Rapport TP4

1. Prétraitement des données

Objectif :

Préparer les données brutes du jeu CoNLL-2003 afin qu'elles soient prêtes à être utilisées pour l'entraînement d'un modèle de reconnaissance d'entités nommées (NER).

Étapes réalisées :

1. **Construction du vocabulaire :**

   À partir des phrases du jeu d'entraînement, nous avons construit un vocabulaire contenant tous les mots uniques. Deux tokens spéciaux ont été ajoutés :

   - `<PAD>` pour le remplissage des séquences courtes pour etre tous en meme taille
   - `<UNK>` pour les mots inconnus

2. **Indexation des mots :**

   Chaque mot du vocabulaire a été associé à un indice numérique (`word2idx`). Cela permet de convertir les séquences textuelles en vecteurs d'entiers.

   ▼ **Exemple**

   Suppose word_counts = {'EU': 5, 'German': 3, ...}

   Then vocab = ['<PAD>', '<UNK>', 'EU', 'German', ...]

   word2idx = {'<PAD>': 0, '<UNK>': 1, 'EU': 2, 'German': 3, ...}

3. **Extraction des étiquettes NER :**

   Les étiquettes nommées (tags) ont été extraites, correspondant aux entités à reconnaître (personne, organisation, lieu, etc.). Le nombre total de classes (`num_tags`) a été identifié.

   Dans notre data se sont: `['O', 'B-PER', 'I-PER', 'B-ORG', 'I-ORG', 'B-LOC', 'I-LOC', 'B-MISC', 'I-MISC']`

4. **Limitation de la séquence :**

   La longueur maximale des séquences a été fixée à 128 tokens afin de limiter la consommation mémoire.

5. **Conversion des Tokens et Étiquettes en Tensors**

- Pour chaque phrase :
  - **Tokens** :
    - Remplacement des mots par leurs indices dans le vocabulaire (`<UNK>` si le mot est inconnu).
    - Remplissage avec `<PAD>` (zéros) pour atteindre `max_len`.
  - **Étiquettes (Tags)** :
    - Troncature à `max_len`.
    - Remplissage avec `100` (valeur ignorée par la fonction de perte de PyTorch).

  ▼ Exemple pour une phrase

  Phrase originale : ['EU', 'rejects', 'German']

  Tokens convertis : [2, 4, 3]

  Tokens après padding : [2, 4, 3, 0, 0, ..., 0] (longueur 128)

  Tags après padding : [3, 0, 7, -100, ..., -100]

1. **Prétraitement de tous les ensembles :**

   Les ensembles `train`, `validation` et `test` ont été convertis en tenseurs PyTorch (`LongTensor`), prêts pour l'entraînement.

Observation :

Cette étape permet de transformer efficacement le texte brut en représentations numériques adaptées à l'entrée dans un réseau de neurones. L'utilisation d'un padding et d'une longueur fixe garantit l'uniformité des séquences.

2. Vectorisation des données

Objectif :

Transformer les données textuelles prétraitées en un format exploitable efficacement par le modèle de deep learning, en les organisant sous forme de *batches* à l'aide de DataLoaders.

Étapes réalisées :

1. **Création des jeux de données PyTorch :**

Les tenseurs contenant les séquences de tokens et les tags (issues de l'étape précédente) ont été combinés en objets `TensorDataset` pour :

- l'entraînement (`train_dataset`)
- la validation (`val_dataset`)
- le test (`test_dataset`)

2. **Mise en batch avec DataLoader :**

   Les jeux de données ont ensuite été chargés dans des `DataLoader` PyTorch :

   - `batch_size` fixé à 32 pour un bon compromis entre performance et capacité mémoire.
   - Les données d'entraînement sont mélangées (`shuffle=True`) pour favoriser une généralisation du modèle.

Observation :

L'utilisation des `DataLoaders` permet de gérer efficacement le chargement des données par lots pendant l'entraînement, la validation et les tests. Cela permet aussi une itération facile dans les boucles d'entraînement du modèle tout en maintenant une performance stable.

3. Entraînement du modèle

Objectif :

Concevoir et entraîner un modèle basé sur un LSTM pour la tâche de reconnaissance d'entités nommées.

Architecture du modèle :

1. **Embedding Layer** :
   - Convertit les mots en vecteurs denses de dimension `embedding_dim` (= 100).
   - Gère les mots inconnus (`<UNK>`) et le padding (`<PAD>`).
2. **Couche BiLSTM** :
   - **Type** : Bidirectionnel (analyse du contexte gauche et droit).
   - **Dimensions** :
     - `hidden_dim` = 256 (taille des états cachés).
     - `n_layers` = 2 (2 couches LSTM empilées).

- `dropout` = 0.3 (pour éviter le surapprentissage).

