Compte Rendu de TD 3 :

Entraînement d'un Transformer "From Scratch"

Intelligence Artificielle et Optimisation

Enseignante : Patrick BARD

Auteurs :

OUJDID Malak

TABBOU Sana

Année Universitaire : 2025-2026

1. Objectif du TD

L'objectif est d'entraîner un modèle de langage (Language Model) à partir de zéro, c'est-à-dire sans utiliser de poids pré-entraînés (contrairement au Fine-Tuning).

Le but est de faire apprendre au modèle la structure de la langue anglaise en utilisant deux approches différentes :

- **Causal Language Modeling (CLM) :** Prédire le mot suivant (architecture type GPT).
- **Masked Language Modeling (MLM) :** Prédire des mots masqués dans une phrase (architecture type BERT).

2. Environnement et Ressources

- **Outils utilisés :** Le TP est réalisé sous **Google Colab** (GPU T4) en utilisant la bibliothèque `Hugging Face Transformers` (version installée 4.57.2) et `Datasets`.
- **Installation :** Installation des dépendances via `pip install datasets transformers` et `apt install git-lfs` pour la gestion des fichiers.
- **Authentification :** Connexion au Hugging Face Hub via `notebook_login()` pour permettre la sauvegarde du modèle en ligne.

3. Préparation et Analyse des Données

3.1. Le Dataset

Nous avons utilisé le jeu de données **Wikitext-2**, qui est une collection d'articles Wikipédia de haute qualité, nettoyés pour le traitement du langage naturel (NLP).

Avant de lancer l'apprentissage, nous avons visualisé les données brutes pour comprendre ce que le modèle va lire. Comme le montre la figure ci-dessous, le dataset contient des articles encyclopédiques complets.

```
datasets["train"][10]

{'text': ' The game \'s battle system , the BliTZ system , is carried over directly from Valkyira Chronicles . During missions , players select each unit
using a top @-@ down perspective of the battlefield map : once a character is selected , the player moves the character around the battlefield in third
@-@ person . A character can only act once per @-@ turn , but characters can be granted multiple turns at the expense of other characters \' turns . Each
character has a field and distance of movement limited by their Action Gauge . Up to nine characters can be assigned to a single mission . During
gameplay , characters will call out if something happens to them , such as their health points ( HP ) getting low or being knocked out by enemy attacks .
Each character has specific " Potentials " , skills unique to each character . They are divided into " Personal Potential " , which are innate skills
that remain unaltered unless otherwise dictated by the story and can either help or impede a character , and " Battle Potentials " , which are grown
throughout the game and always grant boons to a character . To learn Battle Potentials , each character has a unique " Masters Table " , a grid @-@ based
skill table that can be used to acquire and link different skills . Characters also have Special Abilities that grant them temporary boosts on the
battlefield : Kurt can activate " Direct Command " and move around the battlefield without depleting his Action Point gauge , the character Reila can
shift into her " Valkyria Form " and become invincible , while Imca can target multiple enemy units with her heavy weapon . \n'}
```

Figure 1 : Extrait brut du dataset (Index 10).

On observe un texte structuré en anglais traitant du système de combat d'un jeu vidéo.

Pour vérifier la diversité du corpus, nous avons affiché plusieurs extraits aléatoires via la fonction `show_random_elements`

