RCP 209

le cnam

CONSERVATOIRE NATIONAL DES ARTS ET MÉTIERS

Centre de Paris

Master 2 TRIED

Projet UE RCP209 - Apprentissage Statistique 2

Présenté par

Elimane Y. SEIDOU

Génération de texte par RNN : approche expérimentale de la méthode multiplicative sur les GRU, minGRU et LRU pour la génération de caractères

Responsable de l'UE :

Arnaud BRELOY – Professeur des universités

2024-2025

Abstract

Ce projet explore l'utilisation des réseaux de neurones récurrents (RNN) pour la génération de texte, en se concentrant sur trois architectures spécifiques : le GRU (Gated Recurrent Unit), le minGRU (une version simplifiée du GRU), et le LRU (Linear Recurrent Unit). L'objectif principal est d'évaluer l'efficacité de ces modèles dans la génération de texte, en particulier en appliquant une méthode multiplicative pour améliorer la gestion des dépendances à long terme. La méthode multiplicative, qui permet une interaction dynamique entre l'entrée et la matrice de transition des états cachés, a été appliquée aux GRU, minGRU et LRU pour évaluer son impact sur la génération de texte. Les résultats montrent que, malgré des tentatives de simplification et d'optimisation, les modèles réduits (comme le minGRU et le LRU) souffrent d'une perte significative de précision par rapport au GRU traditionnel. De plus, l'application de la méthode multiplicative n'a pas apporté d'amélioration notable, et les modèles ont montré une faible confiance lors de la génération de texte. Ces résultats soulignent la nécessité d'approches plus sophistiquées, comme l'intégration de mécanismes d'attention, pour améliorer les performances des modèles récurrents dans des tâches complexes de génération de texte. Les expériences menées dans ce projet ont également révélé des défis liés à l'overfitting et à la généralisation, suggérant que des techniques de régularisation plus robustes et une augmentation des données pourraient être nécessaires pour améliorer les performances des modèles. Enfin, les taux d'erreur élevés, tant par caractères que par mots, indiquent que la capacité de correction automatique des modèles est limitée, ce qui met en lumière la nécessité de continuer à explorer des techniques d'optimisation et de régularisation pour améliorer la robustesse et la généralisation des modèles récurrents dans des tâches de génération de texte.

Introduction

Depuis la conférence de Dartmouth en 1956, qui a marqué le début de l'intelligence artificielle moderne, de nombreux efforts ont été déployés pour doter les machines de compétences linguistiques. Les réseaux de neurones récurrents (RNN) ont longtemps été utilisés pour des tâches de génération de texte, mais ils sont confrontés à des limites, notamment en ce qui concerne la gestion des séquences longues et la parallélisation. Ces limitations ont conduit à l'émergence de nouvelles architectures, comme les LSTM (Long Short-Term Memory) et les GRU (Gated Recurrent Unit), qui ont montré des performances supérieures dans la gestion des dépendances à long terme. Plus récemment, des variantes simplifiées comme le minGRU et le LRU (Linear Recurrent Unit) ont été proposées pour améliorer l'efficacité computationnelle et la parallélisation des modèles récurrents.

Ce projet vise à explorer ces architectures, en particulier en appliquant une méthode multiplicative pour améliorer leur capacité à gérer les dépendances à long terme. La méthode multiplicative, qui permet une interaction dynamique entre l'entrée et la matrice de transition des états cachés, a été appliquée aux GRU, minGRU et LRU pour évaluer son impact sur la génération de texte. Nous examinons également l'impact de la réduction des paramètres sur les performances des modèles et évaluons leur capacité à générer du texte de manière fiable. Les résultats de cette étude fournissent des insights précieux sur les limites et les opportunités d'amélioration des modèles récurrents pour la génération de texte.

I - Problématique, Données et Objectifs

II - Méthodologie

Afin de réaliser les différentes expérimentations, une méthodologie a été suivie allant de l'exploration des données à la génération de texte. Dans cette section, nous exposons ces différentes étapes suivies et les outils nous qui nous ont permis de les implémenter.

II-1- Analyse Exploratoire et Prétraitement

II-1.1- Analyse Exploratoire

Comme dans tout projet impliquant l'exploitation de données, une analyse exploratoire a d'abord été faite afin d'observer les différentes tendances et l'aspect général des données. La particularité de nos données étant leur nature textuelle, notre exploration s'est beaucoup plus centrée sur des aspects de tailles, de distribution et d'identification d'éléments particuliers. Nous avons donc dans notre analyse observé :

- le nombre total de mots présents dans notre document
- le nombre total de mots uniques

- le nombre de phrases contenant un certain nombre de tokens
- les mots avec les plus grandes fréquence d'apparition dans l'ensemble du document

