République Algérienne Démocratique et Populaire
الجمهورية الجزائرية الديمقراطية الشعبية
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
وزارة التعليم العالي و البحث العلمي

ECOLE NATIONALE
SUPÉRIEURE
D'INFORMATIQUE

المدرسة الوطنية العليا للإعلام الآلي
(المعهد الوطني للتكوين في الاعلام الالي سابقا)
École nationale Supérieure d'Informatique
ex. INI (Institut National de formation en Informatique)

Projet BDM

Module :

Big Data Mining

Groupe :

2CS SIL1

Membres de l'équipe :

BENGUIAR Safia Ines
BENKHELIFA Bouchra
SAFSAFI Safa
YOUSFI Sarah

Année universitaire 2024-2025

Contents

Chapter 1

Introduction

Ce rapport explore l'analyse de sentiments sur un dataset de 1,6 million de tweets en utilisant des réseaux de neurones récurrents (RNN) et des LSTM. Comment ces modèles traitent-ils les données textuelles séquentielles ? Quels sont leurs impacts sur la classification des sentiments ? En quoi leurs performances diffèrent-elles ? Ce travail examine ces questions en détaillant le prétraitement des données, la mise en place des modèles et l'analyse des résultats obtenus.

1.1 Définition du Deep Learning

Le Deep Learning (ou apprentissage profond) est un sous-domaine du Machine Learning qui utilise des réseaux de neurones artificiels à plusieurs couches (d'où le terme profond) pour apprendre et extraire automatiquement des caractéristiques complexes à partir de grandes quantités de données. Inspiré du fonctionnement du cerveau humain, le Deep Learning est particulièrement efficace pour traiter des données non structurées comme les images, le texte et les signaux audio.
Les principales caractéristiques du Deep Learning sont :

1. **Réseaux de neurones profonds :** Utilisation de plusieurs couches (entrées, cachées et sorties) pour apprendre des représentations hiérarchiques des données. Plus il y a de couches, plus le réseau peut apprendre des caractéristiques complexes.

2. A**pprentissage supervisé et non supervisé :** Peut être utilisé avec des données étiquetées (supervisé) ou sans étiquettes (non supervisé).

3. **Techniques d'optimisation :** Utilisation d'algorithmes comme la rétropropagation et des optimisateurs (ex : Adam, SGD) pour ajuster les poids du réseau et minimiser l'erreur.

4. **Besoin de grandes quantités de données et de puissance de calcul :** Plus les réseaux sont profonds, plus ils nécessitent de données et de puissance de calcul (souvent fournie par des GPU).

Le Deep Learning est largement utilisé dans des domaines comme la vision par ordinateur, le traitement du langage naturel, la reconnaissance vocale, et les systèmes de recommandation.
Auteur : Safa

1.2 Objectif du Deep Learning

1.2.1 Amélioration de la Précision des Prédictions

Les réseaux de neurones profonds permettent d'identifier des motifs complexes au sein de vastes ensembles de données, conduisant à des prédictions plus précises. Par exemple, dans le domaine de la météorologie, des modèles avancés peuvent prévoir des conditions climatiques avec une précision accrue, contribuant ainsi à une meilleure anticipation des phénomènes météorologiques.

1.2.2 Automatisation de l'Extraction des Caractéristiques

Contrairement aux méthodes traditionnelles nécessitant une intervention humaine pour définir les caractéristiques pertinentes, le Deep Learning apprend automatiquement à

partir des données brutes. Cette capacité est particulièrement utile dans des applications telles que la reconnaissance vocale, où les modèles peuvent transcrire la parole en texte sans nécessiter de prétraitement manuel des signaux audio.

1.2.3 Traitement Efficace de Données Massives

Les architectures de Deep Learning sont conçues pour gérer et analyser de grandes quantités de données en temps réel. Par exemple, dans le secteur de la publicité en ligne, des systèmes peuvent analyser des millions d'interactions utilisateur pour diffuser des annonces ciblées, optimisant ainsi les campagnes publicitaires [**?**].