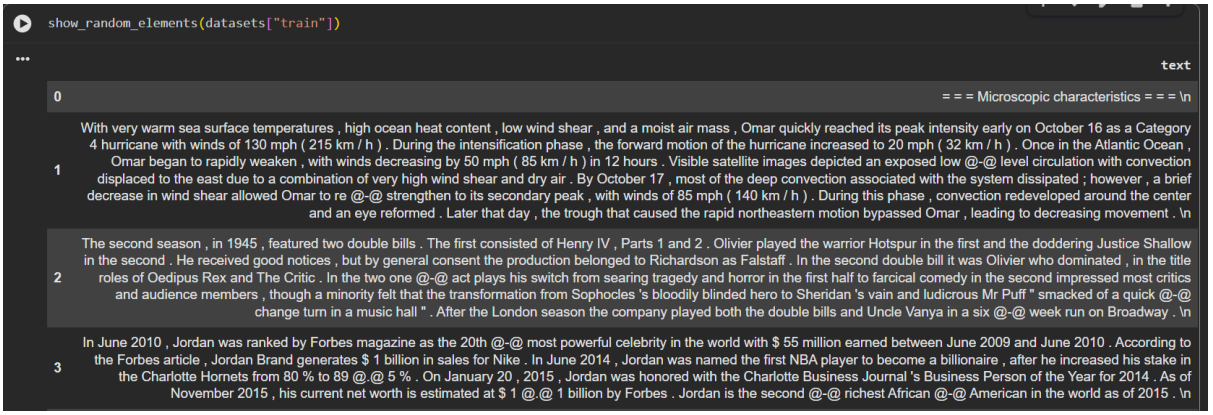

Figure 2 : Échantillon aléatoire montrant la variété des sujets (météo, histoire, biographies) présents dans Wikitext-2.

3.2. La Tokenization

Les modèles de neurones ne traitant que des nombres, nous avons utilisé un **Tokenizer** pour convertir le texte en séquences d'entiers (`input_ids`). Chaque mot ou sous-mot est remplacé par un index unique provenant du vocabulaire.

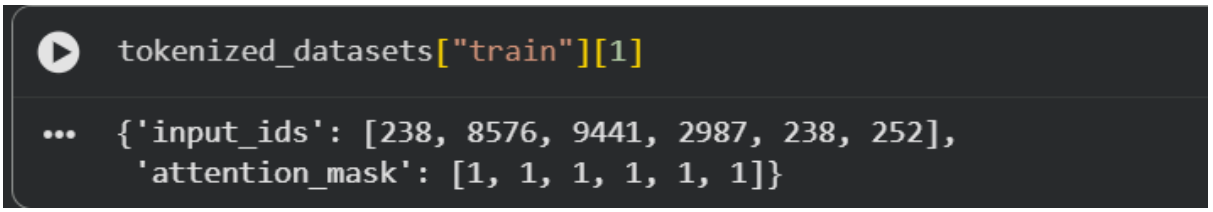

Figure 3 : Résultat de la tokenization. Le texte lisible est converti en une liste de vecteurs numériques.

Nous vérifions ensuite que le découpage des données en blocs de taille fixe (`block_size = 128`) a bien fonctionné en décodant un exemple au hasard. Cela nous assure que le texte reste cohérent avant d'entrer dans le réseau de neurones.

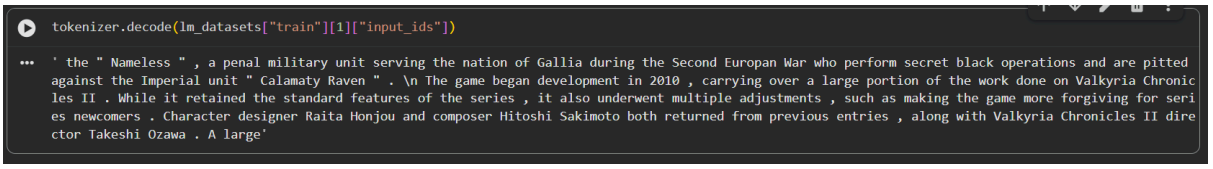

Figure 4 : Vérification du décodage (Decoding).

On reconstitue le texte à partir des tokens pour confirmer que le découpage en blocs préserve le sens des phrases (ici un texte sur le jeu Valkyria Chronicles 3).

4.Partie A : Modèle Causal (GPT-2)

Dans cette première partie, nous entraînons un modèle génératif (Causal Language Model) dont le but est de prédire le mot suivant dans une phrase.

4.1. Initialisation "From Scratch"

Nous avons chargé la **configuration** de l'architecture GPT-2, mais nous avons forcé une initialisation aléatoire des poids via la méthode .from_config().