- **Pourquoi on utilise BiLSTM?**

  Perceque dans le LSTM classique par exemple "Apple" → Difficile de distinguer FRUIT vs ORG sans contexte futur mais BiLSMT utilise le contexte futur ("Apple released iPhone") pour identifier l'ORG.

  - La NER nécessite une compréhension **globale** de la phrase.
  - Les entités comme "European Commission" (ORG) ou "San Francisco" (LOC) dépendent de **l'ensemble du contexte**.

3. **Couche Linéaire (Classification)** :
   - Transforme les sorties du BiLSTM (`hidden_dim*2`) en logits de dimension `output_dim` (nombre de tags NER).

Détails de l'entraînement :

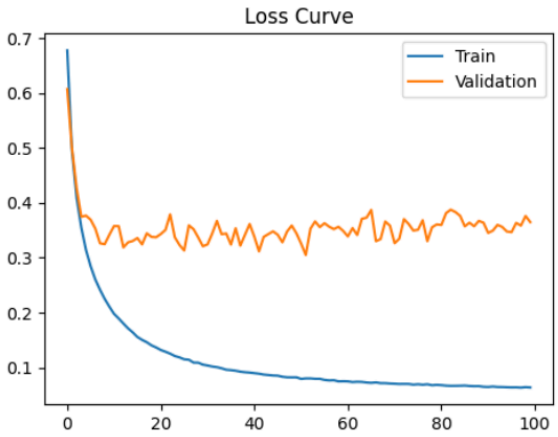
- Optimiseur : `Adam` avec un taux d'apprentissage de 0.001
- Fonction de perte : `CrossEntropyLoss` en ignorant les positions de padding (`-100`)
- Nombre d'époques : 100

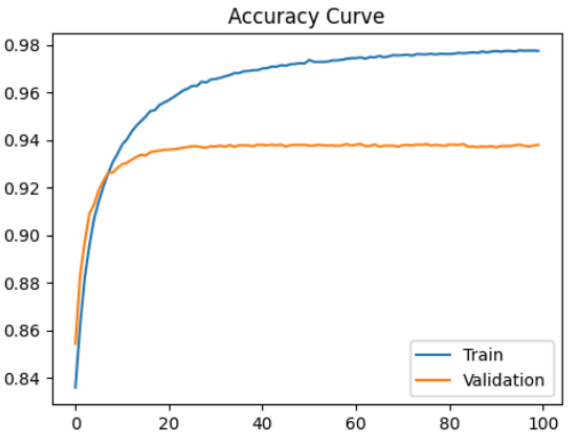
Fonctionnement:

- **Entrée** : Séquence de mots encodés en indices (ex: `[EU, rejects, German]` → `[2, 4, 3]`).
- **Sortie** : Probabilités pour chaque tag NER à chaque position (ex: `B-ORG`, `O`, `B-MISC`).

4. Évaluation du modèle

1. Courbes d'apprentissage





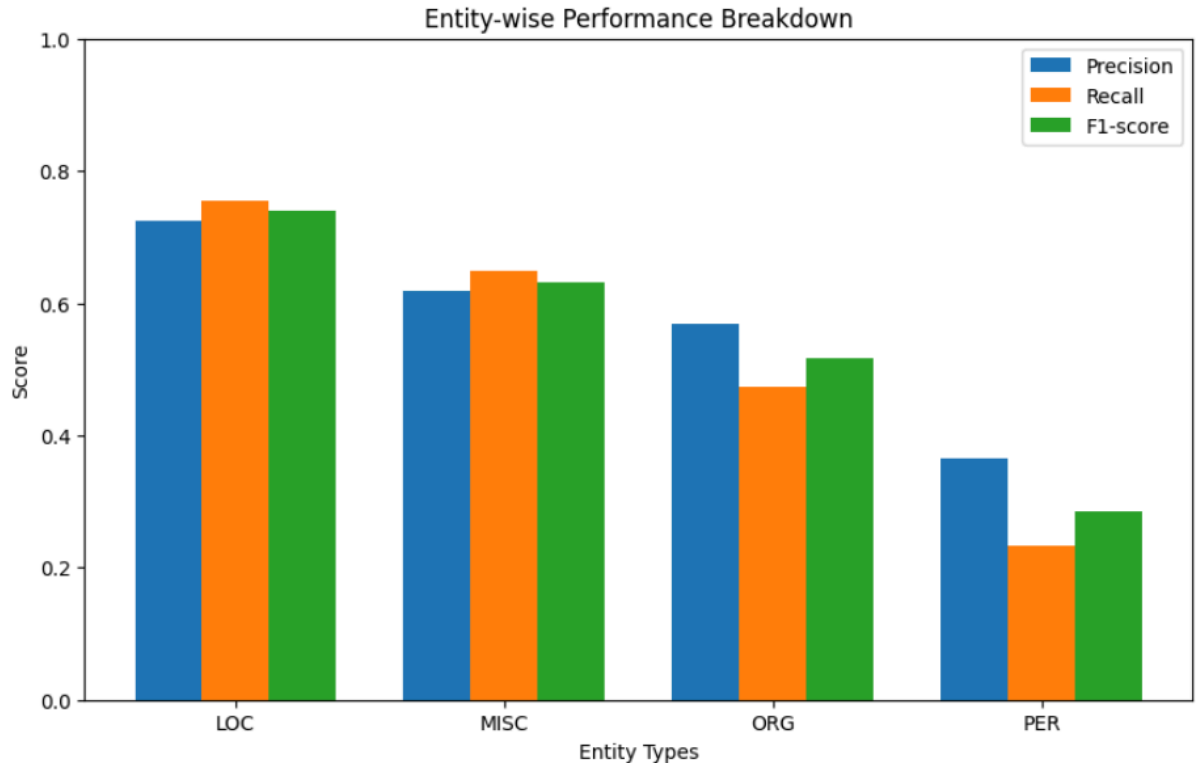
Observations :

