TAIFI
ASSIA

WIFKAOUI
YASMINE

AL JADD
MOHAMMED

EL NABAOUI
NOUHAILA

MOVIE SENTIMENT ANALYSIS

Deep Learning

2020 - 2021

Projet Deep Learning

Sujet de Recherche : Movie sentiment analysis

Encadrement :

Mr. **Ibn Elhaj El Hassane,** Professeur de l'enseignement supérieur à l'Institut Nationale des Postes et de Télécommunications.

Résumé

En raison de l'évolution massive des données et de la quantité de données échangées et produites chaque jour, l'envie de comprendre, d'exploiter et d'analyser ces données a considérablement augmenté.
D'où l'apparition de L'analyse des sentiments qui est aujourd'hui, la technique la plus populaire en développement de systèmes d'exploration d'opinions depuis la naissance d'apprentissage profond, elle utilise l'analyse de texte linguistique, ou ce qu'on appelle traitement du langage naturel NLP.

Dans ce rapport, on s'intéressera à l'analyse des sentiments à l'égard des différents films sur la base des données IMBD en utilisant les modèles de l'apprentissage profond sous Keras.

Mots-clés : **analyse des sentiments, apprentissage profond, traitement du Langage naturel, Keras.**

Abstract

Due to the massive evolution of data and the amount of data exchanged and produced every day, the urge to understand, exploit and analyze this data has increased dramatically.
Hence the emergence of Sentiment Analysis which is today the most popular technique in developing opinion exploration systems since birth deep learning, it uses linguistic text analysis, or what's called NLP natural language processing.

In this report, we will focus on the sentiment analysis for different films based on IMBD data using deep learning models under Keras.

Keywords: **sentiment analysis, deep learning, Natural language processing, Keras.**

Liste des abréviations

DL	Deep Learning
NLP	Natural Language Processing
GloVe	Global Vectors
IMBD	Internet Movie DataBase
DCNN	Densely Connected Neural Network
CNN	Convolutional Neural Network
RNN	Recurrent Neural Network
LSTM	Long-short Term Memory
GRU	Gated Recurrent Units

Sommaire

Introduction générale

Par définition, l'analyse des sentiments ou l'extraction d'opinions est l'utilisation de l'analyse de texte, de la linguistique informatique ou du traitement du langage naturel (NLP) afin d'obtenir une quantification sémantique des informations étudiées.
L'analyse des sentiments vise à indiquer l'opinion d'un texte spécifique (par exemple un tweet ou une critique de produit). Ces indications sont utilisées en conséquence par les décideurs dans la planification et la prise de mesures appropriées telles que les décisions de marketing, la chasse aux clients ou l'expansion des entreprises dans une région géographique spécifique.

En raison de l'évolution massive des données et de la quantité de données échangées et produites chaque seconde, l'envie de comprendre, d'exploiter et d'analyser ces données a considérablement augmenté.
Et comme les techniques habituelles d'apprentissage automatique et les réseaux de neurones n'étaient pas suffisants pour être obtenus sur ce big data, l'apprentissage en profondeur était la clé de l'ère du big data.
L'apprentissage profond est un sous-domaine de l'apprentissage automatique et une alternance de réseaux de neurones. Autrement dit, le réseau neuronal régulier est un réseau unique avec des couches d'entrée et de sortie en plus des couches cachées entre les deux, où le calcul est effectué.

Là où les réseaux de neurones profonds sont, fondamentalement, se composent de plusieurs réseaux de neurones où la sortie d'un réseau est une entrée vers le réseau suivant et ainsi de suite. Ce concept a surmonté la limitation du nombre de couches cachées dans les réseaux de neurones et a rendu le travail avec le Big Data plus faisable.
Les réseaux d'apprentissage en profondeur apprennent les fonctionnalités par eux-mêmes, c'est-à-dire qu'ils sont devenus une technique d'apprentissage automatique robuste qui apprend plusieurs couches de caractéristiques des données et induit des résultats de prédiction. L'apprentissage profond a été récemment utilisé dans diverses applications dans le domaine du traitement du signal et de l'information, en particulier avec l'évolution du big data. En outre, des réseaux d'apprentissage en profondeur ont été utilisés dans l'analyse des sentiments et l'exploration d'opinions.

Notre projet consisté a appliqué l'analyse des sentiments à un ensemble de données de critiques de films anglais (ensemble de données IMDB) à l'aide de techniques d'apprentissage en profondeur afin de classer ces fichiers de données en critiques positives et négatives.

Ce rapport est divisé en quatre chapitres qui sont :
Chapitre I : Cadre théorique
Chapitre II : Cadre Méthodologique

Chapitre 1: Cadre théorique

I. Présentation du sujet :

1. Problématique traitée

L'époque où la publicité directe et la bouche à oreille étaient les seules options pour les clients de choisir le bon produit ont disparu depuis longtemps.
Aujourd'hui, L'Internet fournit une passerelle idéale pour tous ceux qui veulent savoir ce que les autres pensent d'un certain produit avant de réellement l'acheter.
Puisque les gens peuvent exprimer leurs sentiments et leurs émotions de bien des façons : par le biais de plateformes de médias sociaux comme Twitter, Facebook ou Instagram, de sites Web d'avis, de forums…etc. Ils peuvent également fournir librement des commentaires sur divers produits et services.
Ce qui peuvent facilement influencer les décisions d'achat en laissant un examen dévastateur d'un certain produit ou service.
Par conséquent, l'Internet nous génère des tonnes d'informations publiées par des millions de personnes qui est difficile à analyser, mais il reste une source précieuse et lucrative qui peut fournir aux entreprises des idées révélatrices.

Alors, comment exploiter, analyser et classer ces données *?*

2. Objectifs

Le but principal de notre Travail est d'étudier l'analyse des sentiments comme outil nécessaire pour le développement des systèmes d'exploration d'opinions et comment construire des modèles pour la classification des films comme exemple de produit.

II. Généralités :

1. Natural Language Processing

- **Définition** :

  Le traitement naturel du langage, aussi appelé Natural Language Processing ou NLP en anglais, est une technologie permettant aux machines de comprendre le langage humain grâce à l'intelligence artificielle. Découvrez tout ce que vous devez savoir à ce sujet. L'objectif de cette technologie est de permettre aux machines de lire, de déchiffrer, de comprendre et de donner sens au langage humain. D'importants progrès ont été effectués dans ce domaine au fil des dernières années, et le traitement naturel du langage est aujourd'hui exploité pour une large variété de cas d'usage…

- **Fonctionnement** :

  La plupart des techniques de traitement naturel du langage reposent sur le Deep Learning. Les algorithmes d'intelligence artificielle sont entraînés à partir de données afin d'apprendre analyser le langage humain pour y trouver des patterns et des corrélations.
  Les algorithmes ont pour rôle d'identifier et d'extraire les règles du langage naturel, afin de convertir les données de langage non structuré sous une forme que les ordinateurs pourront comprendre.

2. Keras Embedding Layer

- **Keras** propose une couche d'intégration qui peut être utilisée pour les réseaux de neurones sur des données texte.
- Il nécessite que les données d'entrée soient codées en entier, de sorte que chaque mot soit représenté par un entier unique. Cette étape de préparation des données peut être effectuée à l'aide **de l'API Tokenizer** également fournie avec Keras.
- La couche d'incorporation est initialisée avec des pondérations aléatoires et apprendra une incorporation pour tous les mots de l'ensemble de données d'apprentissage.
- C'est une couche flexible qui peut être utilisée de différentes manières, telles que :
  - Il peut être utilisé seul pour apprendre une incorporation de mots qui peut être enregistrée et utilisée ultérieurement dans un autre modèle.
  - Il peut être utilisé dans le cadre d'un modèle d'apprentissage en profondeur dans lequel l'incorporation est apprise avec le modèle lui-même.
  - Il peut être utilisé pour charger un modèle d'intégration de mots pré-entraîné, un type d'apprentissage par transfert.
- La couche d'incorporation est définie comme la première couche masquée d'un réseau. Il doit spécifier 3 arguments :
  - **input_dim:** C'est la taille du vocabulaire dans les données de texte. Par exemple, si vos données sont encodées en nombre entier avec des valeurs comprises entre 0 et 10, la taille du vocabulaire serait de 11 mots.
  - **output_dim**: C'est la taille de l'espace vectoriel dans lequel les mots seront incorporés. Il définit la taille des vecteurs de sortie de cette couche pour chaque mot. Par exemple, il pourrait être 32 ou 100 ou même plus. Testez différentes valeurs pour votre problème.
  - **input_length**: Il s'agit de la longueur des séquences d'entrée, comme vous le définiriez pour n'importe quelle couche d'entrée d'un modèle

Keras. Par exemple, si tous vos documents d'entrée sont composés de 1000 mots, ce serait 1000.

- Ci-dessous, nous définissons la couche d'incorporation (embedding layer) avec un vocabulaire de 200 (par exemple, des mots codés en entier de 0 à 199 inclus), un espace vectoriel de 32 dimensions dans lequel les mots seront incorporés et des documents d'entrée de 50 mots chacun.

e = Embedding (200, 32, input_length = 50)

3. GloVe word embedding

- **Word embedding :**

Après Tomas Mikolov et al ont publié l'outil word2vec, il y avait un boom d'articles sur les représentations vectorielles de mots.
L'un des meilleurs de ces articles est GloVe : Global Vectors for Word Representation de Stanford, qui explique pourquoi de tels algorithmes fonctionnent et reformule les optimisations word2vec comme un type spécial de factorisation pour les matrices de cooccurrence de mots.

- **GloVe algorithm:**

L'algorithme **GloVe** comprend les étapes suivantes :

1- Recueillir des statistiques de cooccurrence de mots sous la forme d'une matrice de cooccurrence de mots X. Chaque élément Xij d'une telle matrice représente la fréquence à laquelle le mot i apparaît dans le contexte du mot j. Habituellement, nous analysons notre corpus de la manière suivante: pour chaque terme, nous recherchons des termes de contexte dans une zone définie par un window_size avant le terme et un window_size après le terme. De plus, nous accordons moins de poids aux mots plus éloignés, en utilisant

2- Généralement cette formule :

$$decay = 1/offset$$

3- Définissez des contraintes souples pour chaque paire de mots:

$$w_i^T w_j + b_i + b_j = log(X_{ij})$$

Ici wi - vecteur pour le mot principal, wj - vecteur pour le mot de contexte, bi, bj sont des biais scalaires pour les mots principaux et contextuels.

4- Définir une fonction de coût :

$$J = \sum_{i=1}^{V} \sum_{j=1}^{V} f(X_{ij})(w_i^T w_j + b_i + b_j - \log X_{ij})^2$$

Ici f est une fonction de pondération qui nous aide à éviter d'apprendre uniquement à partir de paires de mots extrêmement courantes.
Les auteurs de **GloVe** choisissent la fonction suivante :

$$f(X_{ij}) = \begin{cases} (\frac{X_{ij}}{x_{max}})^{\alpha} & \text{if } X_{ij} < XMAX \\ 1 & \text{otherwise} \end{cases}$$

Chapitre 2: Cadre méthodologique

I. Description du Dataset

1. Exploration du dataset:

Etape 1 : Importation des libraires

Commençons par inclure toutes les fonctions et objets dont nous aurons besoin pour notre projet.

```
In [1]: import pandas as pd
        import numpy as np
        import re
        import nltk
        from nltk.corpus import stopwords

        from numpy import array
        from keras.preprocessing.text import one_hot
        from keras.preprocessing.sequence import pad_sequences
        from keras.models import Sequential
        from keras.layers.core import Activation, Dropout, Dense
        from keras.layers import Flatten
        from keras.layers import GlobalMaxPooling1D
        from keras.layers.embeddings import Embedding
        from sklearn.model_selection import train_test_split
        from keras.preprocessing.text import Tokenizer
```

Etape 2 : Importation et analyse du dataset

- Dans cette étape, nous avons chargé notre dataset à partir d'un fichier dans le répertoire local.
- Le dataset est sous format CSV.
- Nous l'avons chargé à l'aide de la bibliothèque pandas.
- Et nous avons vérifié l'existence de valeur null.

```
In [2]: #Importer le dataset
        movie_reviews = pd.read_csv("dataset/IMDB Dataset.csv")

        #Exploration du dataset
        #Vérifier l'existence de valeur null
        movie_reviews.isnull().values.any()

Out[2]: False
```

- Ensuite nous avons vérifié le nombre de ligne et de colonne dans notre dataset (50000 lignes et 2 colonnes).

```
In [3]: movie_reviews.shape
Out[3]: (50000, 2)
```

- Utilisation de la fonction info(), nous a permis de savoir plus d'information sur notre dataset, comme le nombre de lignes les noms des colonnes et vérifie aussi l'existence des valeur null dans le dataset.

```
In [4]: movie_reviews.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50000 entries, 0 to 49999
Data columns (total 2 columns):
 #   Column     Non-Null Count  Dtype 
---  ------     --------------  ----- 
 0   review     50000 non-null  object
 1   sentiment  50000 non-null  object
dtypes: object(2)
memory usage: 781.4+ KB
```

- Une autre façon pour avoir une idée générale sur le dataset, la fonction describe(), qui nous a permis de faire quelques vérifications sur notre dataset.

```
In [5]: movie_reviews.describe()
```

Out[5]:

	review	sentiment
count	50000	50000
unique	49582	2
top	Loved today's show!!! It was a variety and not...	positive
freq	5	25000

- La méthode head(), nous a permis d'afficher les 5 premières lignes du dataset.

```
In [6]: movie_reviews.head()
```

Out[6]:

	review	sentiment
0	One of the other reviewers has mentioned that ...	positive
1	A wonderful little production. The...	positive
2	I thought this was a wonderful way to spend ti...	positive
3	Basically there's a family where a little boy ...	negative
4	Petter Mattei's "Love in the Time of Money" is...	positive

- Afin que nous ayons une idée du texte que nous allons traiter, nous avons exécuté la commande ci-dessous.