1.2.4 Découverte de Modèles Complexes et Cachés

Le Deep Learning excelle dans la détection de relations non évidentes au sein des données. Par exemple, dans le domaine de la santé, des modèles peuvent identifier des biomarqueurs subtils associés à certaines maladies, facilitant ainsi des diagnostics plus précoces et précis.

1.2.5 Adaptabilité et Apprentissage Continu

Les systèmes de Deep Learning peuvent s'adapter à de nouvelles informations, améliorant continuellement leurs performances. Par exemple, dans le secteur financier, des modèles peuvent ajuster leurs prédictions en fonction des fluctuations du marché, offrant ainsi des analyses plus pertinentes.

1.2.6 Réduction des Coûts et Gains d'Efficacité

En automatisant des tâches complexes, le Deep Learning permet de réduire les coûts opérationnels et d'accroître l'efficacité. Par exemple, dans le domaine de la fabrication, des systèmes peuvent surveiller la qualité des produits en temps réel, diminuant ainsi le besoin d'inspections manuelles.

1.2.7 Stimulation de l'Innovation Technologique

Le Deep Learning ouvre la voie à des innovations dans divers secteurs. Par exemple, dans le domaine de la traduction automatique, des modèles avancés permettent des traductions plus fluides et précises, facilitant ainsi la communication entre différentes langues.

1.2.8 Amélioration de la Qualité des Résultats

Grâce à des techniques avancées, le Deep Learning offre des résultats de haute qualité. Par exemple, dans le domaine de la restauration d'images, des modèles peuvent améliorer la résolution et la clarté de photos anciennes ou endommagées, préservant ainsi le patrimoine visuel [**?**].

1.3 Description du Dataset

Le dataset Sentiment140 est une collection largement utilisée de 1,6 million de tweets, étiquetés pour indiquer un sentiment **positif**, **négatif** ou **neutre**. Développé en 2009 par des chercheurs de Stanford, il constitue une ressource précieuse pour la création et l'évaluation de modèles d'analyse de sentiment.

Le dataset comprend :

- **800 000 tweets positifs**
- **800 000 tweets négatifs**

Le dataset est composé de six colonnes :

- **target** : Étiquette de sentiment du tweet (0 = négatif, 2 = neutre, 4 = positif).
- **ids** : Identifiant unique du tweet.
- **date** : Date de publication du tweet.
- **flag** : Terme de requête (mot-clé utilisé pour collecter le tweet). Si aucun mot-clé n'a été utilisé, cette valeur est `"NO_QUERY"`.
- **user** : Nom d'utilisateur de la personne ayant publié le tweet.
- **text** : Contenu du tweet en anglais.

Conçu à la fois pour les consommateurs et les entreprises, **Sentiment140** permet de classer automatiquement les tweets liés à une marque, un produit ou un sujet comme étant **positifs** ou **négatifs**. Il est considéré comme la référence en matière de recherche sur la classification des sentiments, contribuant à des avancées majeures dans le **traitement du langage naturel** (*Natural Language Processing, NLP*).

1.4 Prétraitement des données

Le prétraitement des données est une étape essentielle pour améliorer la qualité des entrées du modèle RNN.

Nous avons nettoyé le texte en appliquant plusieurs transformations :

- Suppression des mentions (`@user`), des liens, de la ponctuation et des chiffres.
- Suppression des contractions et correction des mots allongés.
- Normalisation en convertissant le texte en **minuscules**.
- Suppression des mots vides (*stopwords*) tels que *"the", "is", "and", "in"*.
- Lemmatisation pour réduire les mots à leur forme de base.

- Réduction des mots allongés (comme "coooool" -¿ "cool")

Une fois le texte nettoyé :