II-1.2- Prétraitement

Après l'analyse exploratoire vient le prétraitement des données commun à la plupart des projets en traitement automatique de langue. Les informations textuelles étant brute et sujette à des biais en l'état, il est important de les nettoyer avant de passer à l'entraînement de modèles sur ces données. Les différentes étapes de prétraitement qui ont été appliquées sont les suivantes :

- conversion des mots en minuscules : Unifie le texte (évite "Chat" ≠ "chat"). **Important** pour la cohérence de l'analyse.
- suppression des caractères non alphabétiques grâce aux techniques d'expressions régulières : Élimine ponctuation et symboles inutiles. **Important** pour nettoyer le texte
- tokenization : Découpe le texte en mots ou phrases. Ici nous avons choisi de découper le texte en mots
- retrait des stop words : Supprime les mots fréquents mais peu informatifs (ex. "le", "et"). **Important** pour se concentrer sur les mots clés
- lemmatization : Ramène un mot à sa forme de base (ex. "manges" → "manger"). **Important** pour regrouper les variantes d'un mot

A la suite de notre prétraitement, une nouvelle analyse exploratoire a été effectuée afin d'observer à nouveau la nouvelle distribution des mots et l'aspect global de notre document.
La dernière phase de notre prétraitement consistera en la préparation effective des données que nous souhaitons utiliser dans l'entraînement de notre modèle. Ainsi, nous avons :

- extrait du document prétraité un échantillon textuel sur lequel nous effectuerons nos entraînements et inférences
- concaténé les lignes en une seule grande chaîne de text
- créé un dictionnaire de caractère en récupérant tous les caractères uniques du texte
- créé des séquences de caractères
- encodé les caractères en indices
- effectué une conversion en tenseurs one-hot
- séparé les données d'entraînement en train-test

A la suite de ce prétraitement, nous avons procédé à la création des modèles que nous avons utilisés, à leur entraînement et à l'analyse de leur confiance.

II-2- Modèles explorés

Plusieurs modèles ont été explorés pour la tâche de génération de texte. Parmi les modèles explorés, les modèles neuronaux de types recurrent unit on été choisi pour leur simplicité et leur performances comparé au RNN classiques ou aux LSTM. Les architectures de modèles explorés sont celles du GRU, du minGRU et du LRU.

Les modèles ont été entraînés dans des conditions quasi similaires à quelques différences près. Les méthodes et outils suivants ont été utilisés pour l'entraînement des modèles sont les suivants :

- Cosine Anaeling Scheduler : Diminution du taux d'apprentissage de 0.001 à 0.0001 sur 100 epochs
- Dropout : Désactivation aléatoirement 30% des neurones après chaque couche GRU pour éviter le surapprentissage
- Weigth Decay (0.0001) : Ajout d'une régularisation L2 pour contraindre les poids du modèle
- nombre d'epochs : Entraîne le modèle en parcourant 100 fois l'ensemble de données
- nombre de batchs :Division les données en sous-groupes de 128 échantillons pour chaque mise à jour des poids.
- Early stopping : Optimisation des poids en ajustant les gradients avec une moyenne exponentielle.
- optimiseur (RMSprop puis finalement Adam, learning_rate=0.001) : Optimisation les poids en ajustant les gradients avec une moyenne exponentielle.
- fonction perte : Évaluation la différence entre la sortie du modèle et les classes cibles avec categorical_crossentropy
- skip connection : Ajout de connexions résiduelles permettant de faciliter la propagation du gradient et d'améliorer la convergence du modèle en contournant les problèmes de disparition du gradient. Ces connexions permettent également aux couches plus profondes de mieux exploiter les informations des couches précédentes.
- BatchNormalization : Normalisation des activations après chaque couche récurrente afin de stabiliser l'entraînement, accélérer la convergence et limiter l'explosion ou la disparition des gradients. Cette technique permet également de réduire la sensibilité du modèle aux variations dans les distributions des données d'entrée, améliorant ainsi sa robustesse.
- Test-Time Scaling [0] : Application d'un facteur de mise à l'échelle aux poids du modèle lors de l'inférence pour compenser les effets du dropout utilisé pendant l'entraînement. Cela permet d'assurer que les activations restent cohérentes entre les phases d'entraînement et d'inférence, évitant ainsi une chute des performances du modèle après l'entraînement.

II-2.1- GRU

Introduit par Kyunghyun Cho et al. [1] [2] en 2014, l'architecture GRU (Gated Recurrent Unit) est une variante simplifiée des LSTM (Long Short-Term Memory). Elle vise à résoudre les problèmes de dépendances à long terme dans les réseaux de neurones récurrents classiques tout en étant plus simple et moins coûteuse en calcul. Contrairement aux LSTM, les GRU combinent les portes d'entrée et d'oubli en une seule porte appelée "porte de mise à jour" (update gate). De plus, elles intègrent une porte de réinitialisation (reset gate) pour contrôler le flux d'informations venant de la mémoire précédente.

