jouer l'Accord C Majeur.

2.2 L'Apprentissage Automatique de la Musique

L'apprentissage automatique (Machine Learning) est un champ d'étude de l'intelligence artificielle qui se fonde sur des approches mathématiques et statistiques pour donner aux ordinateurs la capacité de générer des patterns à partir de données et ainsi inférer de nouvelles connaissances, ce qui leur permet d'améliorer leurs performances à résoudre des tâches sans être explicitement programmés pour chacune [16]. Un certain nombre d'approches existent, nous en citons quelques unes :

2.2.1 Chaîne de Markov

Une chaîne de Markov est une suite de variables aléatoires, qui permet de modéliser l'évolution dynamique d'un système aléatoire. La propriété fondamentale des chaînes de Markov, dite propriété de Markov, est que son état future ne dépend que de son état actuel et ne prend pas en compte les états lointains [25].

2.2.2 Apprentissage Profond (Deep Learning)

L'apprentissage profond (deep learning) est l'une des principales branches du Machine Learning et ainsi de l'intelligence artificielle. L'apprentissage profond s'inspire du fonctionnement du cerveau humain, plus particulièrement de l'interconnexion des neurones [33]. L'apprentissage profond consiste à :

- *Apprendre la représentation de caractéristiques complexes à travers plusieurs couches de traitement.*
- *Apprendre la représentation automatiquement et conjointement avec les tâches de classification (ou autres) (optimisation de bout en bout).*
- *La représentation en fonction des tâches est plus performante.*

Dans ce qui suit, des notions de base liées à cette approche seront présentées :

2.2.2.1 Perceptron

Un neurone artificiel ou un perceptron est un concept inspiré du neurone biologique, et constitue l'entité de base d'un réseau de neurones. Les perceptrons ont été créés en vue d'avoir les mêmes capacités d'apprentissage que les neurones biologiques.

Fonctionnement d'un Perceptron Un perceptron reçoit n entrées représentées par un vecteur $\overrightarrow{x} \in \mathbb{R}^n$, où chaque entrée $i, 1 \leq i \leq n$, lui attribuée un poids $w_1, ...w_n$ (Figure 2.6). Le

perceptron calcule d'abord la somme pondérée de ses entrées. Ensuite, la sortie Y est calculée en appliquant une fonction $f(X)$ dite d'*activation* (voir équation 2.1). Le résultat est transmis au prochain neurone de la prochaine couche interne ou la couche de sortie, imitant ainsi le fonctionnement d'un neurone biologique.

$$\begin{aligned} f(X) &= (X.W + B) \\ X.W &= \sum_{i=0}^{n} x_i.w_i \end{aligned} \tag{2.1}$$

où W représente la matrice des poids assignés à chaque entrée de la matrice des entrées X, et B est un biais qui vise à améliorer l'apprentissage d'un modèle en ajustant les poids des arcs ainsi que les poids de sortie.

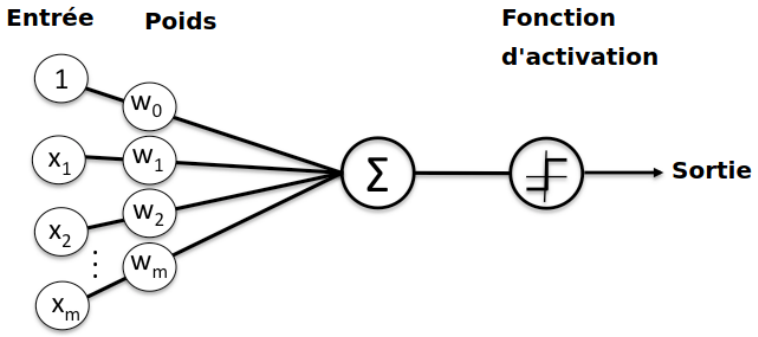


FIGURE 2.6 – Fonctionnement d'un Neurone Artificiel.

2.2.2.2 Multilayer Perceptron (MLP)

C'est une catégorie de réseaux neuronaux artificiels (ANN), constituée d'au moins trois couches de nœuds : une couche d'entrée, une couche cachée et une couche de sortie. À l'exception des nœuds d'entrée, chaque nœud (perceptron) utilise une fonction d'activation (généralement non linéaire e.g la fonction sigmoïde) [30] et génère une probabilité qui influencera la mise-à-jour des poids du réseau.

(a) MLP avec Une Couche

(b) MLP avec Deux Couches

FIGURE 2.7 – MLP avec Différents Architectures.

Cette architecture a montré son efficacité avec les problèmes où les techniques classiques du machine learning atteignent leurs limites.

2.2.2.3 Fonctions d'activation

Elle permet d'ajouter des complexités non-linéaires au réseau. Il existe différentes fonctions d'activation : la fonction linéaire, sigmoïde, tangente hyperbolique, etc.

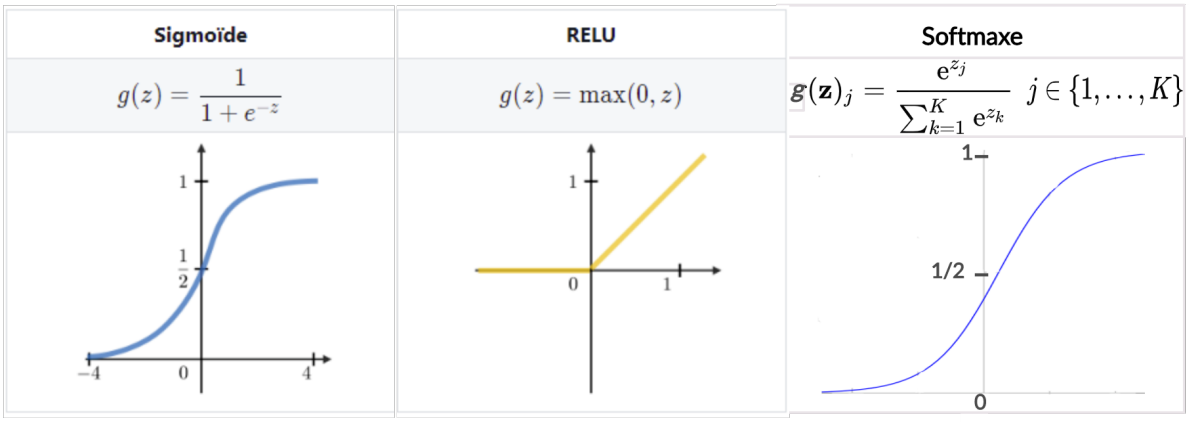


FIGURE 2.8 – Les Fonctions d'Activations [?].

La fonction SIGMOIDE

$$f(\varphi) = \frac{1}{1+e^{-\varphi}}$$

C'est une fonction d'activation qui normalise les valeurs entre 0 et 1. L'inconvénient est que pour des valeurs très élevées ou très faibles de φ, il n'y a pratiquement pas de changement dans la prédiction. Ce qui engendre le problème de disparition du gradient [16]. C'est un phénomène dans lequel les gradients deviennent de plus en plus petits au fur et à mesure que l'algorithme de rétro-propagation progresse vers les couches inférieures (les plus proches de l'entrée) jusqu'à devenir insignifiants. Par conséquent, la mise à jour du gradient laisse les poids des couches inférieures pratiquement inchangés, subséquemment l'algorithme ne pourra pas converger vers une meilleure solution [16]. Le réseau peut alors arrêter d'apprendre, ou devient trop lent pour obtenir une prédiction précise [26].

La Fonction RELU

$$f(\varphi) = max(0, \varphi) = \begin{cases} \varphi \text{ si } \varphi \geqslant 0 \\ 0 sinon \end{cases}$$

C'est une fonction qui a montré de bonnes performances lorsqu'elle est utilisée dans les couches cachées. Cependant, lorsque les entrées ont des valeurs proches du zéro, ou elles sont négatives, le gradient de la fonction devient nul et le réseau ne pourra pas effectuer de la rétro-propagation [16] et ne peut pas apprendre [26].

La Fonction SOFTMAX

$$f(\varphi)_j = \frac{e^{\varphi_j}}{\Sigma_{i=1}^{k} e^{\varphi_i}} j \in \{1...k\}$$

— K est la dimension du vecteur φ.

En général, Softmax n'est utilisé que pour la couche de sortie des réseaux dits "*classifieurs*" . Son avantage est qu'elle normalise les sorties en probabilités pour chaque classe entre 0 et 1. Cette classe est choisie en prenant évidemment la plus grande probabilité [26].

2.2.2.4 Apprentissage

L'apprentissage peut être supervisé dans le cas de la nécessité d'étiqueter un ensemble de données, ou bien non supervisé dans le cas contraire. Dans notre projet, on s'intéresse à l'apprentissage supervisé, où un réseau de neurones essaie de matcher les variables d'entrées X à celles de sorties Y. Ce qui consiste à réduire l'erreur $e_i = y_i - \hat{y}_i$ entre la valeur prédit $\hat{y}$ et la valeur réelle y. L'apprentissage est un processus itératif, où le modèle apprend des patterns et met à jour les différents paramètres du réseau (selon le problème à traiter) en se basant sur la descente du gradient. Afin d'évaluer la pertinence de son entraînement, on réalise des tests à l'aveugle.

«Un perceptron multicouches n'est qu'une fonction mathématique mettant en correspondance un ensemble de valeurs d'entrée avec des valeurs de sortie» [14]

«Un réseau feedforward définit un mappage et apprend la valeur des paramètres qui donnent la meilleure approximation de la fonction» [14]

Les poids sont initialisés uniformément, en général en prenant des valeurs aléatoires. Chaque donnée $(\overrightarrow{x}, \overrightarrow{y})$ est traitée séquentiellement en deux phases (voir le pseudo-code suivant).

Pour chaque $e \in \{1..P\}$ (P nombre d'époques) :
  Pour chaque $n \in \{1..N\}$ (N nombre de segments $(\overrightarrow{x}, \overrightarrow{y})$) :

1. Propagation vers l'avant :
   - Pour tous les neurones dans la couche de l'entrée on leur donne un évènement à partir de $\overrightarrow{x}$.
   - Exécuter le fonctionnement du neurone (voir 2.2.2.1).
   - Utiliser les résultats retournés par chaque neurone de la couche d'entrée comme entrée pour tous les neurones de première couche cachée.
   - Continuer cette propagation de couche en couche successivement jusqu'à avoir une prédiction $\overrightarrow{\hat{y}}$ à la couche de sortie.
2. Propagation vers l'arrière :
   - Comparer le résultat obtenu $\hat{y}$ avec le résultat attendu y et calculer l'erreur e_i à travers une fonction d'erreur (voir 2.2.2.5).
   - Utiliser la valeur de l'erreur pour calculer le gradient.
   - Utiliser le gradient pour mettre à jour les poids w_{ij}.

  fait ; fait ;

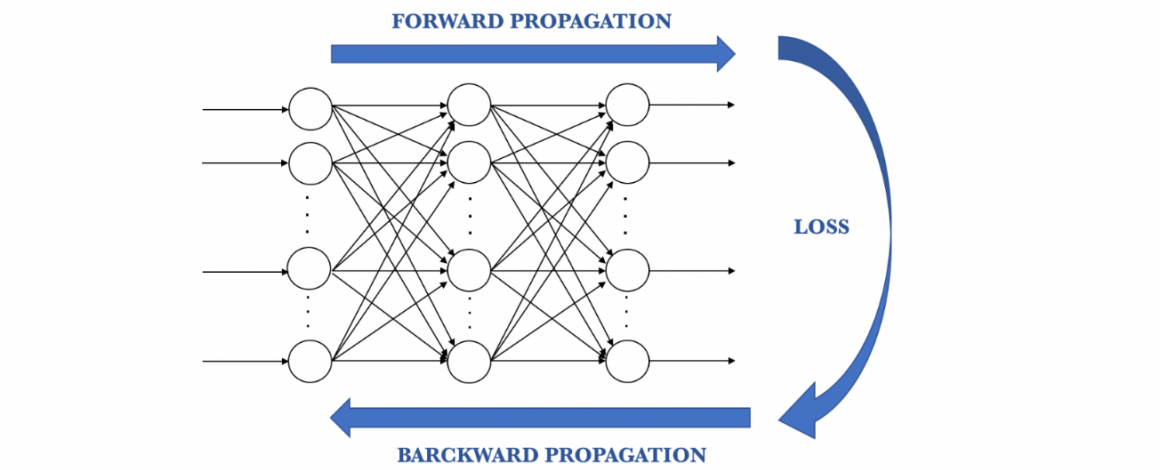


FIGURE 2.9 – Le Processus d'Apprentissage d'un Réseau de Neurones.

En fait, la rétro-propagation est un algorithme d'optimisation qui aide au calcul du gradient. Il y a plusieurs algorithmes d'optimisations tel que la descente stochastique du gradient, qui est utilisée pour effectuer l'apprentissage à l'aide des gradients [14].

2.2.2.5 Composantes de l'Algorithme d'Apprentissage

Entraîner un modèle de réseau neuronal implique de choisir un certain nombre de composants et d'hyper-paramètres. Dans cette partie, nous allons examiner chacun d'eux.

Fonctions d'Erreur (Loss Function) : la fonction utilisée pour calculer l'écart entre les prédictions du modèle et les valeurs réelles de l'ensemble de données d'entraînement [16], aussi appelée "fonction objectif". Généralement, on choisit un cadre probabiliste spécifique pour l'inférence, appelé Maximum Likelihood. Dans ce cadre, les fonctions d'apprentissage couramment choisies sont l'entropie croisée pour les problèmes de classification et l'erreur quadratique moyenne pour les problèmes de régression.

