1- Explication des besoins

2- Solution apportée + Design

A) Solution proposée

Afin de répondre aux différentes problématiques posées précédemment, nous avons élaboré une solution complète, composée de multiples fonctionnalités. Notre plateforme serait composée de deux modes d'utilisation.

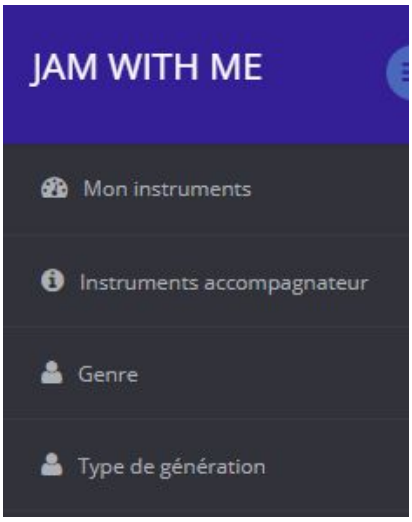
Tout d'abord, un musicien pourrait demander à notre réseau de neurone de générer une musique, composée avec les instruments, le style, et le tempo de son choix. Cette solution permettrait d'aider l'utilisateur à trouver l'inspiration. Une fois la musique générée il pourrait modifier en temps réel le rythme de cette dernière afin de l'adapter à sa guise.

Ensuite, notre plateforme permettrait de générer de la musique en temps réel afin d'accompagner le musicien dans ses entrainements. En effet, il lui suffirait de saisir sur la plateforme avec quel instrument il joue, quels instruments doivent l'accompagner, ainsi que le style de la musique, et Jam With Me générerait de la musique en conséquence qui accompagnerait le musicien en direct. Le réel atout de cette fonctionnalité est qu'elle permet à l'utilisateur d'être accompagné avec des instruments, des styles différents en fonction de ses envies.

B) Design de Jam With Me

Notre but premier est de créer un prototype simple d'utilisation afin que le musicien puisse se focaliser sur sa musique. Nous avons tout d'abord voulu réaliser un boitier électronique, tactile, que l'utilisateur pourrait brancher à son instrument, et qui l'accompagnerait en temps réel en s'adaptant à chaque note jouée par le musicien. Cette musique ressortirait par un haut-parleur incorporé au boîtier.

Cependant, ayant manqué de temps nous avons décidé de développer une plateforme web permettant à l'utilisateur de saisir ses choix d'instrument et de style. Ce site permet tout de même l'utilisation des différentes fonctionnalités citées ci-dessus.



Tout d'abord à l'aide des trois premiers onglets du menu, l'utilisateur pourra saisir : son instrument, les instruments qui l'accompagneront, ainsi que le style de musique qu'il souhaite. Une fois ses choix effectués, en appuyant sur le bouton de validation, l'algorithme se mettrait en route afin l'accompagner en direct. Malheureusement cette partie de notre algorithme ne fonctionnant pas encore, en cliquant sur ce bouton, rien ne se passe.

Ensuite à l'aide du dernier onglet du menu, l'utilisateur pourrait demander à notre plateforme de créer de toute pièce une musique afin de l'aider à être inspiré. Le musicien devrait simplement choisir les instruments devant être présent dans la musique générée, puis il pourrait régler le tempo en direct

afin d'adapter le son à ce qu'il recherche.

Ayant manqué de temps, nous n'avons pas pu faire le lien entre le site et l'algorithme. C'est pourquoi le site qui vous est présenté ici ne représente qu'une ébauche de ce à quoi ressemblerait le projet si nous avions eu plus de temps.

3- A type of Recurrent Neural Network : Long Short Term Memory (LSTM)

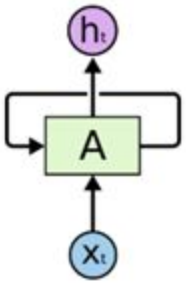
Sources :
https://distill.pub/2016/augmented-rnns/?fbclid=IwAR17i-tp6YTLh2sgnW3YobDmLOJNFP83Qj4Ba8mJiwbjU8IKmLTl-5GXrfQ

http://colah.github.io/posts/2015-08-Understanding-LSTMs/?fbclid=IwAR0YeXrXwTR4ZFozhk4Ov6MWgUpmhskXpsqKrnMNfj8qFbPlwRYSJvxZvYs

