Université Chouaib Doukkali
Ecole Nationale des Sciences Appliquées d'El Jadida
Département Télécommunications, Réseaux et Informatique
Module : **Séries Temporelles et Applications**

TP

Filière : **SDIA**
Niveau : **2ème Année**

Sujet :

TP 1 — Génération de musique en ABC Notation avec un modèle RNN (LSTM)

Réalisé Par :
Wassima RAICHI

Examiné par :
Prof. Youness ABOUQORA

Année Universitaire : 2025/2026

Table des matières

Liste des Figures

Introduction

L'objectif de ce TP est de mettre en œuvre un modèle de **génération de séquences** basé sur les réseaux de neurones récurrents (RNN) afin de produire automatiquement une **partition musicale** au format **ABC notation**.

Le principe consiste à apprendre la distribution des caractères d'une grande collection de chansons (textes ABC), puis à générer de nouveaux morceaux caractère par caractère, à partir d'un *prompt* initial.

1.1 Données utilisées

```
Nombre de chansons dans le train : 214122
Nombre de chansons dans la validation : 2162

--- Première chanson (extrait) ---

X:1
L:1/8
M:4/4
K:Emin
|: E2 EF E2 EF | DEFG AFDF | E2 EF E2 B2 |1 efe^d e2 e2 :|2 efe^d e3 B |: e2 ef g2 fe |
 defg afdf |1 e2 ef g2 fe | efe^d e3 B :|2 g2 bg f2 af | efe^d e2 e2 ||
```

Le dataset utilisé provient de Hugging Face : **sander-wood/irishman** (musiques traditionnelles irlandaises en ABC notation).

- **Train : 214122 chansons**
- **Validation : 2162 chansons**

Chaque exemple contient une chaîne de caractères représentant une partition au format ABC (ex: X:, T:, M:, K: puis les notes).

Exemple (extrait d'une chanson du train) :

- En-tête ABC : index, longueur de mesure, tonalité…
- Corps : notes codées sous forme textuelle (ex: E2 EF E2 EF | DEFG AFDF | ...)

1.2 Prétraitement des données

1.2.1 Construction du vocabulaire

On concatène tout le texte du train afin d'extraire la liste des caractères uniques :

- **Nombre de caractères uniques : 95**
- On ajoute explicitement un token de padding : **PAD_TOKEN = " "** (espace)

```
Nombre de caractères uniques : 95
Exemples: ['\n', ' ', '!', '"', '#', '$', '&', "'", '(', ')', '*', '+', ',', '-', '.', '/', '0', '1', '2', '3',
'4', '5', '6', '7', '8', '9', ':', ';', '<', '=', '>', '?', '@', 'A', 'B', 'C', 'D', 'E', 'F', 'G']
```

Figure 1 : Vocabulaire des caractères du dataset musical (ABC notation)

On construit ensuite :

- char2idx : dictionnaire caractère → index
- idx2char : dictionnaire index → caractère

Exemples affichés dans l'exécution :

- index de 'E' trouvé dans le vocabulaire
- vérification d'un caractère à un index donné

```
Index de 'E' (si présent) : 37
Caractère à l'index 10 : *
```

Figure 2 : Vérification du mapping caractère–index.

1.2.2 Analyse des longueurs

On calcule la longueur des partitions (en nombre de caractères) :

- **Max length train : 2968**
- **Mean length train : ~290**

Cela montre que les morceaux sont très variables, donc on adopte une stratégie de **troncature/padding**.

```
Max length train: 2968
Mean length train: 290
```

Figure 3 : Statistiques de longueur des partitions musicales

1.2.3 Padding + Vectorisation

On fixe une longueur maximale MAX_LEN pour uniformiser les séquences.

Fonctions utilisées :

- pad_or_truncate(text, max_len) :
    - si longueur < max_len → padding avec espaces
    - si longueur > max_len → troncature
- vectorize(text) : conversion en liste d'indices via char2idx

Dans ton notebook, tu utilises :

- d'abord MAX_LEN = 256 pour vectoriser
- puis plus bas tu changes MAX_LEN = 128 pour l'entraînement

⇨ **Remarque :**

Il y a une incohérence : train_vec est construit avec MAX_LEN=256 puis l'entraînement utilise MAX_LEN=128. Comme mon Dataset recalcule x=seq[:-1], ça reste cohérent techniquement, mais c'est mieux de garder une seule valeur de MAX_LEN pour tout le pipeline (ex: 256 partout).

