

RAPPORT



Génération des citations avec GRU

RÉALISÉE PAR

Ezzagrani Habiba

Table des matières :

I - Introduction :	2
1. Présentation générale du projet :	2
2. Objectifs du projet :	2
3. Contexte du projet :	2
II - Deep Learning :	3
1. Architecture des Réseaux Neuronaux Profonds :	3
III - Choix de l'algorithmes :	4
1. RNN :	4
2. GRU :	5
IV - Méthodologie :	6
1. Collecte et Préparation des Données :	6
2. Prétraitement des Données :	7
3. Construction du Modèle GRU :	7
4. Entraînement du Modèle :	7
5. Évaluation du Modèle :	7
6. Génération de Citations :	7
V - Ensemble des données :	8
1. Collecte et Préparation des Données :	8
VI - Implémentation :	10
1. Prétraitement des données :	10
2. Construction du Modèle GRU :	11
3. Entraînement du Modèle :	13
VII - Interprétation & Résultat :	14
1. Résultat :	14
1.1. Évaluation du Modèle :	14
1.2. Génération de Citations :	15
2. Interprétation :	17
VIII - Environnement du travail :	17
IX - Conclusion :	18
X - Défis :	18
XI - Suggestions pour les Prochaines Étapes :	20
Annexes :	21

I - Introduction :

1. Présentation générale du projet :

Les citations, véritables condensés de sagesse, d'expérience et de connaissances accumulées au fil du temps, jouent un rôle crucial dans la transmission de l'information et l'inspiration. Elles incarnent la quintessence d'idées souvent complexes en quelques mots, offrant ainsi des perspectives uniques sur des concepts variés. De ce fait, la génération de citations représente une étape novatrice dans l'univers de la communication et de l'intelligence artificielle. Les citations, bien plus qu'une simple série de mots, ont le pouvoir de captiver l'attention, d'évoquer des émotions et de susciter la réflexion. Elles sont les joyaux littéraires qui transcendent les frontières linguistiques et culturelles, diffusant des idées à travers le temps et l'espace. Leur pertinence s'étend à de nombreux domaines, de l'éducation à la littérature, en passant par la motivation personnelle et le discours public. La génération automatisée de citations représente une convergence remarquable entre l'ingéniosité humaine et la puissance de l'intelligence artificielle. Elle vise à combler un besoin croissant dans un monde où l'accès à l'information est rapide, mais où la création de contenus originaux et inspirants demeure un défi. La pertinence de cette génération réside dans sa capacité à produire des énoncés uniques et percutants, offrant ainsi une source infinie d'inspiration et d'informations.

2. Objectifs du projet :

L'objectif principal de ce projet de génération de citations est de créer un modèle capable de produire des énoncés pertinents, authentiques et significatifs. J'aspire à développer un modèle novateur capable de générer des citations dans différents domaines, répondant aux besoins variés des utilisateurs, qu'ils soient éducatifs, professionnels ou personnels. Cette initiative vise à explorer les frontières de la création automatisée de contenu tout en préservant l'essence et la qualité des citations générées.

3. Contexte du projet :

Dans ce projet, nous explorerons diverses méthodes de génération de langage naturel, telles que les modèles de langage basés sur des réseaux neuronaux, les approches de traitement du langage naturel (NLP) et l'apprentissage automatique. Nous recourrons à des ensembles de données variés comprenant des citations célèbres, des ouvrages littéraires et des discours historiques pour entraîner mon modèle. L'utilisation de techniques de prétraitement des données des modèles joue un rôle clé dans la qualité et la diversité des citations générées.

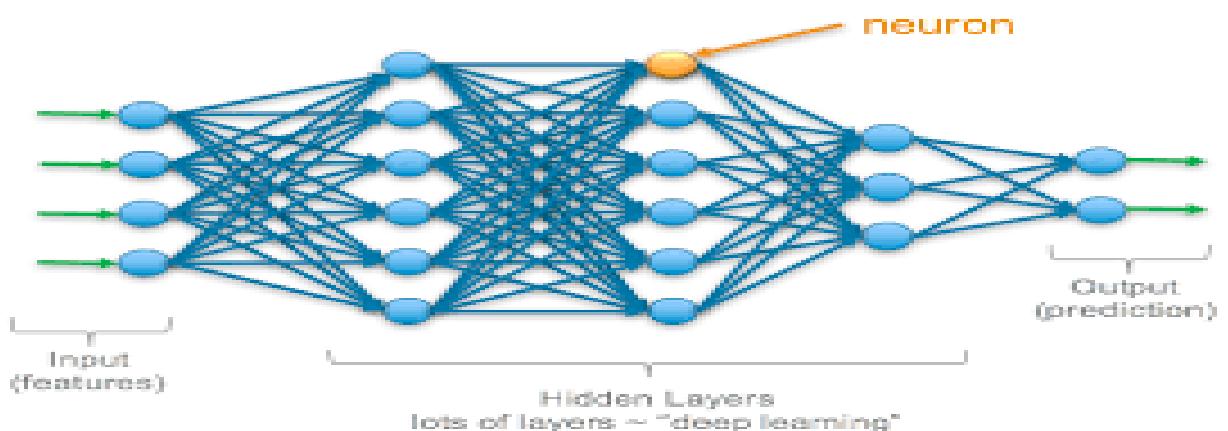
II - Deep Learning :

Le deep learning est une branche de l'intelligence artificielle (IA) qui repose sur l'utilisation de **réseaux neuronaux profonds**, composés de plusieurs couches de neurones. Contrairement aux réseaux de neurones traditionnels, le deep learning exploite des architectures beaucoup plus complexes, nécessitant souvent de grandes quantités de données pour être efficaces. Cette méthode permet aux algorithmes d'apprendre des représentations de données hiérarchiques, en identifiant des caractéristiques de plus en plus abstraites à mesure que l'on progresse dans les différentes couches du réseau. Le deep learning a permis d'obtenir des avancées significatives dans des domaines tels que la vision par ordinateur, le traitement du langage naturel, la reconnaissance vocale et bien d'autres, en réalisant des tâches complexes avec des performances remarquables.

- **Réseaux neuronaux** : Les réseaux neuronaux sont des modèles informatiques inspirés du fonctionnement du cerveau humain. Ils sont composés de plusieurs unités interconnectées appelées neurones artificiels. Chaque neurone reçoit des entrées, effectue des calculs sur ces entrées en utilisant des poids spécifiques, puis transmet une sortie traitée vers d'autres neurones. Ces réseaux sont organisés en couches, comprenant généralement une couche d'entrée, une ou plusieurs couches cachées et une couche de sortie. En utilisant des techniques d'apprentissage, les réseaux neuronaux sont capables d'apprendre à partir de données pour effectuer des tâches telles que la classification, la prédiction ou **la génération de contenu**.

1. Architecture des Réseaux Neuronaux Profonds :

- *Réseaux de neurones convolutifs (CNN)* : Ils sont souvent utilisés pour le traitement d'images en raison de leur capacité à reconnaître des motifs spatiaux.
- *Réseaux de neurones récurrents (RNN)* : Ils sont conçus pour traiter des données séquentielles comme du texte, de la parole ou des séries temporelles.
- *Réseaux neuronaux générateurs adverses (GAN)* : Ces réseaux sont utilisés pour la génération de données réalistes, souvent appliquée dans la création d'images, de textes et de sons synthétiques.



III - Choix de l'algorithmes :

Dans ce projet, j'ai utilisé principalement les **GRU**, une variante de réseaux de neurones récurrents (RNN).