— Erreur quadratique moyenne MSE [14] :

$$MSE(Y, \varphi(x)) = \frac{1}{n}\sum_{i=1}^{n}(W^{\intercal}.x_i - y_i)^2 \tag{2.2}$$

— Entropie croisée (Cross entropy) : l'entropie croisée varie selon la forme de l'étiquette y et de la prédiction $\hat{y}$. Si c'est un vecteur de probabilité où chaque dimension représente une classe à prédire alors on utilise *L'entropie croisée catégorique (categorial crossentropy)* présentée par l'équation (2.3), sinon si y et $\hat{y}$ sont des valeurs entières représentant une classe, on utilise *entropie croisée catégorique clairsemée (sparse categorial crossentropy)* présentée par l'équation (2.4).

$$CCE(Y, \hat{Y}) = -\sum_{i=0}^{m}(\overrightarrow{y_i} * log(\overrightarrow{\hat{y}_i})) \tag{2.3}$$

$$SCCE(Y, \hat{Y}) = -\sum_{i=0}^{m}(y_i * log(\hat{y}_i)) \tag{2.4}$$

Initialisation du Poids : la procédure par laquelle les petites valeurs aléatoires initiales sont attribuées à des poids modèles au début du processus de formation [14]. Comme la surface d'erreur est non convexe, l'algorithme d'optimisation est sensible au point de départ initial. C'est pourquoi de petites valeurs aléatoires sont choisies comme poids initiaux du modèle, bien que différentes techniques puissent être utilisées pour sélectionner l'échelle et la distribution de ces valeurs.

Batch Size : le batch size représente le nombre d'exemples dans chaque segment d'entraînement, qui est aussi un paramètre empirique.

Taux d'Apprentissage : le Taux d'apprentissage est un scalaire positif qui détermine la longueur du pas pendant la descente du gradient [14]. Bien choisir ce scalaire est crucial car avec une trop grande valeur. La descente du gradient pourrait ne jamais converger, et avec une valeur trop petite l'algorithme pourrais atterrir dans un optimum local et ne pas atteindre l'optimum global (Figure 2.10).

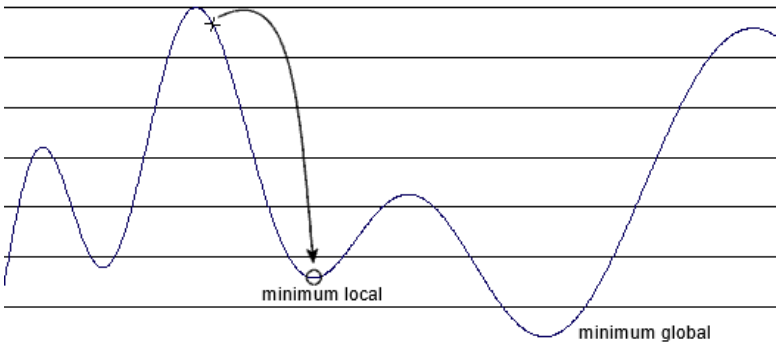


FIGURE 2.10 – Minimum Local et Global.

Nombre d'itérations (Epochs) : une itération est définie par une passe complète sur l'ensemble des données d'entraînement avant que le processus d'entraînement ne soit terminé [14]. A chaque itération, le modèle essaie de trouver un optimum local. Et sur toutes les itérations, il essaiera de trouver l'optimum global.

2.2.2.6 Réseau de Neurones récurrents (RNN)

Il s'agit d'une classe de réseaux neuronaux artificiels qui tiennent compte du temps et de la séquence. Utilisés énormément pour le traitement de données séquentielles (voix, texte, musique, mouvement d'un objets, ...etc), les réseaux récurrent sont fondamentalement les modèles les plus adaptés pour traiter nos données (séquences de notes). Leur architecture flexible permet, contrairement au modèle MLP et CNN de base [1] de travailler avec des séquences de vecteur de tailles variables et de manière différente, séquence de vecteurs en entrée, en sortie ou les deux.

1. il existe des implémentions de CNN qui permettent de traiter des matrice de taille variable [20].

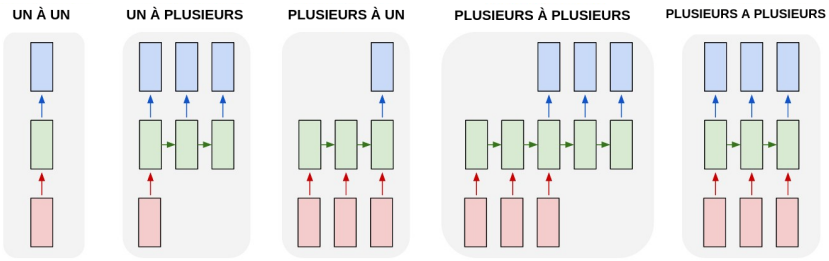
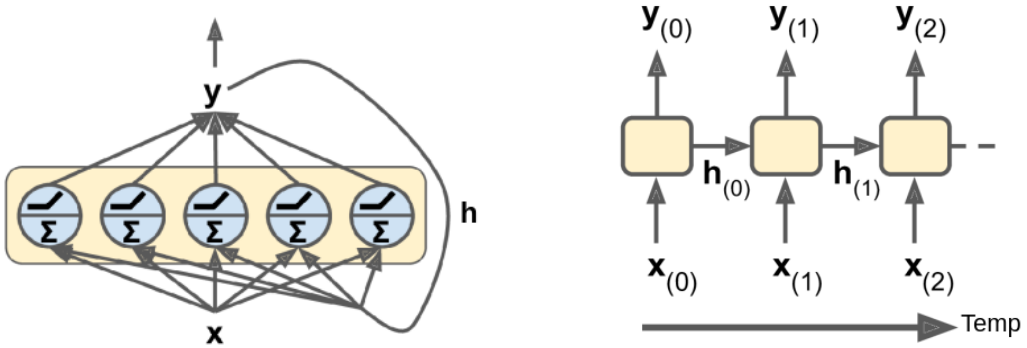


FIGURE 2.11 – Les Différentes Architectures d'Entrées-sorties Prises en Charge par Les RNN [19].

Une cellule RNN possède un nombre d'unités récurrentes (i.e neurones) qui prennent en entrée les éléments (i.e vecteurs) de la séquence l'un après l'autre tout en générant un vecteur d'état H et un vecteur de sortie Y à chaque instant t. Cette façon de traiter les éléments d'une même séquence (un après l'autre) est dite *dérouler le réseau dans le temps.*



(a) Une Cellule RNN. (b) Une Cellule Dérouler dans le Temps.

FIGURE 2.12 – Structure d'un RNN.

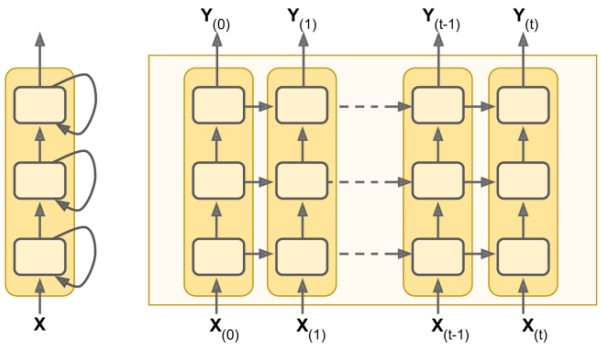


FIGURE 2.13 – Structure de Plusieurs RNN Empiler avec les Noeuds Visible [16].

Le vecteur d'état est ajouté à l'entrée à chaque instant t, où un vecteur de zéros est mis en entrée à l'instant $t = 0$. Le vecteur h pour les *hidden state* est calculé en combinant l'entrée X_t avec h_{t-1}, et le vecteur de sortie est calculé en utilisant h_t :

$$h_t = g(W_{hh}h_{t-1} + W_{hx}X_t + b)$$

$$Y_t = g'(W_{yh}h_t + b)$$

— g **:** est une fonction d'activation (généralement *tanh* ou *ReLu*).
— g' **:** est une fonction d'activation (généralement *sigmoid*).
— W_{hh} **:** matrice des poids correspondant à la connexion entre le vecteur d'état interne précèdent h_{t-1} et le vecteur d'état interne actuel h_t.
— W_{hx} **:** matrice des poids correspondant à la connexion entre le vecteur en entré x_t et le vecteur d'état interne actuel h_t.
— W_{yh} **:** matrice des poids correspondant à la connexion entre le vecteur de l'état interne actuel avec le vecteur de sortie.

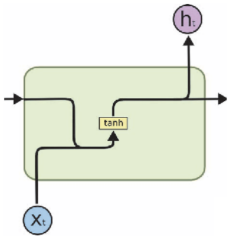


FIGURE 2.14 – Structure Interne d'une Cellule RNN avec $g = tanh$ [19].

Rétro-propagation dans le temps La Rétro-propagation dans le temps [31] est un algorithme similaire à la Rétro-propagation [16], sauf qu'en plus il prend en compte l'évolution du réseau dans le temps en considérant chaque état t du réseau comme une couche. Mais en réalité, c'est juste une application de la règle de la chaîne [4] pour calculer les gradients à chaque instant t, ce qui donne un genre de propagation verticale et horizontale, de la sortie vers l'entrée, et de la fin au début. La rétro propagation dans le temps sur l'axe temporelle met à jours les poids à chaque état peu importe s'il génère un vecteur dans cet état ou pas (Figure 2.15).

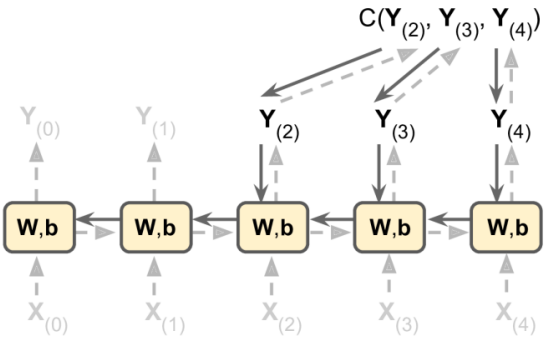


FIGURE 2.15 – Rétro Propagation dans le Temps d'un Réseau Plusieurs à Plusieurs ou les 2 Premières Sortie sont Ignorée [16].

— **W :** les matrices de poids de la cellules.
— **b :** biais.
— **C :** fonction de coûte.

Réseau Récurrent à Mémoire Court/Long (LSTM) En pratique, c'est un peu difficile d'entraîner des RNN standard pour résoudre des problèmes qui nécessitent l'apprentissage de séquences excessivement longues, car ces derniers font face au problème de disparition du gradient [16]. Les réseaux LSTM sont un type des RNN qui utilisent des unités spéciales en plus des unités standard. Les unités LSTM comprennent une «cellule mémoire» qui peut conserver les informations pendant de longues périodes. Cette architecture leur permet d'apprendre des dépendances de plus long périodes.

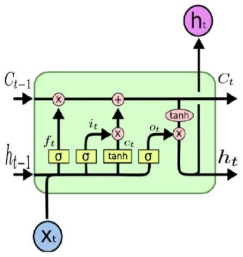


FIGURE 2.16 – Une Cellule LSTM.

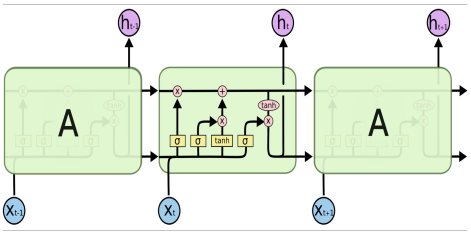


FIGURE 2.17 – Une Suite de Cellules LSTM.

FIGURE 2.18 – Structure d'un LSTM [19].

2.2.2.7 Réseau de Neurones Convolutionnel (CNN)

Un réseau neuronal convolutionnel (CNN) est une classe des réseaux neuronaux profonds, le plus souvent appliqués à la vision par ordinateur. Un réseau neuronal convolutionnel se compose d'une couche d'entrée et d'une couche de sortie, et de plusieurs couches cachées. Les couches cachées sont généralement constituées de succession de couches convolutionnelles suivies de couches de pooling (regroupement). La couche convolutionnelle permet, grâce à ses filtres d'effectuer des opérations de convolution (produit matriciel), ainsi générer une nouvelle matrice interne qui comprend les différentes caractéristiques des données de l'entrée. Cette nouvelle matrice constitue le nouvel espace de recherche. La couche de pooling est une opération de sous-échantillonnage typiquement appliquée après une couche convolutionnelle. Cette opération permet de réduire l'espace de recherche en prenant uniquement les solutions les plus proches d'un optimum local. À la fin des architectures de CNN, nous trouvons généralement des couches entièrement connectées. Elle peuvent être utilisées pour optimiser des objectifs tels que les scores de classe [2].