1.3 Création du Dataset PyTorch

On crée une classe MusicDataset qui retourne :

- **x** : séquence d'entrée = tous les caractères sauf le dernier
- **y** : séquence cible = tous les caractères sauf le premier

Donc le modèle apprend à prédire le prochain caractère (language modeling).

Vérification avec un batch de test :

- x_b shape = [8, 255]
- y_b shape = [8, 255]

Ce qui est logique car :

- séquence initiale : 256 caractères
- après décalage : 255

```
x_b shape: torch.Size([8, 255])
y_b shape: torch.Size([8, 255])
Extrait x: [56, 26, 17, 20, 20, 22, 16, 21, 0, 44, 26, 17, 15, 17, 22, 0, 45, 26, 19, 15, 20, 0, 43, 26, 39, 0, 9
2, 26, 1, 39]
Extrait y: [26, 17, 20, 20, 22, 16, 21, 0, 44, 26, 17, 15, 17, 22, 0, 45, 26, 19, 15, 20, 0, 43, 26, 39, 0, 92, 2
6, 1, 39, 18]
```

Figure 4 : Construction des paires entrée–cible pour l'apprentissage séquentiel.

1.4 Modèle utilisé : LSTM (MusicRNN)

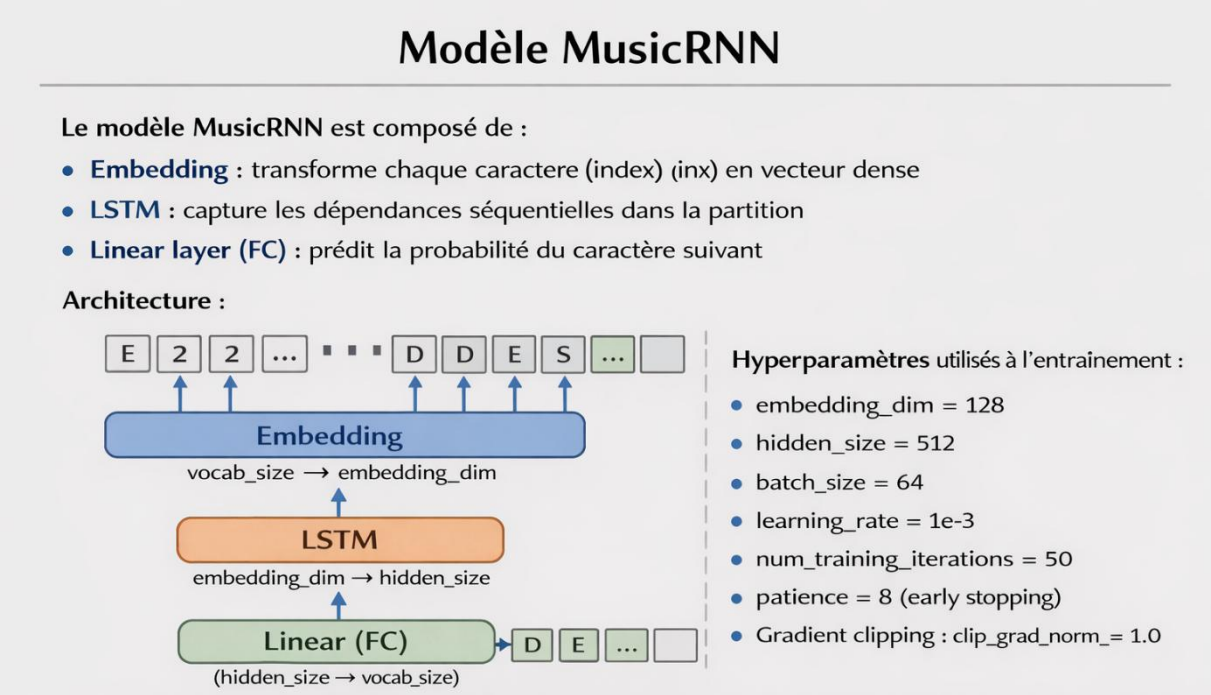

Figure 5 : Architecture du modèle MusicRNN pour la génération de partitions musicales