1. RNN :

Les Réseaux de Neurones Récurrents (RNN) sont une classe de réseaux neuronaux conçus pour modéliser des séquences de données, où la séquence en elle-même est importante pour la tâche à accomplir. Contrairement aux réseaux neuronaux classiques, les RNN ont des connexions récurrentes qui leur permettent de maintenir une mémoire interne, ce qui leur permet de prendre en compte les informations précédentes tout en traitant de nouvelles entrées.

➤ Structure des RNN :

- *Connexions récurrentes* : Chaque couche de neurones renvoie ses sorties précédentes comme entrées à la couche suivante pour tenir compte de l'historique des séquences.
- *Cellule RNN* : Elle effectue des opérations récurrentes sur chaque élément de la séquence en utilisant une fonction d'activation non linéaire (comme la tangente hyperbolique - tanh) pour gérer les états cachés.
- *Mémoire à court terme* : Les RNN sont capables de conserver une mémoire interne à court terme, ce qui leur permet de conserver des informations importantes sur les séquences qu'ils traitent.

➤ Variantes des RNN :

- *Long Short-Term Memory (LSTM)* : Les LSTM ont été développés pour remédier au problème du vanishing gradient (gradient qui s'estompe) dans les RNN traditionnels. Ils utilisent des unités de mémoire spéciales appelées cellules LSTM avec des portes (porte d'oubli, porte d'entrée, porte de sortie) pour contrôler le flux d'informations à travers la cellule, permettant de conserver des informations sur de plus longues séquences. Les LSTM sont capables de gérer des dépendances temporelles à long terme et sont efficaces pour capturer des relations à long terme dans les données séquentielles.
- *Gated Recurrent Unit (GRU)* : Les GRU sont une variante simplifiée des LSTM, avec moins de portes et une structure plus simple. Ils utilisent une porte de mise à jour pour réguler le flux d'informations et une porte de réinitialisation pour contrôler la mémoire des cellules, ce qui leur permet de conserver des informations importantes sur les séquences.

2. GRU :

Les Gated Recurrent Units (GRU) sont une variante de Réseaux de Neurones Récurrents (RNN) conçue pour améliorer les performances des RNN traditionnels tout en simplifiant leur architecture. Proposés en 2014 par Cho et al., les GRU visent à résoudre certains des problèmes des RNN classiques, comme le vanishing gradient, tout en étant plus simples que les LSTM. Les GRU représentent une avancée significative dans la modélisation des séquences de données grâce à leur capacité à capturer des dépendances temporelles complexes tout en étant moins complexes que les LSTM. Leur simplicité, leur efficacité et leur adaptabilité en font une architecture populaire pour une grande variété de tâches de traitement de séquences.

➤ Fonctionnement des GRU :

- **Structure** : Les GRU sont composés de cellules similaires à celles des RNN, mais avec un mécanisme simplifié. Ils disposent d'une porte de mise à jour (update gate) et d'une porte de réinitialisation (reset gate).
- **Portes dans les GRU** : La porte de mise à jour contrôle la quantité d'informations à conserver de l'étape précédente, tandis que la porte de réinitialisation détermine quelle partie de l'information précédente sera oubliée.
- **Avantages des GRU :**
 - ➔ *Moins de paramètres* : Comparés aux LSTM, les GRU ont moins de paramètres à entraîner, ce qui les rend souvent plus rapides à entraîner et moins susceptibles de surapprentissage.
 - ➔ *Performances* : Ils sont efficaces pour capturer des dépendances temporelles à long terme dans les données séquentielles, tout en évitant certains des problèmes de vanishing gradient des RNN classiques.

➤ Applications des GRU :

- *Traitemennt du langage naturel* : Ils sont utilisés pour des tâches telles que la génération de texte, la traduction automatique et la modélisation de langage.
- *Prévision de séries temporelles* : Les GRU sont employés pour la prédiction de séries temporelles dans des domaines tels que la finance, la météorologie et la santé.

Les GRU sont configurables en ajustant ces différents paramètres et en modifiant l'architecture du réseau pour s'adapter aux besoins spécifiques de la tâche à

accomplir. Expérimenter avec ces hyperparamètres peut permettre d'optimiser les performances du modèle pour différentes tâches de traitement de séquences.

➤ Architecture des GRU :

- *Porte de Réinitialisation (Reset Gate)* : Cette porte contrôle quelles informations de l'étape précédente doivent être oubliées ou ignorées.
- *Porte de Mise à Jour (Update Gate)* : Elle détermine quelle proportion des informations actuelles et précédentes doit être combinée pour former l'état caché.
- *État caché (Hidden State)* : Représente la mémoire à court terme du réseau, contenant les informations importantes pour la prédiction à l'étape suivante.

➤ Hyperparamètres des GRU :

- *Single Layer (Une seule couche)* : Une seule couche de GRU peut être utilisée pour des tâches plus simples ou avec moins de données.
- *Nombre d'unités GRU* : Détermine la taille de la représentation latente et la capacité du modèle à capturer les dépendances séquentielles.
- *Taux d'Apprentissage (Learning Rate)* : Contrôle la vitesse d'apprentissage du modèle : Un taux d'apprentissage plus élevé peut accélérer la convergence, mais peut également entraîner une instabilité ou des difficultés à converger.
- *Dropout* :- Régularisation du modèle - Le dropout est utilisé pour réduire le surapprentissage en désactivant aléatoirement des unités pendant l'entraînement.
- *Batch Size* : Nombre d'exemples d'entraînement dans un lot : Influence la stabilité de l'entraînement et la mise à jour des poids du réseau.

IV - Méthodologie :

1. Collecte et Préparation des Données :

- a. **Collecte des Citations** : Rassemblement d'un ensemble de données variées et pertinentes comprenant des citations provenant de différentes sources telles que des livres, des articles, des discours, etc.

- b. **Nettoyage des Données** : Élimination des doublons, correction des erreurs typographiques, suppression des citations incohérentes et formatage homogène pour préparer les données à l'entraînement.

2. Prétraitement des Données :

- a. **Tokenization** : Conversion des citations en séquences de tokens ou de mots pour l'entrée dans le modèle.
- b. **Padding** : Uniformisation de la longueur des séquences en ajoutant des zéros ou d'autres marqueurs pour créer des lots de données homogènes.

3. Construction du Modèle GRU :

- a. **Configuration de l'architecture** : Définition des paramètres du modèle GRU tels que le nombre de couches, la taille de la couche cachée, etc.
- b. **Mise en place des Couches de GRU** : Ajout des couches GRU dans la séquence du modèle, en prenant en compte l'architecture spécifique nécessaire pour la génération de citations.

4. Entraînement du Modèle :

- a. **Division des Données** : Séparation de l'ensemble de données en ensembles d'entraînement, de validation et, éventuellement, de test.
- b. **Choix des Hyperparamètres** : Réglage du taux d'apprentissage, du nombre d'époques, du dropout, etc.
- c. **Entraînement du Modèle** : Utilisation des données d'entraînement pour ajuster les poids du modèle GRU en minimisant la fonction de perte (loss function).

5. Évaluation du Modèle :

- a. **Validation du Modèle** : Évaluation des performances du modèle sur l'ensemble de validation pour vérifier sa capacité à générer des citations cohérentes et pertinentes.
- b. **Mesure de la Qualité des Citations** : Utilisation de métriques appropriées pour évaluer la qualité des citations générées, telles que la diversité, la cohérence et la nouveauté.