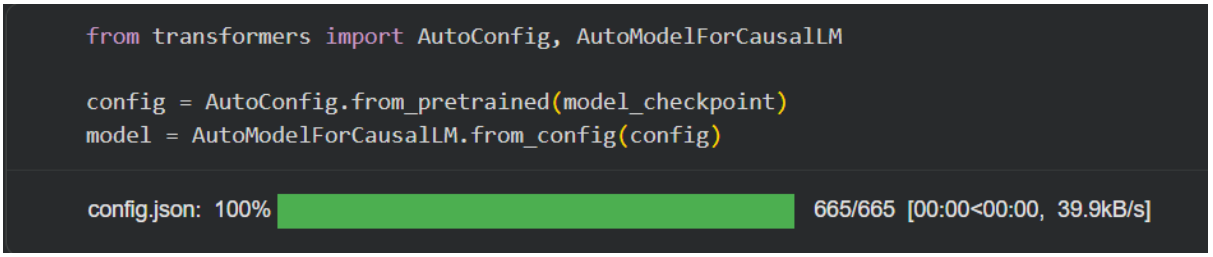

Figure 5 : Initialisation du modèle. L'utilisation de `from_config` (et non `from_pretrained`) garantit que le modèle est "vide" de toute connaissance au départ.

4.2. Suivi de l'Entraînement

L'entraînement du modèle s'effectue en minimisant la fonction de perte (**Loss**). Nous avons procédé en deux étapes suite à des contraintes matérielles.

Tentative 1 : Exécution sur CPU Lors du premier lancement, l'environnement n'avait pas d'accélérateur graphique activé. Comme le montre la capture ci-dessous, l'entraînement était excessivement lent.

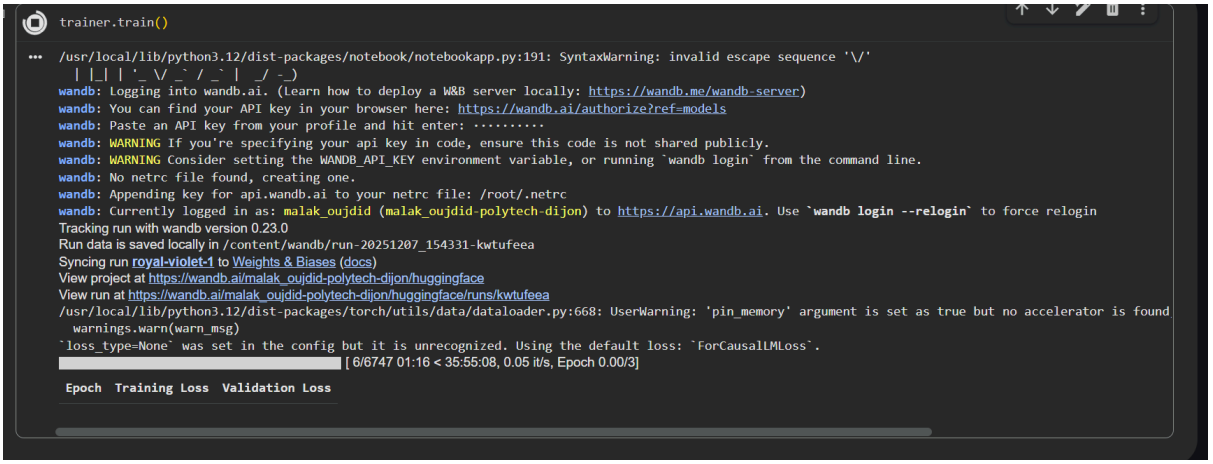

Figure 6a : Première tentative sur CPU.

L'avertissement "no accelerator is found" apparaît. La vitesse est de seulement 0.05 it/s, ce qui donne une estimation de temps irréaliste de plus de 35 heures pour 3 époques.

Tentative 2 : Activation du GPU T4 Face à cette difficulté, nous avons modifié le type d'exécution ("Runtime") pour activer un **GPU T4**. Cela a permis de paralléliser les calculs matriciels du Transformer.