- Et nous avons remarqué que le texte contient également des ponctuations, des crochets et quelques balises HTML.

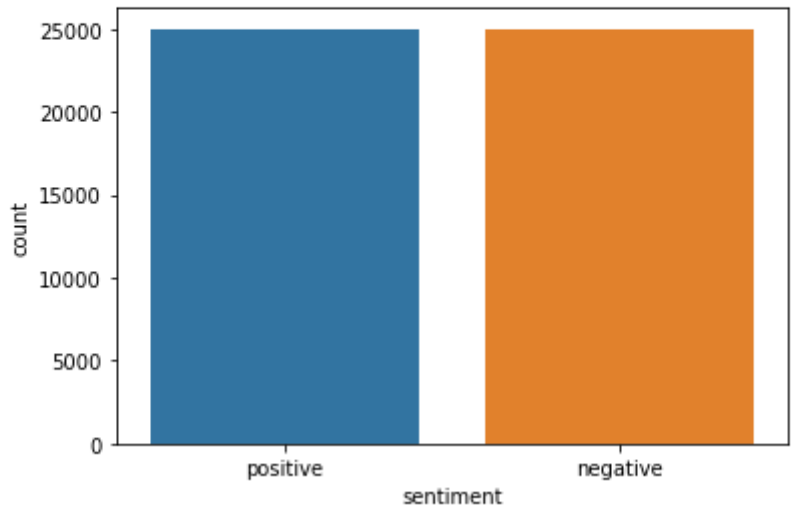
- Nous prétraiterons le texte dans la partie prétraitement des données.

```
In [7]: movie_reviews["review"][3]
```

```
Out[7]: "Basically there's a family where a little boy (Jake) thinks there's a zombie in his closet & his parents are fighting all the
time.<br /><br />This movie is slower than a soap opera... and suddenly, Jake decides to become Rambo and kill the zombie.<br /
><br />OK, first of all when you're going to make a film you must Decide if its a thriller or a drama! As a drama the movie is
watchable. Parents are divorcing & arguing like in real life. And then we have Jake with his closet which totally ruins all the
film! I expected to see a BOOGEYMAN similar movie, and instead i watched a drama with some meaningless thriller spots.<br /><br
/>3 out of 10 just for the well playing parents & descent dialogs. As for the shots with Jake: just ignore them."
```

- Enfin, nous avons utilisé countplot() de seaborn, afin de visualiser la distribution des sentiments positifs et négatifs dans le dataset.

```
In [8]: import seaborn as sns

        sns.countplot(x='sentiment', data=movie_reviews)

Out[8]: <AxesSubplot:xlabel='sentiment', ylabel='count'>
```



- D'après le diagramme à battons, il est clair que le dataset contient un nombre égal d'avis positifs et négatifs.

2. Prétraitement des données

Etape 1 : Suppression des ponctuations et des balises

- Dans la partie précédente, nous avons remarqué que notre dataset contenait des ponctuations et des balises HTML.
- Pour cela, dans cette partie, nous avons défini une fonction qui prend une chaîne de texte comme paramètre, puis effectue un prétraitement sur la chaîne pour supprimer les caractères spéciaux et les balises HTML de la chaîne.
- Enfin, la chaîne est renvoyée à la fonction appelante.

```
In [9]: #Data Preprocessing
        def preprocess_text(sen):
            # Suppression des balises HTML
            sentence = remove_tags(sen)

            # Supprimer les ponctuations et les chiffres
            sentence = re.sub('[^a-zA-Z]', ' ', sentence)

            # Suppression d'un seul caractère
            sentence = re.sub(r"\s+[a-zA-Z]\s+", ' ', sentence)

            # Suppression de plusieurs espaces
            sentence = re.sub(r'\s+', ' ', sentence)

            return sentence
```

```
In [10]: TAG_RE = re.compile(r'<[^>]+>')

         def remove_tags(text):
             return TAG_RE.sub('', text)
```

- Ensuite, nous avons prétraité les avis et nous les avons stockés dans une nouvelle liste.

- Et nous avons vérifié les modifications en affichant la 4ème critique.

```
In [11]: #Prétraiter les avis et les stocker dans une nouvelle liste

         X = []
         sentences = list(movie_reviews['review'])
         for sen in sentences:
             X.append(preprocess_text(sen))

         #Voir la quatrième critique
         X[3]
```

```
Out[11]: 'Basically there a family where little boy Jake thinks there a zombie in his closet his parents are fighting all the time This
         movie is slower than soap opera and suddenly Jake decides to become Rambo and kill the zombie OK first of all when you re going
         to make film you must Decide if its thriller or drama As drama the movie is watchable Parents are divorcing arguing like in rea
         l life And then we have Jake with his closet which totally ruins all the film expected to see BOOGEYMAN similar movie and inste
         ad watched drama with some meaningless thriller spots out of just for the well playing parents descent dialogs As for the shots
         with Jake just ignore them '
```

- À partir de la sortie(**Out[11]**), vous pouvez voir que les balises HTML, les ponctuations et les nombres ont été supprimés.

Etape 2 : Conversion des étiquettes en chiffres

- Ensuite, nous avons converti nos étiquettes en chiffres.
- Puisque nous n'avons que deux étiquettes dans la sortie, à savoir « positif » et « négatif ».
- Nous pouvons simplement les convertir en nombres entiers en remplaçant "positif" par le chiffre 1 et le négatif par le chiffre 0.

```
In [12]: #Convertir les étiquettes en chiffres
         y = movie_reviews['sentiment']

         #Remplacer "positive" par le chiffre 1 et "negative" par le chiffre 0
         y = np.array(list(map(lambda x: 1 if x=="positive" else 0, y)))
```

Etape 3 : Division du dataset

- Dans cette étape, nous avons divisé le dataset en ensembles d'entrainement et de test.
- L'ensemble d'entrainement sera utilisé pour former nos modèles d'apprentissage en profondeur tandis que l'ensemble de test sera utilisé pour évaluer les performances de notre modèle.
- Nous avons utilisé la méthode train_test_split du module sklearn.model.selection.
- Et nous avons divisé les données en 80% pour l'ensemble d'entraînement et 20% pour l'ensemble de test.

```
In [13]: #Diviser l'ensemble de données en ensembles d'entrainement et de test.
         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=42)
```

Etape 4 : Préparation du Embedding Layer

- Dans un premier temps, nous avons utilisé la classe Tokenizer du module keras.preprocessing.text pour créer un dictionnaire de mots à index.
- Dans le dictionnaire mot-à-index, chaque mot du corpus est utilisé comme clé, tandis qu'un index unique correspondant est utilisé comme valeur pour la clé.

```
In [14]: tokenizer = Tokenizer(num_words=5000)
         tokenizer.fit_on_texts(X_train)

         X_train = tokenizer.texts_to_sequences(X_train)
         X_test = tokenizer.texts_to_sequences(X_test)
```

- La variable X_train contient 40 000 listes où chaque liste contient des entiers.
- Chaque liste correspond en fait à chaque phrase de l'ensemble d'apprentissage. Et la taille de chaque liste est différente. C'est parce que les phrases ont des longueurs différentes.
- Nous avons fixé la taille maximale de chaque liste à 100.
- Les listes de taille supérieure à 100 seront tronquées à 100.
- Pour les listes dont la longueur est inférieure à 100, nous ajouterons 0 à la fin de la liste jusqu'à ce qu'elle atteigne la longueur maximale. Ce processus s'appelle le remplissage (**Padding**).
- Donc le script trouve la taille des phrases, puis effectue un remplissage à la fois sur l'ensembles d'entrainement et sur l'ensemble de test.

```
In [15]: # Ajout de 1 en raison de l'index 0 réservé
         vocab_size = len(tokenizer.word_index) + 1

         maxlen = 100

         X_train = pad_sequences(X_train, padding='post', maxlen=maxlen)
         X_test = pad_sequences(X_test, padding='post', maxlen=maxlen)
```

- Maintenant, le X_train ou X_test, contient des listes de même longueur, soit 100.
- De plus, la variable vocabulary_size contient maintenant une valeur 92547 ce qui signifie que notre corpus contient 92547 mots uniques.

- Nous avons utilisé les imbrications **GloVe** pour créer notre matrice de fonctionnalités.
- Et nous avons changé les embeddings de mots **GloVe** et nous avons créé un dictionnaire qui contient les mots comme clés et leur liste d'incorporation correspondante comme valeurs.

```
In [16]: from numpy import array
         from numpy import asarray
         from numpy import zeros

         embeddings_dictionary = dict()
         glove_file = open('dataset/Word Embeddings/glove.6B.100d.txt', encoding="utf8")

         for line in glove_file:
             records = line.split()
             word = records[0]
             vector_dimensions = asarray(records[1:], dtype='float32')
             embeddings_dictionary [word] = vector_dimensions
         glove_file.close()
```

- Enfin, nous avons créé une matrice d'incorporation où chaque numéro de ligne correspondra à l'index du mot dans le corpus.
- La matrice aura 100 colonnes où chaque colonne contiendra les embeddings de mots GloVe pour les mots de notre corpus.

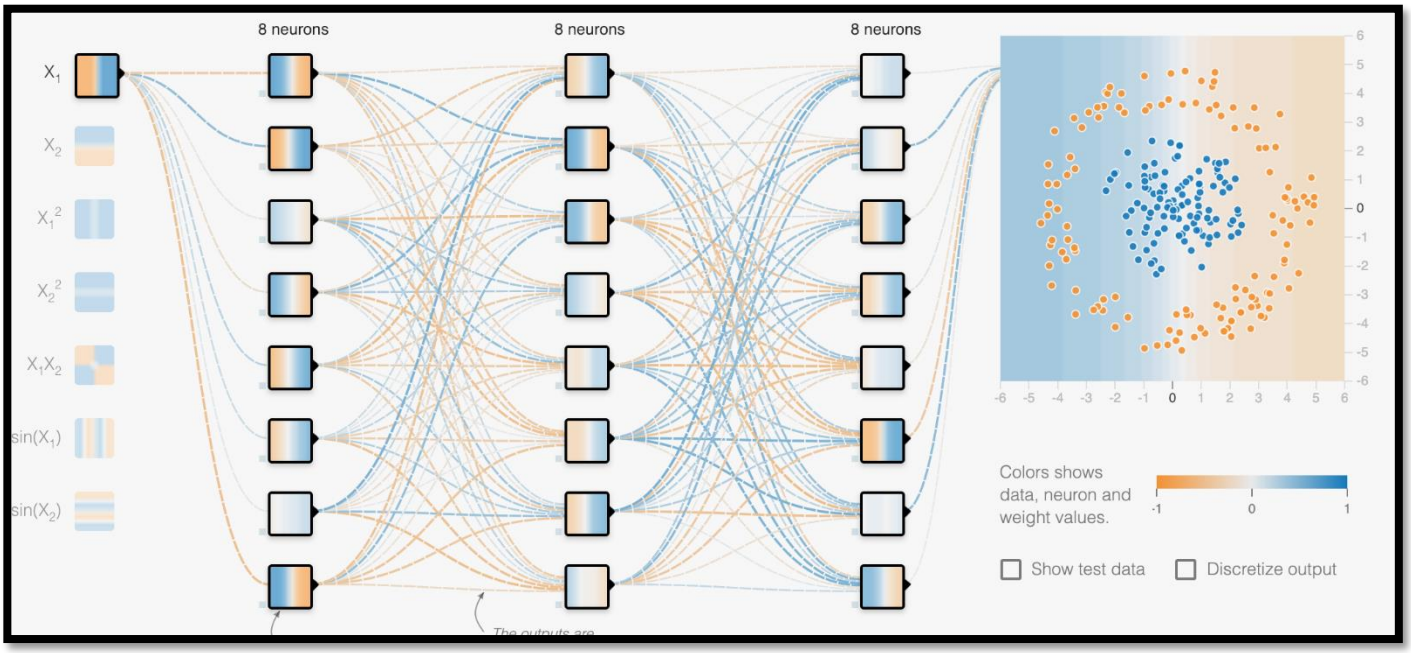
```
In [17]: embedding_matrix = zeros((vocab_size, 100))
         for word, index in tokenizer.word_index.items():
             embedding_vector = embeddings_dictionary.get(word)
             if embedding_vector is not None:
                 embedding_matrix[index] = embedding_vector
```

II. Modèles d'apprentissage profond

1 Densely connected neural network (Basic Neural Network)

a) Présentation générale

- Le nom suggère que les couches sont entièrement connectées (denses) par les neurones dans une couche réseau.
- Chaque neurone d'une couche reçoit une entrée de tous les neurones présents dans la couche précédente - ainsi, ils sont densément connectés.
- En d'autres termes, la couche dense est une couche entièrement connectée, ce qui signifie que tous les neurones d'une couche sont connectés à ceux de la couche suivante.



b) L'application du modèle

Etape 1: Création du modèle

- Dans cette première partie, nous avons créé un modèle Sequential ().
- Ensuite, nous avons créé une couche d'intégration.
- La couche d'intégration aura une longueur d'entrée de 100, la dimension du vecteur de sortie sera également de 100.