- Nous appliquons la **tokenisation**, qui transforme les phrases en séquences numériques exploitables par le modèle.
- Les séquences sont ensuite remplies (*padding*) pour garantir une longueur uniforme.
- Enfin, nous divisons les données en ensembles **d'entraînement** et **de test**.

```python
def clean_text(text):
    text = re.sub(r"@\w+", "", text)  # Supprime les mentions
    text = re.sub(r"http\S+|www.\S+", "", text)  # Supprime les URLs
    text = re.sub(r"[^a-zA-Z\s]", "", text)  # Supprime la ponctuation
    text = re.sub(r'(\w)\1{2,}', r'\1', text) #Modifier les mots allongés
    text = contractions.fix(text) #Enlever les contraction
    text = re.sub(r'\d+', '', text) #Supprimer les chiffres
    text = text.lower().strip()  # Minuscule et suppression espaces inutiles
    text = " ".join([word for word in text.split() if word not in       stop_words])  # Suppression stopwords
    text = " ".join([lemmatizer.lemmatize(word) for word in text.split()])  # Lemmatisation
    return text
df["text"] = df["text"].apply(clean_text)
# Tokenisation des textes
tokenizer = Tokenizer(oov_token="<OOV>")
tokenizer.fit_on_texts(df["text"])


# Convertir les tweets en séquences de tokens
df["tokens"] = tokenizer.texts_to_sequences(df["text"])


#Longueur maximum des vecteurs est de 50
max_length = 50
padded_sequences = pad_sequences(df["tokens"], maxlen=max_length, padding="post")


#Diviser le dataset en un ensemble pour training et test
X_train, X_test, y_train, y_test = train_test_split(padded_sequences, df["target"], test_size=0.2, random_state=42)


#Deux classes en sortie (positif, négatif)
y_train = to_categorical(y_train, num_classes=2)
y_test = to_categorical(y_test, num_classes=2)
```

1.5 RNN

Un réseau neuronal récurrent (Recurrent Neural Network ou RNN) est un type de réseau de neurones profond conçu pour traiter des données séquentielles ou des séries temporelles. Grâce à son architecture spécifique, il peut apprendre des dépendances entre les éléments d'une séquence et effectuer des prédictions basées sur ces relations. Ce modèle de machine learning (ML) est particulièrement adapté aux tâches où l'ordre des données joue un rôle crucial, comme la reconnaissance vocale, la traduction automatique ou la prévision de

séries temporelles. Dans ce qui suit, nous allons présenter l'implémentation du modèle RNN appliqué au dataset Sentiment140.

1.5.1 Librairies utilisées

Afin de prétraiter nos données et de concevoir efficacement notre modèle, nous avons utilisé les bibliothèques suivantes :

- **Pandas** : pour la manipulation et l'analyse des données, ainsi que le chargement et le nettoyage du dataset.
- **NumPy** : pour les opérations mathématiques et le traitement des tableaux multi-dimensionnels.
- **NLTK (Natural Language Toolkit)** : pour le traitement du langage naturel, incluant la tokenisation, la suppression des *stopwords* et la lemmatisation.
- **TensorFlow/Keras** : pour la construction et l'entraînement du modèle RNN, en exploitant ses couches récurrentes comme LSTM ou GRU.

1.5.2 Construction du Modèle

```
model = Sequential([
    Embedding(input_dim=vocab_size, output_dim=embedding_dim, input_shape=(max_length,)),
    SimpleRNN(rnn_units, activation="tanh"),
    Dropout(0.5),
    Dense(64, activation="relu", kernel_regularizer=l2(0.01)),
    Dense(64, activation="relu"),
    Dense(2, activation="softmax")
])
model.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["accuracy"])


model.fit(X_train, y_train, epochs=2 ,batch_size=1024, validation_data=(X_test, y_test))
```

Le modèle comporte les couches suivantes :

- **Couche d'embedding** : Convertit les entiers reçus en entrée en vecteurs denses, aidant ainsi le modèle à comprendre les relations entre les mots.
- **Couche SimpleRNN** : Traite ces séquences en tenant compte du contexte des mots précédents grâce à la fonction d'activation `tanh`.
- **Couche Dropout (50%)** : Désactive aléatoirement la moitié des neurones pendant l'entraînement afin de réduire le risque de surapprentissage.

- **Couche Dense de 64 neurones** : Utilise la fonction d'activation `ReLU` et une régularisation L2 (0.01), suivie d'une seconde couche dense de 64 neurones.
- **Couche de sortie** : Utilise une activation `softmax` pour générer des probabilités sur les deux classes (**positive** et **négative**).