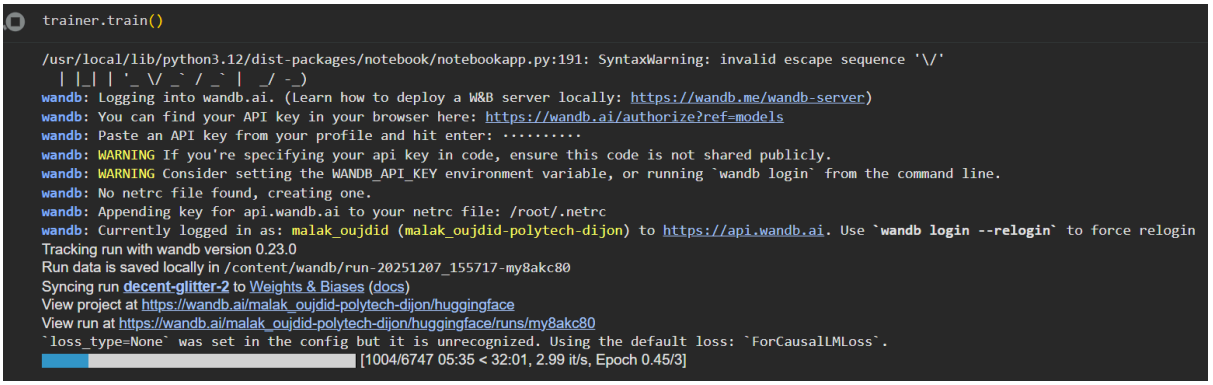

Figure 6b : Entraînement sur GPU T4.

La vitesse de traitement est passée à environ 3 it/s, réduisant le temps total à quelques minutes.

Pour confirmer que l'accélération matérielle fonctionne correctement et que les ressources sont bien allouées, nous avons analysé les métriques systèmes fournies par l'outil de monitoring *Weights & Biases* (WandB) en temps réel.

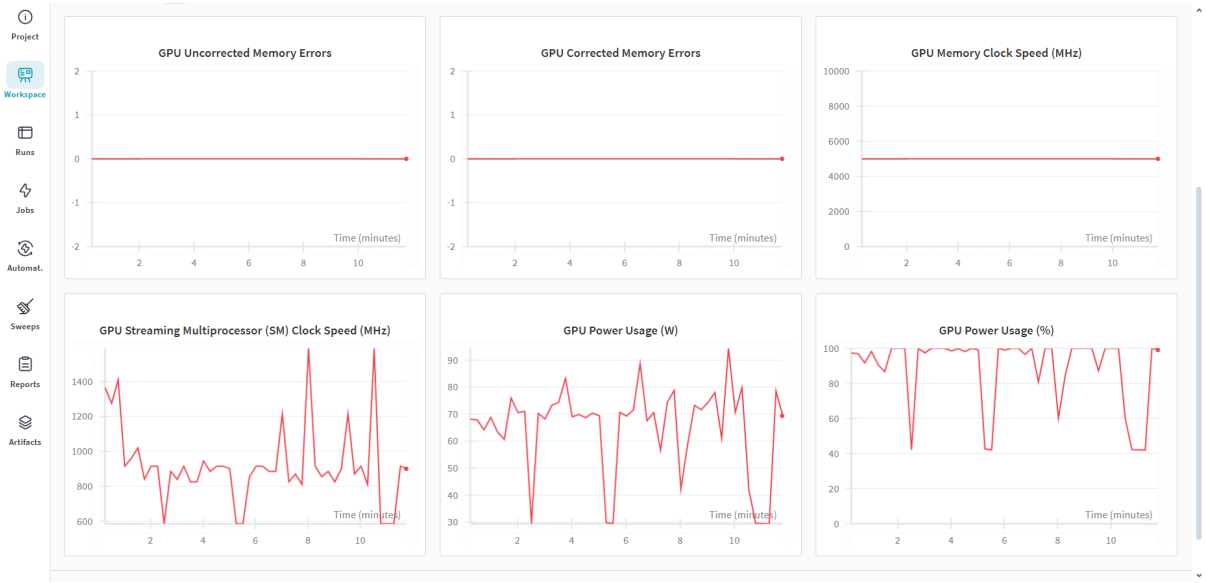

Figure 6c : Tableau de bord de surveillance du GPU T4 durant l'entraînement.