- La taille du vocabulaire sera de 92547 mots.

```
In [18]: #Text Classification with Simple Neural Network

         model = Sequential()
         embedding_layer = Embedding(vocab_size, 100,
                                     weights=[embedding_matrix],
                                     input_length=maxlen ,
                                     trainable=False)
         model.add(embedding_layer)

         model.add(Flatten())
         model.add(Dense(1, activation='sigmoid'))
```

- Puisque nous n'entraînons pas nos propres plongements et que nous n'utilisons pas l'incorporation GloVe, nous définissons « trainable » sur False et dans l'attribut poids, nous avons transmis notre propre matrice d'incorporation.
- La couche d'incorporation est ensuite ajoutée au modèle.
- Ensuite, puisque nous connectons directement notre couche d'intégration à une couche densément connectée, nous aplatissons la couche d'intégration en utilisant Flatten().
- Enfin, nous avons ajouté une couche dense avec une fonction d'activation sigmoïde.

Etape 2 : Compilation du modèle

- Pour compiler notre modèle, nous avons utilisé l'optimiseur « adam », « binary_crossentropy » comme fonction de perte et la « précision » comme métrique, puis nous avons affiché le résumé de notre modèle.
- Comme il y a 92547 mots dans notre corpus et que chaque mot est représenté sous la forme d'un vecteur à 100 dimensions, le nombre de paramètres pouvant être entraînés sera de 92547x100 dans la couche d'intégration.
- Dans la couche d'aplatissement, nous multiplions simplement les lignes et les colonnes.
- Enfin dans la couche dense, le nombre de paramètres est de 10000 (à partir de la couche d'aplatissement) et 1 pour le paramètre de biais, pour un total de 10001.

```
In [19]: model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['acc'])

         print(model.summary())

         Model: "sequential"
         _________________________________________________________________
         Layer (type)                 Output Shape              Param #
         =================================================================
         embedding (Embedding)        (None, 100, 100)          9254700
         _________________________________________________________________
         flatten (Flatten)            (None, 10000)             0
         _________________________________________________________________
         dense (Dense)                (None, 1)                 10001
         =================================================================
         Total params: 9,264,701
         Trainable params: 10,001
         Non-trainable params: 9,254,700
         _________________________________________________________________
         None
```

Etape 3 : Entrainement du modèle

- Dans cette étape, nous avons utilisé la méthode fit pour entraîner le réseau neuronal.
- L'entrainement spécifie à la fois le nombre d'époques sur lesquelles s'entraîner et la taille du batch.
- Epochs (nb_epoch) est le nombre de fois où le modèle est exposé à l'ensemble de données d'apprentissage (ici 6).
- La taille du batch (batch_size) est le nombre d'instances d'entraînement affichées au modèle avant qu'une mise à jour de poids ne soit effectuée. (ici 128)
- La validation_split de 0,2 signifie que 20% des données d'apprentissage sont utilisées pour trouver la précision d'apprentissage de l'algorithme.
- À la fin de l'entrainement, la précision de l'entrainement est d'environ 81,42%.

```
In [20]: history = model.fit(X_train, y_train, batch_size=128, epochs=6, verbose=1, validation_split=0.2)

         Epoch 1/6
         250/250 [==============================] - 1s 5ms/step - loss: 0.5998 - acc: 0.6743 - val_loss: 0.5444 - val_acc: 0.7245
         Epoch 2/6
         250/250 [==============================] - 1s 4ms/step - loss: 0.4951 - acc: 0.7600 - val_loss: 0.5323 - val_acc: 0.7330
         Epoch 3/6
         250/250 [==============================] - 1s 4ms/step - loss: 0.4613 - acc: 0.7841 - val_loss: 0.5247 - val_acc: 0.7376
         Epoch 4/6
         250/250 [==============================] - 1s 4ms/step - loss: 0.4375 - acc: 0.7956 - val_loss: 0.5308 - val_acc: 0.7379
         Epoch 5/6
         250/250 [==============================] - 1s 4ms/step - loss: 0.4176 - acc: 0.8083 - val_loss: 0.5345 - val_acc: 0.7380
         Epoch 6/6
         250/250 [==============================] - 1s 4ms/step - loss: 0.4093 - acc: 0.8142 - val_loss: 0.5345 - val_acc: 0.7408
```

Etape 4 : Evaluation du modèle

- Pour évaluer les performances du modèle, nous avons passé l'ensemble de test à la méthode d'évaluation de notre modèle.

```
In [21]: score = model.evaluate(X_test, y_test, verbose=1)

         313/313 [==============================] - 0s 1ms/step - loss: 0.5335 - acc: 0.7446
```

- Nous avons obtenu une précision de test de 74,46%. Notre précision d'entraînement était de 81,42%. Cela signifie que notre modèle est surajusté sur l'ensemble d'entraînement.

```
In [22]: print("Test Score:", score[0])
         print("Test Accuracy:", score[1])

         Test Score: 0.533453106880188
         Test Accuracy: 0.7445999979972839
```

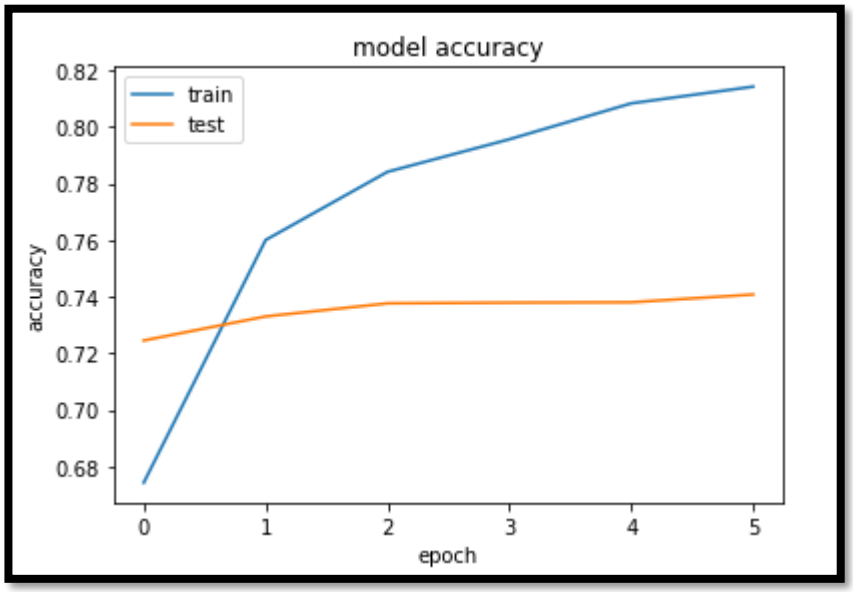
- Le surajustement se produit lorsque le modèle fonctionne mieux sur l'ensemble d'apprentissage que sur l'ensemble de test.
- Idéalement, la différence de performance entre les ensembles d'entraînement et de test devrait être minimale.

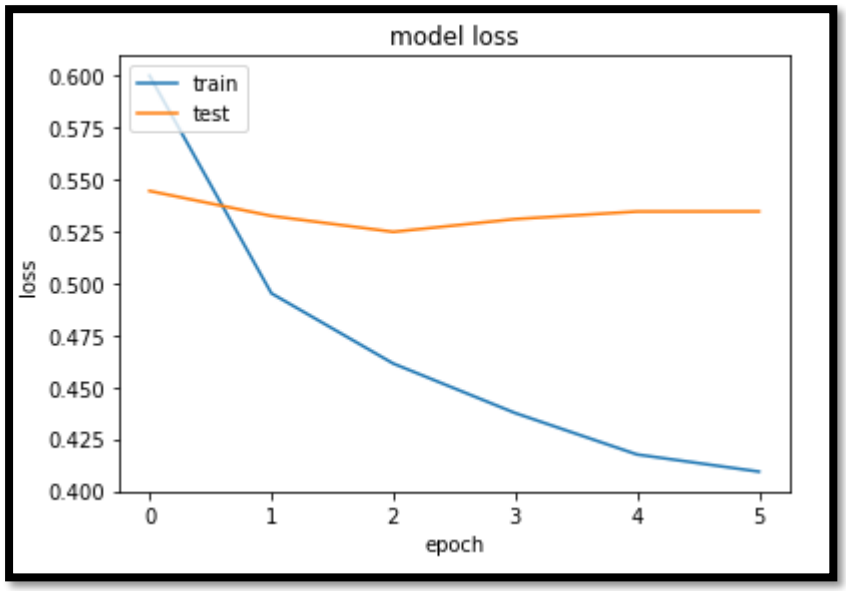
Etape 5 : Plot

- Dans cette partie, nous avons essayé de tracer les différences de perte et de précision pour les ensembles d'entraînement et de test.
- En utilisant les fonctions de pyplot de matplotlib.

```
In [23]: import matplotlib.pyplot as plt

         plt.plot(history.history['acc'])
         plt.plot(history.history['val_acc'])

         plt.title('model accuracy')
         plt.ylabel('accuracy')
         plt.xlabel('epoch')
         plt.legend(['train','test'], loc='upper left')
         plt.show()

         plt.plot(history.history['loss'])
         plt.plot(history.history['val_loss'])

         plt.title('model loss')
         plt.ylabel('loss')
         plt.xlabel('epoch')
         plt.legend(['train','test'], loc='upper left')
         plt.show()
```

- Nous remarquons une grande différence de perte et de précision entre les ensembles de formation et de test.





c) <u>Exemple de prédiction</u>

```
#Prediction
instance = X[200]
print(instance)
```

```
Interesting and short television movie describes some of the machinations surrounding Jay Leno replacing Carson as host of the
Tonight Show Film is currently very topical given the public drama surrounding Conan Brien and Jay Leno The film does good job
of sparking viewers interest in the events and showing some of the concerns of the stakeholders particularly of the NBC executi
ves The portrayal of Ovitz was particularly compelling and interesting thought Still many of the characters were only very brie
fly limned or touched upon and some of the acting seemed perfunctory Nevertheless an interesting story
```

```
instance = tokenizer.texts_to_sequences(instance)

flat_list = []
for sublist in instance:
    for item in sublist:
        flat_list.append(item)

flat_list = [flat_list]

instance = pad_sequences(flat_list, padding='post', maxlen=maxlen)

model.predict(instance)
```

```
array([[0.9997733]], dtype=float32)
```

<u>Résultat:</u> Comme la sortie est supérieur à 0.5 → La critique est positive.

2 Convolutional Neural Network:

a) Définition :

En apprentissage automatique, un réseau de neurones convolutifs ou réseau de neurones à convolution (en anglais CNN) est un type de réseau de neurones artificiels acycliques (feed-forward), dans lequel le motif de connexion entre les neurones est inspiré par le cortex visuel des animaux. Les neurones de cette région du cerveau sont arrangés de sorte qu'ils correspondent à des régions qui se chevauchent lors du pavage du champ visuel. Leur fonctionnement est inspiré par les processus biologiques, ils consistent en un empilage multicouche de perceptrons, dont le but est de prétraiter de petites quantités d'informations. Les réseaux neuronaux convolutifs ont de larges applications dans la reconnaissance d'image et vidéo, les systèmes de recommandation et le traitement du langage naturel.

b) Exécution et explication du code :

Voilà le modèle qu'on a créé :

- Pour Pooling on a choisi maxpooling.
- Les fonctions d'activations pour les couches cachées : relu (convergence rapide par rapport au sigmoid).
- Pour la couche output : sigmoid puisqu'on a classification binaire.
- L'algorithme utilise est adam : combinaison entre RMSprop et Momentum (très rapide)
- Loss function : binary_crossentropy

In [1]

```
from keras.layers.convolutional import Conv1D
model = Sequential()

embedding_layer = Embedding(vocab_size, 100, weights=[embedding_matrix], input_length=maxlen , trainable=False)
model.add(embedding_layer)

model.add(Conv1D(128, 5, activation='relu'))
model.add(GlobalMaxPooling1D())
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['acc'])
```

L'affichage du sommaire de modèle :

- Il y a 92547 mots dans notre corpus.
- Chaque mot est représenté comme un vecteur de 100
- Dimensions.
- Le nombre de paramètres ferroviaires sera dans la couche D'intégration.

- Dans la couche aplatissante, nous multiplions simplement les lignes et les colonnes.
- Enfin, dans la couche dense, le nombre de paramètres est de 10000 (de la couche aplatissante) et 1 pour le paramètre de biais, pour un total de 10001.