Cette simplification permet de réduire le nombre de paramètres et d'améliorer l'efficacité computationnelle, tout en maintenant des performances similaires à celles des LSTM dans de nombreuses tâches séquentielles, telles que la traduction automatique et la reconnaissance vocale.

À chaque étape temporelle, une GRU prend en entrée la donnée actuelle (par exemple, le mot actuel dans une phrase) et l'état caché précédent (qui contient l'information accumulée jusqu'à cette étape). Les deux portes contrôlent le flux d'informations de la manière suivante :

Porte de mise à jour (update gate) : Elle détermine dans quelle mesure l'état caché précédent doit être conservé ou mis à jour avec la nouvelle information.

- **Calcul de la porte de mise à jour :** $z_t = \sigma(W_z \cdot x_t + U_z \cdot h_{t-1} + b_z)$
  où :
  - x_t est l'entrée actuelle.
  - h_{t-1} est l'état caché précédent.
  - W_z et U_z sont des matrices de poids.
  - b_z est un vecteur de biais.
  - σ représente la fonction sigmoïde, qui produit des valeurs entre 0 et 1.

Porte de réinitialisation (reset gate) : Elle décide combien de l'information précédente doit être oubliée.

- **Calcul de la porte de réinitialisation :** $r_t = \sigma(W_r \cdot x_t + U_r \cdot h_{t-1} + b_z)$
  où les notations sont similaires à celles de la porte de mise à jour.

Calcul de l'état candidat (proposition de nouvel état) : Une nouvelle information est générée en tenant compte de l'entrée actuelle et de l'état caché précédent modifié par la porte de réinitialisation.

- **Formule :** $\tilde{h}_t = tanh(W \cdot x_t + r_t * (U \cdot h_{t-1}) + b)$ où :
  - $\tilde{h}_t$ est l'état candidat.
  - $tanh$ est la fonction tangente hyperbolique, qui produit des valeurs entre -1 et 1.
  - * représente la multiplication élément par élément.

Mise à jour de l'état caché : L'état caché actuel est une combinaison entre l'état candidat et l'état caché précédent, pondérée par la porte de mise à jour.

- **Formule :** $h_t = z_t * h_{t+1} + (1 - z_t) * \tilde{h}_t$
  Cette équation signifie que si z_t est proche de 1, le réseau conserve principalement l'état précédent h_{t-1}. Si z_t est proche de 0, il privilégie le nouvel état candidat $\tilde{h}_t$.

II-2.2- minGRU

Introduit dans l'étude *"Were RNNs All We Needed?"* par Leo Feng et al. [3] en 2024, **minGRU** est une version simplifiée du GRU (Gated Recurrent Unit) qui vise à rendre les modèles récurrents plus parallélisables et efficaces en supprimant certaines dépendances aux états précédents. Contrairement aux GRU classiques, **minGRU** élimine la porte de réinitialisation (reset gate) et supprime la dépendance explicite des portes aux états cachés passés. Cette simplification permet une réduction significative du nombre de paramètres et facilite un entraînement entièrement parallélisable, tout en conservant des performances comparables aux modèles récents tels que Mamba ou S4.

À chaque étape temporelle, un minGRU prend en entrée la donnée actuelle x_t et l'état caché précédent h_{t-1}. Il utilise une porte unique de mise à jour pour contrôler le flux d'informations et assurer la transition entre l'état précédent et l'état candidat.

Porte de mise à jour (update gate) :

Elle détermine dans quelle mesure l'état caché précédent doit être conservé ou remplacé par le nouvel état candidat.

$$z_t = \sigma(W_z \cdot x_t + b_z)$$

où :

- x_t est l'entrée actuelle.
- W_t est une matrice de poids.
- b_z est un vecteur de biais.
- σ représente la fonction sigmoïde, qui produit des valeurs entre 0 et 1.

Calcul de l'état candidat :

L'état candidat est directement calculé à partir de l'entrée actuelle, sans dépendance explicite aux états cachés précédents.

$$\tilde{h}_t = W_h \cdot x_t + b_h$$

où :

- $\tilde{h}_t$ est l'état candidat.
- W_h est une matrice de poids.
- b_h est un vecteur de biais.

Mise à jour de l'état caché :

L'état caché actuel est obtenu en combinant l'état candidat et l'état caché précédent, pondérés par la porte de mise à jour.

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t$$

où $\odot$ représente la multiplication élément par élément.