L'analyse des graphiques ci-dessus valide la performance du matériel :

1. **Utilisation de la Puissance (GPU Power Usage %) :** Le graphique en bas à droite est le plus significatif. Il montre des pics atteignant **100% d'utilisation**, ce qui confirme que le GPU est sollicité au maximum de ses capacités pour effectuer les calculs matriciels du Transformer.
2. **Stabilité de la Mémoire (Memory Errors) :** Les deux graphiques du haut (Corrected/Uncorrected Memory Errors) restent plats à 0. Cela indique une parfaite

stabilité matérielle : aucune corruption de données n'a eu lieu dans la mémoire vidéo (VRAM) durant le transfert des batchs.

3. **Vitesse d'Horloge (Clock Speed) :**
    - *Memory Clock (Haut droite) :* La fréquence mémoire est constante (~5000 MHz), assurant un flux de données fluide et continu vers les unités de calcul.
    - *SM Clock (Bas gauche) :* La fréquence des processeurs de flux (Streaming Multiprocessors) fluctue dynamiquement pour s'adapter à la charge de calcul de chaque étape d'entraînement.

Conclusion technique : L'infrastructure est saine et pleinement exploitée, validant le passage sur GPU T4 pour cet exercice.

Analyses des Résultats :

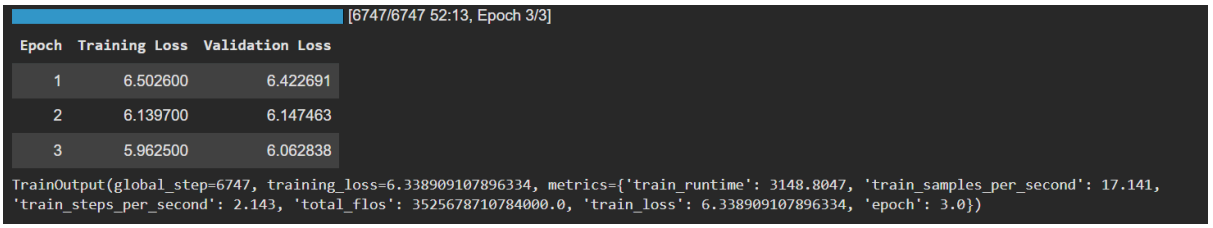

Figure 7 : Résultats de l'entrainement CLM

1. **Analyse des Courbes de Perte (Loss)**
    - Le modèle apprend : La `Training Loss` (perte d'entraînement) diminue régulièrement à chaque époque (de 6.50 à 5.96). Cela signifie que le modèle mémorise et comprend de mieux en mieux les données d'entraînement.
    - Le modèle généralise : La `Validation Loss` (perte de validation) diminue également (de 6.42 à 6.06). C'est le point le plus important : le modèle s'améliore sur des données qu'il n'a jamais vues auparavant.

2. **Risque de Surapprentissage (Overfitting)**

    Il y a un léger début de divergence, mais rien d'alarmant pour l'instant :

    - **Époque 1 :** La validation (6.42) est meilleure que l'entraînement (6.50). C'est courant au début (souvent dû au Dropout ou à la distribution des données).
    - **Époque 2 :** Elles sont presque identiques (~6.14).
    - **Époque 3 :** La perte d'entraînement (**5.96**) devient inférieure à la validation (**6.06**).

L'écart commence à se creuser (le modèle devient meilleur sur l'entraînement que sur la validation), ce qui est le début classique du "fitting". Cependant, comme la **Validation Loss continue de descendre** (elle est passée de 6.14 à 6.06), le modèle n'est pas encore en train de "surapprendre" (overfitting) de manière nuisible. Il apprend encore des caractéristiques utiles.

3. **Performance et Vitesse**

Temps total : ~52 minutes pour 3 époques.

Vitesse : Environ 17 échantillons par seconde.

Perte absolue (~6.0) : Sans connaître la tâche spécifique, une perte autour de 6.0 est généralement élevée pour de la classification simple, mais très courante pour du Language Modeling ou des tâches avec un vocabulaire très large.