In [2]

```
print(model.summary())
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_4 (Embedding)      (None, 100, 100)          9254700
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 96, 128)           64128
_________________________________________________________________
global_max_pooling1d_1 (Glob (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 129
=================================================================
Total params: 9,318,957
Trainable params: 64,257
Non-trainable params: 9,254,700
_________________________________________________________________
None
```

Maintenant on forme et on évalue notre modèle. Pour ce faire, on utilise les méthodes et les méthodes respectivement **Fit** et **evaluate**.

In [3]

```
history = model.fit(X_train, y_train, batch_size=128, epochs=6, verbose=1, validation_split=0.2)

score = model.evaluate(X_test, y_test, verbose=1)
```

```
Train on 32000 samples, validate on 8000 samples
Epoch 1/6
32000/32000 [==============================] - 44s 1ms/step - loss: 0.4824 - acc: 0.7667 - val_loss: 0.4097 - val_acc: 0.8100
Epoch 2/6
32000/32000 [==============================] - 44s 1ms/step - loss: 0.3626 - acc: 0.8432 - val_loss: 0.3820 - val_acc: 0.8244
Epoch 3/6
32000/32000 [==============================] - 44s 1ms/step - loss: 0.3131 - acc: 0.8704 - val_loss: 0.3715 - val_acc: 0.8322
Epoch 4/6
32000/32000 [==============================] - 39s 1ms/step - loss: 0.2778 - acc: 0.8872 - val_loss: 0.3460 - val_acc: 0.8456
Epoch 5/6
32000/32000 [==============================] - 40s 1ms/step - loss: 0.2390 - acc: 0.9100 - val_loss: 0.3437 - val_acc: 0.8468
Epoch 6/6
32000/32000 [==============================] - 44s 1ms/step - loss: 0.2073 - acc: 0.9281 - val_loss: 0.3424 - val_acc: 0.8511
10000/10000 [==============================] - 4s 388us/step
```

Test accuracy

In [4]

```
print("Test Score:", score[0])
print("Test Accuracy:", score[1])
```

```
Test Score: 0.5375254062652588
Test Accuracy: 0.7439000010490417
```

Train accuracy

In [5]

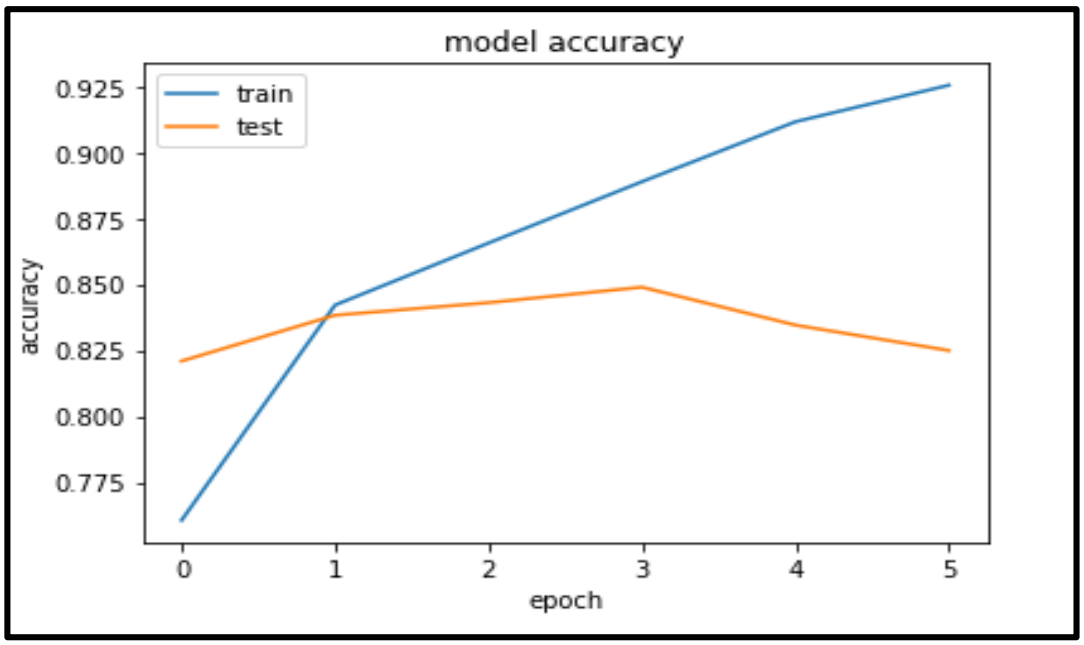
```
print("Train Score:", scoreTrain[0])
print("Train Accuracy:", scoreTrain[1])
```

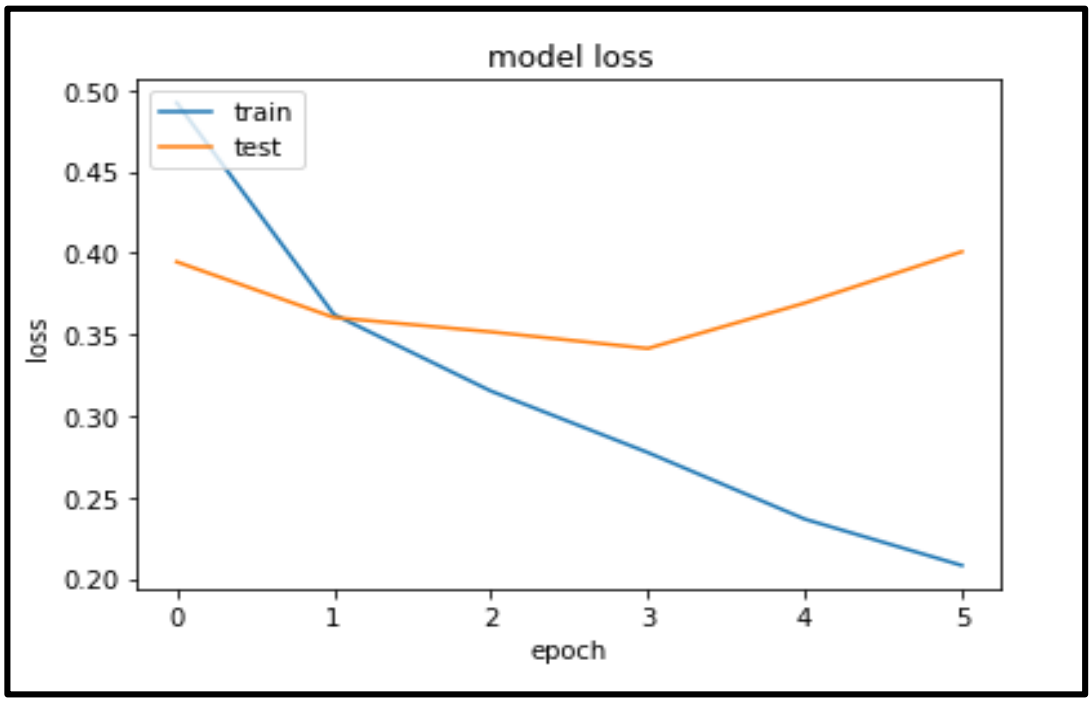
```
Train Score : 0.25041067600250244
Train Accuracy : 0.89267498254776
```

Maintenant c'est mieux d'afficher l'erreur pour chaque Époque pour voir est ce que notre model a bien appris :

In [6]

```
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])

plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train','test'], loc = 'upper left')
plt.show()

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])

plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train','test'], loc = 'upper left')
plt.show()
```





Constat : Notre modèle a très bien fonctionné sur l'ensemble d'entraînement mais si nous voyons les pertes (losses), il est à peu près sûr qu'il souffre d'une variance élevée car test loss est bien plus grande que train loss.

Prédiction sur une seule instance

```
instance = X[200]
print(instance)
```

```
Interesting and short television movie describes some of the machinations surrounding Jay Leno replacing Carson as host of the Tonight Show Film is currently very topical given the public drama surrounding Conan Brien and Jay Leno The film does good job of sparking viewers interest in the events and showing some of the concerns of the stakeholders particularly of the NBC executi ves The portrayal of Ovitz was particularly compelling and interesting thought Still many of the characters were only very brie fly limned or touched upon and some of the acting seemed perfunctory Nevertheless an interesting story
```

```
instance = tokenizer.texts_to_sequences(instance)

flat_list = []
for sublist in instance:
    for item in sublist:
        flat_list.append(item)

flat_list = [flat_list]

instance = pad_sequences(flat_list, padding='post', maxlen=maxlen)

model.predict(instance)
```

```
array([[0.7598282]], dtype=float32)
```

Résultat : La sortie > 0.5 → La critique est positive

3 Long – Short Term Memory

- **Définition :**

Long Short-Term Memory est une version avancée de l'architecture récurrente du réseau neuronal (RNN) : c'est un réseau de neurones artificiels présentant des connexions récurrentes .
En effet, il travaille sur l'entrée actuelle en tenant compte de la sortie précédente (rétroaction) et en stockant dans sa mémoire pendant une courte période de temps (mémoire à court terme).
Parmi ses diverses applications, les plus populaires sont dans les domaines du traitement de la parole, du contrôle non markovien et de la composition musicale.
Afin de modéliser des dépendances à très long terme, il est nécessaire de donner aux réseaux de neurones récurrents la capacité de maintenir un état sur une longue période de temps.

D'où l'apparition du LSTM, qui possèdent une mémoire interne
appelée cellule permettant de maintenir un état aussi longtemps que nécessaire.

Il est largement utilisé en traitement du langage naturel et c'est la raison derrière notre choix comme modèle de classification de texte.

- **Exécution et explication du code :**

On initialise un modèle séquentiel, suivi de la couche d'intégration. Ensuite, nous créons une couche LSTM avec 128 neurones.

Enfin, on ajoute une couche dense avec activation sigmoïde et On compile notre modèle avec l'optimiseur : adam ,binary_crossentropy accuracy, comme notre fonction de perte et comme mesures, puis on imprime le résumé de notre modèle .

Et on affiche le résume du modèle.

In [1]
```
from keras.layers import GRU, LSTM
model = Sequential()
embedding_layer = Embedding(vocab_size, 100, weights=[embedding_matrix], input_length=maxlen , trainable=False)
model.add(embedding_layer)
model.add(LSTM(128))

model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['acc'])
```

In [2]

```
print(model.summary())
```

Out [2]

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_6 (Embedding)      (None, 100, 100)          9254700
_________________________________________________________________
lstm_1 (LSTM)                (None, 128)               117248
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 129
=================================================================
Total params: 9,372,077
Trainable params: 117,377
Non-trainable params: 9,254,700
_________________________________________________________________
None
```

On forme notre modèle, en travaillant sur 20 des données d'entrainement, puis on évalue les performances.

In [3]

```
history = model.fit(X_train, y_train, batch_size=128, epochs=6, verbose=1, validation_split=0.2)

score = model.evaluate(X_test, y_test, verbose=1)
```

Afin de la formation, on voit bien que la Précision de la Formation est d'environ 85,40 %.

Out [3]

```
Train on 32000 samples, validate on 8000 samples
Epoch 1/6
32000/32000 [==============================] - 149s 5ms/step - loss: 0.5487 - acc: 0.7181 - val_loss: 0.4513 - val_acc: 0.7950
Epoch 2/6
32000/32000 [==============================] - 139s 4ms/step - loss: 0.4371 - acc: 0.7973 - val_loss: 0.4098 - val_acc: 0.8086
Epoch 3/6
32000/32000 [==============================] - 141s 4ms/step - loss: 0.3899 - acc: 0.8245 - val_loss: 0.3958 - val_acc: 0.8152
Epoch 4/6
32000/32000 [==============================] - 127s 4ms/step - loss: 0.3667 - acc: 0.8371 - val_loss: 0.3669 - val_acc: 0.8316
Epoch 5/6
32000/32000 [==============================] - 133s 4ms/step - loss: 0.3450 - acc: 0.8468 - val_loss: 0.3439 - val_acc: 0.8476
Epoch 6/6
32000/32000 [==============================] - 129s 4ms/step - loss: 0.3258 - acc: 0.8602 - val_loss: 0.3378 - val_acc: 0.8533
10000/10000 [==============================] - 20s 2ms/step
```

On vérifie la précision et la perte du test

In [4]

```
print("Test Score:", score[0])
print("Test Accuracy:", score[1])
```

Notre précision de test est d'environ 85,07%

Out [4]

