**CI4 : Réseaux récurrent**

Le but de ce TP est de mettre en pratique les concepts théoriques étudiés sur les RNN pour résoudre un problème concret : la génération de musique au format ABC. Vous apprendrez à manipuler des données textuelles, à concevoir un modèle RNN avec PyTorch, à entraîner ce modèle, et enfin, à l'utiliser pour générer de nouvelles séquences musicales.

**Objectifs**

- Prétraiter les données pour les rendre compatibles avec un modèle de RNN.
- Créer un dataset PyTorch et gérer les séquences avec un DataLoader.
- Implémenter un modèle RNN avec une architecture LSTM pour la prédiction de notes.
- Configurer et exécuter une boucle d'entraînement avec logging dans TensorBoard.
- Appliquer des techniques avancées comme l'early stopping et la sauvegarde du modèle.
- Générer de nouvelles séquences musicales à l'aide du modèle entraîné.
- (Bonus) Explorer des stratégies d'augmentation de données musicales.

**Chargement et exploration des données**

Dans cet exercice, vous allez explorer les données qui serviront à entraîner nos modèles de RNN.

Les données d'entraînement et de validation sont disponibles dans le dossier **/mnt/ssd/irishman**. Chaque fichier (**train.json** et **validation.json**) est au format JSON et contient une **liste d'objets**. Chaque objet possède au minimum la clé **abc notation** (une chaîne) qui contient la partition en notation ABC.

La notation ABC est un format texte compact pour représenter des partitions (surtout des mélodies monodiques, comme le répertoire traditionnel irlandais). Elle encode les *notes*, leurs *durées*, les *mesures* et des *métadonnées* (titres, métrique, tonalité) avec des caractères ASCII. On trouve en général des lignes d'en-tête telles que **X:** (identifiant), **T:** (titre), **M:** (métrique) et **K:** (tonalité), suivies du corps de la mélodie.

Pour en savoir plus, consultez l'article Wikipédia sur la notation ABC.

Chargez **train.json** et **validation.json** avec la bibliothèque Python **json** et affichez le nombre total de chansons dans chaque ensemble.

Affichez la première chanson du fichier d'entraînement. Que pouvez-vous en déduire sur sa structure ?

Pour mieux comprendre la notation ABC, visualisez une chanson. Copiez la première chanson du fichier d'entraînement et jouez-la sur ce site web.

**Prétraitement des données**

Dans cet exercice, vous allez préparer les données pour qu'elles puissent être utilisées avec un modèle RNN.

**Étape 1 : Extraction des caractères uniques**

Pour transformer les chansons en données utilisables par un réseau de neurones, vous devez identifier les caractères uniques utilisés dans le dataset.

Trouvez tous les caractères uniques présents dans le dataset d'entraînement.

Combien y a-t-il de caractères uniques ?

Pourquoi est-il nécessaire de travailler avec des indices plutôt que directement avec des caractères ?

**Étape 2 : Mapping caractères-index**

Pour convertir les caractères en vecteurs numériques, nous allons créer une liste et un dictionnaire permettant de faire les correspondances.

Écrivez un dictionnaire permettant de passer d'un caractère à un index.

Écrivez une liste permettant de passer d'un index à un caractère.

**Étape 3 : Vectorisation des chaînes**

Maintenant, nous allons écrire une fonction qui transforme une chaîne de caractères en une liste d'indices correspondants.

Implémentez une fonction **vectorize_string** prenant en entrée une chaîne de caractères et retournant une liste d'indices. Testez votre fonction avec la première chanson du dataset d'entraînement.

**Étape 4 : Padding des séquences**

Pour former des batches, toutes les séquences doivent avoir la même longueur. Nous allons donc ajouter artificiellement des espaces pour atteindre la longueur maximale : c'est du padding.

Trouvez la longueur maximale des séquences dans le dataset d'entraînement.

Écrivez une fonction qui prend en entrée une chaîne de caractères et une longueur maximale, et ajoute des espaces à la fin si nécessaire. Sinon, il faut couper la fin de la chaîne pour qu'elle ait la bonne longueur.

**Création du dataset PyTorch**

Dans cet exercice, vous allez écrire un dataset PyTorch pour gérer les séquences.

**Étape 1 : Préparation des données**

Nous voulons rassembler tout le prétraitement en une fonction qui retourne les mappings et les données vectorisées prêtes à l'emploi.

**Étape 2 : Dataset et DataLoader**

Dans un problème de génération de séquences, comme celui de la prédiction de notes musicales, le modèle apprend à prédire le prochain élément de la séquence en se basant sur les éléments précédents. Pour entraîner le modèle, nous utilisons la séquence d'entrée pour lui fournir le contexte, et la séquence de sortie correspond à la même séquence décalée d'un pas vers la gauche. Cela signifie que, pour chaque étape de l'entraînement, le modèle apprend à prédire le prochain caractère en se basant sur tous les caractères précédents. Par exemple, si la séquence d'entrée est "ABCD", la séquence de sortie sera "BCDE". Ce décalage est essentiel, car il permet au modèle de comprendre la relation temporelle entre les éléments successifs, ce qui est la base même des architectures RNN comme les LSTM et les GRU.