Ainsi :

- minGRU **supprime** $U_z \cdot h_{t-1}$, ce qui permet une parallélisation complète du modèle dans la porte de mise à jour
- minGRU **n'utilise plus de porte de réinitialisation**, ce qui réduit encore le nombre de paramètres et simplifie l'architecture.
- *minGRU supprime* $U_t \cdot (r_t \odot h_{t-1})$ et enlève la fonction \tanh pour ne plus contraindre l'échelle des états cachés.
- Même si la formule est la même, dans minGRU $\tilde{h}_t$ **ne dépend plus de** h_{t-1} , ce qui réduit le besoin de calcul séquentiel.

Grâce à ces simplifications, **minGRU** peut être entraîné en parallèle à l'aide de l'algorithme de scan parallèle, éliminant ainsi le besoin de la rétropropagation à travers le temps (BPTT). Cette approche rend l'entraînement des modèles récurrents plus rapide et plus scalable, tout en maintenant une capacité d'apprentissage compétitive sur des tâches complexes telles que la modélisation du langage et l'apprentissage par renforcement.

II-2.3- LRU

Introduit par Antonio Orvieto et al. en 2023 [4], l'architecture **Linear Recurrent Unit (LRU)** est une variante optimisée des réseaux neuronaux récurrents (RNN). Elle vise à résoudre les problèmes de gradient et d'optimisation des RNN classiques tout en égalant les performances des modèles d'état spatial (SSM) modernes comme **S4 et S5**. Contrairement aux architectures récurrentes traditionnelles, le **LRU** adopte une approche basée sur des récurrences linéaires et des matrices diagonales dans l'espace complexe, permettant un entraînement parallélisable et plus efficace.

Cette simplification réduit la complexité computationnelle, facilite la convergence du modèle et améliore la stabilité sur les tâches de modélisation de séquences longues, tout en conservant une expressivité élevée. Le **LRU** est particulièrement performant dans des applications nécessitant un raisonnement à long terme, comme la modélisation du langage et la classification de séquences.

À chaque étape temporelle, un **LRU** prend en entrée la donnée actuelle x_t et l'état caché précédent h_{t-1} , avec une mise à jour reposant sur une paramétrisation exponentielle et une normalisation adaptée

Récurrence linéaire avec matrice diagonale

Contrairement aux RNN classiques, où la mise à jour de l'état suit une transformation dense et non linéaire, le **LRU** utilise une matrice de transition diagonale dans l'espace complexe :

$$h_t = \Lambda h_{t-1} + B x_t$$

où :

- $\Lambda = diag(\lambda_1, \lambda_2, ..., \lambda_N)$ est une matrice diagonale avec des valeurs complexes.

- B est une matrice de projection des entrées.

Paramétrisation exponentielle pour la stabilité

Afin d'éviter l'explosion des gradients et d'assurer la stabilité du modèle, les valeurs de Λ sont paramétrées exponentiellement :

$$\lambda_j = \exp^{(-exp^{\nu_j} + i\theta_j)}$$

où :

- ν_j contrôle l'amortissement (stabilité).
- θ_j contrôle la fréquence des oscillations.

Normalisation des activations

Le LRU applique une normalisation dynamique sur l'état caché pour éviter l'explosion des activations :

$$h_t = \Lambda h_{t-1} + \gamma \cdot B x_t$$

où \gamma est un facteur d'échelle appris pour garantir une propagation stable des gradients.

Ainsi, le LRU:

- **Supprime les portes de mise à jour et de réinitialisation** pour une structure plus simple.
- **Utilise une récurrence linéaire avec une matrice diagonale dans l'espace complexe**, permettant un calcul plus rapide et stable.
- **Paramètre les transitions d'état de manière exponentielle**, garantissant une **meilleure stabilité et un contrôle précis des gradients**.
- **Ajoute une normalisation dynamique** pour éviter l'explosion ou la disparition des gradients.

II-4- Méthode multiplicative appliquée au GRU, LRU et minGRU

Un RNN standard utilise une relation additive entre l'entrée et l'état caché précédent pour mettre à jour son état caché. Cependant, cette approche présente des limites pour la modélisation des dépendances à long terme et ne permet pas une interaction flexible entre l'entrée et les transitions cachées. L'idée clé du MRNN est d'introduire une **interaction multiplicative** entre l'entrée et la matrice de transition W_{hh} . Concrètement, plutôt que d'avoir une matrice de transition fixe, le MRNN laisse l'entrée x_t influencer la matrice de transition entre les états cachés.

Ainsi, la méthode multiplicative des MRNNs améliore significativement les performances des RNNs classiques pour des tâches complexes comme la modélisation du langage à l'échelle des caractères. Elle permet aux entrées de moduler dynamiquement la transition des états cachés, augmentant ainsi la flexibilité et la puissance prédictive du modèle, tout en nécessitant des techniques d'optimisation avancées pour être entraînée efficacement.

Dans notre expérience, nous appliquerons la méthode multiplicative au GRU, minGRU et LRU.