1.5 Entraînement et validation

1.5.1 Fonction d'entraînement

La fonction train_model_csv :

- entraîne sur train
- évalue sur validation à chaque époque
- enregistre les métriques dans un fichier CSV (training_log.csv)
- sauvegarde le meilleur modèle (best_music_rnn.pt) selon la val_loss
- applique early stopping

Fonction de coût :

- CrossEntropyLoss (classification multi-classes sur vocabulaire)

Optimiseur :

- Adam

1.5.2 Résultats observés

D'après :

- **Train batches : 3345**
- **Val batches : 34**

Après ~25 époques :

- Train Loss ≈ 0.6427
- Train Acc ≈ 0.7879
- Val Loss ≈ 0.6743
- Val Acc ≈ 0.7826

```
Epoch 1/50 | Train Loss 0.9642 Acc 0.6915 | Val Loss 0.8283 Acc 0.7308
Epoch 2/50 | Train Loss 0.7929 Acc 0.7379 | Val Loss 0.7741 Acc 0.7477
Epoch 3/50 | Train Loss 0.7505 Acc 0.7517 | Val Loss 0.7490 Acc 0.7560
Epoch 4/50 | Train Loss 0.7280 Acc 0.7592 | Val Loss 0.7302 Acc 0.7624
Epoch 5/50 | Train Loss 0.7135 Acc 0.7640 | Val Loss 0.7165 Acc 0.7663
Epoch 6/50 | Train Loss 0.7026 Acc 0.7677 | Val Loss 0.7199 Acc 0.7664
Epoch 7/50 | Train Loss 0.6950 Acc 0.7703 | Val Loss 0.7042 Acc 0.7710
Epoch 8/50 | Train Loss 0.6879 Acc 0.7727 | Val Loss 0.7098 Acc 0.7695
Epoch 9/50 | Train Loss 0.6824 Acc 0.7746 | Val Loss 0.6969 Acc 0.7740
Epoch 10/50 | Train Loss 0.6780 Acc 0.7760 | Val Loss 0.6942 Acc 0.7744
Epoch 11/50 | Train Loss 0.6726 Acc 0.7778 | Val Loss 0.6890 Acc 0.7762
Epoch 12/50 | Train Loss 0.6692 Acc 0.7790 | Val Loss 0.6880 Acc 0.7771
Epoch 13/50 | Train Loss 0.6657 Acc 0.7801 | Val Loss 0.6867 Acc 0.7772
Epoch 14/50 | Train Loss 0.6626 Acc 0.7812 | Val Loss 0.6843 Acc 0.7782
Epoch 15/50 | Train Loss 0.6594 Acc 0.7823 | Val Loss 0.6817 Acc 0.7790
Epoch 16/50 | Train Loss 0.6570 Acc 0.7831 | Val Loss 0.6825 Acc 0.7794
Epoch 17/50 | Train Loss 0.6552 Acc 0.7837 | Val Loss 0.6792 Acc 0.7801
Epoch 18/50 | Train Loss 0.6543 Acc 0.7840 | Val Loss 0.6793 Acc 0.7802
Epoch 19/50 | Train Loss 0.6514 Acc 0.7850 | Val Loss 0.6847 Acc 0.7781
Epoch 20/50 | Train Loss 0.6495 Acc 0.7856 | Val Loss 0.6770 Acc 0.7808
Epoch 21/50 | Train Loss 0.6481 Acc 0.7861 | Val Loss 0.6775 Acc 0.7813
Epoch 22/50 | Train Loss 0.6469 Acc 0.7865 | Val Loss 0.6762 Acc 0.7813
Epoch 23/50 | Train Loss 0.6449 Acc 0.7872 | Val Loss 0.6740 Acc 0.7815
Epoch 24/50 | Train Loss 0.6454 Acc 0.7870 | Val Loss 0.6738 Acc 0.7823
Epoch 25/50 | Train Loss 0.6427 Acc 0.7879 | Val Loss 0.6743 Acc 0.7826
```

Figure 6 : Évolution des performances du modèle MusicRNN au cours de l'entraînement

Interprétation :

- Les courbes montrent une diminution régulière de la loss train/val.
- L'accuracy progresse et se stabilise autour de **78%**.
- L'écart train vs val reste faible → **peu de surapprentissage** (bonne généralisation).