```
Test Score: 0.34200601801872255
Test Accuracy: 0.8507999777793884
```

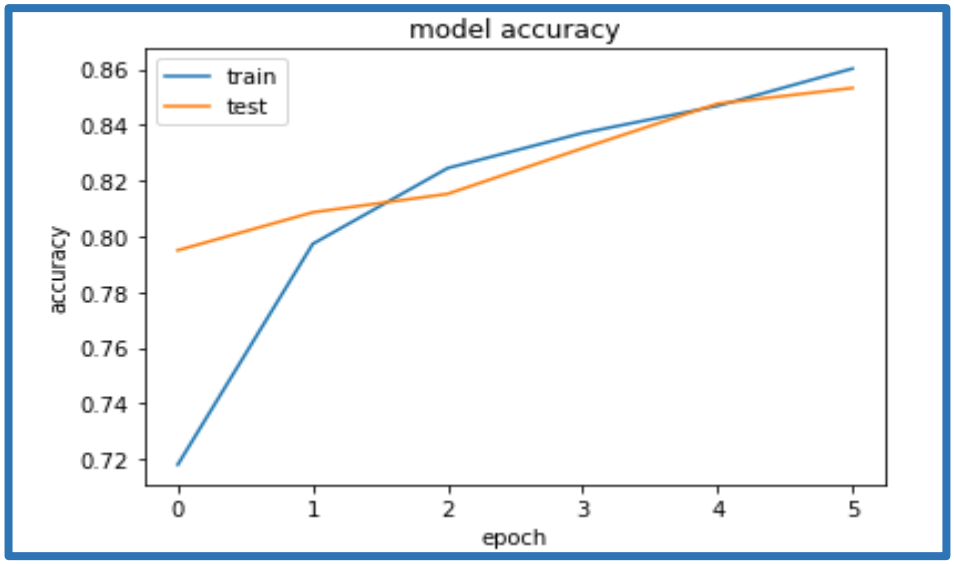
On trace les différences de perte et de précision entre
L'entraînement et les ensembles de tests.

In [5]

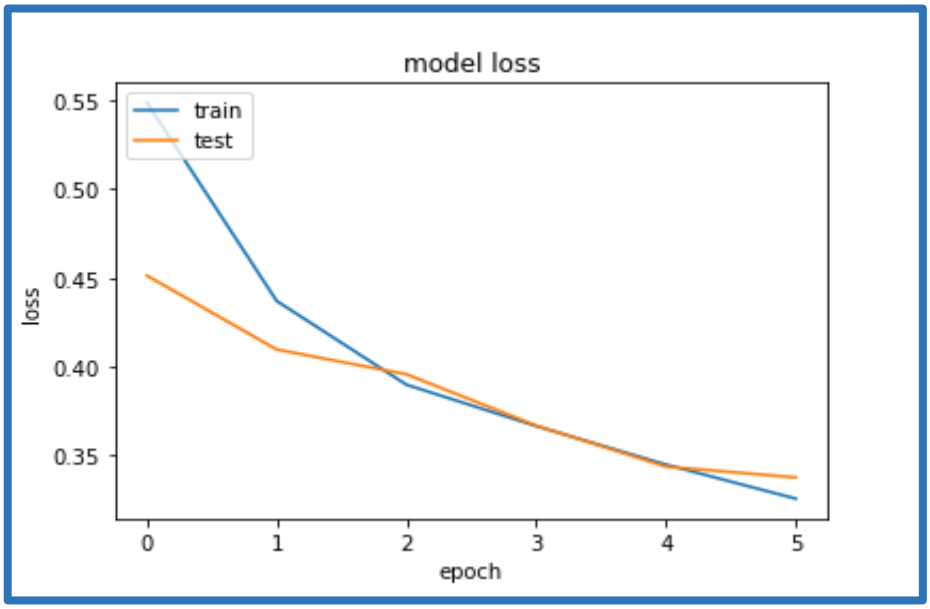
```
import matplotlib.pyplot as plt

plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])

plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train','test'], loc='upper left')
plt.show()

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])

plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train','test'], loc='upper left')
plt.show()
```

On obtient ces 2 graphes en sortie

Out [5]



Out [5]



Prédiction sur une seule instance :

On a sur un exemple pour la prédiction et on a obtenu le résultat suivant :

```
instance = X[57]
print(instance)
```

```
I laughed all the way through this rotten movie It so unbelievable woman leaves her husband after many years of marriage has br
eakdown in front of real estate office What happens The office manager comes outside and offers her job Hilarious Next thing yo
u know the two women are going at it Yep they re lesbians Nothing rings true in this Lifetime for Women with nothing better to
do movie Clunky dialogue like don want to spend the rest of my life feeling like had chance to be happy and didn take it doesn
help There a wealthy distant mother who disapproves of her daughter new relationship sassy black maid unbelievable that in the
year film gets made in which there a sassy black maid Hattie McDaniel must be turning in her grave The woman has husband who fr
eaks out and wants custody of the snotty teenage kids Sheesh No cliche is left unturned
```

```
instance = tokenizer.texts_to_sequences(instance)

flat_list = []
for sublist in instance:
    for item in sublist:
        flat_list.append(item)

flat_list = [flat_list]

instance = pad_sequences(flat_list, padding='post', maxlen=maxlen)