RNN : take as input their hidden state from the previous timestep along with whatever else they have, it allows them to remember information in a period of time by passing the information through the hidden states

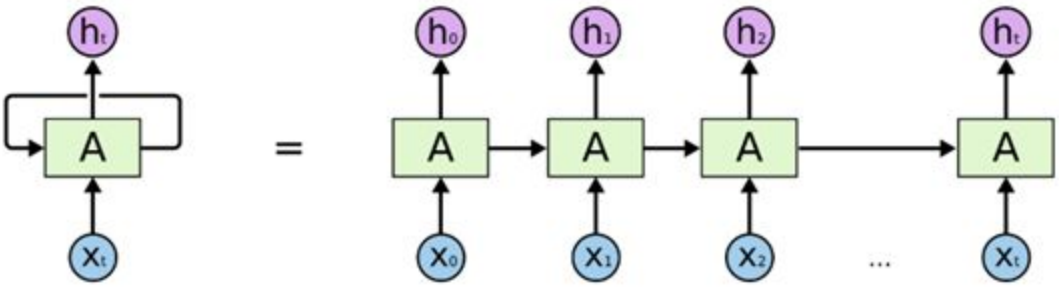
L'être humain ne commence pas son raisonnement sur une base vide, au fur et à mesure que vous lisez ce texte vous comprenez chaque mot par rapport aux mots précédents que vous avez lu, c'est ce qui donne un sens à la phrase. Les réseaux de neurones traditionnelles ne sont pas formés de cette manière, les réseaux de neurones récurrents (RNN) ont justement été abordés pour palier à cette problématique.

Un RNN est construit de manière cyclique :

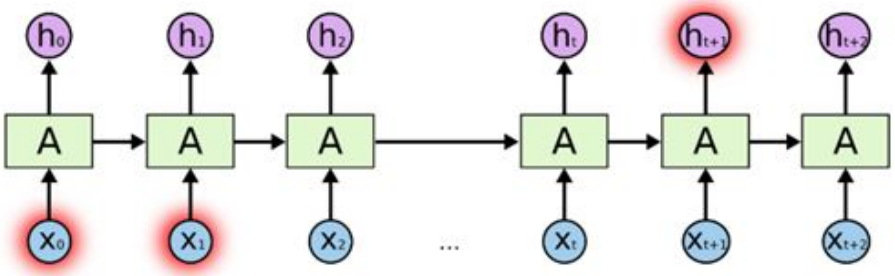


Une entrée « xt » passe par le réseau de neurone (composé d'un seul hidden layer) et prédit une sortie « ht », cette prédiction est gardé et remise en entrée au prochain module de la séquence. On commence à parler de « mémoire ».
Les RNN ont été créés afin de gérer des données en entrée séquentielles, où la sortie prédite à l'instant « t-1 » dépend du contexte et de la sortie à l'instant « t »

La séquence temporelle de module est définit de façon suivante :

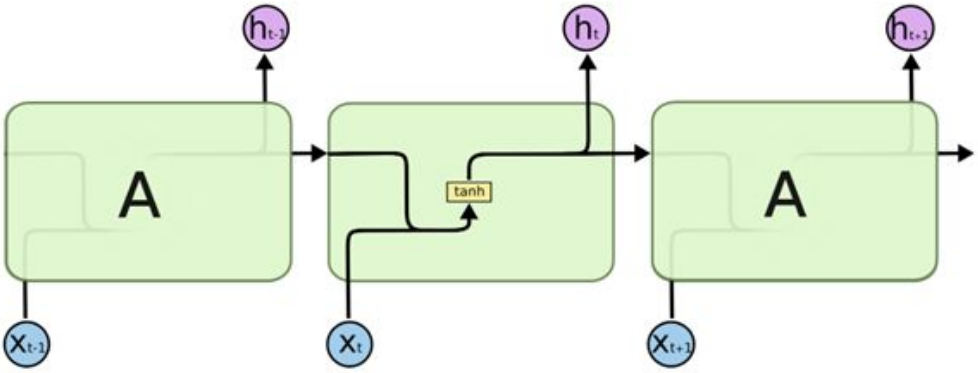


Prenant un exemple textuel :
« J'habite en France, je parle couramment le … » ceci est notre entrée, un RNN peut facilement prédire que le mot manquant est « français » puisque l'information pertinente dans ce contexte, dans cette phrase est le mot France. Néanmoins, si le gap entre l'information pertinente et ce qu'il faut prédire est trop grand, notre réseau de neurone rencontre des difficultés à garder des données en mémoire pour une durée trop longue, ce qui devient problématique en pratique.