Pour la fonction de coût, nous avons opté pour `categorical_crossentropy`, qui est la plus appropriée pour `softmax`, ainsi que la métrique **Accuracy** pour évaluer la performance du modèle.

Layer (type)	Output Shape	Param #
embedding (Embedding)	?	0 (unbuilt)
lstm (LSTM)	?	0 (unbuilt)
dropout (Dropout)	?	0
dense (Dense)	?	0 (unbuilt)

```
Total params: 0 (0.00 B)
Trainable params: 0 (0.00 B)
Non-trainable params: 0 (0.00 B)
Epoch 1/5
16000/16000 ━━━━━━━━━━━━━━━━━━━━ 1986s 124ms/step - accuracy: 0.5985 - loss: 0.6259 - val_accuracy: 0.7789 - val_loss: 0.4633
Epoch 2/5
16000/16000 ━━━━━━━━━━━━━━━━━━━━ 2060s 128ms/step - accuracy: 0.7832 - loss: 0.4579 - val_accuracy: 0.7797 - val_loss: 0.4669
Epoch 3/5
16000/16000 ━━━━━━━━━━━━━━━━━━━━ 2072s 128ms/step - accuracy: 0.7912 - loss: 0.4429 - val_accuracy: 0.7841 - val_loss: 0.4547
Epoch 4/5
16000/16000 ━━━━━━━━━━━━━━━━━━━━ 2058s 128ms/step - accuracy: 0.7976 - loss: 0.4317 - val_accuracy: 0.7841 - val_loss: 0.4572
Epoch 5/5
16000/16000 ━━━━━━━━━━━━━━━━━━━━ 2044s 128ms/step - accuracy: 0.8029 - loss: 0.4226 - val_accuracy: 0.7779 - val_loss: 0.4745
```

Nous avons également utilisé d'autres métriques pour évaluer notre modèle, telles que la précision (Precision), le rappel (Recall) et le F1-score, qui nous permettent de mieux comprendre la qualité des prédictions du modèle au-delà de la simple exactitude (Accuracy).

	precision	recall	f1-score	support
Negative	0.78	0.80	0.79	159494
Positive	0.79	0.77	0.78	160506
accuracy			0.78	320000
macro avg	0.78	0.78	0.78	320000
weighted avg	0.78	0.78	0.78	320000

1.5.3 Test du modèle

La figure ci-dessous représente le résultat des prédictions du modèle sur une série de tweets. Chaque tweet est associé à une classe prédite:

- *"I'm so happy because we helped those poor people"* → **Positive**
- *"I thought the movie would be terrible, but it was actually fantastic!"* → **Positive**
- *"This movie could have been better without that actor"* → **Negative**
- *"I loved it!"* → **Positive**