- La **loss d'entraînement** diminue continuellement, indiquant une bonne convergence du modèle.
- La **loss de validation** stagne légèrement à partir de l'époque 20, ce qui peut indiquer un début de surapprentissage.
- La **précision de validation** se stabilise autour de **0.935**, ce qui montre une bonne généralisation avec un léger écart par rapport à l'entraînement.

---

2. Performances par entité

Nous avons analysé les performances du modèle sur le jeu de test pour chaque entité nommée (`LOC`, `MISC`, `ORG`, `PER`) en termes de **précision**, **rappel** et **F1-score**.



Visualisation des scores par type d'entité :

Analyse :

- Le modèle obtient de meilleurs résultats sur les entités `LOC` et `MISC`, avec des F1-scores autour de 0.75 et 0.65 respectivement.
- Les performances chutent significativement pour `ORG` et surtout `PER`, ce qui peut être dû à un déséquilibre de classes ou à une difficulté du modèle à capturer leurs contextes.

---

3. Scores globaux

En résumé, les performances globales sur le jeu de test sont :

- **F1-score** global : **0.546**
- **Précision** : **0.588**
- **Rappel** : **0.509**

Ces résultats montrent que le modèle est capable de reconnaître les entités avec une performance raisonnable, bien que des améliorations soient possibles, notamment via un meilleur équilibrage des données, des embeddings contextuels ...

5. Amélioration du modèle

Pour améliorer les performances du modèle NER de base, une nouvelle architecture enrichie a été développée en combinant **CNN**, **LSTM** et **Attention multi-têtes**. Cette version vise à mieux capturer les structures contextuelles complexes et à généraliser efficacement sur les entités nommées.

Modifications et techniques appliquées :

Nouveaux Composants

1. **Couche CNN** :
   - **But** : Extraire des motifs locaux (ex: n-grammes comme "New York").
   - **Paramètres** :
     - `Conv1d` : 64 filtres de taille 3 (`kernel_size=3`).
     - `padding=1` pour préserver la longueur des séquences.
   - **Fonctionnement:**
     a. **Texte** : Une matrice de mots (longueur de phrase × dimension d'embedding).
     b. **Opération de Convolution**
     - Un **filtre** (kernel) de taille `k` glisse sur les embeddings de mots pour capturer des **n-grammes** (groupes de `k` mots consécutifs).
     - **Exemple** :
       - `kernel_size=3` → Détecte des trigrammes comme "New York City" (LOC) ou "Mr. John Smith" (PER).
2. **Couche BiLSTM** :
   - Identique au modèle original, mais travaille sur les **features CNN** au lieu des embeddings bruts.
3. **Mécanisme d'Attention Multi-Têtes** :
   - **But** : Focus sur les mots clés (ex: entités).
   - **Paramètres** :
     - `num_heads=4` (4 mécanismes d'attention parallèles).
     - Combine les contextes locaux (CNN) et globaux (BiLSTM).