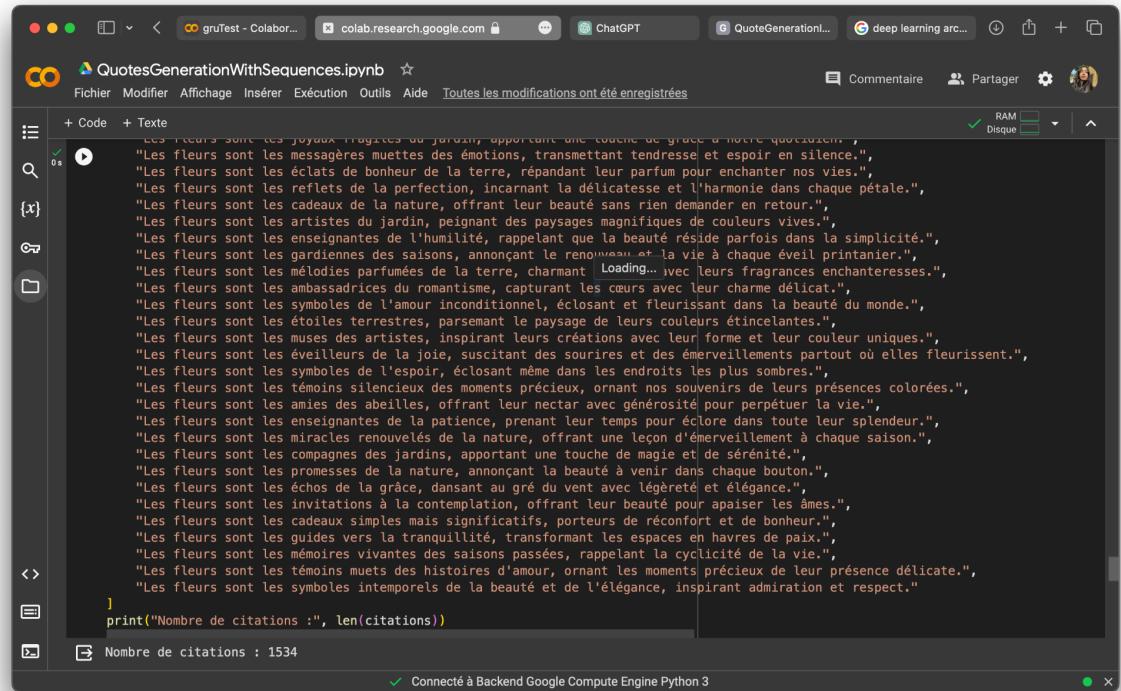
6. Génération de Citations :

- a. **Utilisation du Modèle Entraîné** : Application du modèle entraîné pour générer de nouvelles citations à partir de déclencheurs ou de contextes spécifiques.
- b. **Validation manuelle** : Vérification manuelle pour évaluer la qualité et la pertinence des citations générées par le modèle.

V - Ensemble des données :

1. Collecte et Préparation des Données :

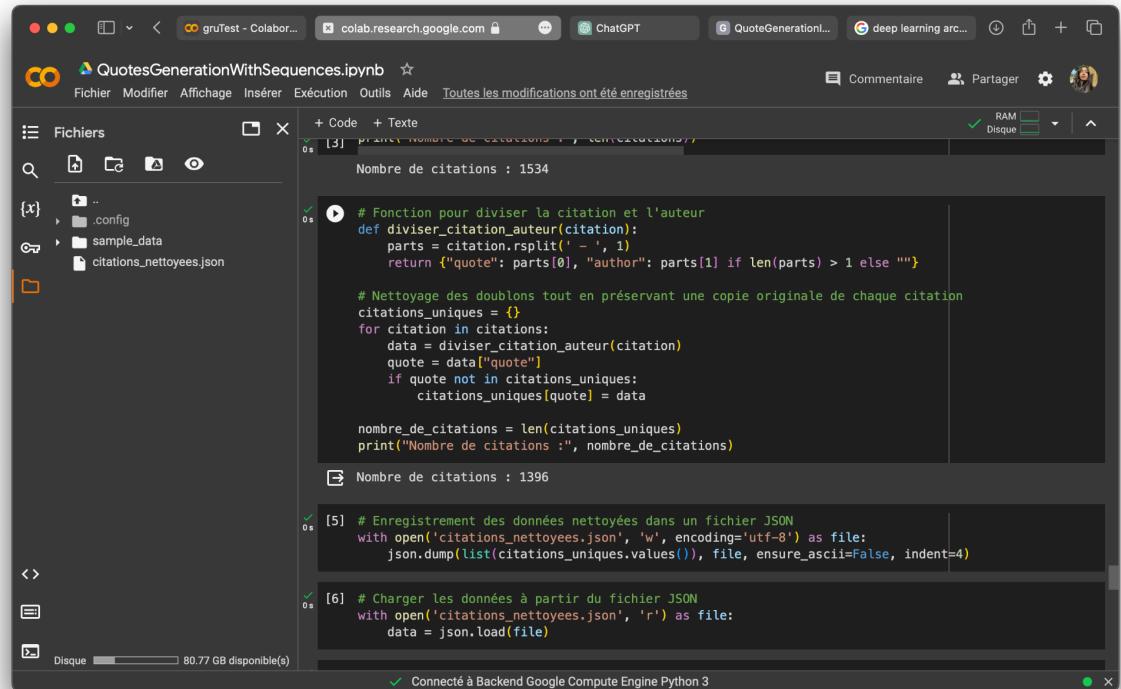
a. Collecte des données : La collecte des données consiste à rassembler des citations à partir de diverses sources. Cela peut inclure des livres, des articles, des sites web, des bases de données spécialisées, des archives en ligne, etc. L'objectif est de constituer un ensemble de données diversifié, couvrant un large éventail de styles, de thèmes et d'auteurs pour enrichir la variété des citations. Il est important de sélectionner des sources fiables et de vérifier l'authenticité des citations pour maintenir la qualité de l'ensemble de données.



```
+ Code + Texte
[0] 0s
Les fleurs sont les joyaux fragiles du jardin, apportant une touche de grâce à notre quotidien., "Les fleurs sont les messagères muettes des émotions, transmettant tendresse et espoir en silence.", "Les fleurs sont les éclats de bonheur de la terre, répandant leur parfum pour enchanter nos vies.", "Les fleurs sont les reflets de la perfection, incarnant la délicatesse et l'harmonie dans chaque pétale.", "Les fleurs sont les cadeaux de la nature, offrant leur beauté sans rien demander en retour.", "Les fleurs sont les artistes du jardin, peignant des paysages magnifiques de couleurs vives.", "Les fleurs sont les enseignantes de l'humilité, rappelant que la beauté réside parfois dans la simplicité.", "Les fleurs sont les gardiennes des saisons, annonçant le renouveau et la vie à chaque éveil printanier.", "Les fleurs sont les mélodies parfumées de la terre, charmant Loading... avec leurs fragrances enchanteresses.", "Les fleurs sont les ambassadrices du romantisme, capturant les cœurs avec leur charme délicat.", "Les fleurs sont les symboles de l'amour inconditionnel, éclosant et fleurissant dans la beauté du monde.", "Les fleurs sont les étoiles terrestres, parstenant le paysage de leurs couleurs étincelantes.", "Les fleurs sont les muses des artistes, inspirant leurs créations avec leur forme et leur couleur uniques.", "Les fleurs sont les éveilleurs de la joie, suscitant des sourires et des émerveillements partout où elles fleurissent.", "Les fleurs sont les symboles de l'espoir, éclosant même dans les endroits les plus sombres.", "Les fleurs sont les témoins silencieux des moments précieux, ornant nos souvenirs de leurs présences colorées.", "Les fleurs sont les amies des abeilles, offrant leur nectar avec générosité pour perpétuer la vie.", "Les fleurs sont les enseignantes de la patience, prenant leur temps pour éclore dans toute leur splendeur.", "Les fleurs sont les miracles renouvelés de la nature, offrant une leçon d'émerveillement à chaque saison.", "Les fleurs sont les compagnes des jardins, apportant une touche de magie et de sérénité.", "Les fleurs sont les promesses de la nature, annonçant la beauté à venir dans chaque bouton.", "Les fleurs sont les échos de la grâce, dansant au gré du vent avec légèreté et élégance.", "Les fleurs sont les invitations à la contemplation, offrant leur beauté pour apaiser les âmes.", "Les fleurs sont les cadeaux simples mais significatifs, porteurs de réconfort et de bonheur.", "Les fleurs sont les guides vers la tranquillité, transformant les espaces en havres de paix.", "Les fleurs sont les mémoires vivantes des saisons passées, rappelant la cyclicité de la vie.", "Les fleurs sont les témoins muets des histoires d'amour, ornant les moments précieux de leur présence délicate.", "Les fleurs sont les symboles intemporels de la beauté et de l'élegance, inspirant admiration et respect."
]
print("Nombre de citations :", len(citations))

Nombre de citations : 1534
```