1.6 Visualisation des performances

Deux graphiques sont générés :

1. **Loss train/validation**

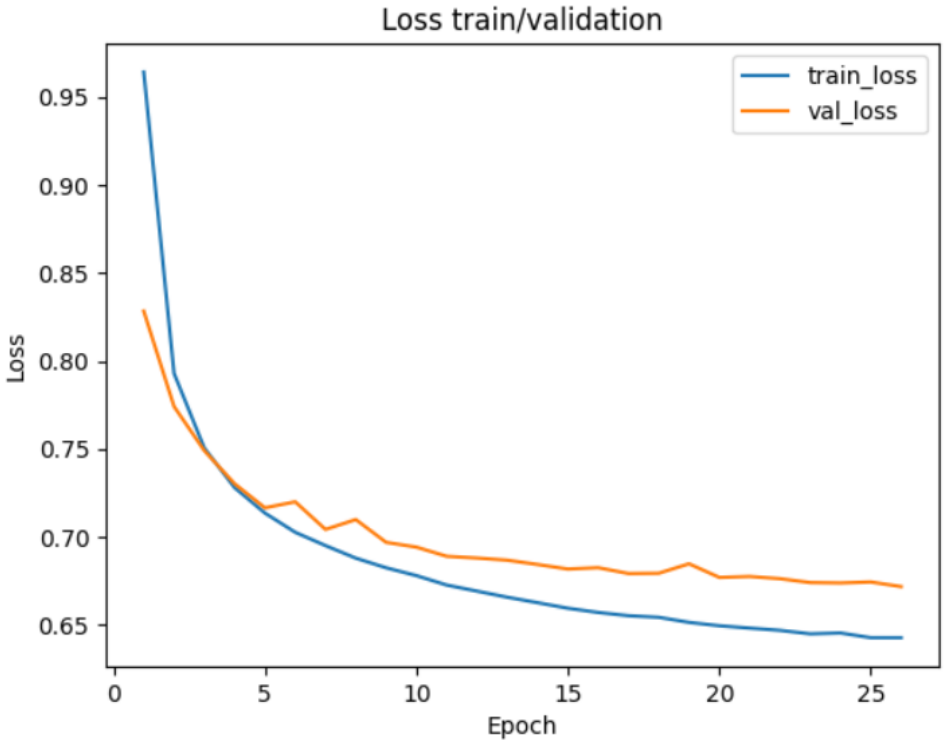

Figure 7 : Évolution de la fonction de perte (loss) sur les ensembles d'entraînement et de validation

La loss diminue rapidement au début puis continue à diminuer doucement. La loss validation suit la même tendance, ce qui indique un apprentissage stable.