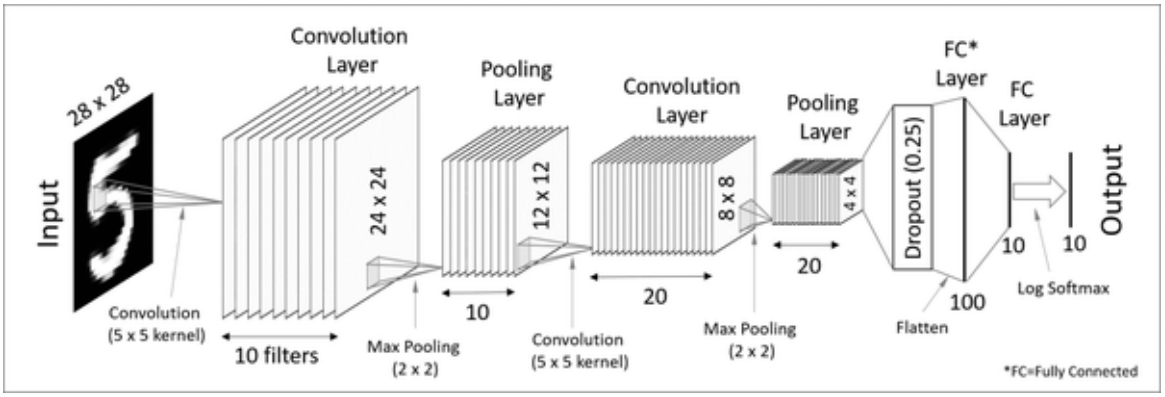


FIGURE 2.19 – Structure d'un Réseau Neuronal Convolutionnel.

2.2.2.8 Réseaux de Neurones Antagonistes Génératifs (GANs)

Introduit en 2014 par Ian J. Goodfellow [15], les GANs sont une nouvelle approche pour générer des données (entre autres des données musicales). Les GANs reposent sur deux architectures de différents réseaux neuronaux. Le premier appelé un générateur qui prend en entrée des variables aléatoires et essaye de générer des données "sans contexte". Le second, dit discriminant, calcule la différence entre les données générées par le modèle génératif et les données réelles. Tant que les résultats restent non indicatifs, le modèle discriminant ajustera les poids des neurones du modèle génératif par l'algorithme de rétro-propagation [15].

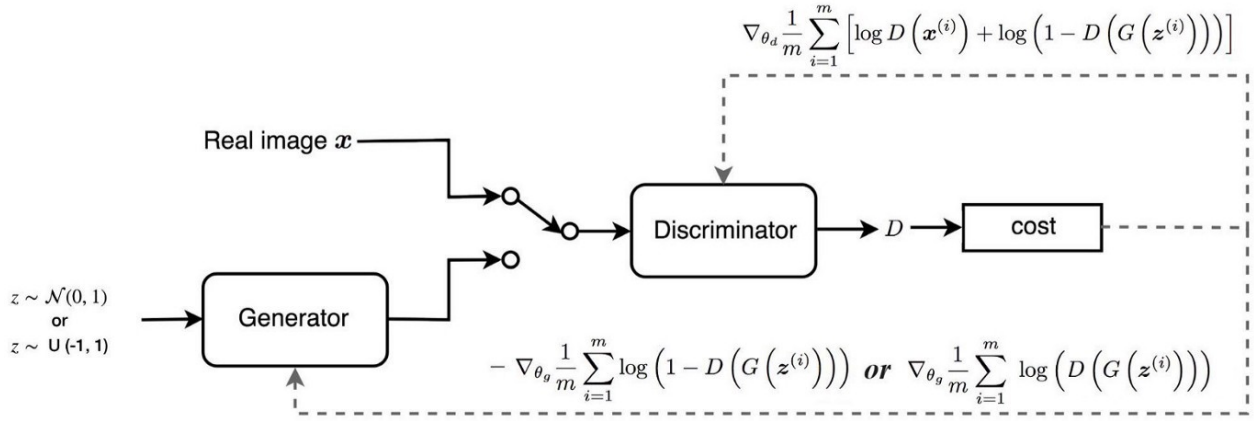


FIGURE 2.20 – Structure d'un Apprentissage Machine avec GAN.

L'erreur du générateur est calculée comme suit :

$$L_G = \frac{1}{m} \sum_{i=1}^{m} log(1 - D(G(z^i)))$$

où :

- z^i est une séquence de vecteurs aléatoires uniformes.
- $D(G(z^i))$ est la prédiction du discriminateur pour la sortie du générateur à partir de z^i.
- m est le nombre d'échantillons générés.

Par ailleurs, l'erreur du générateur est donné par la formule suivante :

$$L_G = \frac{1}{m} \sum_{i=1}^{m} [+logD(x^i) + log(1 - D(G(z^i)))]$$

où :

- x^i est une séquence des données d'entraînement.
- $D(G(z^i))$ est la prédiction du discriminateur pour x^i.

Pour simplifier, le processus d'entraînement est le suivant [15] :

- Le générateur génère un échantillon de musique à partir de données aléatoires.
- Le discriminateur examinera la sortie de G et l'ensemble de données d'entraînement (données réelles) et essaiera de déterminer les "réelles" des "générées".
- En utilisant la fonction d'erreur, nous dérivons les gradients pour mettre à jour les poids de nos modèles.

2.2.2.9 Auto Encodeur

Un auto-encodeur, ou auto-associateur est un réseau de neurones artificiels utilisé pour l'apprentissage non supervisé de caractéristiques discriminantes. L'objectif d'un auto-encodeur est d'apprendre une représentation (encodage) d'un ensemble de données, généralement dans le but de réduire la dimension de cet ensemble. La forme la plus simple d'un auto-encodeur est un réseau de neurones non récurrents qui se propage vers l'avant, très semblable au MLP - ayant une couche d'entrée, une couche de sortie ainsi qu'une ou plusieurs couches cachées les reliant. Mais avec toutefois une couche de sortie possédant le même nombre de nœuds que la couche d'entrée. Son objectif étant de reconstruire ses entrées [16].

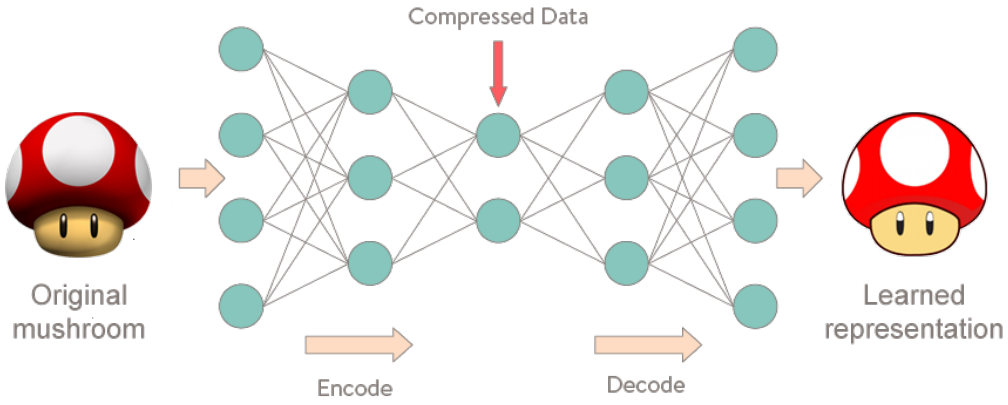


FIGURE 2.21 – Structure d'un Auto Encodeur.

Possédant deux composants : un encodeur et un décodeur. Le premier compresse l'information et le deuxième doit la reconstituer aussi précise que possible, dans le but de trouver des features de représentations plus riches.

2.3 Génération de Musique Grâce aux Techniques du Machine Learning

Les algorithmes sont utilisés pour composer de la musique depuis des siècles. Certains algorithmes qui n'ont pas de pertinence musicale immédiate sont utilisés par les compositeurs comme source d'inspiration pour leur musique. Une façon de classer les algorithmes de composition est de les classer en fonction de leur structure et de la façon dont ils traitent les données. À travers l'histoire, on trouve différents types. Les plus courants sont :

— les modèles mathématiques.
— Les modèles à base d'apprentissage automatique.

2.3.1 La Génération de Musique Avec Les HMMs

Iannis Xenakis a utilisé des chaînes de Markov dans ses compositions en 1958. Il décrit son procédé dans "Formalized Music : Thought and Mathematics in Composition", jusqu'aux détails des matrices de transition qui définissent les probabilités de production de certaines notes [18]. Les chaînes de Markov supposent la propriété Markov "sans mémoire", car la probabilité du symbole suivant est calculée sur la base des K symboles précédents. En pratique, K est limité à des valeurs faibles (3-5), par conséquent, les séquences musicales générées par un modèle de Markov sont soit incohérentes (comme le reflète cette étude de l'Université de Duke [32]), ou elles ne peuvent produire que des sous-séquences qui existent également dans les données d'entraînement [13] comme le montre David Cope en combinant les chaînes de Markov et d'autres techniques (grammaires musicales et combinatoires) dans un système semis-automatique qu'il appelle "Experiments in Musical Intelligence", ou Emmy [7], qui était célèbre pour apprendre et imiter d'autres compositeurs.

MTPZ

↓	A $(f_0g_0d_0)$	B $(f_0g_0d_1)$	C $(f_0g_1d_0)$	D $(f_0g_1d_1)$	E $(f_1g_0d_0)$	F $(f_1g_0d_1)$	G $(f_1g_1d_0)$	H $(f_1g_1d_1)$
$A(f_0g_0d_0)$	0.021	0.357	0.084	0.189	0.165	0.204	0.408	0.096
$B(f_0g_0d_1)$	0.084	0.089	0.076	0.126	0.150	0.136	0.072	0.144
$C(f_0g_1d_0)$	0.084	0.323	0.021	0.126	0.150	0.036	0.272	0.144
$D(f_0g_1d_1)$	0.336	0.081	0.019	0.084	0.135	0.024	0.048	0.216
$E(f_1g_0d_0)$	0.019	0.063	0.336	0.171	0.110	0.306	0.102	0.064
$F(f_1g_0d_1)$	0.076	0.016	0.304	0.114	0.100	0.204	0.018	0.096
$G(f_1g_1d_0)$	0.076	0.057	0.084	0.114	0.100	0.054	0.068	0.096
$H(f_1g_1d_1)$	0.304	0.014	0.076	0.076	0.090	0.036	0.012	0.144

FIGURE 2.22 – Un Extrait du Chapitre 3 du Livre de Iannis Xenakis, "La musique stochastique markovienne : Applications".

2.3.2 La Génération de Musique avec les RNNs

Tandis que les chaînes de Markov entraînées sur un ensemble de composition ne peuvent produire que des sous-séquences qui existent dans les données d'origine [13], les réseaux neuronaux récurrents (RNN) tentent d'améliorer le processus en apprenant a combiner les notes pour générer des suites plus cohérentes. En 1989, les premières tentatives de génération de musique avec les RNN, développées d'abord par Peter M. Todd [28], puis par Michael C. Mozer et d'autres, avaient une cohérence limitée à de courts morceaux. En février 2017, une équipe de Montréal dirigée par Yoshua Bengio a publié SampleRNN [24] pour générer du son en utilisant un ensemble de réseaux récurrents dans une structure hiérarchique.

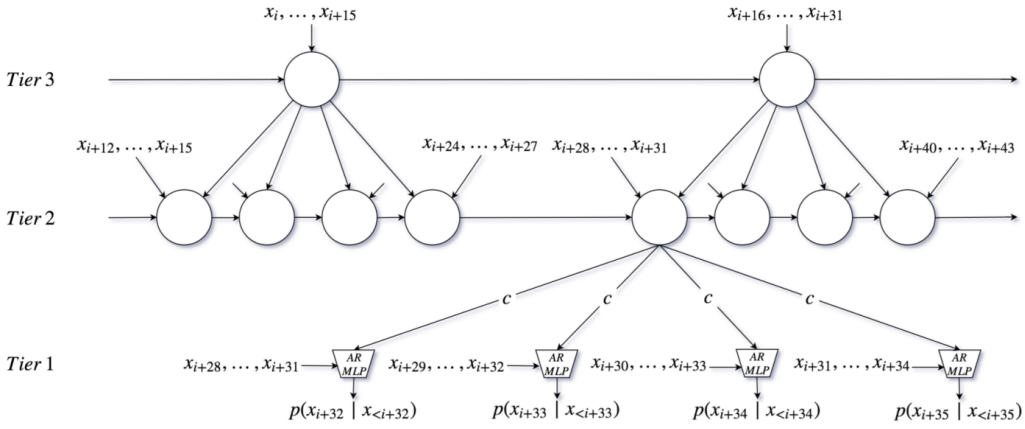


FIGURE 2.23 – Aperçu Simplifié de l'Architecture SampleRNN : une hiérarchie de réseaux récurrents (niveaux (tier) 2 et 3), combinés à un réseau neuronal standard (niveau 1).

2.3.3 La Génération de Musique avec les LSTMs

En 2002, Doug Eck a actualisé cette approche [10] en passant des cellules RNN aux cellules LSTM où il a utilisé son architecture pour improviser du blues. Il écrit : "*le LSTM est capable de jouer du blues avec un bon timing et une structure appropriée tant que l'on est prêt à écouter*". Il dirige maintenant l'équipe Magenta de Google Brain, où ils développent et partagent du code lié à l'apprentissage machine et à la créativité depuis le début de 2016. Un autre travail de l'Université de Columbia [22] utilise un LSTM pour générer de la musique. Le modèle accorde également une attention particulière à la saisie de l'utilisateur pour sélectionner les différents hyperparamètres. Un autre article de l'université de Stanford utilise une structure très similaire :

"une architecture LSTM à deux couches pour produire un modèle au niveau des personnages pour prédire la prochaine note dans une séquence".

2.3.4 La Génération de Musique avec les CNNs

En septembre 2016, DeepMind a publié ses recherches sur WaveNet [29] démontrant une architecture qui peut construire des abstractions audio de haut niveau.