b. Nettoyage des données : Le nettoyage des données est une étape cruciale pour garantir la qualité des données utilisées. Cela implique plusieurs processus tels que l'élimination des doublons, la correction des erreurs typographiques, la suppression des citations incohérentes ou non pertinentes, et la normalisation du format des citations pour une uniformité. Cette étape vise à purger l'ensemble de données de tout bruit ou de toute irrégularité afin de créer une base de données propre et cohérente pour l'entraînement du modèle.



The screenshot shows a Google Colab notebook titled "QuotesGenerationWithSequences.ipynb". The code cell [3] contains Python code to clean quotes from a JSON file. It defines a function to split quotes and authors, then iterates through the data to remove duplicates while keeping one copy per quote. It prints the count of unique quotes and saves the cleaned data to a JSON file. The output shows the count reduced from 1534 to 1396. Cell [5] shows the JSON dump command, and cell [6] shows the JSON load command. A status bar at the bottom indicates "Connected to Backend Google Compute Engine Python 3".

```

Nombre de citations : 1534
# Fonction pour diviser la citation et l'auteur
def diviser_citation_auteur(citation):
    parts = citation.rsplit(' - ', 1)
    return {'quote': parts[0], 'author': parts[1] if len(parts) > 1 else ""}

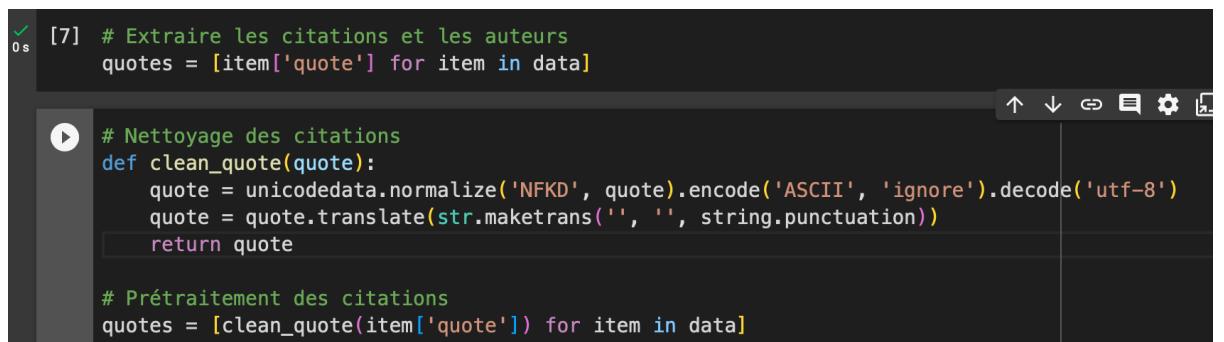
# Nettoyage des doublons tout en préservant une copie originale de chaque citation
citations_uniques = {}
for citation in citations:
    data = diviser_citation_auteur(citation)
    quote = data["quote"]
    if quote not in citations_uniques:
        citations_uniques[quote] = data

nombre_de_citations = len(citations_uniques)
print("Nombre de citations :", nombre_de_citations)

# Enregistrement des données nettoyées dans un fichier JSON
with open('citations_nettoyees.json', 'w', encoding='utf-8') as file:
    json.dump(list(citations_uniques.values()), file, ensure_ascii=False, indent=4)

# Charger les données à partir du fichier JSON
with open('citations_nettoyees.json', 'r') as file:
    data = json.load(file)

```



The screenshot shows a Jupyter Notebook cell with Python code for quote cleaning. It extracts quotes and authors from a JSON file, defines a function to clean quotes by normalizing them and removing punctuation, and then applies this function to all quotes in the data. The code uses the unicodedata module for normalization and the string module for punctuation removal.

```

# Extraire les citations et les auteurs
quotes = [item['quote'] for item in data]

# Nettoyage des citations
def clean_quote(quote):
    quote = unicodedata.normalize('NFKD', quote).encode('ASCII', 'ignore').decode('utf-8')
    quote = quote.translate(str.maketrans('', '', string.punctuation))
    return quote

# Prétraitement des citations
quotes = [clean_quote(item['quote']) for item in data]

```

VI - Implémentation :

1. Prétraitement des données :

c. Tokenization :

La tokenization est le processus de conversion d'une séquence de texte en une séquence de tokens ou de sous-unités, telles que des mots, des caractères ou des sous-mots. Le but principal est de découper le texte en unités plus petites et significatives pour faciliter le traitement par un modèle d'apprentissage automatique.

- *La tokenization par séquence* est un processus dans lequel des séquences de texte sont divisées en unités plus petites, telles que des séquences de mots, de caractères ou de sous-mots, plutôt que d'isoler chaque mot individuellement. Contrairement à la tokenization habituelle par mots, où chaque mot est un token distinct, la tokenization par séquence segmente le texte en séquences continues, ce qui peut être utile pour capturer des informations à un niveau plus élevé que le mot individuel.
- *Exemple de Tokenization par Séquence* : Prenons l'exemple d'une phrase : "La vie est belle."
 - *Tokenization par Mots* : ["La", "vie", "est", "belle"] (Chaque mot est un token distinct.)
 - *Tokenization par Séquence* : ["La vie", "vie est", "est belle"] (Les tokens représentent des paires ou des triplets de mots contigus.)

Dans ce cas, la tokenization par séquence crée des tokens qui sont des séquences continues de mots. Cela permet au modèle d'apprentissage automatique de capturer des relations plus complexes et des nuances qui pourraient être perdues lors de la tokenization par mots individuels.

d. Padding :

Le padding consiste à ajouter des marqueurs spéciaux ou des zéros à des séquences de longueurs variables pour uniformiser leur longueur. Dans le contexte des modèles de séquences comme les réseaux neuronaux récurrents (RNN), les séquences doivent avoir une longueur fixe pour être traitées efficacement en lot (batch).

- *Remplissage de Zéros* : Ajout de zéros à la fin des séquences pour les rendre de la même longueur que la plus longue séquence de l'ensemble de données.