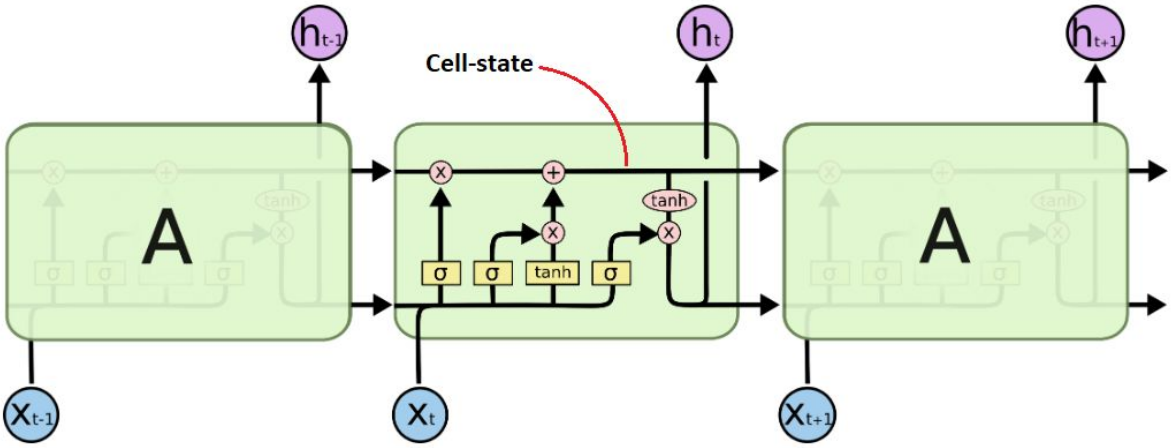


C'est pour cette raison que les LSTM (Long Short Term Memory) sont particulièrement intéressants. LSTM est donc un type de RNN qui est capable d'apprendre sur des dépendances à long termes. Ce qui est d'ailleurs beaucoup plus efficace en musique où l'on doit rigoureusement se rappeler de l'enchaînement des notes pour créer une mélodie agréable à l'écoute.

Cette différence est dûe au fait qu'un RNN est composé que d'un seul layer (représenté par le carré jaune) qui s'occupe uniquement de relier les nouvelles entrées aux précédentes sorties.



Un LSTM quand à lui est composé de 4 layers :



La première chose à comprendre dans ce réseau de neurone est ce qu'on appelle le « cell state », c'est la ligne qui fait le pont entre chaque module. Il agit comme un tapis roulant, où l'on peut ajouter ou enlever des informations. C'est ce qui permet au final, de faire passer la sortie d'un module à l'instant « t » comme entrée du module suivant, à l'instant « t+1 ».

Dans cette séquence, chaque module représentant un réseau de neurone est donc définit par deux types d'entrée : les sorties des modules précédents et les nouvelles informations ajoutées. Ces nouvelles informations - input « Xt » - passent par 4 layers avant de rejoindre le « cell-state »
Ces layers agissent comme des « portails » qui filtrent les informations. Ils sont organisés de manière suivante :

- Forget gate :

4- Explication du code

Le code de ce projet est en 2 parties, une partie afin de s'occuper du web scraping et une deuxième pour le traitement de données, l'entraînement du modèle et la génération de fichier.

-Web scraping et construction du dataset:

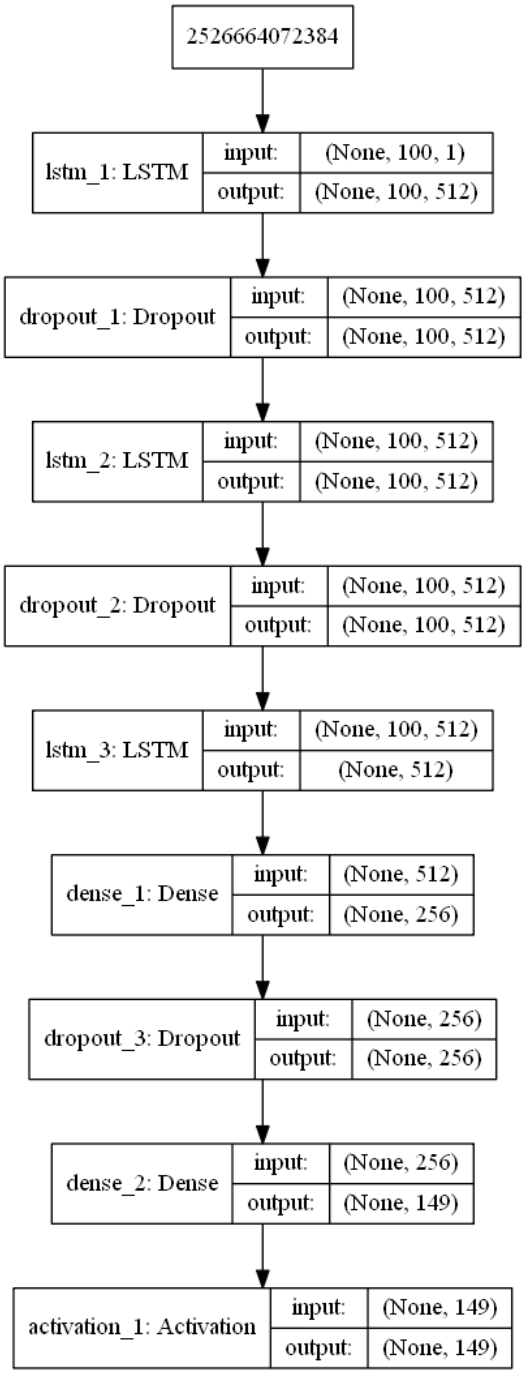
Un besoin fondamental dans les algorithmes de Machine Learning sont les données. En effet, les modèles de machine learning nécessitent une grande quantité de données afin de bien généraliser le concept qu'on tente de leur apprendre. Ici, on utilise la librairie python Sélénium qui va utiliser un web driver chrome afin de simuler un vrai utilisateur sur internet. On pourrait faire du scraping avec d'autres librairies mais sur le site que nous utilisons le téléchargement démarre avec l'évènement javascript click().

-géneration du dataset:

Une fois les fichiers MIDI récoltés, il faut en extraire les pistes qu'on veut garder. Dans le cadre de notre projet, on a testé de construire le dataset avec les pistes de piano et de guitare électrique. Une fois la piste sélectionnée, on y extrait les notes et les accords qui sont stockés dans un tableau notes[] à la suite.
Une fois le dataset finit, on génère un dictionnaire à partir de toutes les notes différentes qu'on a pu rencontrer dans notres dataset afin de pouvoir ensuite vectoriser nos inputs. Ceci est souvent fait dans le machine learning afin de donner à l'algorithme une représentation des notes qui sera plus compréhensible qu'une simple chaîne de caractères. On appelle ce type de vectorisation "one hot encoding".

-Entrainement du modèle:

Comme évoqué dans la partie précédente, le modèle de machine learning utilisé ici est un LSTM ce qui nous permet de lire et de générer des séquences de notes. L'architecture du modèle construite grâce à la librairie Keras ressemble à la figure à droite. Il y a deux couches de LSTM intercalés de couches de dropout qui est une façon de régulariser les poids du réseau de neurone et d'éviter le overfitting.



Ensuite on donne modèle un séquence de notes d'une taille fixe (50 notes) extraite de notre dataset et il doit prédire la prochaine note dans cette séquence. Après un certain nombre de prédiction, on fait la moyenne de ses erreurs qu'on propagera dans les paramètres de notre modèle afin de mettre à jour les paramètres.
Après environs une centaines d'époques (1 époque = traverser toutes les données du dataset) on arrive vers une précision du modèle supérieure à 90%.

-Génération de fichiers MIDI:

Une fois le modèle entraînée avec une précision suffisamment haute on peut passer à l'étape de génération de musique. On donne au modèle une séquence de notes aléatoire de notre dataset afin de lui donner quelque chose sur quoi démarrer. Ensuite on prendra la note prédite par le réseau et on la raccroche à la séquence de départ avant de la faire repasser dans le réseau qui générera une nouvelle note et ainsi de suite. On se retrouve avec une séquence de longueure spécifié à l'avance avec une suite de note qui est inspiré du dataset mais qui ne le copie pas exactement.