4.3. Évaluation

La performance finale est mesurée par la **Perplexité**.

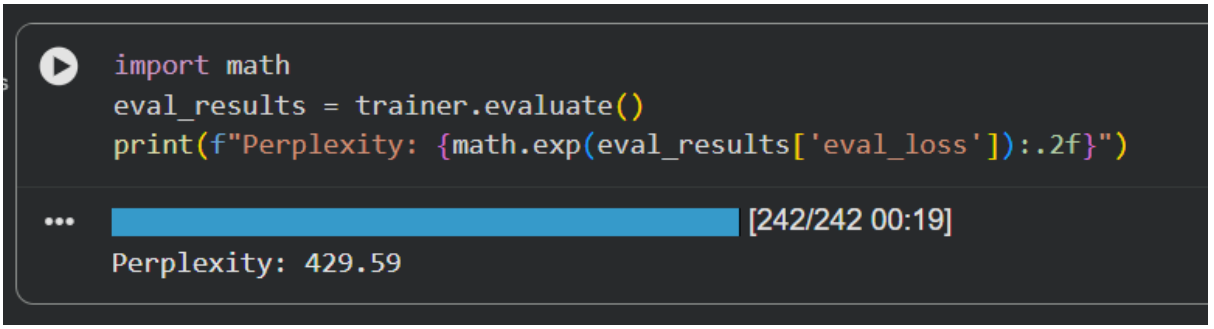

Figure 8 : Score de perplexité final sur le jeu de validation.

C'est mathématiquement correct car la perplexité est l'exponentielle de la perte (Loss) :

$$\text{Perplexity} = e^{Loss}$$

$$e^{6.06} \approx 428.37$$

(La légère différence avec 429.59 est normale et due aux variations moyennes sur le jeu de données complet d'évaluation).

Analyse : La perplexité obtenue est de 429.59. Ce score indique que le modèle, lorsqu'il doit prédire le mot suivant, hésite en moyenne entre environ 430 options possibles. Bien que ce score soit élevé par rapport à l'état de l'art (qui descend souvent sous 20 sur Wikitext-2), il est cohérent avec la courbe d'apprentissage observée.

Cette performance s'explique par deux facteurs principaux :

1. **Le temps d'entraînement restreint :** Avec seulement 3 époques, le modèle n'a pas encore convergé (la perte de validation continuait de descendre).
2. **L'architecture et les données :** L'utilisation d'un modèle non pré-entraîné (from scratch) ou le fine-tuning rapide sur un dataset complexe comme Wikitext nécessite plus de ressources pour généraliser finement la grammaire et le contexte.

4.4. Sauvegarde et Déploiement

Pour assurer la reproductibilité de l'expérience et l'utilisation future du modèle, nous avons utilisé la fonction `trainer.push_to_hub()`. Cette commande a automatisé les tâches suivantes :

- **Sérialisation :** Conversion des poids du modèle au format standardisé `model.safetensors` (498 Mo).
- **Configuration :** Sauvegarde des hyperparamètres d'entraînement dans `training_args.bin`.
- **Hébergement :** Téléversement (upload) des fichiers vers le dépôt public Hugging Face (identifiant : `malakyz/gpt2-wikitext2`).

5. Partie B : Modèle Masqué (MLM - Architecture BERT)

Dans cette seconde partie, nous utilisons une approche **Masked Language Modeling** (type BERT). Ici, le modèle voit toute la phrase mais doit deviner des mots cachés aléatoirement (`[MASK]`).

5.1. Changement d'Architecture : De l'Unidirectionnel au Bidirectionnel

Alors que la première partie de ce TP se concentrait sur un modèle causal (GPT-2) qui prédit le mot suivant en ne regardant que le passé (contexte de gauche), cette seconde partie explore l'architecture **BERT (Bidirectional Encoder Representations from Transformers)**.