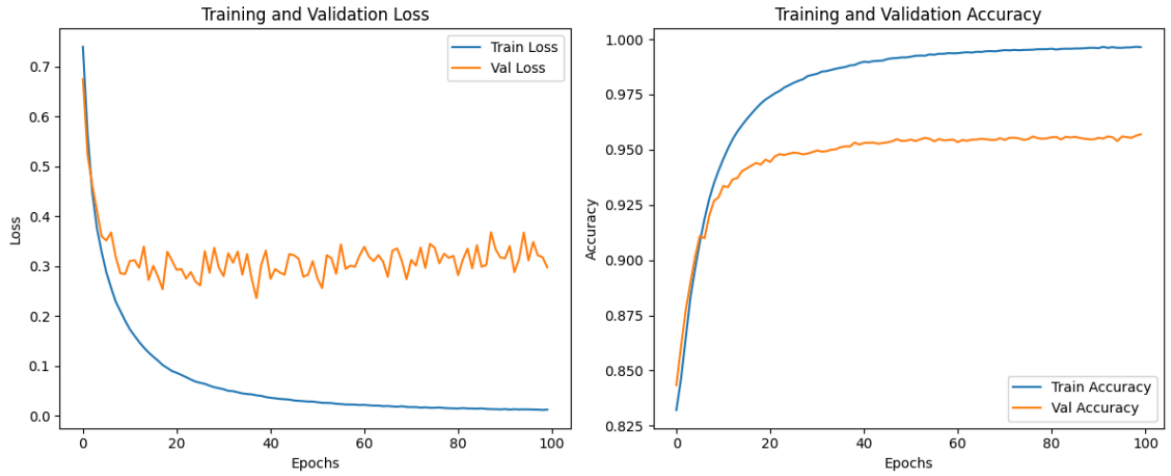
4. **Couche Linéaire (Classification)** :
    - Identique au modèle original.

Fonctionnement

- **Entrée** : Identique au modèle original.
- **Processus** :
    1. **Embedding** → **CNN** (extraction de motifs locaux).
    2. **BiLSTM** → **Attention** (pondération des mots importants).
    3. **Classification** (prédiction des tags).

6. Évaluation du modèle amélioré

1. Courbes d'apprentissage

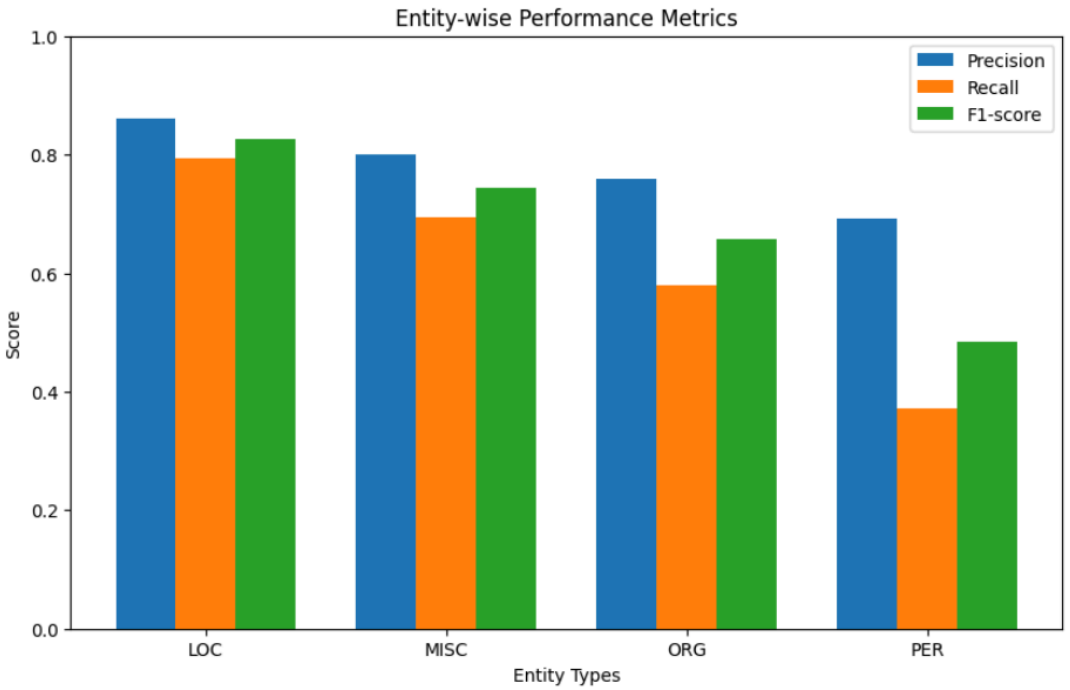


Observations :

- La **précision de validation** atteint environ **0.95**, un net progrès par rapport au modèle de base.

---

2. Performances par entité



Analyse :

- L'entité PER (personne) a les performances les plus faibles, en particulier en termes de rappel, ce qui pourrait suggérer qu'il manque beaucoup d'entités personnes.
- Le LOC est le plus fort dans toutes les mesures.

---

3. Scores globaux

Voici les scores globaux atteints sur le jeu de test :

- **F1-score**: 0.680
- **Precision**: 0.788
- **Recall**: 0.598

Cela reflète à nouveau le point précédent : le modèle est précis (lorsqu'il fait une prédiction, elle est généralement juste) mais manque un bon nombre de vrais positifs, en particulier dans certains types d'entités (comme PER et ORG).