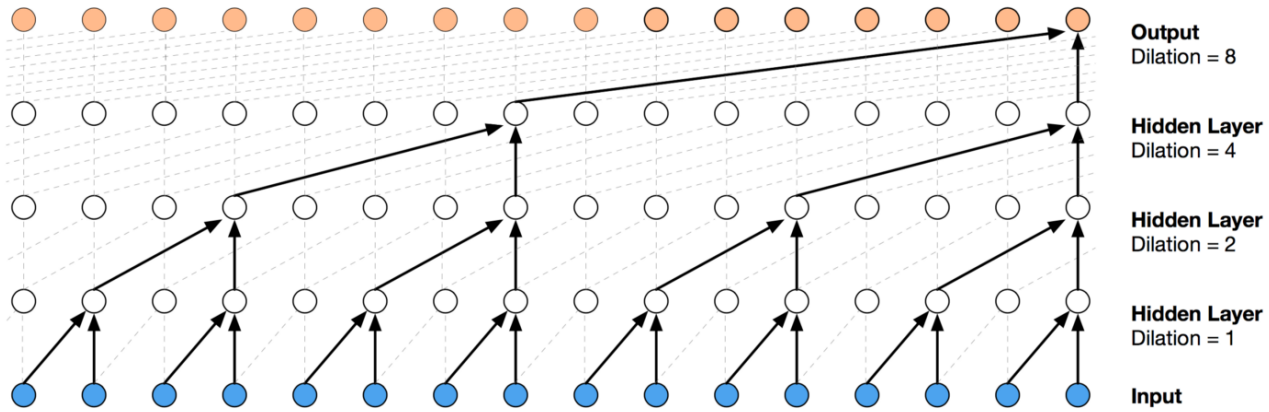


FIGURE 2.24 – Diagramme des Convolutions Dilatées Utilisées dans l'Architecture WaveNet.

2.3.5 La Génération de Musique avec les GANs

Réalisant d'excellents résultats dans la génération d'image (deepFake) en 2017, Hao-Wen Dong et al se sont basés sur une architecture de GAN et un réseau convolutionnel pour créer leur modèle "MuseGan" [17] qui a permis de générer des morceaux de musique polyphonique multi-track (plusieurs instruments). Ils ont entraîné le modèle à l'aide de données collectées dans l'ensemble de données du piano de lakh pour générer des morceaux de chansons pop composées de pistes de basse, batterie, guitare, piano et de cordes.

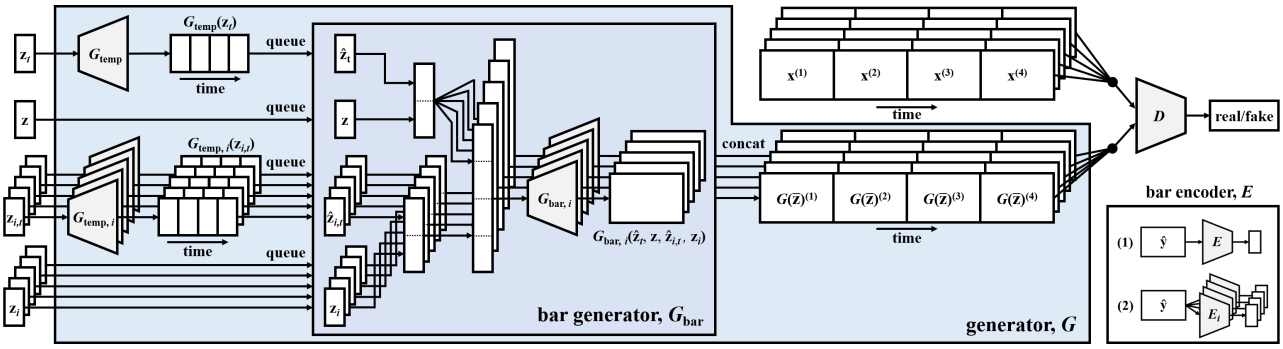


FIGURE 2.25 – Structure de MuseGAN.

La particularité de ce modèle évolué est de considérer les mesures comme élément de base de la génération de musique au lieu de notes individuelles.

2.4 Conclusion

Pour aborder notre projet, nous avons présenté dans ce chapitre des notions essentielles sur la théorie de la musique et l'apprentissage profond en examinant des travaux de l'état de l'art

sur la question. Ceci, nous a permis d'avoir une meilleure vision pour la réalisation de notre système pour lequel la préparation des données ainsi que le choix des modèles sont une étape primordiale. C'est ce qui fera l'objet de présentation dans le chapitre suivant.

Chapitre 3

Données et Modèles

Dans ce chapitre, nous parlerons des choix de l'ensemble des données d'entraînement et de modèles retenus.

3.1 Préparation des données d'entraînement

L'ensemble de données que nous avons choisi est de la musique pour piano. Elle est principalement composée de musique provenant des bandes sonores de Final Fantasy. Nous avons choisi la musique de Final Fantasy en raison d'une part des mélodies belles et distinctes de la plupart des morceaux et de la quantité de morceaux qui existent et d'autre part en raison de contraintes de ressources (mémoire), nous obligeant à limiter la taille de notre ensemble de données. Les notes des morceaux de ce dataset ont pour la plupart la même durée ce qui va faciliter le traitement. Il contient 92 morceaux de piano au format audio MIDI qui est une norme technique pour décrire un protocole de communication, une interface numérique et des connecteurs électriques qui relient une grande variété d'instruments de musique électroniques, d'ordinateurs et de dispositifs audio connexes pour jouer, éditer et enregistrer de la musique [27]. On trouve avec 65377 sons (note, accord, silence) et 359 sons uniques.

L'un des défis principaux dans l'entraînement d'un modèle pour la génération de musique est de choisir la bonne représentation de données. Dans ce projet, nous utiliserons Music21 pour extraire le contenu de nos fichiers MIDI, et créer des fichiers MIDI depuis les valeurs en sortie du réseau neuronal. Tout d'abord, nous allons charger les données dans un tableau comme on peut le voir dans l'extrait de code ci-dessous :

```
files = glob(files_path+'/*.mid*')
note_list = []
try :
    F = open('NoteList','rb')
    note_list = pickle.load(F)
    F.close
    print("List de notes Trouvée et chargé")
    print("le nombre total de son est : ",len(note_list))
except :
    n_chords = 0
    n_notes = 0
    n_silences = 0
    i = 0
    #on boucle sur la liste des chanson et on recuper les notes de chaqune
    for f in files :
        print("\rprocessing","."*((i+1)%4),end='')
        song = converter.parse(f)
        #on verifie si il y'a plusieur instrument si c'est le cas on prend que la partie piano
        instrument_parts = instrument.partitionByInstrument(song)
        if instrument_parts :
            song_notes = instrument_parts.parts[0].recurse()
        else:
            song_notes = song.flat.notes
        note_list.append('<start>')
        # on boucle sur les notes de chaque chanson et on les ajoutes a note_liste
        for n in song_notes :
            if isinstance(n,note.Note):
                n_notes += 1
                note_list.append(str(n.pitch))
            elif isinstance(n,note.Rest):
                n_silences += 1
                note_list.append('SC')
            elif isinstance(n,chord.Chord):
                n_chords += 1
                chrd = '.'.join(str(x) for x in n.normalOrder)
                note_list.append(chrd)
        note_list.append('<end>')
        i += 1
```

FIGURE 3.1 – Le Chargement de Données dans un Tableau.

Nous commençons par charger chaque fichier dans un objet Music21 en utilisant la fonction *converter.parse(file)*. Avec cet objet, nous obtenons une liste de toutes les notes et accords du fichier comme nous pouvons le voir ci-dessous :

```
<music21.note.Note B>
<music21.note.Rest rest>
<music21.chord.Chord E3 B3>
<music21.chord.Chord C#4 F#4 B-4>
<music21.chord.Chord B-3 F#3>
<music21.chord.Chord C#4 E-4 F#4>
<music21.chord.Chord F#3 C#3>
<music21.chord.Chord E4 C#4 G#4>
<music21.chord.Chord C#3 G#3>
<music21.note.Note B>
<music21.chord.Chord E3 B3>
<music21.chord.Chord C#4 F#4 B-4>
<music21.chord.Chord F#3 B-3>
<music21.note.Note G#>
<music21.note.Note B->
```

FIGURE 3.2 – Un Extrait d'un Fichier MIDI Lu avec Music21.

On boucle sur la liste des chansons et pour chaque chanson on récupère toutes les notes, accords et silences pour les ajouter à notre liste. Nous ajoutons la hauteur de chaque objet de note en utilisant sa notation en chaîne de caractères. Et nous ajoutons chaque accord en encodant l'id de chaque note de l'accord ensemble en une seule chaîne, chaque note étant

séparée par un point. La méthode *normalOrder()* de l'objet *Chord* permet d'obtenir les notes de l'accord au format de nombre entier, et pour les silences on a choisi un token 'SC' pour les représenter vu qu'ils n'ont pas de hauteur.

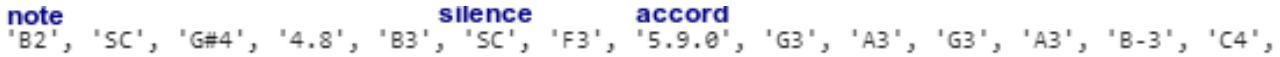


FIGURE 3.3 – Un Extrait de la Liste des Notes Apres la Recuperation.

Enfin, nous allons créer deux fonctions de mappage, une permettant de passer de données basées sur des chaînes de caractères à des données numériques *note2index()* et vice-versa *index2note()* pour décoder le résultat généré par le réseau de neurones. Cela est fait parce que les réseaux de neurones fonctionnent beaucoup mieux avec des données numériques qu'avec des données basées sur des chaînes de caractères [16].

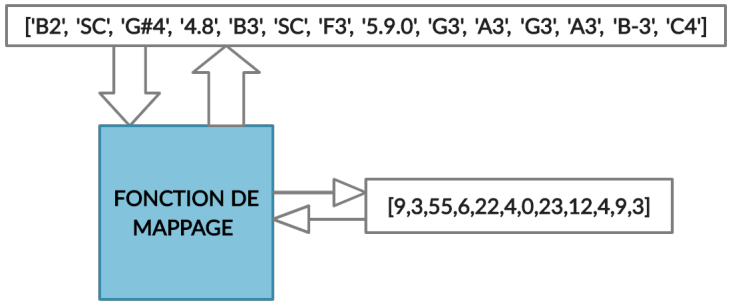


FIGURE 3.4 – La Conversion de Données Catégorielles en Données Numériques les Données sont Converties en Indices de Nombres Entiers Représentant la Position de la Catégorie dans l'Ensemble des Valeurs Distinctes.

3.2 Modèles

Pour ce projet et après une documentation sélective sur ce qui a tendance à donner les meilleurs résultats à la fois pour le traitement et la génération de texte, image et musique, nous avons choisi 3 modèles : le premier étant un simple RNN qui a été construit comme ligne de base pour ce travail. Nous avons ensuite amélioré ce modèle en passant à un modèle LSTM. Et enfin, nous avons essayé d'améliorer encore plus nos résultats en essayant un modèle GAN. L'architecture de chaque modèle sera décrite ci-dessous.

3.2.1 Expérience 1

Pour notre ligne de base, nous commençons par un simple RNN d'une seule couche, avec 512 unités récurrentes. Cette grande complexité permettra au modèle d'apprendre efficacement puisque notre nombre de classes est déjà assez élevé (359 sons uniques). Nous avons initialement choisi un taux d'abandon de 0,1 qui serait ajusté par l'expérimentation. Notre fonction d'activation est softmax puisque la sortie du modèle est la probabilité de la prochaine note qui sera jouée. Le tableau suivant récapitule l'architecture de ce modèle :

Paramètres	Valeur utilisée
Nombre de couches cachées	1
Nombre d'unités récurrentes	512
Taux d'abandon	0.1
Fonction d'activation	softmax

Paramètres d'entraînement :

— Algorithme d'optimisation : adam.
— Taux d'apprentissage : 0.01.
— Fonction de perte : entropie croisée catégorielle.

Pour l'entraînement, nous avons choisi Adam comme algorithme d'optimisation car il est populaire dans le domaine de l'apprentissage profond grâce à sa capacité à donner de bons résultats rapidement et avec peu de nécessité de mémoire. On a laissé le taux d'apprentissage par défaut et pour la fonction de perte, nous avons choisi entropie croisée catégorielle car notre problématique consiste à définir la classe (parmi les 359) à laquelle appartient notre prochaine note dans la séquence.

3.2.2 Expérience 2

Dans cette expérience, nous avons optimisé notre modèle de ligne de base en utilisant un LSTM, avec différents nombre de couches, différent nombres d'unités récurrentes et différent taux d'apprentissage. Pour l'architecture finale, nous avons opté pour 2 couches de LSTM, dont 1024 unités récurrentes sur la première et 512 sur la seconde. Nous avons également augmenté le Taux d'abandon à 0,3 pour éviter le sur-apprentissage (overfitting) c'est à dire lorsqu'un modèle, trop proche de données particulières, ne peut plus être généralisé. On pourrait faire le parallèle avec un humain qui apprend par cœur sans comprendre. Il lui est alors impossible de répondre à une question qu'il n'a encore jamais vue, bien que similaire à ce qu'il a appris. Nous avons gardé la fonction d'activation car nous traitions toujours la même problématique. Le tableau suivant récapitule l'architecture de ce modèle :

Paramètres	Valeur testée	Valeur retenue
Nombre de couches cachées	1, 3	2
Nombre d'unités récurrentes	256, 512, 1024	1024, 512
Taux d'abandon	0.2, 0.3	0.3
Fonction d'activation	softmax	softmax

Paramètres de d'entraînement : l'entraînement dans cette expérience a suivi la même structure que la première.