II-3- Analyse et Evaluation

Dans la suite de notre expérience, nous mènerons des analyses sur le modèle à travers sa capacité à se généraliser sur les données test, à travers la variance des poids des couches en vue de détecter d'éventuelles variations brutes et d'essayer de tirer des conclusions ou de faire des liens avec la qualité des résultats obtenus. Nous tenterons ensuite une approche pour évaluer la confiance du modèle lors de la génération des outputs. Ainsi nous observerons :

- la train accuracy
- la validation accuracy
- la variance des poids par neurones
- la confiance du modèle pour un nombre d'échantillons générés par calcul de l'entropie des logprobs
- les performances pour la correction orthographique automatique par calcul des taux d'erreurs par caractères et par mots à partir de la distance Levenshtein

III - Résultats

III-1- Analyse Exploratoire et Prétraitement

Analyse exploratoire pré et post traitement

De nos analyses, il ressort que le document contient 32.562 mots incluant 6.975 mots uniques.

Observons la distribution de ces mots par phrases.

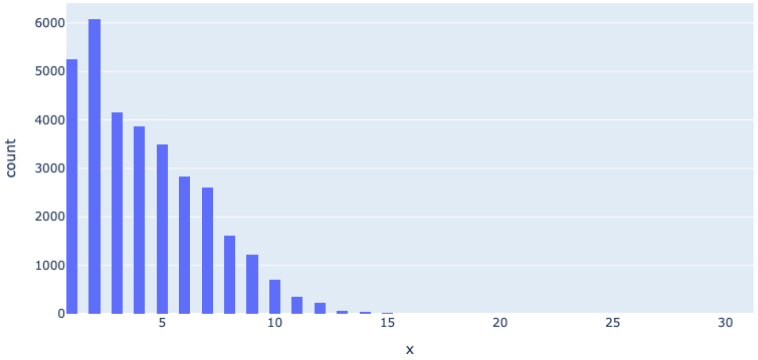

Image 1 : Distribution du nombre de phrase par nombre de mots avant le prétraitement du corpus

Ce graphique présente le nombre de phrases contenant un certain nombre de mots.
Le graphique montre la distribution du nombre de phrases en fonction de leur longueur (en mots). La majorité des phrases sont très courtes, avec un pic autour de 5 mots, ce qui suggère un style concis, typique des poésies.

Image 2 : Top 100 des mots les plus fréquents avant le prétraitement du corpus

Ce graphique présente les mots les plus fréquemment rencontrés dans le document. Comme un peu s'y attendre, ce sont les stopwords qui apparaissent le plus. Un nettoyage s'impose avant tout traitement.
Après le prétraitement, nous obtenons la distribution suivante :

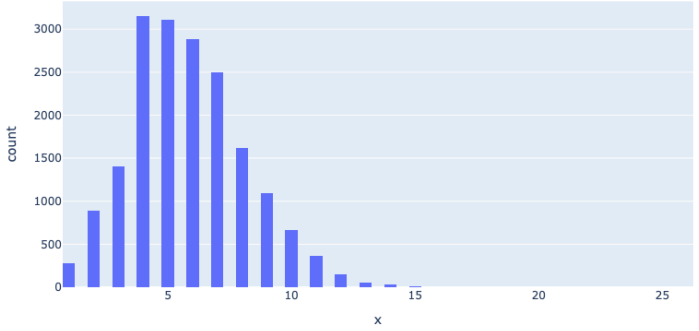

Image 3 : Distribution du nombre de phrase par nombre de mots après le prétraitement du corpus

On observe un changement de la distribution. On peut déduire que les stopwords influençaient la distribution précédente et que leur suppression lors de l'étape du prétraitement a permis l'obtention de cette nouvelle distribution qui reste toujours typique de poésies.

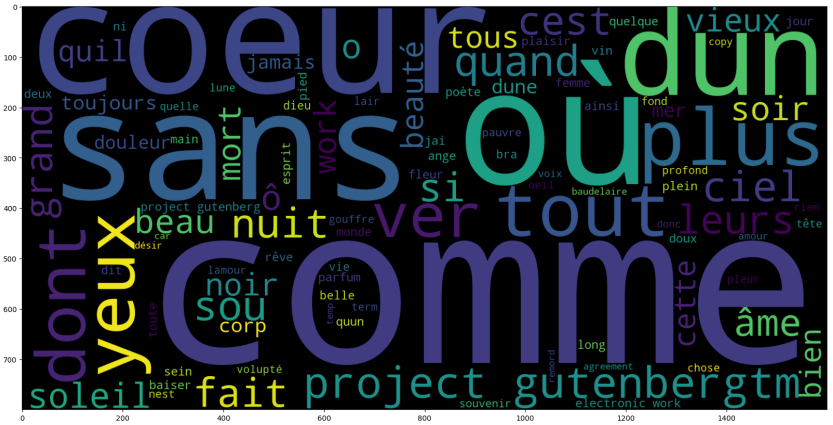

Image 4 : Top 100 des mots les plus fréquents après le prétraitement du corpus

On peut ici constater la réduction de la présence de stopwords.