Nous devons créer une classe **MusicDataset** héritant de **torch.utils.data.Dataset**. Ce dataset doit retourner la séquence d'entrée (tout sauf le dernier caractère) et retourner la séquence cible (tout sauf le premier caractère, décalée d'un indice).

Implémentez la classe **MusicDataset**. Le constructeur prendra simplement les données que nous avons créées plus haut. Il vous faudra ensuite écrire la méthode **__len__(self)** et **__getitem__(self, idx)** pour créer le dataset. C'est dans **__getitem__** que nous récupérons l'élément à l'indice donné, puis extrayons la séquence d'entrée (tout jusqu'à l'avant-dernier élément) et la séquence de sortie (tout à partir du deuxième élément). 2. Initialisez un DataLoader pour le training set et un autre pour le validation set avec une taille de batch de 8. 3. Vérifiez que votre dataset fonctionne correctement en affichant un batch.

**Implémentation du modèle**

Dans cet exercice, vous allez implémenter un modèle LSTM pour prédire la prochaine note.

**Étape 1 : Architecture du modèle**

Nous allons utiliser une couche d'Embedding pour convertir les indices en vecteurs. Ces vecteurs seront ensuite passés dans un LSTM, suivi d'une couche dense pour produire les prédictions.

Les embeddings sont une méthode pour représenter des données discrètes, comme des mots ou des caractères, sous forme de vecteurs continus dans un espace de dimension fixe. Au lieu de représenter chaque caractère par un simple index (par exemple, "A" = 0, "B" = 1), les embeddings permettent de

projeter ces caractères dans un espace vectoriel où les relations entre les éléments peuvent être capturées. Par exemple, des caractères ou mots ayant un rôle similaire peuvent se retrouver proches dans cet espace. Avant d'entrer dans un LSTM, ces représentations vectorielles sont essentielles car elles fournissent une information riche et dense qui facilite l'apprentissage des relations complexes entre les éléments d'une séquence. Le LSTM, qui travaille avec des données continues, peut alors mieux modéliser les dépendances temporelles et sémantiques grâce à ces embeddings, par opposition à des indices discrets qui ne contiennent aucune information relationnelle.

Implémentez une classe **MusicRNN** héritant de **torch.nn.Module**. Votre architecture devra comprendre :

- Une couche d'Embedding
- Un LSTM
- Une couche dense pour les prédictions.

La classe prendra en argument la taille du vocabulaire, la dimension des embeddings, la taille de l'état du LSTM.

**Étape 2 : Boucle d'entraînement**

Nous voulons maintenant écrire une boucle d'entraînement pour entraîner le modèle. Assurez-vous de logger les informations pertinentes dans Tensorboard.

Écrivez une fonction **train_model** qui :

- Entraîne le modèle pour un nombre d'itérations donné
- Logge la loss et l'accuracy à chaque époque sur le train et le val
- Implémente un early stopping et enregistre le meilleur modèle.

Entraînez le modèle avec les hyperparamètres suivants :

- **num_training_iterations = 3000**
- **batch_size = 256**
- **learning_rate = 5e-3**
- **embedding_dim = 256**
- **hidden_size = 1024**

**Génération de musique**

Nous voulons utiliser le modèle entraîné pour générer une nouvelle chanson.

La génération avec un LSTM repose sur l'idée de prédire le prochain élément d'une séquence à partir de ses éléments précédents. Une fois le modèle entraîné, le processus de génération commence avec une séquence de départ qui est donnée en entrée au modèle. À chaque étape, le modèle prédit le prochain caractère en fonction des entrées précédentes. Cette prédiction est ensuite utilisée comme nouvelle entrée pour la prochaine étape, permettant ainsi au modèle de générer une séquence itérativement. Cette méthode, où la sortie d'une étape est réutilisée comme entrée pour l'étape suivante, permet de produire des séquences arbitrairement longues.

Dans l'approche greedy, à chaque étape de génération, le caractère ayant la probabilité la plus élevée dans la distribution de sortie est sélectionné directement. Bien que cette méthode soit simple et rapide, elle peut parfois conduire à des séquences peu variées ou incohérentes, car elle ne tient pas compte de l'incertitude dans la prédiction et ne laisse pas place à l'exploration d'autres chemins possibles dans l'espace des séquences.

Écrivez une fonction **generate_music** qui :

- Prend en entrée le modèle, une séquence de départ, et une longueur souhaitée
- Génère une séquence en échantillonnant les probabilités à chaque étape

Testez votre fonction en générant une chanson de 200 caractères.

(Bonus) Implémentez une stratégie d'augmentation des données en transposant les partitions ou en modifiant les valeurs rythmiques.