2. Construction du Modèle GRU :

- a. **Configuration de l'architecture manuellement:** Définition des paramètres du modèle GRU tels que le nombre de couches, la taille de la couche cachée, etc.

```
[16] # Définir les valeurs par défaut pour les hyperparamètres
default_units = 10
default_activation = 'tanh'
default_learning_rate = 1e-4

def build_model(num_words, max_length, units=default_units, activation=default_activation, learning_rate=default_learning_rate):
    model = keras.Sequential()
    model.add(keras.layers.Embedding(num_words, 100, input_length=max_length))
    model.add(keras.layers.GRU(units=units, activation=activation, return_sequences=False))
    model.add(Dropout(rate=0.2)) # Ajouter une couche Dropout après GRU
    model.add(keras.layers.Dense(num_words, activation='softmax'))
    model.compile(optimizer=keras.optimizers.Adam(learning_rate),
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
    return model

best_model = build_model(num_words, max_length=max_sequence_length - 1)
```

- #### b. Configuration de l'architecture en utilisant RandomSearch:

- La technique de recherche aléatoire (Random Search) est une méthode d'optimisation des hyperparamètres utilisée dans l'entraînement de modèles de machine learning, y compris ceux développés avec Keras. Cette méthode est utilisée pour trouver les

combinaisons d'hyper paramètres qui maximisent les performances d'un modèle.

■ *Fonctionnement de la Random Search :*

1. *Définition de l'Espace des Hyperparamètres* : Pour chaque hyperparamètre du modèle (comme le taux d'apprentissage, la taille des couches, le dropout, etc.), un espace de recherche est défini. Par exemple, un taux d'apprentissage peut être choisi entre 0.001 et 0.1.
2. *Échantillonnage aléatoire* : La technique de recherche aléatoire sélectionne aléatoirement des combinaisons de hyperparamètres dans cet espace défini.
3. *Entraînement et Évaluation* : Chaque combinaison des hyperparamètres est utilisée pour entraîner un modèle sur un ensemble d'entraînement et évaluée sur un ensemble de validation pour mesurer ses performances.
4. *Choix du Meilleur Modèle* : Après avoir évalué plusieurs combinaisons d'hyper paramètres, la combinaison qui produit le modèle avec les meilleures performances est sélectionnée.

```
def build_model(hp):
    model = keras.Sequential()
    model.add(keras.layers.Embedding(num_words, 100, input_length=max_length - 1))
    model.add(keras.layers.GRU(units=hp.Int('units', min_value=1, max_value=50, step=10),
                               activation='tanh',
                               return_sequences=False))
    # Ajout d'une couche Dropout pour la régularisation
    model.add(Dropout(rate=0.2)) # Vous pouvez ajuster le taux de dropout selon vos besoins
    model.add(keras.layers.Dense(num_words, activation='softmax'))
    model.compile(optimizer=keras.optimizers.Adam(hp.Choice('learning_rate', values=[1e-2, 1e-3, 1e-4])),
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
    return model
```

```
▶ tuner = RandomSearch(
        build_model,
        objective='val_accuracy',
        max_trials=10,
        executions_per_trial=2,
        directory='my_dir',
        project_name='gru_tuning'
    )

    tuner.search(X_train, y_train, epochs=15, validation_data=(X_val, y_val))

    best_hps = tuner.get_best_hyperparameters(num_trials=1)[0]
    best_activation = best_hps.get('activation')
    best_learning_rate = best_hps.get('learning_rate')
    best_units = best_hps.get('units')

    print(f"Meilleure fonction d'activation : {best_activation}")
    print(f"Meilleur taux d'apprentissage : {best_learning_rate}")
    print(f"Meilleur nombre d'unités GRU : {best_units}")

    ⏎ Reloading Tuner from my_dir/gru_tuning/tuner0.json
    Meilleure fonction d'activation : tanh
    Meilleur taux d'apprentissage : 0.001
    Meilleur nombre d'unités GRU : 140
```

3. Entraînement du Modèle :

- a. **Division des Données** : Séparation de l'ensemble de données en ensembles d'entraînement, de validation et, éventuellement, de test.

```
✓ 0s [10] # Création des features et labels
    xs, labels = padded_sequences[:, :-1], padded_sequences[:, -1]
    num_words = len(tokenizer.word_index) + 1 # +1 pour inclure le padding
    ys = to_categorical(labels, num_classes=num_words)

✓ 0s [11] # Division des données
    X_train, X_test, y_train, y_test = train_test_split(xs, ys, test_size=0.2, random_state=42)
    X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.25, random_state=42)

✓ 0s ⏎ print(f"Nombre d'échantillons d'entraînement : {len(X_train)},{len(y_train)}")
    print(f"Nombre d'échantillons de test : {len(X_test)},{len(y_test)}")
    print(f"Nombre d'échantillons de validation : {len(X_val)},{len(y_val)}")

    ⏎ Nombre d'échantillons d'entraînement : 837,837
    Nombre d'échantillons de test : 280,280
    Nombre d'échantillons de validation : 279,279
```

- b. **Définition de early stopping** : Early stopping est une technique utilisée dans l'entraînement des modèles de machine learning pour éviter le surapprentissage (overfitting) en surveillant la performance du modèle sur un ensemble de validation pendant l'entraînement et en arrêtant l'apprentissage lorsque les performances cessent de s'améliorer. Lors de l'entraînement d'un modèle, une partie des données est généralement réservée à la validation pour évaluer les performances du modèle à chaque époque. L'early stopping se base sur ces performances de validation pour prendre des décisions d'arrêt anticipé de l'apprentissage.

```
✓ 0s [28] # Définition de l'arrêt précoce
    early_stopping = EarlyStopping(monitor='val_loss', patience=3)
```

- c. **Entraînement du Modèle** : Utilisation des données d'entraînement pour ajuster les poids du modèle GRU en minimisant la fonction de perte (loss function).

```

history=best_model.fit(X_train, y_train, epochs=50,batch_size=34, validation_data=(X_val, y_val),callbacks=[early_stopping])
Epoch 1/50
25/25 [=====] - 9s 104ms/step - loss: 7.2301 - accuracy: 0.0526 - val_loss: 7.2225 - val_accuracy: 1.0000
Epoch 2/50
25/25 [=====] - 1s 57ms/step - loss: 7.2151 - accuracy: 0.7790 - val_loss: 7.2059 - val_accuracy: 1.0000
Epoch 3/50
25/25 [=====] - 1s 49ms/step - loss: 7.1966 - accuracy: 0.9988 - val_loss: 7.1847 - val_accuracy: 1.0000
Epoch 4/50
25/25 [=====] - 1s 41ms/step - loss: 7.1718 - accuracy: 1.0000 - val_loss: 7.1571 - val_accuracy: 1.0000
Epoch 5/50
25/25 [=====] - 1s 47ms/step - loss: 7.1401 - accuracy: 1.0000 - val_loss: 7.1211 - val_accuracy: 1.0000
Epoch 6/50
25/25 [=====] - 1s 44ms/step - loss: 7.0992 - accuracy: 1.0000 - val_loss: 7.0746 - val_accuracy: 1.0000
Epoch 7/50
25/25 [=====] - 1s 40ms/step - loss: 7.0468 - accuracy: 1.0000 - val_loss: 7.0150 - val_accuracy: 1.0000
Epoch 8/50
25/25 [=====] - 1s 26ms/step - loss: 6.9806 - accuracy: 1.0000 - val_loss: 6.9395 - val_accuracy: 1.0000
Epoch 9/50
25/25 [=====] - 1s 21ms/step - loss: 6.8958 - accuracy: 1.0000 - val_loss: 6.8449 - val_accuracy: 1.0000
Epoch 10/50
25/25 [=====] - 1s 21ms/step - loss: 6.7889 - accuracy: 1.0000 - val_loss: 6.7298 - val_accuracy: 1.0000
Epoch 11/50
25/25 [=====] - 1s 21ms/step - loss: 6.6736 - accuracy: 1.0000 - val_loss: 6.5967 - val_accuracy: 1.0000
Epoch 12/50
25/25 [=====] - 0s 20ms/step - loss: 6.5328 - accuracy: 1.0000 - val_loss: 6.4504 - val_accuracy: 1.0000
Epoch 13/50
25/25 [=====] - 1s 22ms/step - loss: 6.3894 - accuracy: 1.0000 - val_loss: 6.2994 - val_accuracy: 1.0000
Epoch 14/50
25/25 [=====] - 1s 22ms/step - loss: 6.2242 - accuracy: 1.0000 - val_loss: 6.1499 - val_accuracy: 1.0000
Epoch 15/50
25/25 [=====] - 1s 23ms/step - loss: 6.0830 - accuracy: 1.0000 - val_loss: 6.0076 - val_accuracy: 1.0000
Epoch 16/50
25/25 [=====] - 1s 21ms/step - loss: 5.9433 - accuracy: 1.0000 - val_loss: 5.8750 - val_accuracy: 1.0000
Epoch 17/50

```

✓ Connecté à Backend Google Compute Engine Python 3

VII - Interprétation & Résultat :

1. Résultat :

1.1. Évaluation du Modèle :

1.1.1. Validation du Modèle : Évaluation des performances du modèle sur l'ensemble de test pour vérifier sa capacité à générer des citations cohérentes et pertinentes.