III-2- Entraînement, Inférence et Évaluation

L'entraînement des modèles a été fait sur 100 epochs, cependant en raison de l'application d'un Early stopping, l'entraînement s'est plusieurs fois arrêté sans atteindre les 100 epochs pour certains modèles.Dans la suite de cette section, nous présenterons les résultats de l'entraînement de quelques modèles ainsi que des échantillons de l'inférence. Nous présenterons également les variations variances des poids de certains de nos modèles

Modèles	Nombres de paramètres	Train accuracy	Test accuracy	Confiance des modèles	Taux d'erreur par caractères	Taux d'erreur par mots
GRU	485 704	69,19%	60,09%	4,2711	63,41%	520%
minGRU	326 472	60,80%	54,86%	4,2711	68,29%	560%
MGRU	104 393	63,63%	53,76%	3,8377	60,98%	500%
MminGRU	49 096	51,08%	48,83%	4,2724	63,41%	520%
MLRU	252 783	62,77%	40,40%	3,8398	75,61%	620%
LRU	248 495	52,32%	39,91%	3,8417	73,17%	600%

Table 1 : Résultats de l'entraînement, de l'inférence et de l'évaluation des modèles explorés

Comme observé, les modèles tels que présentés ont eu des difficultés à atteindre la barre des 90% d'accuracy sur les données de validation. De plus, malgré diverses méthodes de contrôle de l'entraînement et de régularisation des paramètres, certains modèles affichent un overfitting. De

plus, nous remarquons que les modèles font preuve d'une très faible confiance lors de la génération des sorties.

Le GRU reste le modèle le plus performant avec une précision en test de 60,09%, mais son coût en paramètres est élevé (485K). MGRU (104K) réduit considérablement la taille du modèle, mais sa précision chute à 53,76%, et sa confiance à peine variable (3,83 contre 4,27 pour GRU). L'introduction de la méthode multiplicative ne semble pas apporter de gains significatifs et complique l'entraînement. MLRU et LRU sont clairement inefficaces, avec des précisions test autour de 40% et des taux d'erreur très élevés (600-620%). Dans l'ensemble, la réduction du nombre de paramètres entraîne une dégradation notable des performances, et l'approche multiplicative ne compense pas suffisamment cette perte

Afin de comprendre ou à tout le moins de tenter d'expliquer le score obtenu nous avons effectué, observons la distribution de la variance des poids obtenus pour le modèle GRU