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 68ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 70ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 102ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 105ms/step
```

Tweet	Expected Sentiment	Predicted Sentiment
I'm so happy because we helped those poor people	Positive	Negative
I thought the movie would be terrible, but it was actually fantastic!	Positive	Positive
This movie could have been better without that actor	Negative	Positive
I loved it!	Positive	Positive

1.6 LSTM

Les Long Short-Term Memory (LSTM) sont un type spécial de réseaux de neurones récurrents (RNN) conçus pour mieux gérer les dépendances à long terme dans les séquences de données. Contrairement aux RNN classiques, qui souffrent du problème de gradient qui disparaît ou explose, les LSTM peuvent apprendre efficacement les relations dans des séquences longues grâce à leur architecture unique. Les RNN classiques sont bons pour les séquences courtes, mais ils ont du mal à retenir des informations sur de longues périodes en raison du problème de gradient qui disparaît. Les LSTM résolvent ce problème en introduisant des cellules mémoire et des mécanismes de portes qui contrôlent le flux d'information.

Librairies Utilisées

Pour prétraiter nos données et concevoir efficacement notre modèle, nous avons utilisé les bibliothèques suivantes :

- **Pandas** : Pour la manipulation et l'analyse des données, ainsi que le chargement et le nettoyage du dataset.
- **NumPy** : Pour les opérations mathématiques et le traitement des tableaux multi-dimensionnels.

- **NLTK (Natural Language Toolkit)** : Pour le traitement du langage naturel, incluant la tokenisation, la suppression des stopwords et la lemmatisation.
- **TensorFlow/Keras** : Pour la construction et l'entraînement du modèle LSTM, en exploitant ses couches récurrentes avancées.

1.6.1 Construction du Modele

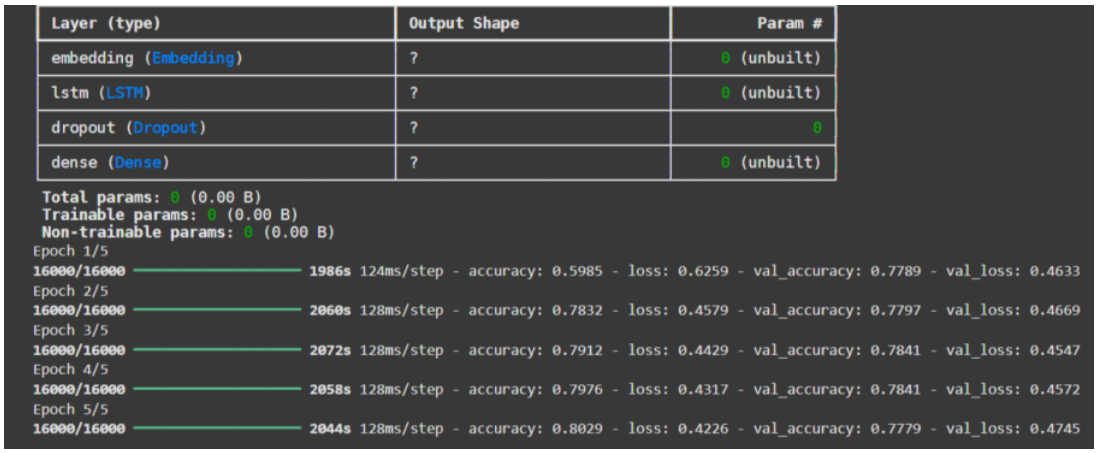
```
model = Sequential([
    Embedding(input_dim=vocab_size, output_dim=embedding_dim, input_shape=(max_length,)),
    SimpleRNN(rnn_units, activation="tanh"),
    Dropout(0.5),
    Dense(64, activation="relu", kernel_regularizer=l2(0.01)),
    Dense(64, activation="relu"),
    Dense(2, activation="softmax")
])


model.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["accuracy"])

model.summary()

model.fit(X_train, y_train, epochs=2 ,batch_size=1024, validation_data=(X_test, y_test))
```

Le modèle LSTM comporte les couches suivantes :

- **Couche d'embedding** : Transforme des mots (représentés par des entiers) en vecteurs denses, aidant ainsi le modèle à comprendre les relations entre les mots.
- **Couche LSTM** : Capture les relations séquentielles dans les données`tanh`.
- **Couche Dropout (70%)** : Prévenir le surapprentissage en désactivant aléatoirement 50
- **Couche Dense** : La fonction d'activation sigmoïde est utilisée pour produire une probabilité entre 0 et 1 (une sortie pour la classification binaire)

Pour la fonction de coût, nous avons opté pour binary crossentropy, qui est la plus appropriée pour les problèmes de classification binaire avec une activation de type sigmoïde en sortie. Cette fonction mesure l'écart entre les prédictions du modèle (probabilités) et les étiquettes réelles (0 ou 1). Elle pénalise fortement les prédictions incorrectes, ce qui permet au modèle d'apprendre efficacement à distinguer entre les classes positive et négative.

Comme métrique d'évaluation, nous avons choisi Accuracy (précision) pour mesurer la proportion de prédictions correctes par rapport à l'ensemble des prédictions.



Layer (type)	Output Shape	Param #
embedding (Embedding)	?	0 (unbuilt)
lstm (LSTM)	?	0 (unbuilt)
dropout (Dropout)	?	0
dense (Dense)	?	0 (unbuilt)

Total params: 0 (0.00 B)
Trainable params: 0 (0.00 B)
Non-trainable params: 0 (0.00 B)