— Algorithme d'optimisation : adam.
— Taux d'apprentissage : 0.01.
— Fonction de perte : entropie croisée catégorielle.

3.2.3 Expérience 3

Dans cette troisième expérience, on se tourne vers un type de réseaux plus complexe que les précédents, les réseaux antagonistes génératifs 2.2.2.8. Le modèle est composé de deux autres modèles, un discriminant avec les deux premières couches comme des couches LSTM. Sans ces

couches, nous avons constaté que le discriminant était incapable de distinguer la vraie musique de la fausse, tant que le générateur était capable de déterminer le domaine de distribution des données d'entrée. Avec le LSTM, le générateur doit maintenant faire plus que simplement déterminer le domaine des données réelles. Il doit également déterminer que la musique suit un certain style. La fonction d'activation est sigmoïde puisque la sortie du discriminant est soit 1 soit 0. Le générateur est simplement un perceptron multicouche (MLP 2.7) qui reçoit des entrées aléatoires et produit des séquences de nombres compris entre -1 et 1 (indice tanh comme fonction d'activation). Cela a été fait pour aider le générateur car les données d'entraînement ont été normalisées entre -1 et 1 dans ce cas. Le tableau suivant récapitule l'architecture de ce modèle :

Paramètres	Valeur du générateur	Valeur du discriminant
Nombre de couches cachées	3×Dense	2×LSTM et 2×Dense
Nombre de noeuds/unités récurrentes	256, 512, 1024	2×512, 256, 1
Taux d'abandon	0.3	0.3
Fonction d'activation	tanh	sigmoid

Paramètres d'entraînement : pour l'entraînement, nous avons conservé les mêmes paramètres que les deux premières expériences. Mais, nous avons remplacé la fonction de perte par l'entropie croisée binomiale car notre problème consiste à choisir la plus forte probabilité entre la classification d'une séquence comme étant réelle ou fausse. On ne spécifie pas de fonction de perte ou d'algorithme d'optimisation pour le modèle du générateur parce que le générateur n'est pas entraîné directement.

— Algorithme : adam.
— Taux d'apprentissage : 0.01.
— Fonction de perte du discriminant et du modèle imbriqué : entropie croisée binomiale.

3.3 Conclusion

Dans ce chapitre, nous avons présenté les modèles retenus ainsi que les données sur lesquelles nos modèles vont être entraînés. Par la suite, nous allons voir plus en détails comment nous avons entraîné ces modèles ainsi que l'interface réalisée pour présenter notre travail.

Chapitre 4

Développement

Dans ce chapitre, nous parlerons de la mise en oeuvre des trois modèles (expérimentations) et du développement de l'interface réalisée en précisant les outils utilisés.

4.1 Technologies Utilisée

Cette expérience utilisera :

— Python 3.7 : un langage de programmation interprété, multiparadigme et multiplate-formes.
— Tensorflow v2.0 : un outil open source d'apprentissage automatique développé par Google.
— Music21 : une bibliothèque python pour manipuler et créer des fichiers MIDI.
— Colaboratory : un environnement gratuit de notebook Jupyter qui nécessite aucune installation et fonctionne entièrement dans le cloud équipé avec un GPU Tesla K80 ou même un TPU.
— Html 5 : un langage de balisage conçu pour représenter les pages web.
— Css 3 : un langage informatique qui décrit la présentation des documents HTML et XML.
— Flask 1.1.2 : un micro framework open-source de développement web en Python.

4.2 Expérience 1 et 2

Dans cette section, nous examinerons comment nous avons entraîné nos deux modèles RNN et LSTM, puis nous passerons au processus de génération de musique avec ces modèles.

4.2.1 L'entraînement

Maintenant que nous avons nos données prêtes et nos modèles construits, nous devons créer nos entrées de réseau avec leurs sorties respectives, afin que nous puissions commencer l'entraînement.

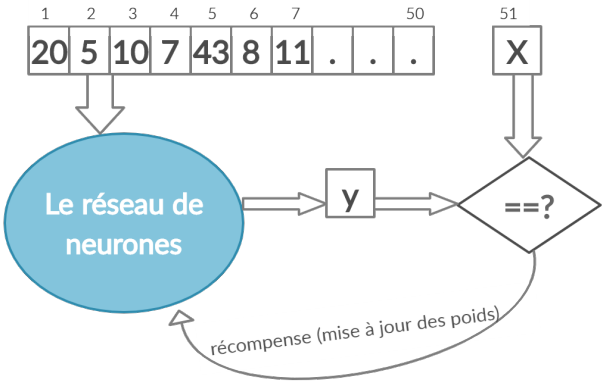


FIGURE 4.1 – Le Processus d'Entraînement du Réseau

Pour nos deux modèles récurrents (RNN, LSTM), nous choisissons d'utiliser une couche d'embedding au lieu d'encoder nos données d'entrée en vecteurs one-hot. Notre entrée sera donc au format (batch-size, seq-len). L'entrée du réseau sera formée de séquences de 50 notes, accords et silences, et la sortie de chaque séquence sera la première note, accord et silence qui la suivra dans notre liste de notes. Cela signifie que pour prédire la note suivante dans la séquence, le réseau dispose des 50 notes précédentes pour l'aider à faire la prédiction. Après avoir essayé différentes longueurs de séquence, nous avons choisi 50 comme longueur de la note.

4.2.2 La Production de Musique

Pour pouvoir générer de la musique, nous devons recréer un même modèle mais avec un batch-size de 1 et charger les poids de notre modèle entraîné dans celui-ci. Nous le faisons en employant la méthode *model.load_ weights()*.

```
1 Gmodel = create_model(1)
2 Gmodel.load_weights(tf.train.latest_checkpoint(checkpoint_dir))
```

FIGURE 4.2 – La Création du Modèle Générateur.

Nous pouvons maintenant utiliser le modèle entraîné pour commencer à générer des notes. Pour cela, nous allons choisir une liste de départ de notes pour nourrir notre réseau. Ce dernier produira une note que nous ajouterons à notre liste de départ. Ensuite, nous alimenterons notre réseau avec cette nouvelle liste. Nous répétons ce processus n fois, n étant le nombre de notes générées qui déterminera la durée de la chanson. Comme le montre la figure suivante :

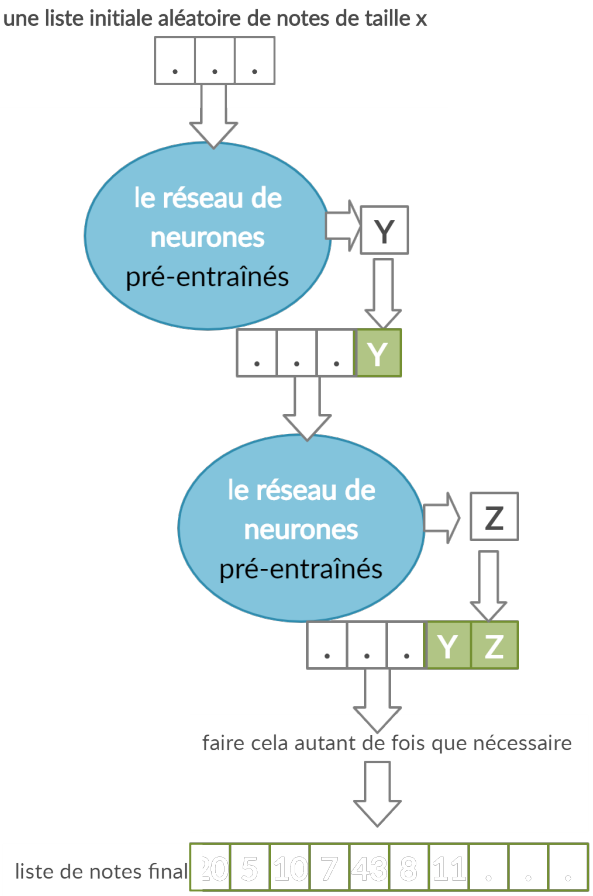


FIGURE 4.3 – Le Processus de Génération de Musique.

Maintenant que nous avons notre liste de notes, accords et silence générés par le réseau, nous pouvons créer nos fichiers MIDI en inversant le processus décrit précédemment (voir 3.1).

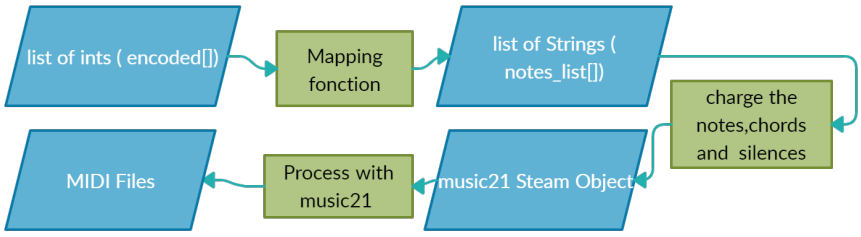


FIGURE 4.4 – La Création des Fiches Midi.

Chacun des modèles RNN et LSTM peut être entraîné. Et pour produire de la musique, il suffit d'exécuter les programmes dans les liens ci-dessous :
RNN : https://colab.research.google.com/drive/1IqKgDgvwet0wj_BSB3Bv3IMvKUPq8Aet?usp=sharing.
LSTM : https://colab.research.google.com/drive/1Y54iGLlSmfVdawlbbMNFoEoytoDFFU8a?usp=sharing.

4.3 Expérience 3

Dans cette section, nous examinerons comment nous avons entraîné notre modèle GAN, puis nous passerons au processus de génération de musique avec ce dernier.

4.3.1 L'Entraînement

L' entraînement de notre modèle GAN nécessite l'entraînement des deux modèles imbriqués séparément pour chaque itération. Nous commençons par entraîner le discriminant sur un segment (batch) comme indiqué dans la figure ci-dessous 4.5, puis nous entraînons le générateur sur un segment à l'aide du discriminant comme indiqué dans la figure après 4.6.

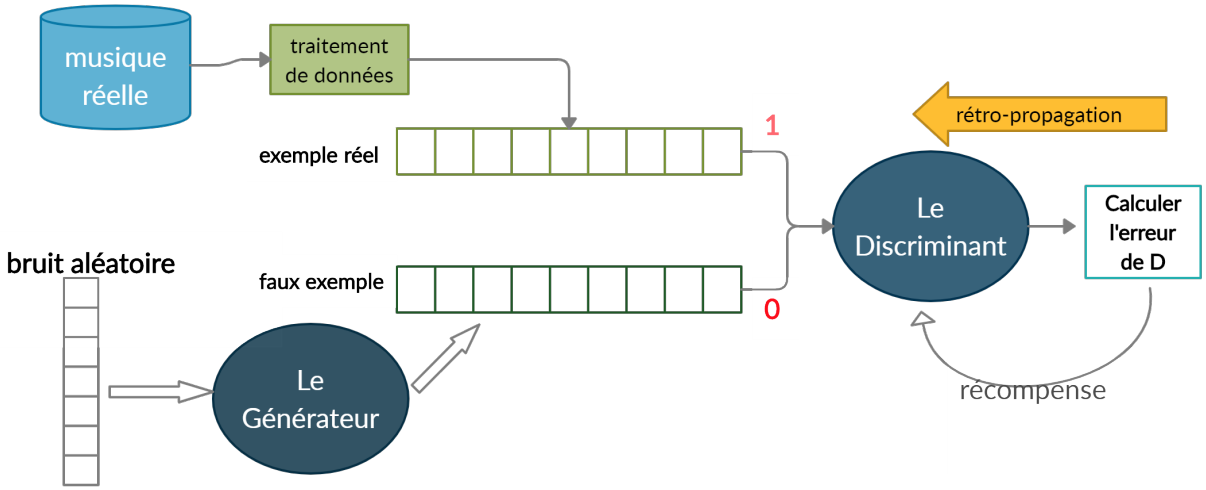


FIGURE 4.5 – La Première Phase d'Entraînement de notre GAN.

Pour entraîner le discriminant, nous commençons par nourrir le générateur avec des entrées aléatoires pour générer un segment de séquences de 100 notes (encodées sous forme de nombres) et nous étiquetons ces dernières comme étant des fausses données. Nous faisons également un segment de séquences à partir de données réelles et nous les étiquetons comme étant vraies. Ensuite, le discriminant passe par une itération d'entraînement classique, comme vu en 2.2.2.4. avec les deux segments et ajuste ses poids et ses biais.

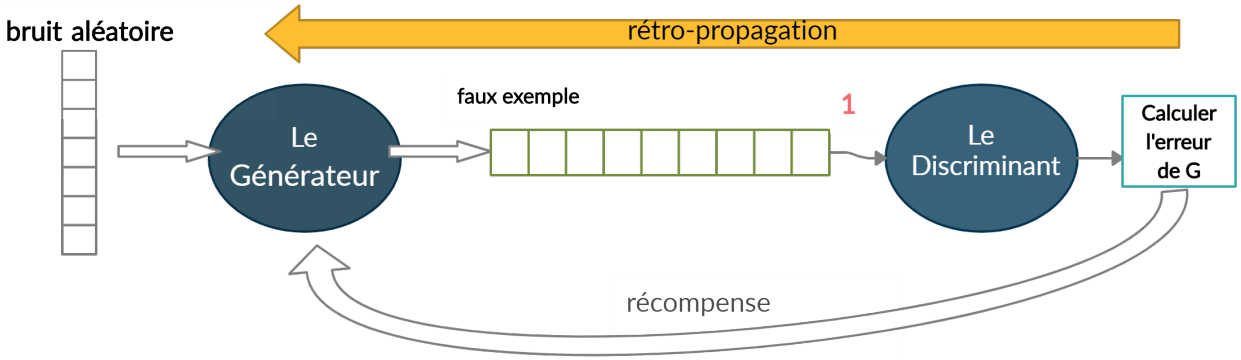


FIGURE 4.6 – La Seconde Phase d'Entraînement de Notre GAN.