Analysons le GRU

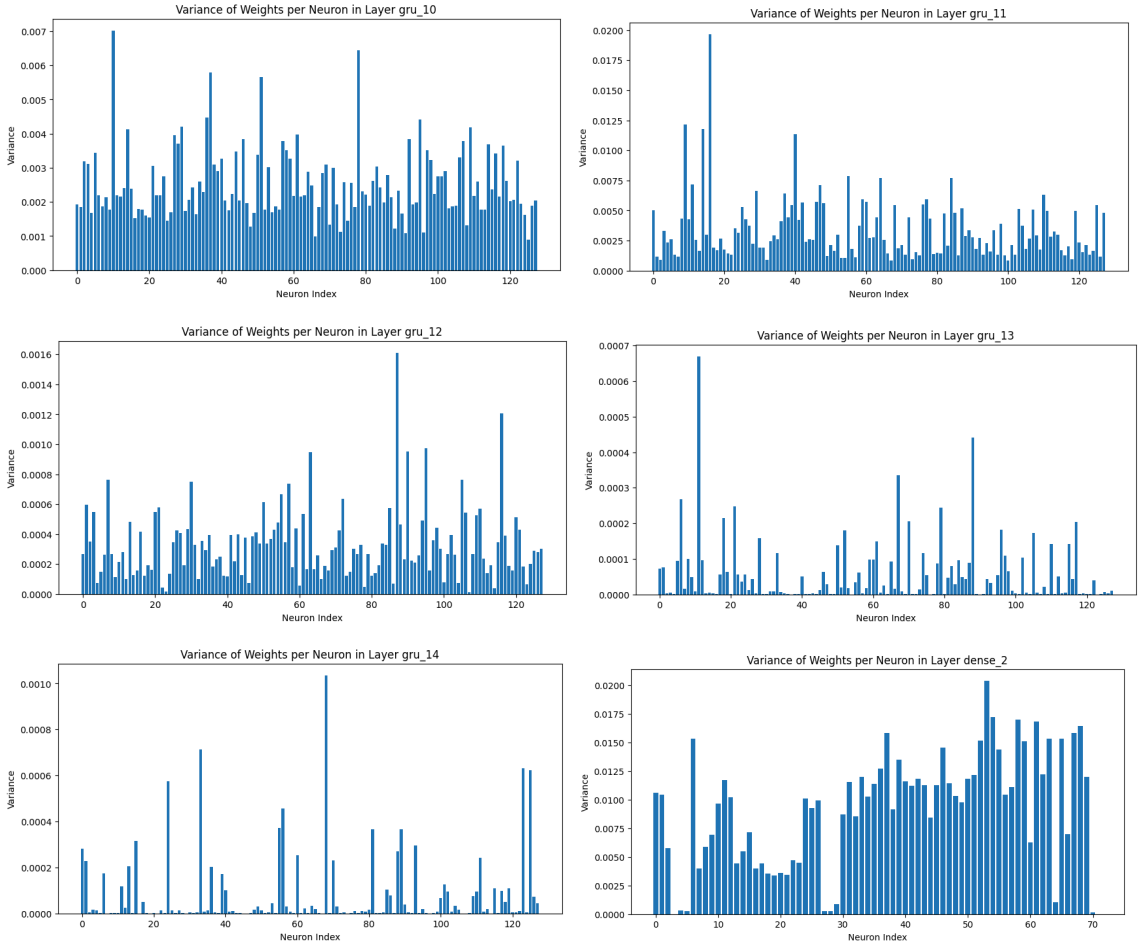

Image 5 : Variance des poids du modèle GRU

Les visualisations de variance des poids du modèle GRU (485K paramètres) pour "Les Fleurs du Mal" révèlent une architecture qui se spécialise progressivement, avec une forte diminution de

variance dans les couches profondes où seuls quelques neurones restent actifs. Cette distribution explique les performances mitigées (60% de précision en test, 69% en entraînement) et surtout le taux d'erreur par mots alarmant de 520%. Le modèle souffre simultanément de surapprentissage et de sous-capacité face à la richesse linguistique baudelairienne, nécessitant éventuellement une régularisation beaucoup plus renforcée et potentiellement l'ajout d'un mécanisme d'attention pour mieux capturer les dépendances complexes propres à la poésie.

Exemples de sorties générées par modèles :

Modèle	Texte généré:
GRU	t gutenberg-tm electronic works in the united states works in any project gutenberg-tm electronic works in any halmon on le jour de notre souvenir de la forme de la vie et du bour et de ces couleurs des deux po
minGRU	, bien qu'il est des leurs repours de la tombe et des démons de l'amour de la parfum, plus en des coups par l'ondre et sans rêves des beaux flambes de vos souvenirs et vaincu, et qui s'étale et jamais le vieux
MminGRU	is ce temps et par le mort infernel se pour le sorte, plaiser de les pour les coeurs des morts et ses riches de le vient de pas d'un temps avec le foument les cares sur le dieu qui me les vites de l'infanté par
MLRU	t faire époqued desseins parain chire beauté notent plus toujours premiamble horre fait voir demp luveur semblable parer reportique sétande comme navital douleur lieux homme ligne mensonge charmant comme femme
LRU	t faire éplus paroi sombre perve hore comme beauté nuit enferaient méterne aprésibre plute car implange vie persoi pois profond ples cour désir troute plint prendeur lenfreuse profond belle derrière ongerre étr
MGRU	author charmant meurtre ainsi comme long inconnent car charmant seulement leurs grandes laisseaux parfum dégard suit mal parfum tout coeur comme rien souvenir parment prassquait charles satin matire noir destin

Table 2 : Exemple de textes générés par les modèles

IV- Discussion et perspectives

L'analyse des résultats met en lumière plusieurs éléments essentiels quant à l'efficacité des modèles explorés.

- Impact de la réduction des paramètres : On observe que la diminution du nombre de paramètres entraîne une perte de précision significative, particulièrement pour minGRU et MminGRU. Bien que l'objectif initial de ces variantes soit une meilleure parallélisation et efficacité, la simplification architecturale se traduit par une dégradation des performances.

- L'approche multiplicative appliquée aux GRU et LRU visait à renforcer la capacité d'apprentissage des dépendances complexes. Cependant, les résultats ne montrent pas d'amélioration significative. Au contraire, cette complexification semble alourdir l'entraînement sans apporter de gains tangibles en test accuracy. Cela suggère que l'ajout d'une interaction multiplicative ne compense pas les limitations inhérentes aux architectures réduites.

- Un autre point préoccupant est la faible confiance affichée par les modèles lors de l'inférence. Un niveau de confiance aussi bas du fait d'une homogénéité des logprobs indique une grande incertitude dans les prédictions générées, ce qui pourrait être lié à une mauvaise généralisation et un surajustement aux données d'entraînement.