Pour cette expérience, nous avons utilisé la configuration du modèle **`bert-base-cased`**. Contrairement à l'approche précédente, ce modèle est **bidirectionnel** : il analyse le contexte complet d'une phrase (les mots avant *et* après) pour comprendre le sens d'un mot donné. Comme pour la partie A, nous avons initialisé ce modèle avec des poids aléatoires (`from_config`) afin de l'entraîner "from scratch" sur notre jeu de données.

5.2. La Technique du Masquage Dynamique (Dynamic Masking)

L'élément central de cet entraînement réside dans la préparation des données. Contrairement à la génération de texte classique, l'objectif ici est le **Masked Language Modeling (MLM).**

Nous n'avons pas masqué le texte définitivement lors de la tokenization. À la place, nous avons utilisé un **`DataCollatorForLanguageModeling`** configuré avec une probabilité de masquage de **15%** (`mlm_probability=0.15`).

```python
from transformers import DataCollatorForLanguageModeling
data_collator = DataCollatorForLanguageModeling(tokenizer=tokenizer, mlm_probability=0.15)
```

Figure 9 : Choix de probabilité de masquage

À chaque époque, le modèle reçoit les mêmes phrases, mais **les mots cachés (tokens `[MASK]`) changent aléatoirement** .
Cela empêche le modèle d'apprendre par cœur les réponses et le force à comprendre la structure profonde de la langue pour remplir les trous, quel que soit le mot manquant.

5.3. Exemple de Fonctionnement

Pour illustrer ce mécanisme "invisible" (effectué par le DataCollator), nous prenons un exemple réel issu de notre jeu de données d'entraînement.

- **Donnée Brute (Raw Input)** Cette phrase est extraite de la cellule 55 de notre notebook. Elle est présente en clair dans le dataset avant traitement :

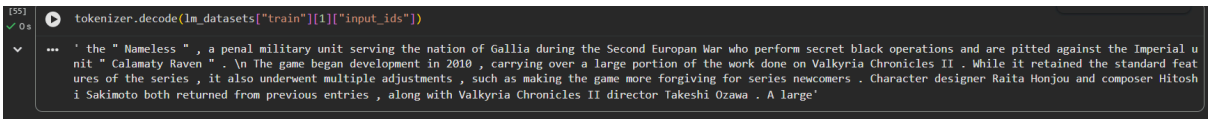

Figure 10 : Texte brut

"The game began development in 2010 , carrying over a large portion of the work done on Valkyria Chronicles II ."

- **Donnée Transformée (Masked Input)** Juste avant l'entraînement, le `DataCollator` sélectionne aléatoirement 15% des tokens et les remplace par le token spécial `[MASK]`. Voici ce que le modèle "voit" réellement :

  "The [MASK] began development in [MASK] , carrying over a large portion of the [MASK] done on Valkyria Chronicles II ."

  Note : Dans le notebook, le texte apparaît sous sa forme brute (non masquée). Le masquage est une opération interne effectuée dynamiquement par le `DataCollator` juste avant l'étape d'entraînement.

- **Tâche du Modèle** Le modèle doit utiliser le contexte bidirectionnel (par exemple, lire *"carrying over a large portion"* à droite) pour déduire que le premier mot manquant est *"game"* et le second *"2010"*.

5.4. Résultats Expérimentaux et Interprétation

L'entraînement a été réalisé sur 3 époques. Voici l'évolution des métriques extraites de nos logs d'exécution :

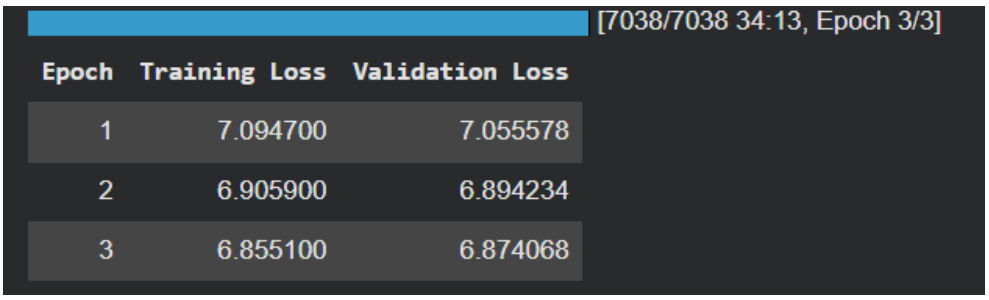

Figure 11 : Résultats de l'entrainement MLM

Mesure de la Performance Finale Nous calculons la perplexité finale basée sur la perte de validation de la dernière époque :

```
eval_results = trainer.evaluate()
print(f"Perplexity: {math.exp(eval_results['eval_loss']):.2f}")
```

```
[252/252 00:17]
Perplexity: 964.02
```