model.predict(instance)
```

```
array([[0.45174888]], dtype=float32)
```

Résultat : La sortie ≤ 0.5 → La critique est négative

4 Gated Reccurent Units

- **Introduction :**

Nous avons utilisé dans les divers TPs précédents les réseaux de neurones pour résoudre des problèmes de classification et de régression, mais nous n'avons pas encore vu comment les réseaux de neurones peuvent traiter les informations de séquence. Tout comme les CNNs étaient plus efficaces pour les données d'images 2D, les RNNs sont plus efficaces pour les données de séquence par exemple :

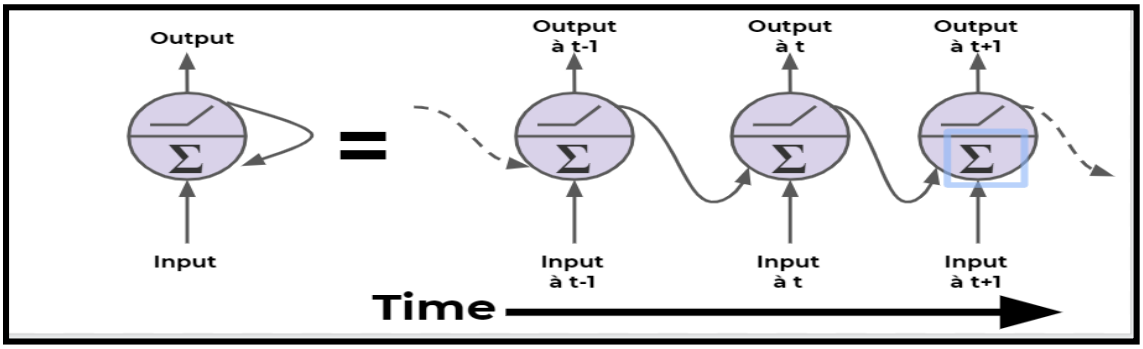
  - Données de Time Series (Ventes)
  - Phrases
  - Audio
  - Trajectoires de Voiture

- **Définition RNN :**

Un réseau de neurones récurrent (RNN, recurrent neural network) est un type de réseau de neurones artificiels principalement utilisé dans la reconnaissance vocale et le traitement automatique du langage naturel (TAL, NLP, TNL). Les RNN sont conçus de manière à reconnaître les caractéristiques séquentielles et les modèles d'utilisation des données requis pour prédire le scénario suivant le plus probable.

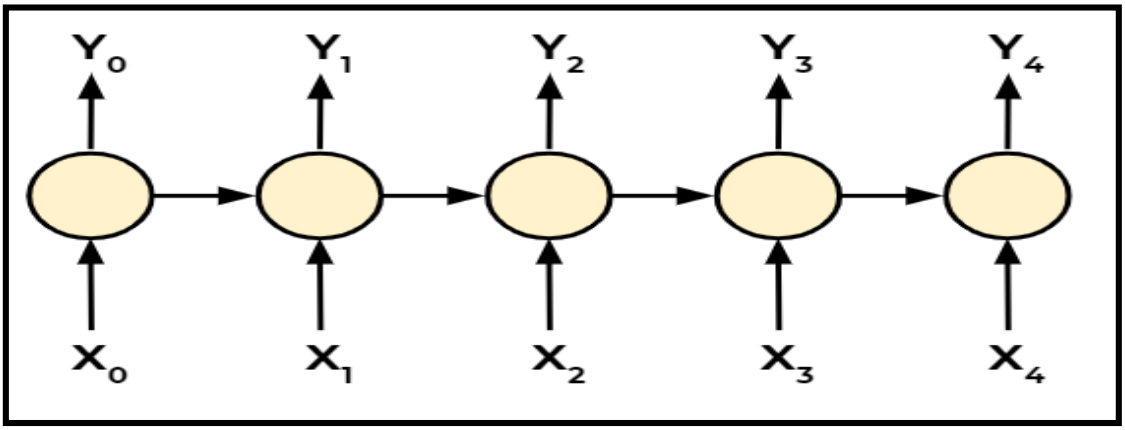
- **Fonctionnement de RNN :**

Un neurone récurrent Renvoie la sortie (output) à elle-même
Voyons à quoi cela ressemble dans le temps :

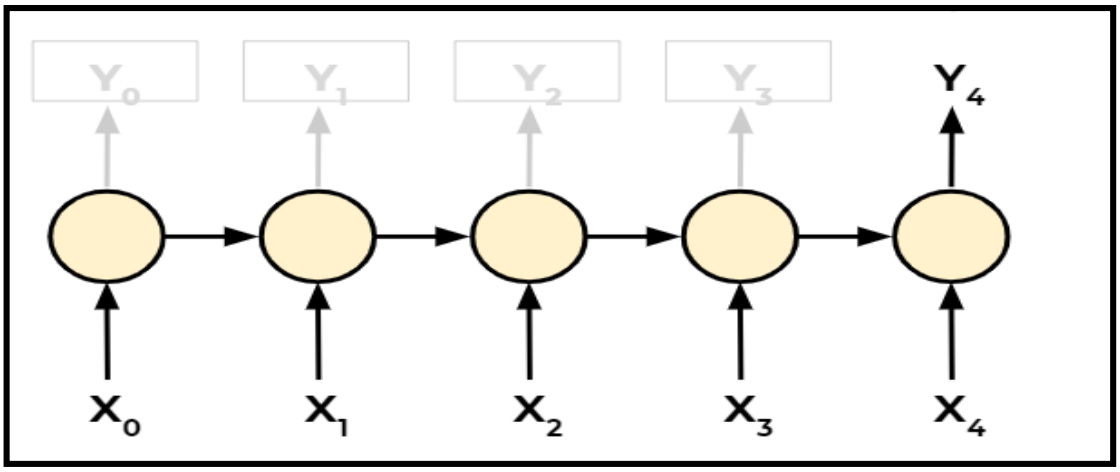


- **Types de RNN :**

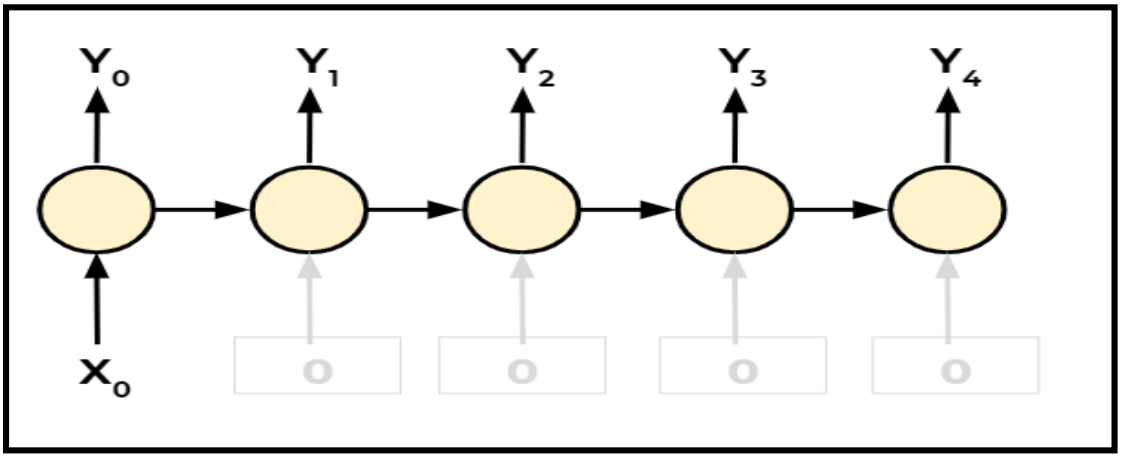
Séquence à Séquence (Many to Many) :



Séquence à Vecteur (Many to One) :



Vecteur vers Séquence (One to Many) :



- **Problème de RNN :**

le RNN souffre du problème de dissipation du gradient ,en effet Pour “apprendre”, un RNN utilise La backpropagation qui va à l'envers de la couche de sortie vers la couche d'entrée, en propageant le gradient d'erreur.Pour les réseaux plus profonds, les problèmes peuvent provenir de la backpropagation, des vanishing et exploding gradients !En remontant vers les couches "inférieures", les gradients deviennent souvent plus petits,

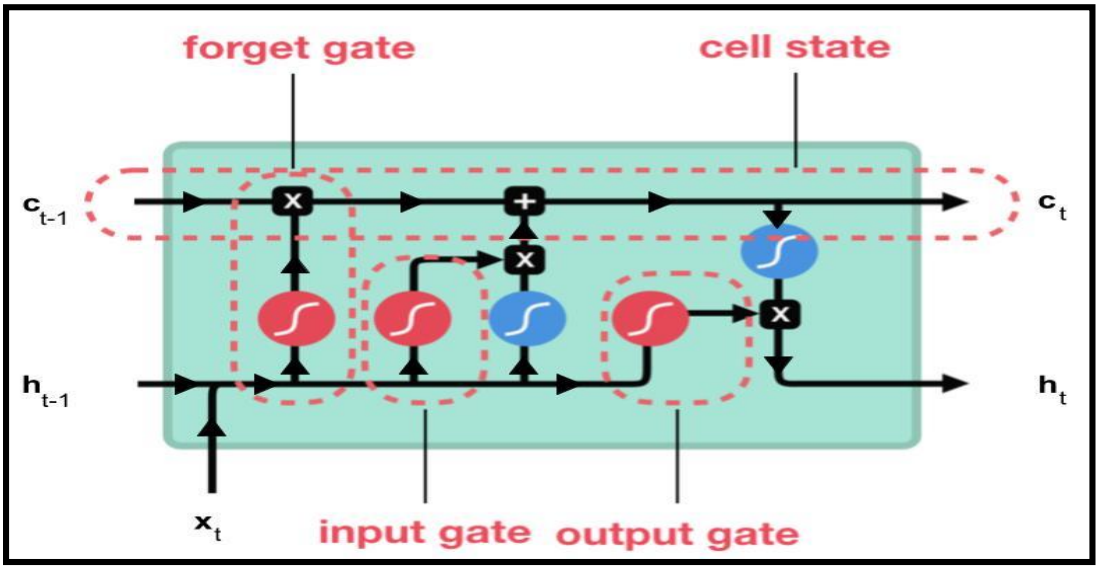
ce qui fait que les poids ne changent jamais aux niveaux inférieurs.L'inverse peut aussi se produire, des gradients explosent au retour, causant des problèmes.

Et par conséquent, le RNN peut facilement oublier des données un petit peu anciennes (ou des mots assez éloignés du mot courant dans un texte) lors de la phase d’apprentissage : sa mémoire est courte.

- **Vers des solutions meilleures :**

Les LSTM et GRU ont été créés comme méthode permettant de gérer efficacement la mémoire à court et long terme grâce à leurs systèmes de portes. Mais comment fonctionnent généralement ses structures ?

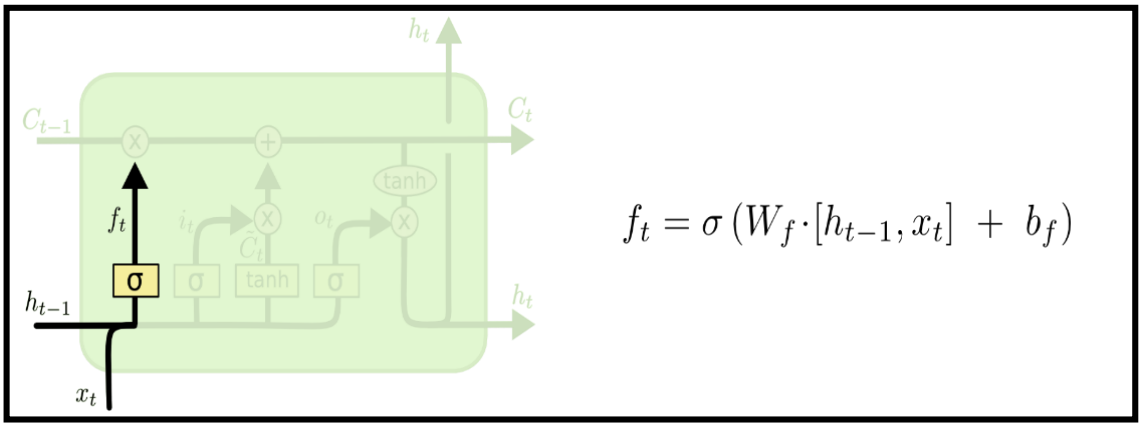
- **LSTM (Long Short Term Memory):**



Les réseaux neuronaux LSTM sont des RNNs qui permettent de limiter le Vanishing Gradient problem par un ensemble de 3 portes (gates). On peut distinguer trois types de portes :

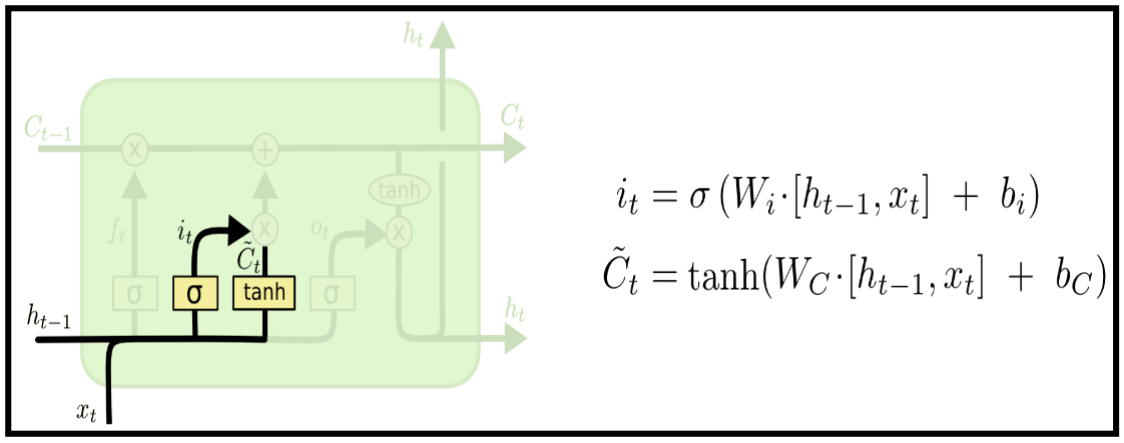
- **Forget gate layer :**

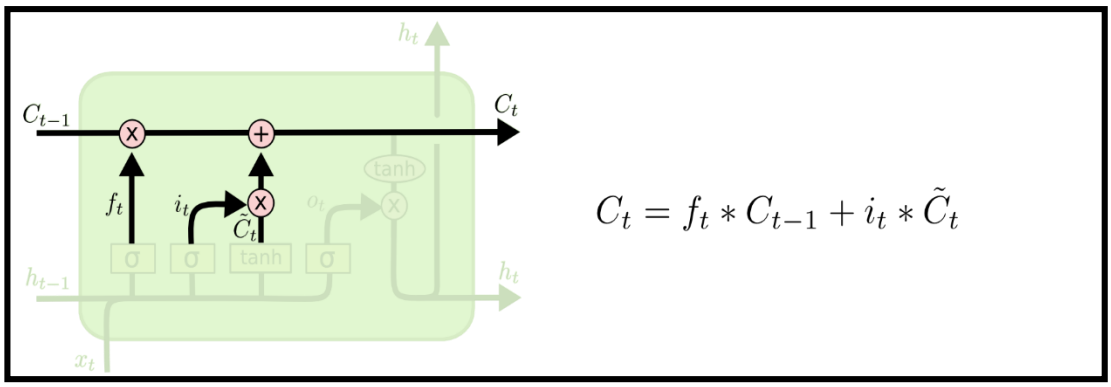
Cette porte décide de quelle information doit être conservée ou jetée : l'information de l'état caché précédent est concaténé à la donnée en entrée (par exemple le mot "des" vectorisé) puis on y applique la fonction sigmoïde afin de normaliser les valeurs entre 0 et 1. Si la sortie de la sigmoïde est proche de 0, cela signifie que l'on doit oublier l'information et si on est proche de 1 alors il faut la mémoriser pour la suite.



- **Input gate :**

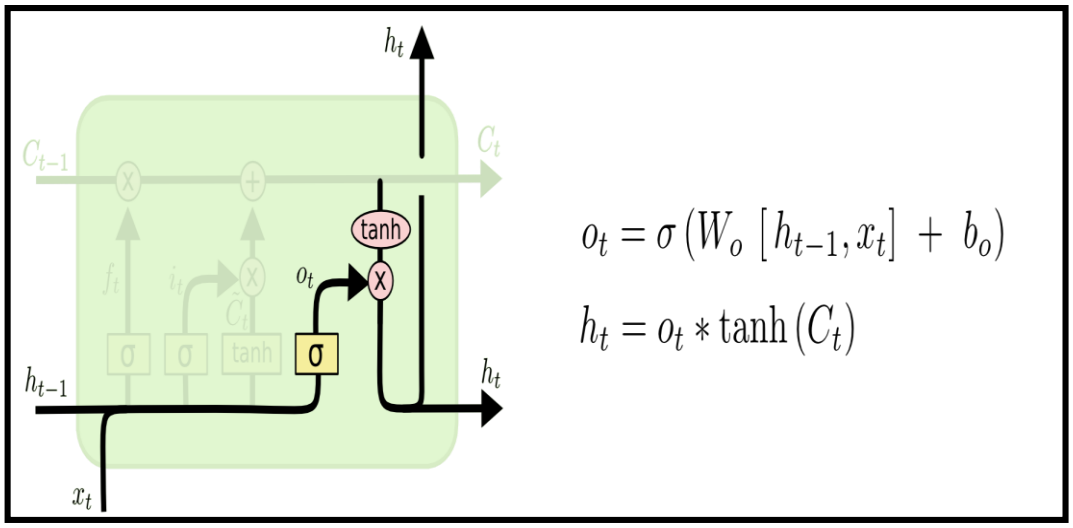
  À présent que le LSTM a éliminé ce qu'il ne souhaitait pas se souvenir, il doit décider de ce qu'il veut garder, cad ajouter au vecteur d'état de cellule. L'état de la cellule se calcule assez simplement à partir de la porte d'oubli et de la porte d'entrée : d'abord on multiplie coordonnée à coordonnée la sortie de l'oubli avec l'ancien état de la cellule. Cela permet d'oublier certaines informations de l'état précédent qui ne servent pas pour la nouvelle prédiction à faire. Ensuite, on additionne le tout (coordonnée à coordonnée) avec la sortie de la porte d'entrée, ce qui permet d'enregistrer dans l'état de la cellule ce que le LSTM (parmi les entrées et l'état caché précédent) a jugé pertinent.Le schéma suivant explicite comment calculer ce qui doit lui être ajouté**.**





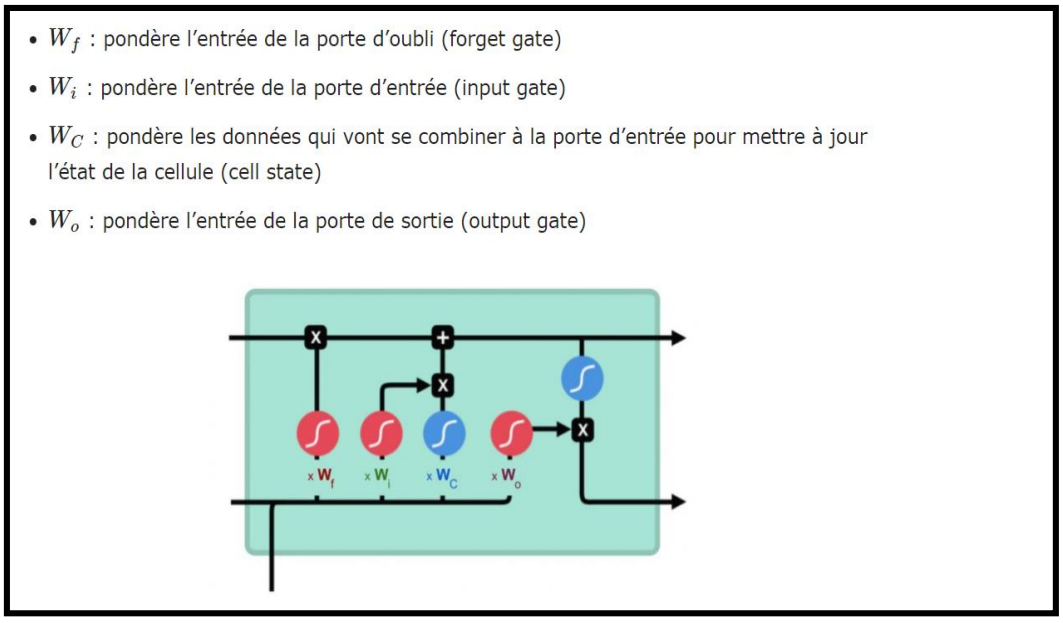
- **Output gate :**

La porte de sortie doit décider de quel sera le prochain état caché, qui contient des informations sur les entrées précédentes du réseau et sert aux prédictions. Pour ce faire, le nouvel état de la cellule calculé juste avant est normalisé entre -1 et 1 grâce à tanh. Le vecteur concaténé de l'entrée courante avec l'état caché précédent passe, pour sa part, dans une fonction sigmoïde dont le but est de décider des informations à conserver (proche de 0 signifie que l'on oublie, et proche de 1 que l'on va conserver cette coordonnée de l'état de la cellule).
Tout cela peut sembler magique en ce sens où on dirait que le réseau doit deviner ce qu'il doit retenir dans un vecteur à la volée, mais rappelons bien qu'une matrice de poids est appliquée en entrée. C'est cette matrice qui va, concrètement, stocker le fait que telle information est importante ou non à partir des milliers d'exemples qu'aura vu le réseau !