2. **Accuracy train/validation**

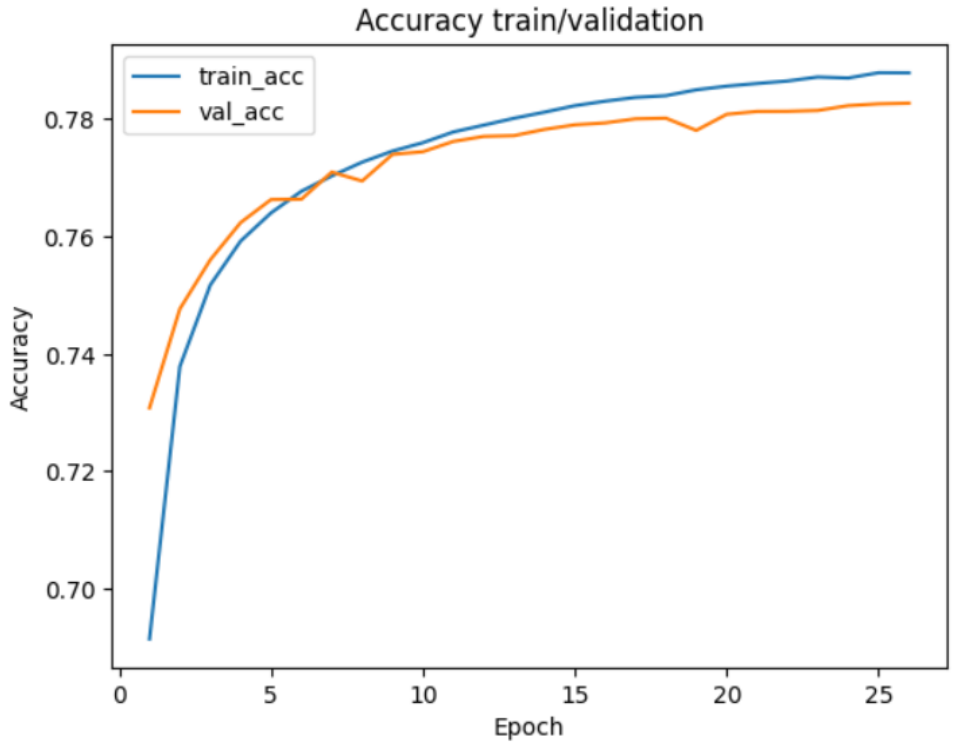

Figure 8 : Évolution de l'accuracy sur les ensembles d'entraînement et de validation

L'accuracy augmente progressivement et se stabilise, avec des valeurs proches entre train et val.

1.7 Génération de musique

1.7.1 Chargement du meilleur modèle

On charge le checkpoint best_music_rnn.pt qui contient :

- model_state
- char2idx, idx2char
- embedding_dim, hidden_size, vocab_size

1.7.2 Stratégie d'échantillonnage

La fonction generate_music génère caractère par caractère avec :

- **Temperature = 0.8**

  → augmente la diversité (plus créatif, mais plus risqué)

- **Top-k sampling = 20**

  → on ne choisit le prochain caractère que parmi les 20 plus probables

1.7.3 Exemple généré

Le modèle génère une nouvelle partition avec des symboles ABC cohérents (barres |, notes, chiffres de durée, accords, etc.).

Cependant, on peut parfois observer :

- des incohérences musicales (répétitions étranges, transitions brusques)
- des tokens moins réalistes selon la température

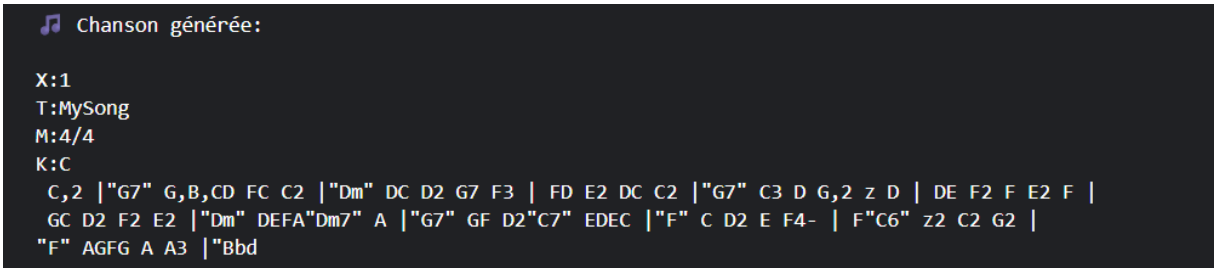

Figure 9 : exemple de partition musicale générée par le modèle MusicRNN

1.8 Discussion et améliorations possibles

Pour améliorer la qualité des musiques générées, on peut :

1. **Augmenter MAX_LEN** (ex : 512 au lieu de 128/256)

   capture des structures plus longues (phrases musicales)

2. **Utiliser un modèle plus profond**
   - LSTM avec num_layers=2 ou 3 + dropout
3. **Meilleure tokenisation**

   Au lieu de caractères, utiliser des tokens “musicaux” (notes, durées, barres, etc.)

4. **Réglage de la génération**
   - Baisser temperature (ex : 0.8–1.0) → plus cohérent
   - Ajuster top_k ou utiliser top_p (nucleus sampling)
5. **Enrichir le prompt**

   Ajouter quelques mesures réelles comme contexte initial.

Conclusion

Dans ce TP, nous avons entraîné un modèle **LSTM** pour apprendre la structure des partitions au format **ABC notation** à partir d'un grand dataset de musique traditionnelle. Le modèle atteint environ **78% d'accuracy** sur validation et permet de générer de nouvelles partitions à partir d'un prompt.

Les résultats montrent un apprentissage stable (loss décroissante, faible écart train/val). Des améliorations sont possibles pour obtenir des compositions plus longues et plus cohérentes musicalement.