```
Epoch 1/5
16000/16000 ━━━━━━━━━━━━━━━━━━━━ 1986s 124ms/step - accuracy: 0.5985 - loss: 0.6259 - val_accuracy: 0.7789 - val_loss: 0.4633
Epoch 2/5
16000/16000 ━━━━━━━━━━━━━━━━━━━━ 2060s 128ms/step - accuracy: 0.7832 - loss: 0.4579 - val_accuracy: 0.7797 - val_loss: 0.4669
Epoch 3/5
16000/16000 ━━━━━━━━━━━━━━━━━━━━ 2072s 128ms/step - accuracy: 0.7912 - loss: 0.4429 - val_accuracy: 0.7841 - val_loss: 0.4547
Epoch 4/5
16000/16000 ━━━━━━━━━━━━━━━━━━━━ 2058s 128ms/step - accuracy: 0.7976 - loss: 0.4317 - val_accuracy: 0.7841 - val_loss: 0.4572
Epoch 5/5
16000/16000 ━━━━━━━━━━━━━━━━━━━━ 2044s 128ms/step - accuracy: 0.8029 - loss: 0.4226 - val_accuracy: 0.7779 - val_loss: 0.4745
```

1.6.2 Test du modèle

La figure ci-dessous présente les résultats des prédictions du modèle LSTM sur une série de tweets. Chaque tweet est associé à une classe prédite :

- *"I'm so happy because we helped those poor people"* → **Positive**
- *"I thought the movie would be terrible, but it was actually fantastic!"* → **Positive**
- *"This movie could have been better without that actor"* → **Negative**
- *"I loved it!"* → **Positive**

	precision	recall	f1-score	support
Negative	0.78	0.80	0.79	159494
Positive	0.79	0.77	0.78	160506
accuracy			0.78	320000
macro avg	0.78	0.78	0.78	320000
weighted avg	0.78	0.78	0.78	320000

Au-

teur : Safa ou Ines

1.7 Comparaison entre RNN et LSTM

1.7.1 Comparaison basée sur les résultats obtenus

Modèle	Accuracy	F1-score	Temps d'entraînement
RNN	78%	76%	**20 minutes**
LSTM	**80%**	**80%**	45 minutes

Table 1.1: Comparaison entre RNN et LSTM sur l'analyse de sentiments des tweets

Nous avons observé que LSTM surpasse RNN dans l'analyse de sentiments des tweets .Voici quelques exemples de tweets mal classés par RNN, mais correctement identifiés par LSTM :

Un tweet :*" I'm so happy because we helped those poor people!"*

- RNN → Négatif (prend en compte " poor " mais ignore la fin)
- LSTM → Positif (comprend le retournement du sentiment)

Analyse : RNN a tendance à oublier les premières parties de la phrase lorsqu'il traite la fin. Ici, le mot "poor" lui fait penser à une émotion négative, alors qu'en réalité, LSTM gère mieux ce type de relation grâce à ses cellules mémoire.

Un autre tweet :*"This movie could have been better without that actor."*

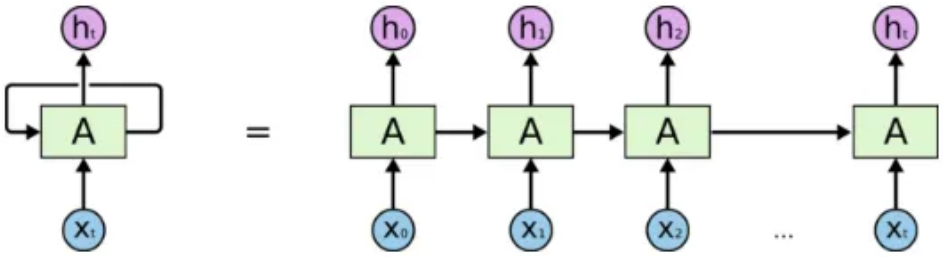
- RNN → Positif (se focalise sur "better" sans comprendre la négation)
- LSTM → Négatif (comprend que "without that actor" exprime une critique)

Analyse : Les RNN simples ont du mal à gérer les négations complexes et les nuances subtiles. LSTM, en gardant la mémoire de la relation entre les mots "better" et "wothout that actor", comprend que c'est une critique déguisée.