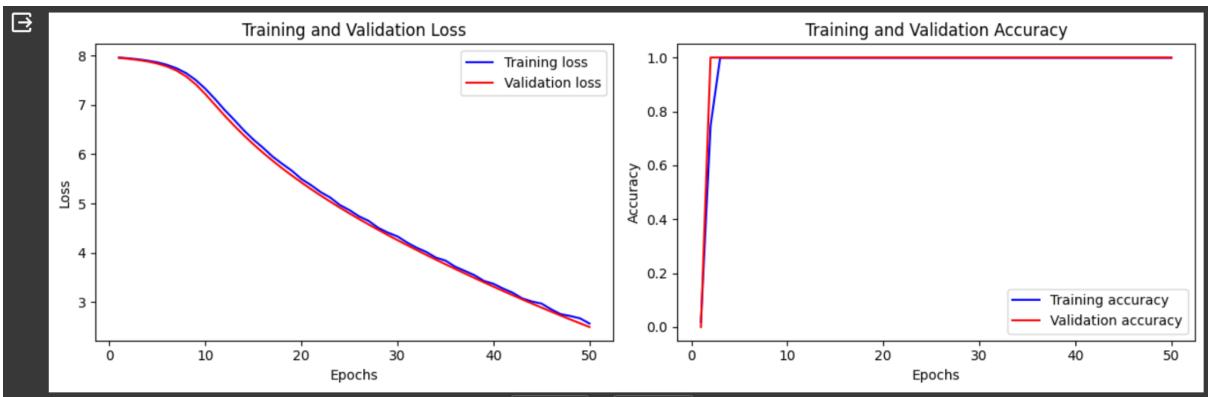
➤ **Sur le modèle configuré manuellement :** les résultats indiquent une précision (accuracy) de 100% sur l'ensemble de test avec une perte (loss) de 3.3078. Une précision de 100% pourrait suggérer que le modèle a parfaitement prédit les étiquettes de l'ensemble de test, ce qui peut sembler prometteur. On remarque aussi que la courbe d'apprentissage a un sens et qu'il n'y a ni de overfitting ni de underfitting .

```

[19] # Évaluation finale du modèle sur l'ensemble de test
test_loss, test_accuracy = best_model.evaluate(X_test, y_test)
print(f"Accuracy on test set: {test_accuracy},, Test Loss: {test_loss}")

9/9 [=====] - 0s 5ms/step - loss: 3.3078 - accuracy: 1.0000
Accuracy on test set: 1.0,, Test Loss: 3.307765483856201

```



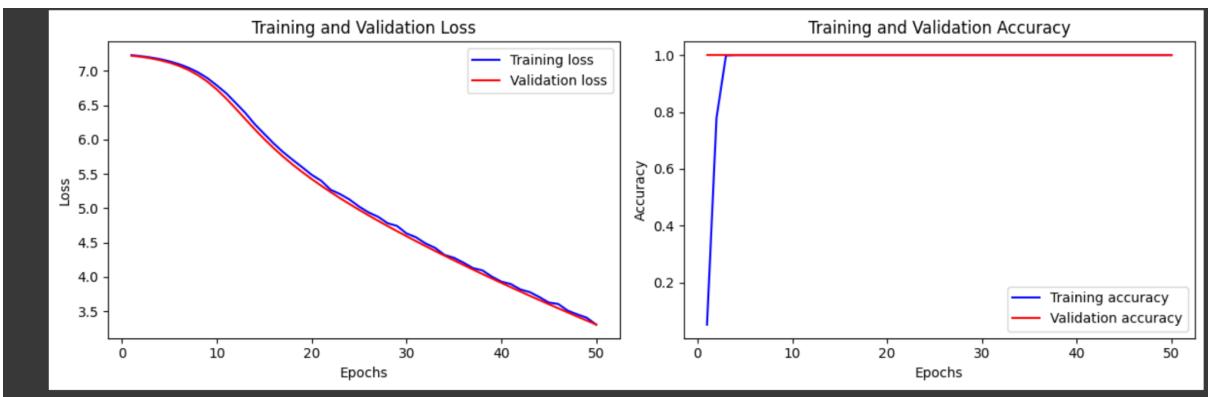
- Sur le modèle configuré avec RandomSearch : les résultats indiquent une précision (accuracy) de 100% sur l'ensemble de test avec une perte (loss) de 2.4948. Une précision parfaite peut sembler prometteuse, indiquant que le modèle a correctement prédit toutes les étiquettes de l'ensemble de test. Ici , on peut remarquer que c'est un peu bizarre que "validation accuracy" reste stable durant toutes les époques .

```

[22] # Évaluation finale du modèle sur l'ensemble de test
test_loss, test_accuracy = best_model.evaluate(X_test, y_test)
print(f"Accuracy on test set: {test_accuracy},, Test Loss: {test_loss}")

9/9 [=====] - 0s 7ms/step - loss: 2.4948 - accuracy: 1.0000
Accuracy on test set: 1.0,, Test Loss: 2.49480938911438

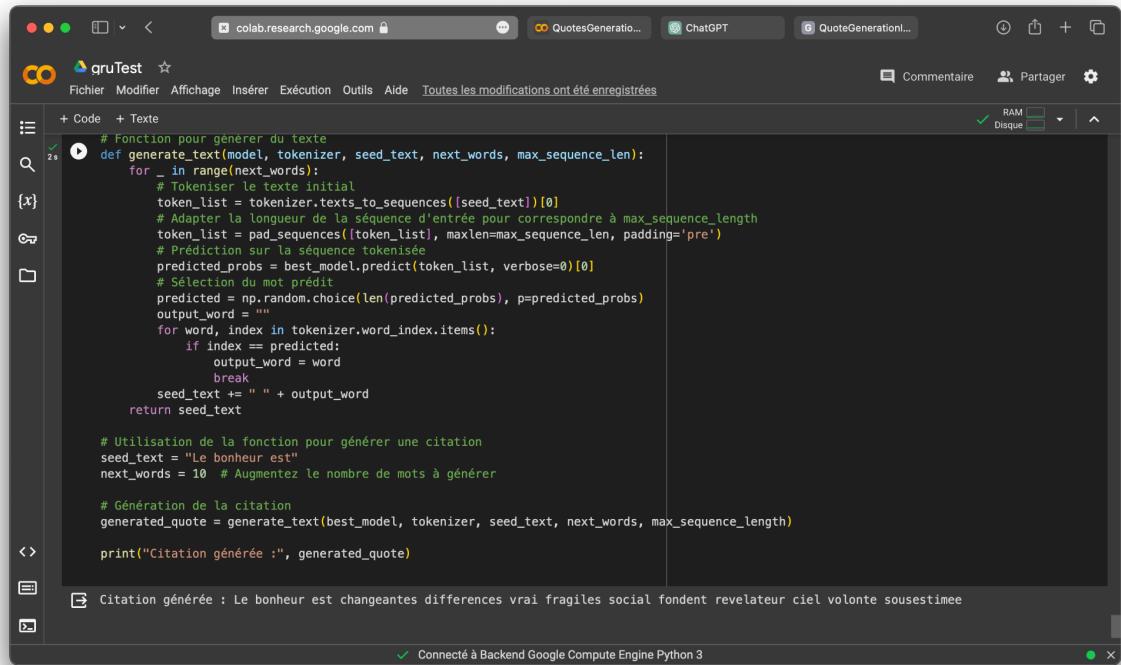
```