Après avoir entraîné le discriminant pour une itération, le générateur est entraîné en formant tout le modèle imbriqué, et en gelant les poids et les biais du discriminant par l'utilisation de *discriminator.trainable = False* avant de compiler le modèle imbriqué. Pour ce faire, nous nourrissons le modèle imbriqué avec des entrées aléatoires afin que le générateur produise un autre segment de fausses données, mais cette fois-ci, nous les étiquetons comme étant vraies. Une fois que ces données ont été traitées par le discriminant et que l'erreur a été calculée, la rétro-propagation commence à la sortie de notre modèle imbriqué et revient dans le discriminant vers le générateur. Et comme nous avons gelé le discriminant, seul le générateur sera affecté par la mise à jour des poids et des biais.
L'entraînement s'arrête dès que le discriminant ne peut plus distinguer les données réelles (1) des fausses (0) (il produit 0,5).

4.3.2 La Production de Musique

Lorsque nous terminons l'entraînement de notre modèle, on prend seulement notre générateur et on lui donne une nouvelle entrée aléatoire à chaque fois pour obtenir un résultat différent. Nous convertissons ensuite ce résultat en fichier MIDI, comme nous l'avons déjà vu (figure 4.4).

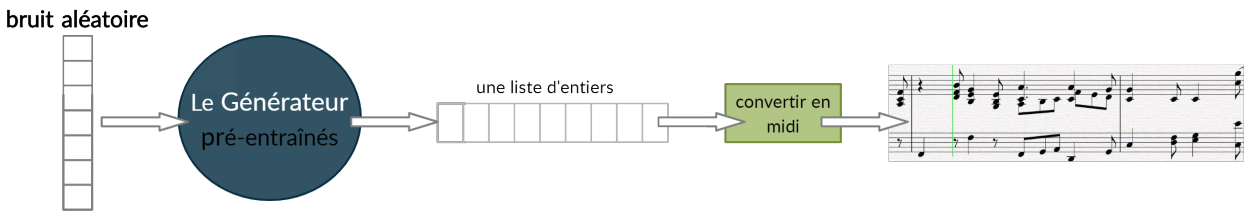


FIGURE 4.7 – La Génération de Musique avec le GAN.

Vous pouvez entraîner notre modèle de GAN et produire de la musique en exécutant le programme dans le lien suivant : https://colab.research.google.com/drive/1Y54iGLlSmfVdawlbbMNFoEo
usp=sharing.

4.4 Développement de l'interface

Dans cette section, les étapes principales de création de l'interface du projet sont présentées.

4.4.1 Partie Serveur

Pour générer de la musique avec nos modèles sans devoir les entraîner à chaque fois, nous avons sauvegardé nos modèles et leurs poids après leur entraînement. Puis, nous avons ajouté ces modèles pré entraînés à notre serveur. Désormais lors de chaque génération d'une chanson avec un modèle particulier, une demande est envoyée au serveur en utilisant le frontal de notre interface (html + css), puis on utilise flask afin d'exécuter le code python sur le serveur générant la musique. Cette musique est ensuite renvoyée au frontal pour l'utilisateur.

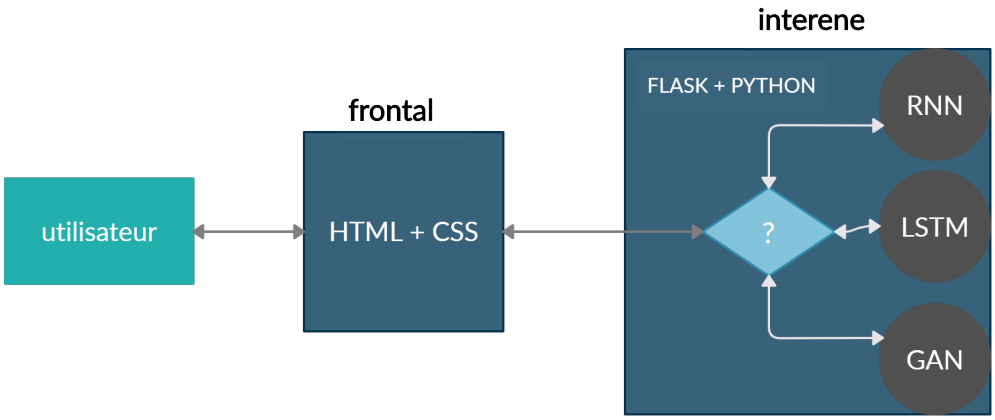


FIGURE 4.8 – Diagramme de l'interface.

4.4.2 Partie Interface

Notre interface se compose de deux pages. Dans la première page, nous trouvons un résumé rapide du projet et un clique sur la deuxième page.

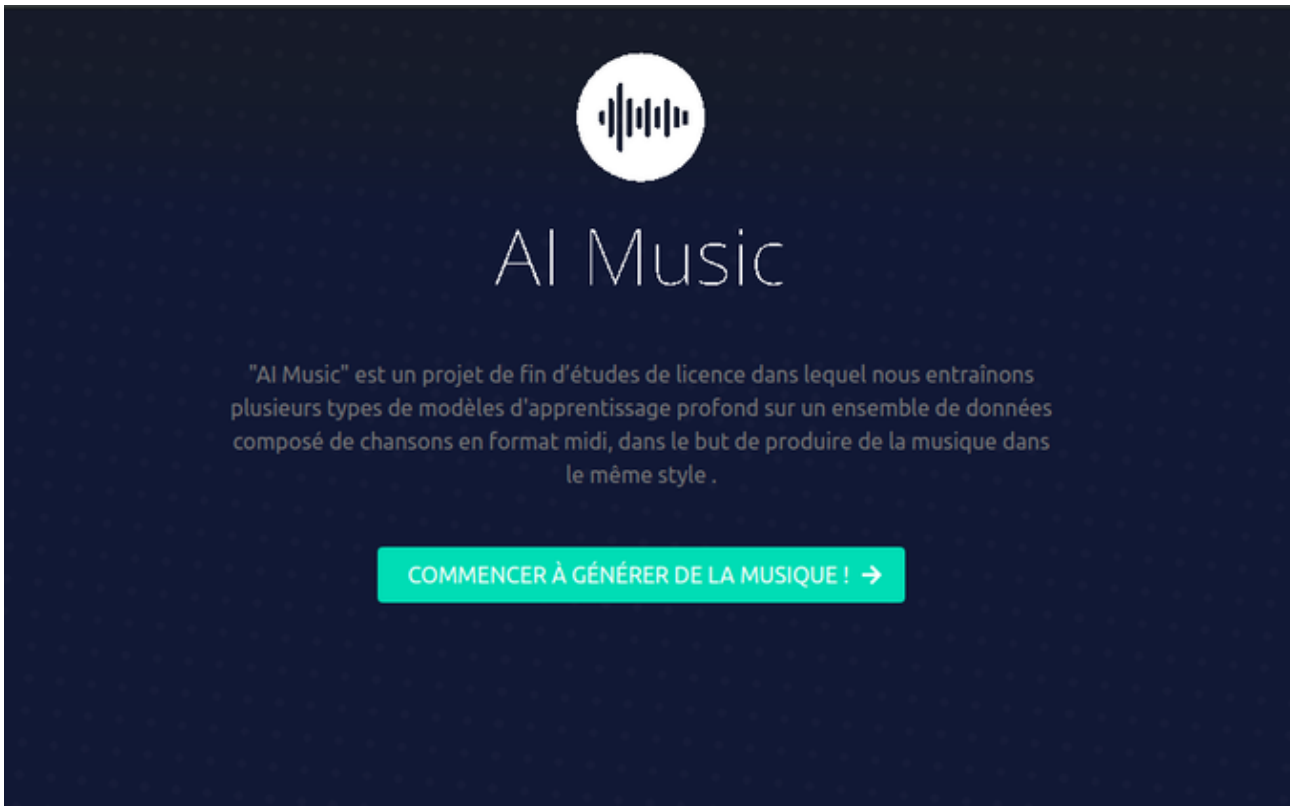


FIGURE 4.9 – Première page de notre interface.

La deuxième page de l'interface propose trois moyens de générer de la musique chacun correspondant à un de nos modèles. Par chaque clique sur un des boutons, un nouveau morceau de musique est généré. Il est également possible de télécharger le morceau si on le trouve intéressant.

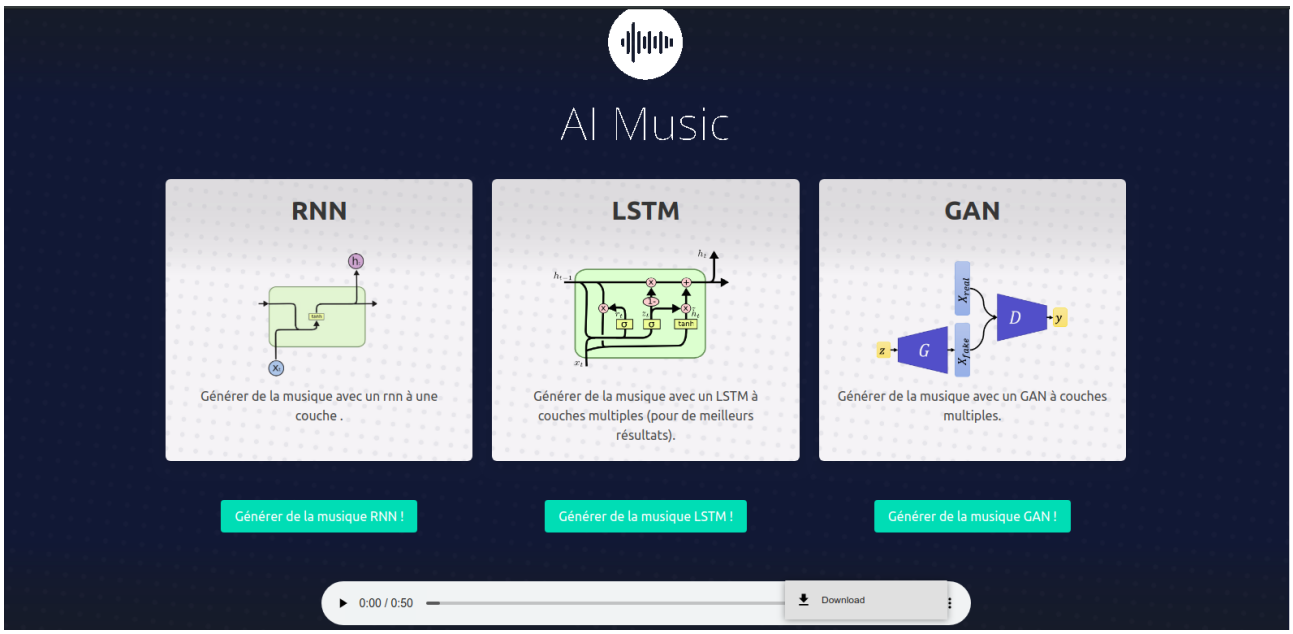


FIGURE 4.10 – Deuxième page de notre interface.

4.5 Conclusion

Dans ce chapitre, nous avons détaillé le processus d'entraînement de chacun de nos modèles, nous avons ensuite abordé le développement de notre interface et la façon dont elle travaille. Dans le prochain chapitre, nous examinerons les résultats de l'entraînement ainsi que l'évaluation de la musique générée.

Chapitre 5

Résultats

Dans ce chapitre, nous parlerons des résultats de l'entraînement, puis nous évaluerons la musique produite pour l'ensemble des modèles retenus après une analyse des résultats.

5.1 Expérience 1 et 2

Nous avons utilisé la méthode de l'entropie croisée catégorielle pour nos modèles RNN et LSTM afin de choisir la note ayant la plus forte probabilité de se produire la suivante. En termes de performance, nos modèles RNN (LSTM y compris) fonctionnent comme prévu, les pertes diminuant progressivement jusqu'à ce qu'elles convergent vers une valeur quasi-constante, comme le montre la figure 5.1.

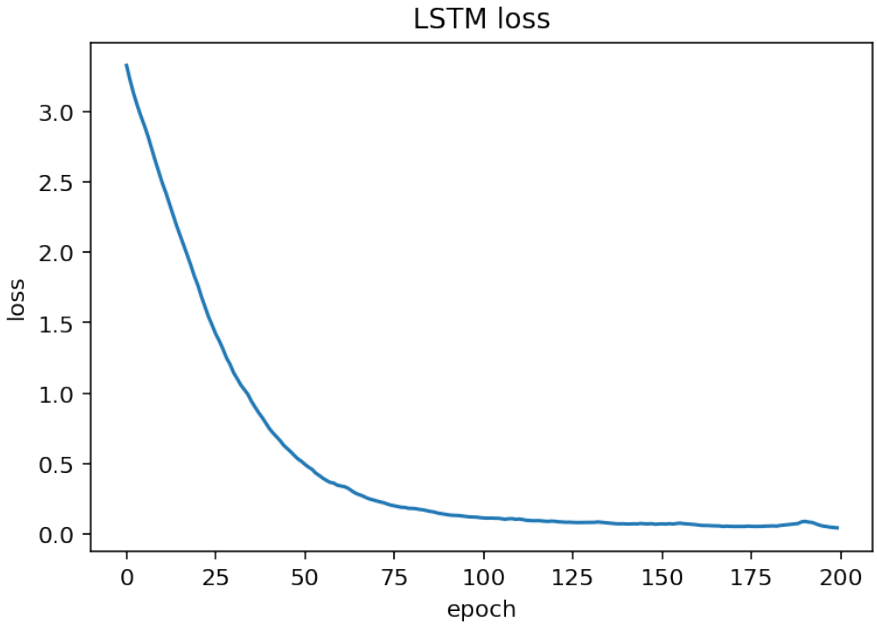


FIGURE 5.1 – La Perte de Modèle de LSTM par Époque.