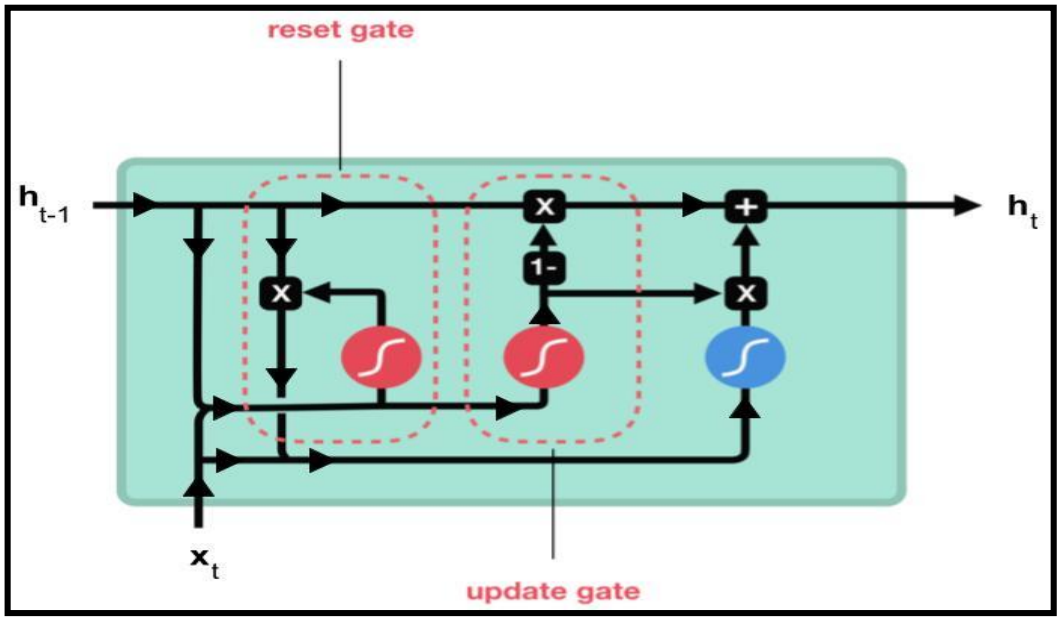


- <u>Apprentissages de LSTM :</u>

Les entrées de chaque porte sont pondérées par des poids liés aux portes ainsi que par un biais on distingue :



- GRU (Gated Recurrent Unit):

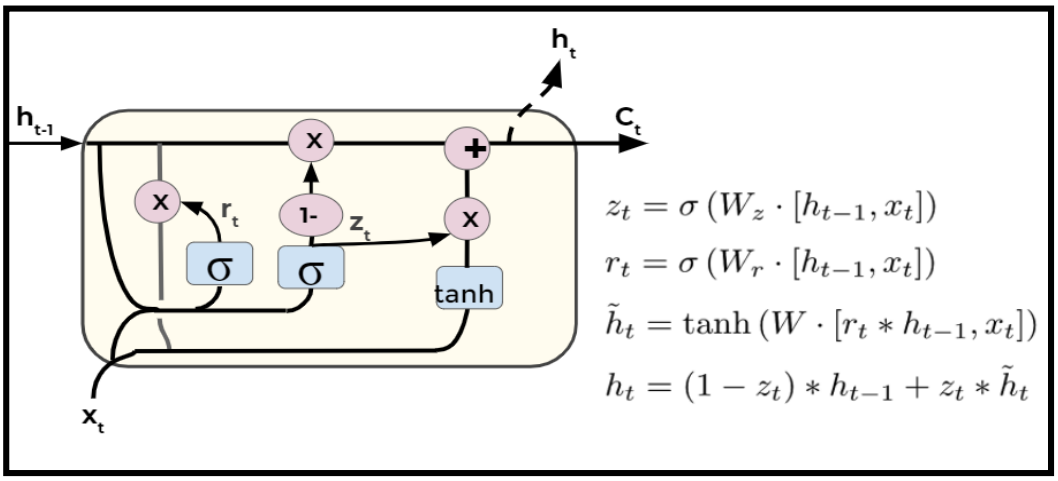


Les GRUs sont similaires aux LSTM mais ils n'ont pas de vecteur d'état de cellule et ils ont seulement deux portes (gates), une porte de réinitialisation r (reset gate) et une porte de mise à jour z (update gate):

Porte de reset (reset gate) :
Cette porte sert à contrôler combien d'information passée le réseau doit oublier. L'état caché précédent, concaténé avec les données d'entrée, passe par une sigmoïde (pour ne conserver que les coordonnées pertinentes) puis est multiplié par l'ancien état caché : on n'en conserve donc que les coordonnées importantes (telles qu'elles) de l'état précédent (on a donc perdu une partie de l'état précédent dans cette porte).

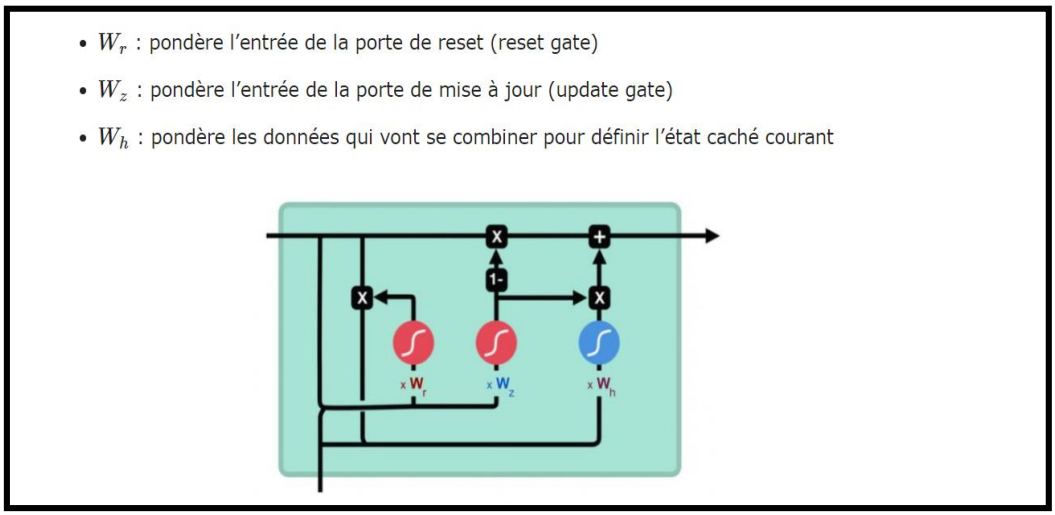
Porte de mise à jour (update gate) :
Cette porte agit exactement de la même manière que les portes oublis et d'entrée du LSTM : elle décide des informations à conserver et de celles à oublier. Les données d'entrées et l'ancien état caché sont concaténés et passent par une fonction sigmoïde dont le rôle est de déterminer quelles sont les composantes importantes.
La sortie du réseau GRU :
L'état caché précédent (partiellement effacé par la porte de reset) est combiné avec l'entrée du réseau et normalisé par un tanh entre -1 et 1. On vient ensuite annuler toutes ses coordonnées jugées "inutiles pour les prédictions" (grâce à la sortie de la

porte de mise à jour), puis on y ajoute les coordonnées de l'état caché précédent jugées "inutiles" (en ayant, cette fois, annulé toutes les coordonnées pertinentes).



- <u>Apprentissages de GRU:</u>

  Voici un schéma de des matrices de poids qui viennent pondérer les vecteurs entrant dans les portes.



- LSTM ou GRU:

  Il est clair qu'à travers les équations précédentes les LSTM ont une porte de mise à jour et une porte d'oubli séparées . Cela rend clairement les LSTM plus sophistiqués mais en même temps plus complexes. Il n'y a pas de moyen simple de décider lequel utiliser pour votre cas d'utilisation particulier. Vous devez toujours faire des essais et examiner les erreurs pour tester les performances. Cependant, étant donné que GRU est plus simple que LSTM, les GRU prendront beaucoup moins de temps à se former et seront plus efficaces.

- GRU et l'analyse des sentiments :

  L'analyse des sentiments est le processus d'extraction de l'attitude des visiteurs envers l'entité où le texte écrit. L'analyse des sentiments le plus simple est une tâche de classification binaire ce qui implique de décider si un texte est positif ou négatif. Elle est effectuée à la fois au niveau de la phrase et au niveau des documents ou paragraphes. Les deux niveaux offrent des défis et nécessitent donc différentes techniques.

  Il y a une disponibilité croissante de divers sites Web qui permet aux clients d'écrire des avis sur divers produits, films ou hôtels. Dans les avis, les clients partagent leur expérience d'utiliser ce service particulier. L'analyse des sentiments est appliquée à ces avis pour découvrir l'opinion des clients à ce sujet. Ceci est d'une valeur importante pour les entreprises.

- Optimisation de GRU:

  Le réseau GRU peut être optimisé comme suit:

  5 Choix d'un meilleur optimisateur :

  Une variété d'optimisateurs modernes ont été développés récemment qui ont à la fois des taux de convergence plus rapides et également une meilleure précision. Nous pouvons utiliser plusieurs types afin d'évaluer le plus performant.Parmi les plus connu pour ce type de tâche est Ada delta.

- **Ajout de couches de Dropout :**

C'est une méthode très efficace pour prévenir le surentraînement en réseaux de neurones. Cette méthode consiste à « désactiver » des sorties de neurones aléatoirement (avec une probabilité prédéfinie pendant la phase d'apprentissage).

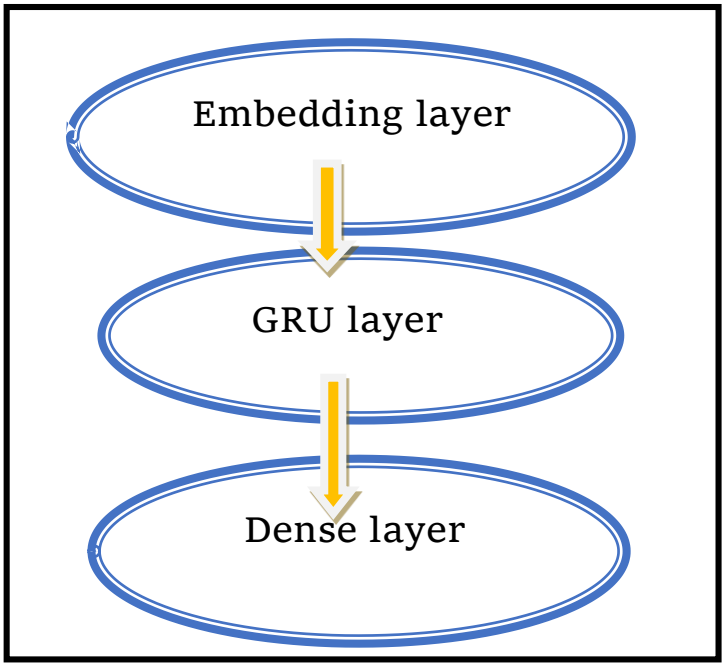
- **INITIALISATION ORTHOGONALE :**

Il a été montré que l'initialisation des matrices de poids avec les matrices orthogonales aléatoires fonctionnent mieux que les matrices aléatoires gaussiennes (ou uniformes).

- **Implémentation du modèle** :

Dans cette partie on va effectuer l'implémentation d'un réseau GRU pour l'analyse des commentaires sur des films en utilisant la dataset du IMBD.

- **La structure du modèle :**



- Exécution et explication du code :

Dans le script ci-dessous, nous créons un modèle Sequential (). Ensuite, nous créons notre Embdedding Layer. La

couche d'intégration « Embdedding Layer » aura une longueur d'entrée de 100, la dimension du vecteur de sortie sera également de 100. Étant donné que nous n'entraînons pas nos propres incorporations et que nous n'utilisons pas l'intégration GloVe, nous définissons l'entraînement sur Faux et dans l'attribut de poids nous transmettons notre propre embedding matrix. Ensuite on ajoute notre GRU model avec 128 neuron.Finalement on ajoute une couche dense avec un seul neuron(une seule sortie) en utilisant une fonction d'activation sigmoid.