1.2. Génération de Citations :

1.2.1. Utilisation du modèle avec RandomSearch:

Application du modèle entraîné à l'aide de random search nous a renvoyé une citation incohérente



The screenshot shows a Google Colab notebook titled "gruTest". The code cell contains Python code for generating a quote. It starts by defining a function `generate_text` that takes a model, tokenizer, seed text, next words, and max sequence length. Inside, it tokenizes the seed text, pads the sequence, makes a prediction, and then iterates to generate words until the output word matches the predicted one. It then prints the generated quote. The output cell shows the generated quote: "Citation générée : Le bonheur est changeantes differences vrai fragiles social fondent revelateur ciel volente sousestimee". A status bar at the bottom indicates "Connecté à Backend Google Compute Engine Python 3".

```
# Fonction pour générer du texte
def generate_text(model, tokenizer, seed_text, next_words, max_sequence_len):
    for _ in range(next_words):
        # Tokeniser le texte initial
        token_list = tokenizer.texts_to_sequences([seed_text])[0]
        # Adapter la longueur de la séquence d'entrée pour correspondre à max_sequence_length
        token_list = pad_sequences(token_list, maxlen=max_sequence_len, padding='pre')
        # Prédiction sur la séquence tokenisée
        predicted_probs = best_model.predict(token_list, verbose=0)[0]
        # Sélection du mot prédict
        predicted = np.random.choice(len(predicted_probs), p=predicted_probs)
        output_word = ""
        for word, index in tokenizer.word_index.items():
            if index == predicted:
                output_word = word
                break
        seed_text += " " + output_word
    return seed_text

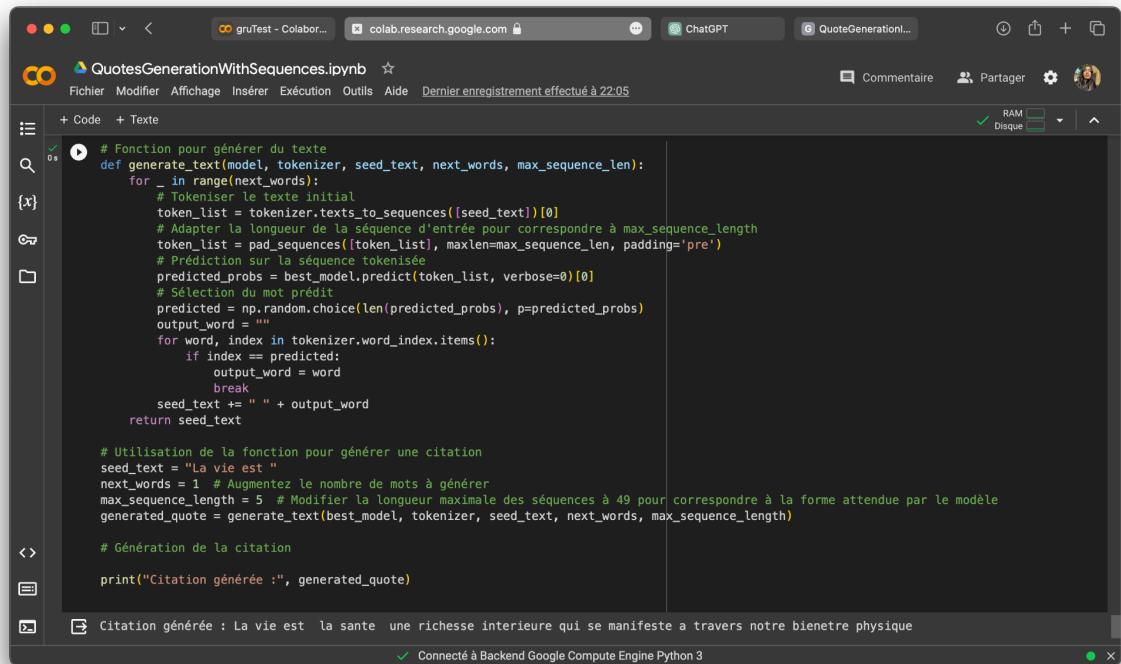
# Utilisation de la fonction pour générer une citation
seed_text = "Le bonheur est"
next_words = 10 # Augmentez le nombre de mots à générer

# Génération de la citation
generated_quote = generate_text(best_model, tokenizer, seed_text, next_words, max_sequence_length)

print("Citation générée :", generated_quote)
```

1.2.2. Validation manuelle :

On peut voir que la citation générée par le modèle configuré manuellement est cohérente et a un sens .



The screenshot shows a Google Colab notebook titled "QuotesGenerationWithSequences.ipynb". The code cell contains Python code for generating a quote. It uses the same `generate_text` function as before but with different parameters: seed text "La vie est", next words 1, max sequence length 5, and a modified max sequence length of 49. The output cell shows the generated quote: "Citation générée : La vie est la sante une richesse interieure qui se manifeste a travers notre bienetre physique". A status bar at the bottom indicates "Connecté à Backend Google Compute Engine Python 3".

```
# Fonction pour générer du texte
def generate_text(model, tokenizer, seed_text, next_words, max_sequence_len):
    for _ in range(next_words):
        # Tokeniser le texte initial
        token_list = tokenizer.texts_to_sequences([seed_text])[0]
        # Adapter la longueur de la séquence d'entrée pour correspondre à max_sequence_length
        token_list = pad_sequences(token_list, maxlen=max_sequence_len, padding='pre')
        # Prédiction sur la séquence tokenisée
        predicted_probs = best_model.predict(token_list, verbose=0)[0]
        # Sélection du mot prédict
        predicted = np.random.choice(len(predicted_probs), p=predicted_probs)
        output_word = ""
        for word, index in tokenizer.word_index.items():
            if index == predicted:
                output_word = word
                break
        seed_text += " " + output_word
    return seed_text

# Utilisation de la fonction pour générer une citation
seed_text = "La vie est"
next_words = 1 # Augmentez le nombre de mots à générer
max_sequence_length = 5 # Modifier la longueur maximale des séquences à 49 pour correspondre à la forme attendue par le modèle
generated_quote = generate_text(best_model, tokenizer, seed_text, next_words, max_sequence_length)

# Génération de la citation
print("Citation générée :", generated_quote)
```

2. Interprétation :

D'après ces résultats et après avoir effectué plusieurs expérimentations avec les hyperparamètres, il est clair que le nombre d'unités dans les couches GRU joue un rôle significatif dans la précision et la capacité de prédiction du modèle. Cette influence varie en fonction de la taille et du type de données utilisées. Dans le contexte spécifique de mon étude, j'ai constaté qu'un réglage du nombre d'unités GRU entre 10 et 15 était optimal pour que le modèle génère des citations cohérentes et pertinentes. Toute déviation par rapport à cette fourchette, que ce soit avec un nombre inférieur ou supérieur d'unités, a un impact notable sur la capacité prédictive du modèle.