Nous étions prudents sur l'entraînement des modèles au-delà du point de convergence pour éviter le risque de sur-apprentissage.

5.2 Expérience 3

Pour le GAN, le générateur a initialement une perte élevée et le discriminant a une perte faible comme prévu. Ces valeurs commencent à converger jusqu'à ce qu'elles le fassent après environ 550 époques.

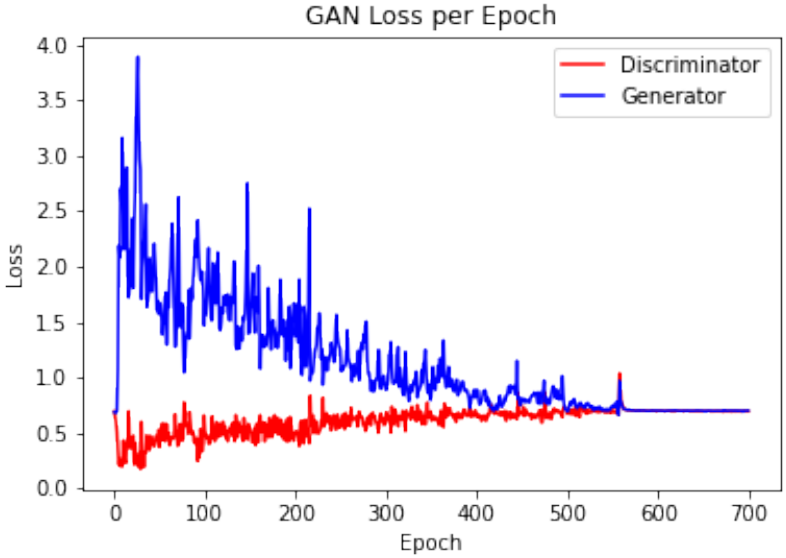


FIGURE 5.2 – La Perte de Modèle de GAN par Époque.

Il est intéressant de noter qu'après environ 2500 époques, le générateur et le discriminant commencent à diverger vers leur point de départ initial (fig 5.3). Une explication possible de cette incohérence peut être liée à la petite taille des données ou à la formation simultanée des deux modèles.

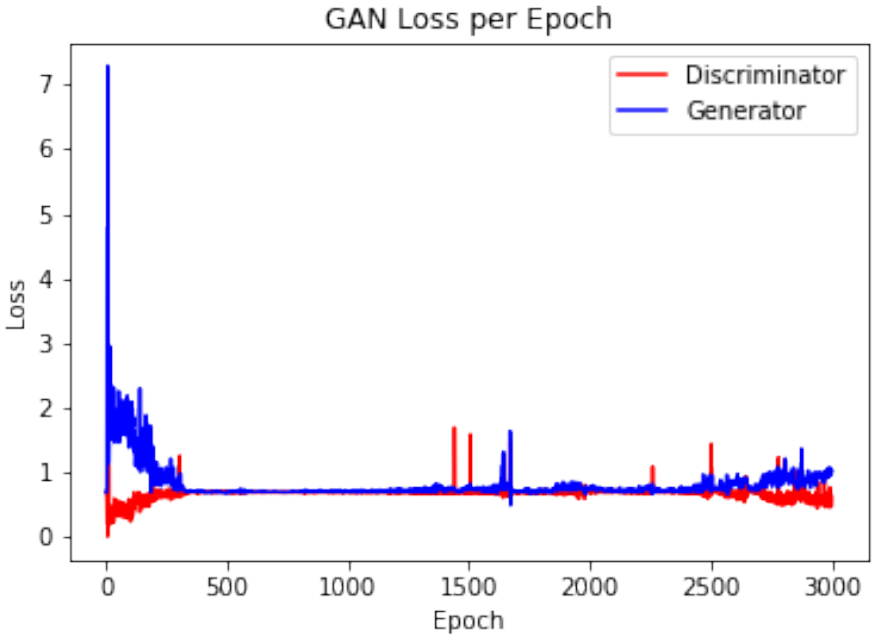


FIGURE 5.3 – La Perte de Modèle de GAN par Époque.

Pour générer notre musique, nous prenons les poids du modèle après 600 époques car cela

semble le point le plus convenable.

5.3 L'Évaluation de Musique

L'évaluation de nos modèles a été réalisée à partir d'un certain nombre de tests sur les résultats générés.

Nombre de sons uniques : c'est la taille de vocabulaire que nos morceaux utilisent. Ce facteur a été calculé manuellement dans notre code.

Plage de sons : c'est le nombre de demi-tons entre la note la plus basse et la note la plus haute dans un morceau. Ce facteur a été calculé manuellement dans notre code.

Polyphonie : c'est la possibilité de jouer plusieurs sons distincts simultanément. Ce facteur a été constaté à partir du "piano-roll".

Répétition des séquences : c'est le nombre de fois où K notes identiques sont apparues dans une suite. Ce facteur a été constaté à partir de la liste des notes produites et du "piano-roll".

Audience humaine : c'est le facteur le plus important, car le but de notre travail est de faire de la musique qui sonne agréable. Pour ce facteur, nous avons joué notre musique pour quelques personnes et des collègues de l'université.

Pour l'audience humaine, elle était exprimée par le pourcentage de gens qui ont aimé la pièce et qui ont trouvé qu'elle ressemblait à une chanson authentique. Quant à la polyphonie et la répétition de séquences, nous avons choisi trois valeurs pour exprimer les résultats obtenus : absence, convenable, exagéré.

5.3.1 L'Analyse des Résultats

Le tableau suivant récapitule les résultats obtenus. Les chiffres du tableau représentent les résultats moyens obtenus à partir de certains morceaux générés. Pour les résultats sur la musique réelle, on a pris un exemple de l'ensemble de données d'entraînement et on l'a analysé.

Critères d'évaluation	Musique réelle	Expérience 1	Expérience 2	Expérience 3
Nombre de sons uniques	40	11	27	35
Plage de sons	54	30	60	63
Polyphonie	convenable	absence/convenable	convenable	convenable
Répétition des séquences	convenable	convenable/exagéré	convenable	convenable/exagéré
Audience humaine	100%	40%	90%	60%

Expérience 1 : pour notre modèle RNN, la plupart des cas, notre modèle s'est coincé sur la répétition d'une séquence encore et encore (voir la figure 5.4), expliquant la baisse dans la moyenne des notes uniques et l'absence de polyphonie parfois. Mais en filtrant la musique générée, nous pouvons trouver de bons morceaux que les gens pourront apprécier. Ce modèle étant principalement employé comme ligne de base, nous ne croyions pas qu'il serait aussi

performant. Visitez https://soundcloud.com/kheireddine-fergani/sets/rnn-music pour écouter quelques morceaux générés par notre modèle RNN.

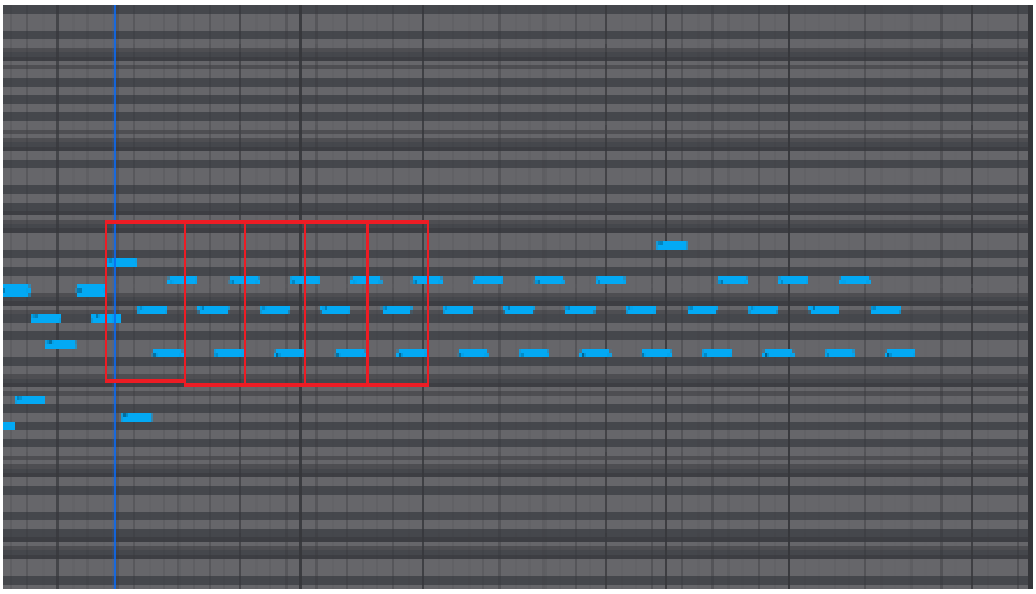


FIGURE 5.4 – Le "piano-roll" du 2ème Morceau dans notre Liste des Chansons Généré avec notre Modèle de RNN.

Expérience 2 : Notre modèle LSTM tend à montrer les meilleurs résultats avec une musique qui peut paraître humaine. Visitez https://soundcloud.com/kheireddine-fergani/sets/lstm-music pour écouter quelques morceaux générés par notre modèle LSTM. Le premier morceau de la liste des chansons généré semble être celui qui a eu le plus de succès, car il est le plus apprécié par les gens.

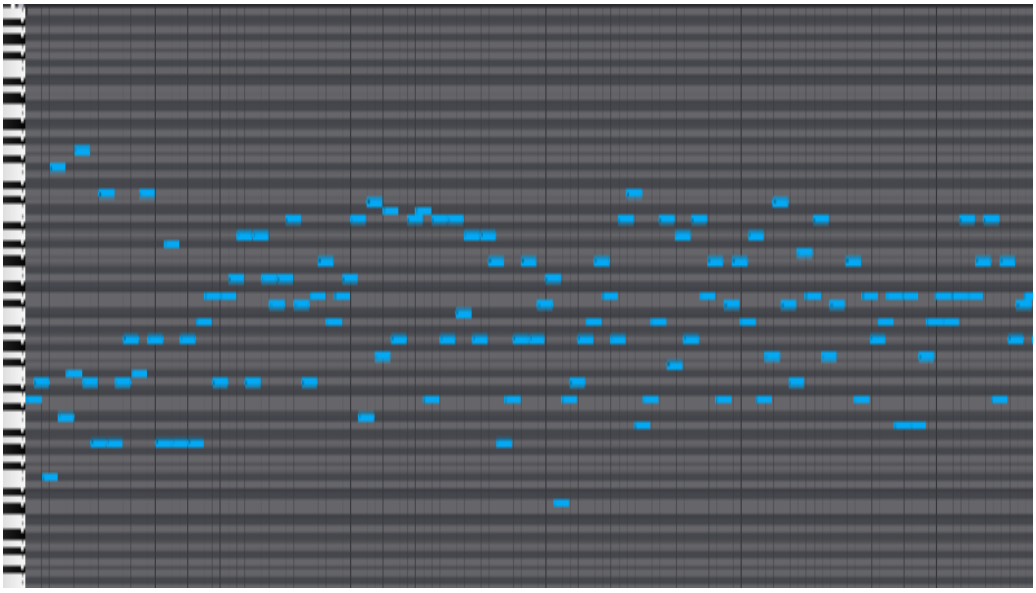


FIGURE 5.5 – Le "piano-roll" du Premier Morceau dans notre Liste des Chansons Généré avec notre Modèle de LSTM.

Expérience 3 : malgré le fait que notre modèle de GAN montre de bons résultats statiques, dans la majorité des cas, il prend une ou plusieurs notes et les répète tout au long de la chanson (voir la figure 5.6), ceci limitant à la fois le rythme et l'harmonie du morceau. Visitez https:

//soundcloud.com/kheireddine-fergani/sets/gan-music pour écouter quelques morceaux générés par notre modèle GAN.

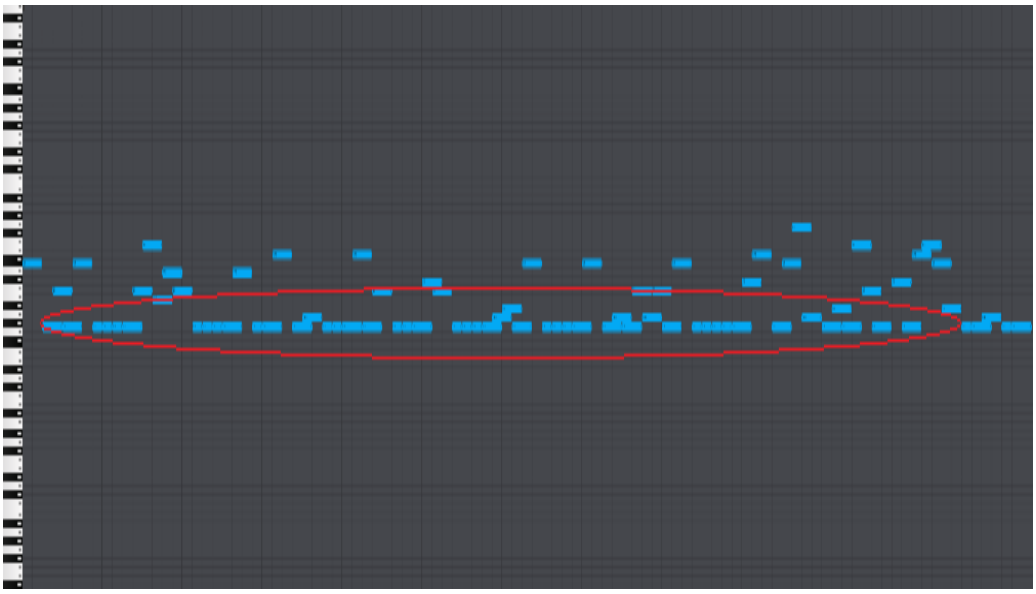


FIGURE 5.6 – Le "piano-roll" du 2ème Morceau dans notre Liste des Chansons Généré avec notre Modèle de GAN.