```
#création du modèle:
model = Sequential()
embedding_layer = Embedding(vocab_size, 100, weights=[embedding_matrix], input_length=maxlen , trainable=False)
model.add(embedding_layer)
model.add(GRU(128))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['acc'])
```

Sommaire du modèle

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 100, 100)          9254700
_________________________________________________________________
gru (GRU)                    (None, 128)               88320
_________________________________________________________________
dense (Dense)                (None, 1)                 129
=================================================================
Total params: 9,343,149
Trainable params: 88,449
Non-trainable params: 9,254,700
_________________________________________________________________
```

Entrainement du modèle du modèle :

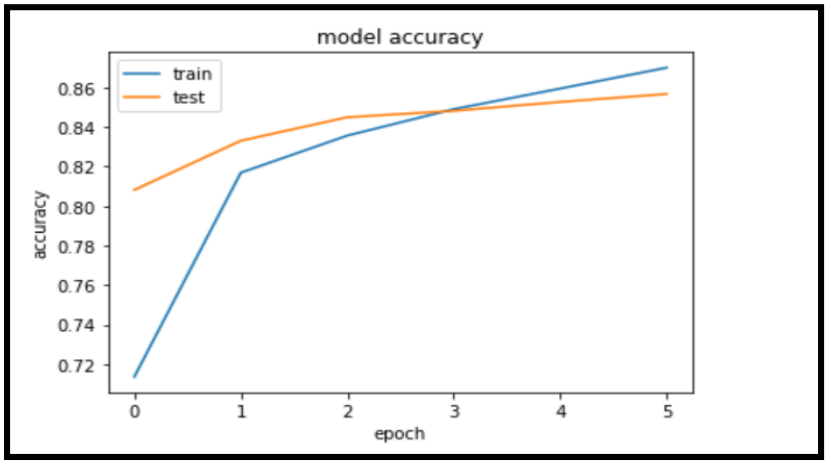
Dans le scipt ci-dessus, nous utilisons la méthode fit pour former notre réseau neuronal et la méthode evaluate pour évaluer les performances du modèle, 20% des données d'apprentissage sont utilisées pour trouver la précision d'apprentissage de l'algorithme.

```
history = model.fit(X_train, y_train, batch_size=128, epochs=6, verbose=1, validation_split=0.2
score = model.evaluate(X_test, y_test, verbose=1)

Epoch 1/6
250/250 [==============================] - 34s 136ms/step - loss: 0.5353 - acc: 0.7137 - val_loss: 0.4185 - val_ac
c: 0.8081
Epoch 2/6
250/250 [==============================] - 25s 99ms/step - loss: 0.4031 - acc: 0.8168 - val_loss: 0.3716 - val_acc:
0.8330
Epoch 3/6
250/250 [==============================] - 27s 110ms/step - loss: 0.3659 - acc: 0.8356 - val_loss: 0.3491 - val_ac
c: 0.8449
Epoch 4/6
250/250 [==============================] - 27s 109ms/step - loss: 0.3406 - acc: 0.8490 - val_loss: 0.3407 - val_ac
c: 0.8481
Epoch 5/6
250/250 [==============================] - 28s 112ms/step - loss: 0.3223 - acc: 0.8594 - val_loss: 0.3316 - val_ac
c: 0.8526
Epoch 6/6
250/250 [==============================] - 28s 113ms/step - loss: 0.2987 - acc: 0.8699 - val_loss: 0.3269 - val_ac
c: 0.8566
313/313 [==============================] - 4s 13ms/step - loss: 0.3217 - acc: 0.8589
```

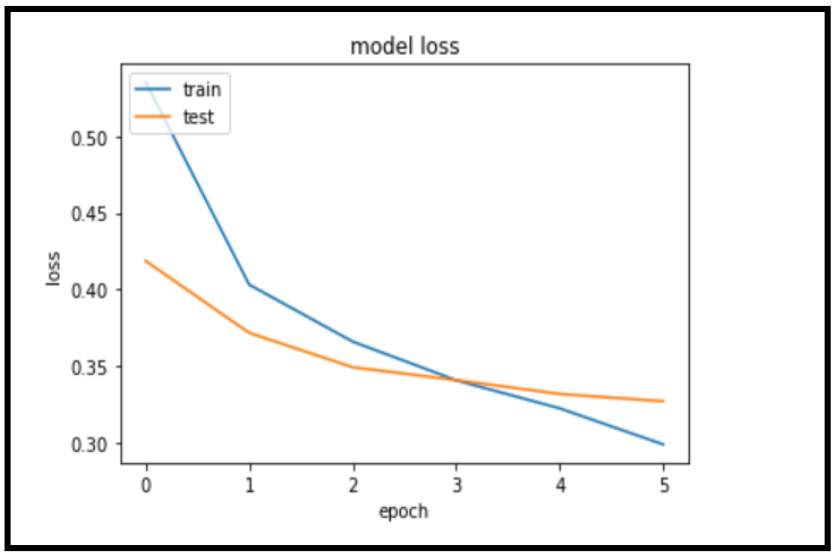
Evaluation du modèle :

```
print("Test Score:", score[0])
print("Test Accuracy:", score[1])

Test Score: 0.321696400642395
Test Accuracy: 0.8589000105857849
```

Nous pouvons voir à partir des résultat précédents qu'il y a une très petite différence entre la précision d'entraînement et la précision du test, ce qui signifie qu'il n'y a pas de sur apprentissage dans le modèle implémenter .Pour s'assurer encore on va faire le plot des différents métriques de perte et de précision entre les ensembles d'entraînement et de test.



L'affichage des métriques :



Prédiction sur une seule instance :

Après avoir créé le modèle il sera intéressant de l'utiliser sur un commentaire choisi aléatoirement afin de savoir est ce que le modèle est capable de bien prédire le résultat correct. Pour cela on a choisi le commentaire suivant

```
If you like original gut wrenching laughter you will like this movie If you are young or old then you will love thi
s movie hell even my mom liked it Great Camp
```

Il s'agit bien d'un commentaire positif et est
ce que le modèle sera capable de bien prédire c'est ce qu'on va voir.

```
model.predict(instance)
```

```
array([[0.9649406]], dtype=float32)
```

le modèle a bien prédis le resultat il s'agit bien d'un commentaire positif

Résultat : Après avoir effectué la prédiction on constate que notre modèle a bien prédis le résultat puisque à la sortie de la fonction sigmoïde on a obtenu 0 ,96 qui indique un résultat positif

III] Evaluation des performances :

Analyse des résultats :

Pour une meilleure comparaison, il n'y a rien de bon qu'un tableau récapitulatif où l'on peut voir tous les résultats :

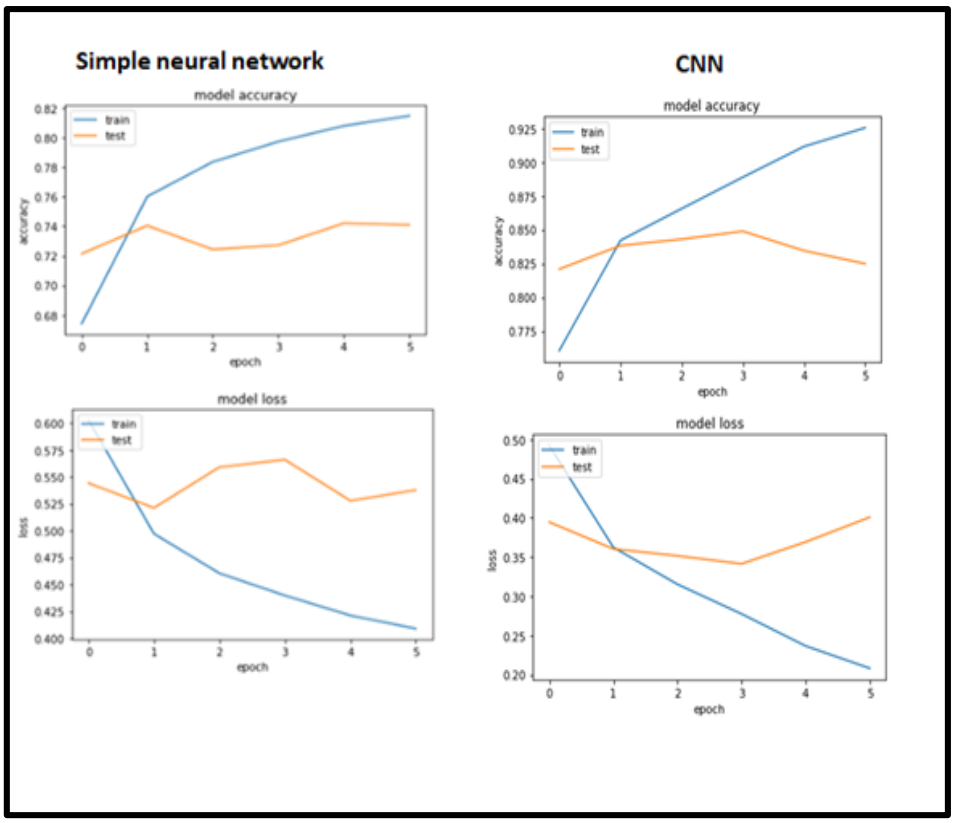
Le code est simple, nous créons un tableau comme en utilisant le package astropy :

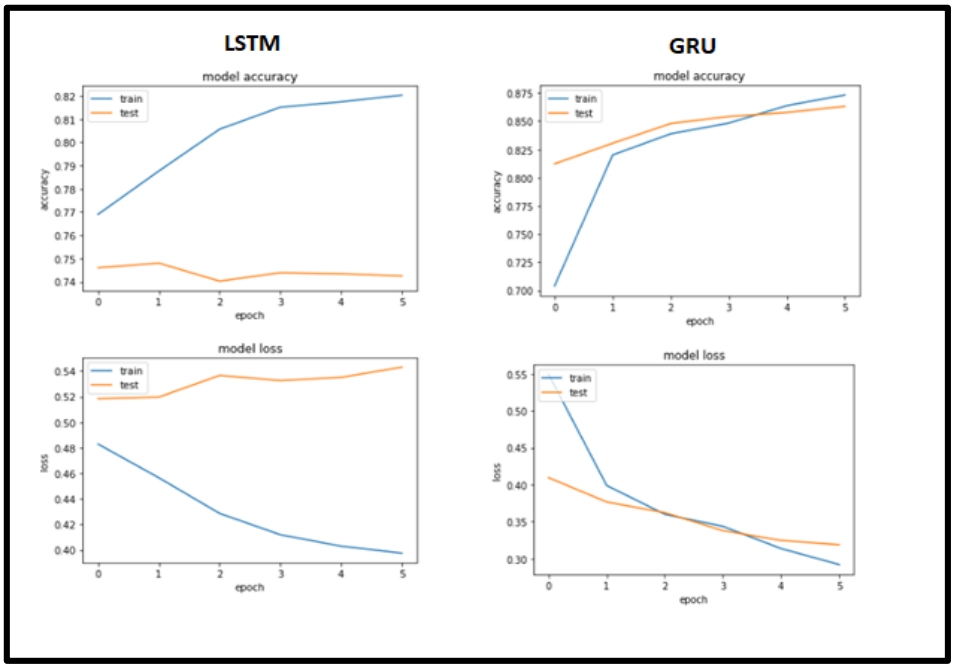
```
data_rows = [('training time(sec)',round(time1), round(time2), round(time3), round(time4)),
             ('','','','',''),
            ('accuracy %',round(100*accuracy_NN), round(100*accuracy_CNN), round(100*accuracy_LSTM), round(100*accuracy_GRU)),
             ('','','','','')]
t = Table(rows=data_rows, names=('metric','Simmple Neural Network', 'CNN', 'LSTM', 'GRU'))
print('------------------------------Tableau récapitulatif----------------------\n\n')
print(t)
```

Après avoir exécuter le code voilà les résultats :

```
--------------------------------Tableau récapitulatif-------------------------


      metric       Simmple Neural Network CNN LSTM GRU
----------------- ---------------------- --- ---- ---
training time(sec)                     13 293   19 627

        accuracy %                     74  83   74  74
```

Pour les pertes (losses) :





Conclusion

Ce projet nous a permis de mettre en pratique nos connaissances que nous avons acquises durant notre formation en Deep Learning et de les enrichir : nous avons exploré le domaine de l'analyse des sentiments qui, comme tous les autres domaines du traitement du langage naturel NLP, a connu une évolution majeure depuis les années 2000 et un grand intérêt depuis la naissance de l'apprentissage profond.
En fait, On a travaillé sur l'analyse des sentiments pour la critique des films en utilisant la classification du texte sous keras (bibliothèque d'apprentissage profond).
On a utilisé quatre différents types de réseaux neuronaux pour classer le sentiment du public à l'égard des différents films :

- **Réseau neuronal densément connecté (Réseau neuronal de base)**
- **Réseau neuronal convolutionnel (CNN)**
- **Réseau neuronal récurrents à mémoire court-terme et long terme (LSTM)**
- **Réseau neuronal récurrents à portes (GRU)**

La comparaison des résultats est faite à l'aide des différents métriques d'évaluation et elle nous a montré que LSTM, qui est une variante de RNN, a obtenu le meilleur résultat puisqu'il a prouvé son efficacité dans tous les mesures d'évaluation prises en considération.
Pour conclure, on peut dire qu'aujourd'hui Sentiment Analysis est devenue la meilleure porte d'entrée vers la compréhension des besoins des consommateurs aussi un outil vital pour les organisations pour l'amélioration de leurs services, elle gagne alors en popularité l'exploration d'opinion.

Références

- Python for NLP: Movie Sentiment Analysis using Deep Learning in Keras (stackabuse.com)
- Text classification using CNN. In this article, we are going to do… | by vijay choubey | Voice Tech Podcast | Medium
- What is Sentiment Analysis and Why Use It? | LITSLINK Blog
- http://dspace.univ-eloued.dz/bitstream/123456789/4271/1/Analyse%20des%20sentiments%20dans%20les%20r%C3%A9seaux%20sociaux%20en%20dialecte%20alg%C3%A9rien.pdf
- https://www.researchgate.net/publication/333607586_SENTIMENT_ANALYSIS_FOR_MOVIES_REVIEWS_DATASET_USING_DEEP_LEARNING_MODELS
- http://bib.univ-oeb.dz:8080/jspui/bitstream/123456789/6819/1/m%C3%A9moire%20lamamri%20nadia.pdf