1.7.2 Comparaison théorique

RNN

Un réseau de neurones récurrent (RNN) est une généralisation du réseau de neurones à propagation avant qui possède une mémoire interne. Le RNN est récurrent par nature, car il effectue la même fonction pour chaque entrée de données, tandis que la sortie de l'entrée actuelle dépend du calcul précédent. Après avoir produit la sortie, celle-ci est copiée et renvoyée dans le réseau récurrent. Pour prendre une décision, il prend en compte l'entrée actuelle ainsi que la sortie qu'il a apprise à partir de l'entrée précédente.

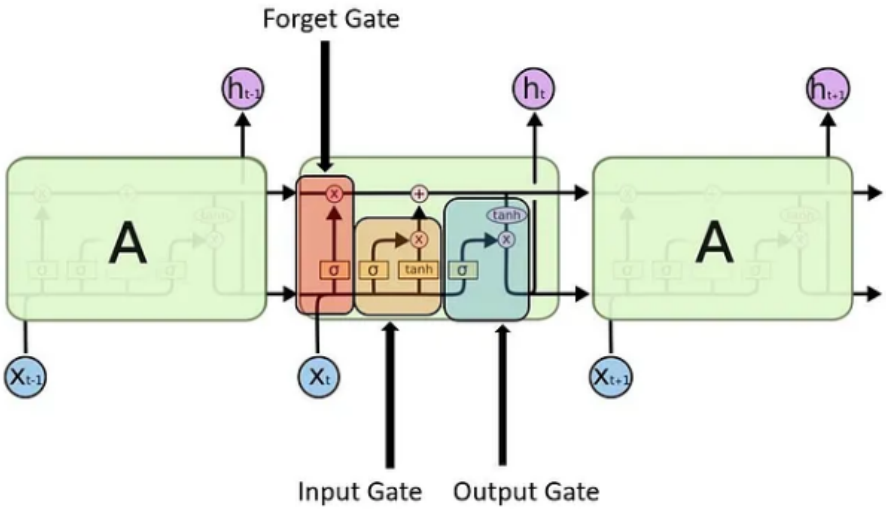


Inconvénients des réseaux de neurones récurrents (RNN) :

- Problèmes de disparition et d'explosion du gradient.
- Il ne peut pas traiter de très longues séquences lorsqu'on utilise tanh ou ReLU comme fonction d'activation.

LSTM

Les réseaux de mémoire à long court terme (LSTM) sont une version modifiée des réseaux de neurones récurrents, ce qui facilite la mémorisation des données passées en mémoire. Le problème de gradient qui disparaît des RNN est résolu ici. Les LSTM sont bien adaptés pour classer, traiter et prédire des séries temporelles en tenant compte des décalages temporels de durée inconnue. Ils entraînent le modèle en utilisant la rétropropagation. Dans un réseau LSTM .



Deduction

Les réseaux de neurones récurrents (RNN) et les réseaux de mémoire à long court terme (LSTM) sont tous deux utilisés pour le traitement des séquences et des séries temporelles. Cependant, Les LSTM sont souvent préférés aux RNN classiques pour des tâches nécessitant une mémoire à long terme, comme la traduction automatique ou la reconnaissance vocale.

Chapter 2

Concluion

Ce travail a permis d'explorer l'analyse de sentiments en utilisant des modèles de réseaux de neurones récurrents (RNN) et des LSTM. L'étude a mis en évidence les limites des RNN en matière de mémorisation sur de longues séquences, ce qui impacte négativement la classification des sentiments. En revanche, les LSTM, grâce à leurs mécanismes de portes (forget gate, input gate, et output gate), parviennent à mieux capturer le contexte et à conserver les informations pertinentes sur de longues séquences, améliorant ainsi la précision des prédictions.

Les résultats obtenus confirment l'efficacité des LSTM pour le traitement du langage naturel et soulignent l'importance du choix de l'architecture neuronale en fonction des caractéristiques des données. Ces conclusions ouvrent la voie à des améliorations potentielles, notamment en explorant des modèles encore plus avancés comme les Transformers, qui surpassent les LSTM dans de nombreuses tâches de NLP.