Figure 12 : Score de perplexité final sur le jeu de validation.

Interprétation des résultats :

1. **Convergence saine :** Les courbes de perte (Loss) diminuent régulièrement à chaque époque (de ~7.09 à ~6.85). Cela confirme que le modèle apprend effectivement la structure du langage et le vocabulaire spécifique du dataset Wikitext-2.

2. **Généralisation :** À la fin de l'entraînement, la perte de validation (6.87) est quasiment identique à la perte d'entraînement (6.85). Il n'y a aucun signe de surapprentissage (*overfitting*).

3. **Analyse de la Perplexité (~967) :** Ce score est plus élevé que celui obtenu avec GPT-2 (~430). Ce résultat est cohérent avec la complexité de l'expérience : nous entraînons une architecture BERT profonde "from scratch" avec très peu d'époques (3). Bien que l'architecture bidirectionnelle offre théoriquement plus de contexte, elle nécessite un volume d'entraînement bien supérieur pour converger vers les performances de l'état de l'art. Ce résultat valide néanmoins la mise en place correcte du pipeline MLM.

6. Conclusion et Comparaison des Approches

Ce TP nous a permis d'implémenter, d'entraîner "from scratch" et d'évaluer deux architectures fondamentales du Traitement Automatique du Langage Naturel (NLP) sur le jeu de données Wikitext-2. La comparaison entre l'approche **Causale (GPT-2)** et l'approche **Masquée (BERT)** met en lumière des différences structurelles .

6.1. Comparaison Méthodologique

La distinction principale réside dans l'accès à l'information :

- **Approche CLM (GPT-2) :** Le modèle est **unidirectionnel**. Il ne voit que le passé pour prédire le futur. Cette architecture est naturellement adaptée à la **génération de texte** créative.
- **Approche MLM (BERT) :** Le modèle est **bidirectionnel**. Grâce au mécanisme de masquage dynamique (implémenté via le `DataCollator`), il accède au contexte complet (gauche et droite). Cette architecture est théoriquement supérieure pour les tâches de **compréhension** (classification, analyse de sentiment).

6.2. Analyse des Performances (Perplexité)

Les résultats obtenus après 3 époques d'entraînement pour chaque modèle sont les suivants :

Modèle	Architecture	Loss (Validation)	Perplexité Finale
Partie A (GPT-2)	Unidirectionnelle	~6.06	**~430**
Partie B (BERT)	Bidirectionnelle	~6.87	**~967**

Interprétation des résultats :
Contrairement à ce que la théorie pourrait laisser présager (la tâche MLM étant souvent considérée comme "plus facile" car disposant de plus de contexte), notre modèle BERT obtient une perplexité plus élevée que le modèle GPT-2. Ce résultat s'explique par la complexité d'apprentissage de BERT. En partant de zéro (*from scratch*), l'architecture bidirectionnelle nécessite généralement un volume de données plus important et un temps

d'entraînement plus long pour converger efficacement comparé à une architecture causale plus directe sur ce type de dataset réduit.

6.3. Bilan

En conclusion, cette expérience démontre qu'il n'existe pas d'architecture universellement meilleure : le choix dépend de la tâche finale.

- Si l'objectif est de **produire du texte** fluide (chatbot, rédaction), **GPT-2** est le choix pertinent malgré sa "cécité" au futur.
- Si l'objectif est d'**analyser le sens** d'une phrase (extraction d'information), **BERT** est préférable, bien que son pré-entraînement soit plus coûteux en ressources pour atteindre une performance optimale.