- Les taux d'erreur, tant par caractères que par mots, restent élevés pour l'ensemble des modèles, ce qui indique que la capacité de correction automatique est limitée. Cela est particulièrement visible pour MLRU et LRU, où les erreurs explosent malgré des mécanismes censés stabiliser les gradients.

- L'écart entre la précision d'entraînement et celle en test montre que certains modèles, notamment GRU et MGRU, souffrent d'overfitting. Des stratégies comme le dropout, la régularisation L2 ayant déjà été appliquée, un renforcement de ces dernières l'augmentation des données pourraient être envisagées pour atténuer cet effet.

Ces résultats montrent qu'une simple réduction des paramètres, l'application de la méthode multiplicative ou une modification de l'architecture ne garantit pas une amélioration des performances. Pour optimiser ces modèles, plusieurs pistes peuvent être envisagées :

- Intégration d'un mécanisme d'attention pour mieux capturer les dépendances longues et complexes, particulièrement dans des textes poétiques.
- Augmentation de données et techniques de régularisation plus agressives afin d'éviter l'overfitting et d'améliorer la généralisation.
- Expérimentation avec des fonctions d'activation alternatives ou des transformations plus robustes pour limiter la perte d'information lors des mises à jour d'état.

En conclusion, bien que certaines approches explorées ici aient permis de mieux comprendre les limites des architectures récurrentes simplifiées, elles n'ont pas produit d'amélioration substantielle des performances sur la tâche ciblée. Des optimisations plus avancées sont nécessaires pour garantir à la fois l'efficacité computationnelle et la qualité des résultats générés.

Conclusion

En conclusion, ce projet a exploré les performances de trois architectures de réseaux de neurones récurrents (GRU, minGRU et LRU) dans la tâche de génération de texte. Les résultats montrent que la simplification des modèles, comme dans le cas du minGRU et du LRU, entraîne une perte significative de précision, malgré des avantages potentiels en termes de parallélisation et d'efficacité computationnelle. L'application de la méthode multiplicative n'a pas apporté d'amélioration notable, et les modèles ont montré une faible confiance lors de la génération de texte, ce qui suggère une mauvaise généralisation et un surajustement aux données d'entraînement. Pour améliorer les performances, des approches plus avancées, comme l'intégration de mécanismes d'attention et des techniques de régularisation plus robustes, devraient être envisagées. Ces résultats soulignent la complexité de la génération de texte et la nécessité de continuer à explorer des architectures et des méthodes innovantes pour surmonter les limitations actuelles des modèles récurrents.

Les expériences menées dans ce projet ont également révélé des défis liés à l'overfitting et à la généralisation. Par exemple, le GRU, bien qu'étant le modèle le plus performant avec une précision en test de 60,09%, souffre d'un écart important entre la précision d'entraînement (69,19%) et la précision en test, indiquant un surajustement aux données d'entraînement. Les modèles réduits, comme le minGRU et le LRU, ont montré des performances encore plus faibles, avec des précisions en test autour de 40%. Ces résultats suggèrent que la réduction des paramètres, bien qu'utile pour améliorer l'efficacité computationnelle, peut entraîner une perte significative de capacité de modélisation, en particulier pour des tâches complexes comme la génération de texte.

Enfin, les taux d'erreur élevés, tant par caractères que par mots, indiquent que la capacité de correction automatique des modèles est limitée. Cela est particulièrement visible pour les modèles MLRU et LRU, où les erreurs explosent malgré des mécanismes censés stabiliser les gradients. Ces résultats mettent en lumière la nécessité de continuer à explorer des techniques d'optimisation et de régularisation pour améliorer la robustesse et la généralisation des modèles récurrents dans des tâches de génération de texte.

Bibliographie

- Cho, Kyunghyun, et al. "Learning phrase representations using RNN encoder-decoder for statistical machine translation." *arXiv preprint arXiv:1406.1078* (2014)
- Chung, Junyoung, et al. "Empirical evaluation of gated recurrent neural networks on sequence modeling." *arXiv preprint arXiv:1412.3555* (2014)
- Feng, Leo, et al. "Were rnns all we needed?." *arXiv preprint arXiv:2410.01201* (2024)
- Orvieto, Antonio, et al. "Resurrecting recurrent neural networks for long sequences." *International Conference on Machine Learning*. PMLR, 2023.
- Sutskever, Ilya, James Martens, and Geoffrey E. Hinton. "Generating text with recurrent neural networks." *Proceedings of the 28th international conference on machine learning (ICML-11)*. 2011.
- Maupomé, Diego, and Marie-Jean Meurs. "Multiplicative models for recurrent language modeling." *International Conference on Computational Linguistics and Intelligent Text Processing*. Cham: Springer Nature Switzerland, 2019.
- Liu, Runze, et al. "Can 1B LLM Surpass 405B LLM? Rethinking Compute-Optimal Test-Time Scaling." *arXiv preprint arXiv:2502.06703* (2025).