5.4 Conclusion

Globalement, les expériences 2 et 3 ont le potentiel de générer de la musique considérablement complexe. Cependant, l'instabilité du processus d'entraînement des GANs rend l'atteinte d'un état où il peut être compétitif face à un LSTM bien plus difficile. Peut-être qu'avec un modèle plus complexe, plus de temps, un ensemble de données plus important et des ressources de calcul plus grandes, un résultat plus impressionnant peut être atteint.

Conclusion Générale et Futurs Travaux

Ce projet a été une grande opportunité d'apprendre à la fois l'apprentissage profond et la théorie de la musique. Dans l'ensemble, nous sentons bien que nos expériences sur la génération musicale ont été une réussite. Parmi les trois modèles que nous avons testés, le modèle LSTM a donné les meilleurs résultats et a pu générer des motifs musicaux avec une mélodie et une harmonie cohérentes. Les GANs sont une nouvelle technologie qui n'a pas encore été correctement explorée car elle est très récente, et a beaucoup de potentiel pour se développer dans les prochaines années, notamment dans le domaine de la génération musicale. En plus, nous pensons que ce projet permet vraiment de démonter la réalité de la musique ou de la réintroduire comme une langue facilement accessible et qui, pour certains, vaut peut-être la peine d'être apprise, et inversement, pour les musiciens, afin de les amener à s'intéresser aux langages de programmation.
Malgré les résultats intéressants obtenus dans le cadre de ce travail, de nombreuses améliorations pourraient être apportées. Le domaine de l'apprentissage profond pour la musique manque encore d'un ensemble de données standard comme le MNIST pour la reconnaissance d'images. Pour notre modèle LSTM, nous pensons qu'il pourrait être amélioré en utilisant une architecture d'encodeur-décodeur que nous avons présenté en 2.2.2.9. Notre modèle actuel ne permet pas de faire varier la durée des notes et les écarts entre les notes. Pour y parvenir, nous pourrions ajouter d'autres classes pour chaque durée différente, ce qui nécessiterait un ordinateur beaucoup plus puissant.
En termes d'amélioration du GAN, nous voulons mettre en place une architecture CNN pour le modèle de générateur, car elle donne de meilleurs résultats dans le domaine du traitement des images. Nous pensons également qu'un ensemble de données beaucoup plus important peut mieux stabiliser le processus d'entraînement.
Les entrées peuvent également être améliorées en permettant d'inclure d'autres informations contenues dans les fichiers MIDI, telles que la vitesse et les décalages. Enfin, nous pouvons ajouter d'autres instruments à l'ensemble des données. Actuellement, le réseau ne prend en charge que les pièces qui n'ont qu'un seul instrument. Il serait intéressant de voir s'il pourrait être développé pour soutenir tout un orchestre.

Bibliographie

[1] C. Abromont and E. de Montalembert. *Guide de la théorie de la musique*. Musique. Fayard, 2001.

[2] Shervine Amidi Afshine Amidi. réseaux de neurones récurrents. https://stanford.edu/~shervine/l/fr/teaching/cs-230/pense-bete-reseaux-neurones-recurrents.

[3] Anastasia Azurra Baharudin. What is an art block and how to get out of one. https://medium.com/@aa13707/what-is-an-art-block-and-how-to-get-out-of-one-9e00ab3f238c, 2017.

[4] Stéphane Le Borgne. Sur la règle de dérivation en chaîne. https://perso.univ-rennes1.fr/stephane.leborgne/Derivation-en-chaine.pdf.

[5] Jean-Pierre Briot, Gaëtan Hadjeres, and François Pachet. Deep learning techniques for music generation - A survey. *CoRR*, abs/1709.01620, 2017.

[6] J.P. Clendinning and E.W. Marvin. *The Musician's Guide to Theory and Analysis : Third Edition*. W.W. Norton, 2016.

[7] David Cope. Experiments in musical intelligence (emi) : Non-linear linguistic-based composition. *Interface*, 18(1-2) :117–139, 1989.

[8] Shlomo Dubnov, Gérard Assayag, Olivier Lartillot, and Gill Bejerano. Using machine-learning methods for musical style modeling. *Computer*, 36 :73– 80, 11 2003.

[9] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12 :2121–2159, 07 2011.

[10] D. Eck and J. Schmidhuber. Finding temporal structure in music : blues improvisation with lstm recurrent networks. In *Proceedings of the 12th IEEE Workshop on Neural Networks for Signal Processing*, pages 747–756, 2002.

[11] Jesse Engel, Kumar Krishna Agrawal, Shuo Chen, Ishaan Gulrajani, Chris Donahue, and Adam Roberts. Gansynth : Adversarial neural audio synthesis, 2019.

[12] Jesse Engel, Cinjon Resnick, Adam Roberts, Sander Dieleman, Douglas Eck, Karen Simonyan, and Mohammad Norouzi. Neural audio synthesis of musical notes with wavenet autoencoders, 2017.

[13] Yoav Goldberg. The unreasonable effectiveness of character-level language models. https://nbviewer.jupyter.org/gist/yoavg/d76121dfde2618422139.

[14] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[15] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

[16] Aurélien Géron. *Hands-On Machine Learning*. O'Reilly Media, Inc., 2019.

[17] Li-Chia Yang Yi-Hsuan Yang Hao-Wen Dong, Wen-Yi Hsiao. MuseGAN : Demonstration of a Convolutional GAN Based Model for Generating Multi-track Piano-rolls. *Late-Breaking Demos of the 18th International Society for Music Information Retrieval Conference (ISMIR)*, 2, 2017.

[18] Sharon Kanach. The writings of iannis xenakis (starting with "formalized music"). *Perspectives of New Music*, 41(1) :154–166, 2003.

[19] Andrej Karpathy. The unreasonable effectiveness of recurrent neural networks. http://karpathy.github.io/2015/05/21/rnn-effectiveness/, 2015.

[20] Yoon Kim. Convolutional neural networks for sentence classification. *CoRR*, abs/1408.5882, 2014.

[21] Diederik P. Kingma and Jimmy Ba. Adam : A method for stochastic optimization, 2017.

[22] P. Y. Nikhil Kotecha. Generating music using an lstm network. <https://github.com/nikhil-kotecha/Generating_Music/>, 2018.

[23] Yann Lecun, Leon Bottou, Y. Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86 :2278 – 2324, 12 1998.

[24] Soroush Mehri, Kundan Kumar, Ishaan Gulrajani, Rithesh Kumar, Shubham Jain, Jose Sotelo, Aaron Courville, and Yoshua Bengio. Samplernn : An unconditional end-to-end neural audio generation model, 2017.

[25] S.P. Meyn and R.L. Tweedie. *Markov Chains and Stochastic Stability*. Communications and Control Engineering. Springer London, 2012.

[26] Chigozie Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. Activation functions : Comparison of trends in practice and research for deep learning. *CoRR*, abs/1811.03378, 2018.

[27] J. Rothstein. *MIDI :A Comprehensive Introduction, 2nd ed.* Computer Music and Digital Audio Series. A-R Editions, 1992.

[28] Peter M. Todd. Computer music journal, vol. 13, no. 4 (winter, 1989), pp. 27-43. 1989.

[29] Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew W. Senior, and Koray Kavukcuoglu. Wavenet : A generative model for raw audio. *CoRR*, abs/1609.03499, 2016.

[30] Bc. Ján Vojt. Deep neural networks and their implementation. https://is.cuni.cz/webapps/zzp/download/120229251, 2016.

[31] P. J. Werbos. Backpropagation through time : what it does and how to do it. *Proceedings of the IEEE*, 78(10) :1550–1560, 1990.

[32] Anna Yanchenko. Classical music composition using hidden markovmodels. <https://dukespace.lib.duke.edu/dspace/bitstream/handle/10161/15245/Yanchenko_duke_0066N_13967.pdf?sequence=1>, 2017.

[33] Dong Yu. Deep learning : Premise, philosophy, and its relation to other techniques. 2013.

Annexe A

Les Architectures des Modèles Fournies par Keras

Les architectures ci-dessous sont obtenues en utilisant la méthode *model.summary()*.

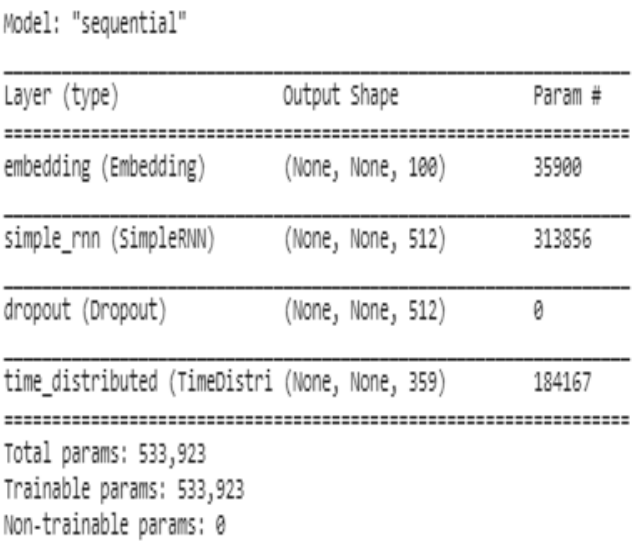


FIGURE 5.7 – L'architecture du Modèle RNN.

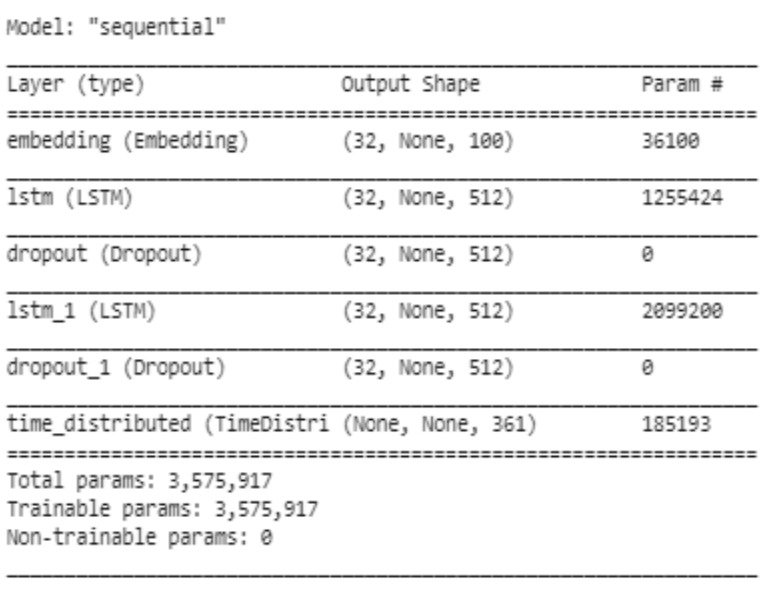


FIGURE 5.8 – L'architecture du Modèle LSTM.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
cu_dnnlstm (CuDNNLSTM)       (None, 100, 512)          1054720
_________________________________________________________________
bidirectional (Bidirectional (None, 1024)              4202496
_________________________________________________________________
dense (Dense)                (None, 512)               524800
_________________________________________________________________
leaky_re_lu (LeakyReLU)      (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               131328
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 256)               0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 257
=================================================================
Total params: 5,913,601
Trainable params: 5,913,601
Non-trainable params: 0
_________________________________________________________________
```

FIGURE 5.9 – Le Modèle Discriminant.

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_3 (Dense)              (None, 256)               256256
_________________________________________________________________
leaky_re_lu_2 (LeakyReLU)    (None, 256)               0
_________________________________________________________________
batch_normalization (BatchNo (None, 256)               1024
_________________________________________________________________
dense_4 (Dense)              (None, 512)               131584
_________________________________________________________________
leaky_re_lu_3 (LeakyReLU)    (None, 512)               0
_________________________________________________________________
batch_normalization_1 (Batch (None, 512)               2048
_________________________________________________________________
dense_5 (Dense)              (None, 1024)              525312
_________________________________________________________________
leaky_re_lu_4 (LeakyReLU)    (None, 1024)              0
_________________________________________________________________
batch_normalization_2 (Batch (None, 1024)              4096
_________________________________________________________________
dense_6 (Dense)              (None, 100)               102500
_________________________________________________________________
reshape (Reshape)            (None, 100, 1)            0
=================================================================
Total params: 1,022,820
Trainable params: 1,019,236
Non-trainable params: 3,584
_________________________________________________________________
```

FIGURE 5.10 – Le Modèle Générateur.

FIGURE 5.11 – L'architecture du Modèle GAN.