Concernant le choix de la fonction d'activation, j'ai observé que la fonction tangente hyperbolique (tanh) est la plus adaptée pour la génération de texte, surtout lorsqu'elle est utilisée avec un taux d'apprentissage très petit. Cette fonction d'activation semble favoriser la génération de séquences de texte plus cohérentes et mieux structurées, ce qui est crucial pour la création de citations significatives.

Il est important de noter que ces observations sont spécifiques à mon expérience et peuvent varier en fonction du type de données, de la complexité de la tâche et des caractéristiques propres au problème étudié. Ces conclusions ont été tirées à la suite d'une série de tests et d'ajustements d'hyper paramètres, et elles représentent une base solide pour configurer le modèle afin d'obtenir des performances optimales dans la génération de citations cohérentes.

VIII - Environnement du travail :



Python est un langage de programmation interprété, multi paradigme et multiplateformes. Il favorise la programmation impérative structurée, fonctionnelle et orientée objet.



Colaboratory, souvent raccourci en "Colab", est un produit de Google Research. Colab permet à n'importe qui d'écrire et d'exécuter le code Python de son choix par le biais du navigateur. C'est un environnement particulièrement adapté au machine learning, à l'analyse de données et à l'éducation.

IX - Conclusion :

La réalisation de ce projet de génération de citations à l'aide d'un modèle GRU a été une expérience enrichissante, mettant en lumière l'importance des hyperparamètres dans la qualité des prédictions du modèle. À travers une série d'expérimentations et de réglages, il est devenu évident que le nombre d'unités GRU a un impact crucial sur la précision et la pertinence des citations générées. Une fourchette de 10 à 15 unités s'est révélée optimale pour obtenir des prédictions cohérentes, tandis que des valeurs plus élevées ou plus basses ont montré une dégradation significative des performances.

En parallèle, le choix de la fonction d'activation s'est avéré tout aussi crucial. La fonction tangente hyperbolique (\tanh), en particulier avec un taux d'apprentissage bas, a démontré une capacité supérieure à générer des citations plus structurées et pertinentes, contribuant ainsi à renforcer la cohérence et la qualité des résultats.

Cependant, il est important de souligner que ces conclusions sont spécifiques à la nature des données et au contexte de ce projet. Les observations faites ici peuvent servir de guide pour optimiser la configuration du modèle, mais il est recommandé d'approfondir davantage les recherches, notamment en explorant d'autres hyperparamètres ou en élargissant l'éventail des données pour obtenir des résultats plus robustes et généralisables.

En somme, ce projet souligne l'impact significatif des choix d'hyperparamètres sur la performance d'un modèle de génération de texte basé sur des réseaux de neurones récurrents. Ces découvertes offrent une base solide pour de futures investigations dans le domaine de la génération de contenu textuel et de la manipulation des hyperparamètres pour améliorer la qualité des prédictions de modèles similaires.

X - Défis :

Dans le cadre de ce projet de génération de citations utilisant un modèle GRU, plusieurs défis ont été rencontrés, mettant en lumière des aspects cruciaux pour améliorer la qualité et la pertinence des résultats :

> Variété et Qualité des Données :

- *Manque de Diversité* : La disponibilité limitée de données variées peut limiter la capacité du modèle à capturer la diversité stylistique et thématique des citations.
- *Nettoyage et Prétraitement* : La qualité des données, y compris la présence d'erreurs typographiques ou de citations mal formatées, nécessite un nettoyage approfondi et des techniques de prétraitement sophistiquées pour garantir la cohérence des entrées du modèle.

> Surapprentissage (Overfitting) :

- *Sensibilité aux Données d'Entraînement* : Le modèle peut avoir tendance à trop s'adapter aux données d'entraînement spécifiques, ce qui réduit sa capacité à généraliser à de nouvelles données.
- *Optimisation des Hyperparamètres* : La recherche de meilleurs hyperparamètres pour réduire le surapprentissage sans sacrifier les performances est un défi crucial.

➤ **Recherche des Meilleurs Hyperparamètres :**

- *Expérimentations Coûteuses* : L'exploration exhaustive de l'espace des hyperparamètres peut nécessiter beaucoup de ressources de calcul et de temps.
- *Évaluation de la Performance* : Déterminer quelles combinaisons d'hyper paramètres conduisent aux meilleurs résultats nécessite une évaluation minutieuse et rigoureuse.

➤ **Évaluation de la Qualité des Résultats :**

- *Subjectivité* : Évaluer la qualité et la pertinence des citations générées peut être sujet à l'interprétation humaine et nécessite des mesures objectives et subjectives.
- *Métriques de Qualité* : Le choix de métriques appropriées pour évaluer la qualité, la nouveauté et la cohérence des citations générées peut être complexe et sujet à interprétation.

Chaque défi rencontré dans ce projet a offert des opportunités d'apprentissage et de croissance, soulignant l'importance de l'exploration approfondie des données, des techniques de modélisation et de l'évaluation pour améliorer la qualité et la pertinence des résultats de génération de citations. En surmontant ces défis, il est possible d'améliorer la capacité du modèle à produire des citations pertinentes et cohérentes, adaptées à divers contextes et styles.

XI - Suggestions pour les Prochaines Étapes :

➤ Exploration des Hyperparamètres :

- *Diversification des Réglages* : Explorez davantage les variations des hyper paramètres tels que le taux d'apprentissage, les stratégies d'optimisation, ou même la taille des lots pour évaluer leur impact sur la qualité des citations générées.
- *Utilisation de Techniques Avancées* : Intégrez des mécanismes d'attention ou d'autres couches récurrentes pour améliorer la qualité des prédictions et la cohérence des citations.

➤ Expansion des Données et Prétraitement :

- *Augmentation des Données* : Enrichissez l'ensemble de données avec plus de citations diversifiées pour permettre au modèle de capturer une plus grande variété de styles et de contenus.
- *Prétraitement Amélioré* : Expérimitez avec des techniques de prétraitement plus sophistiquées, telles que la tokenization par séquence ou des méthodes de nettoyage plus avancées, pour optimiser la qualité des données d'entrée.

➤ Évaluation Approfondie :

- *Évaluation Humaine* : Impliquez des évaluateurs humains pour noter la cohérence, la pertinence et la nouveauté des citations générées par le modèle, afin d'obtenir des retours subjectifs et qualitatifs.
- *Métriques de Qualité* : Explorez l'utilisation de métriques automatiques pour évaluer la diversité, la cohérence sémantique et la nouveauté des citations générées.

➤ Expérimentation avec d'Autres Modèles :

- *Comparaison de Modèles* : Comparez les performances du modèle GRU avec d'autres architectures de réseaux neuronaux récurrents ou des modèles de génération de texte, comme les LSTM, les Transformer, ou même les modèles pré-entraînés tels que GPT.

Ces suggestions visent à approfondir l'exploration et l'amélioration du modèle actuel, ainsi qu'à élargir les horizons pour tirer le meilleur parti de la génération de citations à l'aide de techniques d'apprentissage automatique. À mesure que vous explorez ces prochaines étapes, vous pourrez découvrir de nouvelles avenues pour améliorer la qualité et la pertinence des citations générées par votre modèle.

Annexes :

- Lien vers code source :
[https://colab.research.google.com/drive/1UorCFVIGHGwyTn14r6b0WIFbnbL3IA0C?
usp=sharing](https://colab.research.google.com/drive/1UorCFVIGHGwyTn14r6b0WIFbnbL3IA0C?usp=sharing)